Introduction .......................................................................................................................... 12
A brief Lua overview ............................................................................................................... 12
The CARA Object Model (CARM) ....................................................................................... 16
The CARA/Lua API Reference ............................................................................................ 18
  Global Variable cara .......................................................................................................... 18
  Class Atom ......................................................................................................................... 18
    getAtomType .................................................................................................................. 18
    getGroup ......................................................................................................................... 19
    getMagnitude .................................................................................................................. 19
    getName ........................................................................................................................... 19
    getNeighbours ................................................................................................................ 19
    getValue ........................................................................................................................... 19
  Class AtomGroup .............................................................................................................. 20
    getAtoms ......................................................................................................................... 20
    getName ........................................................................................................................... 20
  Class Buffer ........................................................................................................................ 20
    clone ................................................................................................................................ 20
    getAt ................................................................................................................................. 20
    getAtomType .................................................................................................................. 21
    getAtPpm ......................................................................................................................... 21
    getDimCount .................................................................................................................. 21
    getFreq ............................................................................................................................ 21
    getIndex ........................................................................................................................... 22
    getPpmRange .................................................................................................................. 22
    getSampleCount ............................................................................................................ 22
    resample .......................................................................................................................... 22
    setAt ................................................................................................................................ 23
  Class Button ....................................................................................................................... 23
    getText ............................................................................................................................... 23
    isOn .................................................................................................................................. 23
    isToggleButton ............................................................................................................... 23
    setAccel ............................................................................................................................ 23
    setCallback ..................................................................................................................... 23
    setIcon ............................................................................................................................. 23
    setText ............................................................................................................................. 23
  Class ButtonGroup ............................................................................................................ 24
    addButton ......................................................................................................................... 24
    setCallback ..................................................................................................................... 24
    setExclusive .................................................................................................................... 24
  Class Canvas ....................................................................................................................... 24
    begin ................................................................................................................................ 25
    commit ............................................................................................................................. 25
    drawContour ................................................................................................................... 25
    drawEllipse ..................................................................................................................... 25
    drawIcon .......................................................................................................................... 25
    drawImage ....................................................................................................................... 25
    drawLine .......................................................................................................................... 25
    drawPoint ......................................................................................................................... 25
    drawRect .......................................................................................................................... 26
    drawSlice .......................................................................................................................... 26
    drawText .......................................................................................................................... 26
    fillRect ............................................................................................................................ 26
    getBounding .................................................................................................................... 26
    lineTo .............................................................................................................................. 26
    moveTo .............................................................................................................................. 26
    setBgColor ..................................................................................................................... 26
    setBrush ........................................................................................................................... 26
    setCaption ....................................................................................................................... 26
    setFont ............................................................................................................................ 26
    setPen .............................................................................................................................. 27
    setSize ............................................................................................................................. 27
  Class CheckBox ................................................................................................................... 27
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>isChecked</td>
<td>27</td>
</tr>
<tr>
<td>setChecked</td>
<td>27</td>
</tr>
<tr>
<td>Class ComboBox</td>
<td>27</td>
</tr>
<tr>
<td>addItem</td>
<td>27</td>
</tr>
<tr>
<td>clear</td>
<td>27</td>
</tr>
<tr>
<td>getCurrentItem</td>
<td>27</td>
</tr>
<tr>
<td>getCurrentText</td>
<td>27</td>
</tr>
<tr>
<td>setCallback</td>
<td>28</td>
</tr>
<tr>
<td>setCurrentItem</td>
<td>28</td>
</tr>
<tr>
<td>setEditable</td>
<td>28</td>
</tr>
<tr>
<td>setEditText</td>
<td>28</td>
</tr>
<tr>
<td>Class ContourPlot</td>
<td>28</td>
</tr>
<tr>
<td>setBuffer</td>
<td>28</td>
</tr>
<tr>
<td>setLineWidth</td>
<td>28</td>
</tr>
<tr>
<td>setNegColor</td>
<td>29</td>
</tr>
<tr>
<td>setParams</td>
<td>29</td>
</tr>
<tr>
<td>setPosColor</td>
<td>29</td>
</tr>
<tr>
<td>toPoint</td>
<td>29</td>
</tr>
<tr>
<td>toPpm</td>
<td>29</td>
</tr>
<tr>
<td>Class Dialog</td>
<td>29</td>
</tr>
<tr>
<td>accept</td>
<td>29</td>
</tr>
<tr>
<td>exec</td>
<td>29</td>
</tr>
<tr>
<td>reject</td>
<td>29</td>
</tr>
<tr>
<td>Class DomDocument</td>
<td>30</td>
</tr>
<tr>
<td>getDocumentElement</td>
<td>30</td>
</tr>
<tr>
<td>getXml</td>
<td>30</td>
</tr>
<tr>
<td>saveToFile</td>
<td>30</td>
</tr>
<tr>
<td>Class DomElement</td>
<td>30</td>
</tr>
<tr>
<td>createElement</td>
<td>31</td>
</tr>
<tr>
<td>createText</td>
<td>31</td>
</tr>
<tr>
<td>getAttribute</td>
<td>31</td>
</tr>
<tr>
<td>getAttributes</td>
<td>31</td>
</tr>
<tr>
<td>getChildren</td>
<td>31</td>
</tr>
<tr>
<td>getFirstChild</td>
<td>31</td>
</tr>
<tr>
<td>getLastChild</td>
<td>31</td>
</tr>
<tr>
<td>getName</td>
<td>31</td>
</tr>
<tr>
<td>getNextSibling</td>
<td>31</td>
</tr>
<tr>
<td>getPrevSibling</td>
<td>32</td>
</tr>
<tr>
<td>hasAttribute</td>
<td>32</td>
</tr>
<tr>
<td>isElement</td>
<td>32</td>
</tr>
<tr>
<td>isText</td>
<td>32</td>
</tr>
<tr>
<td>normalize</td>
<td>32</td>
</tr>
<tr>
<td>removeAttribute</td>
<td>32</td>
</tr>
<tr>
<td>removeThis</td>
<td>32</td>
</tr>
<tr>
<td>setAttribute</td>
<td>32</td>
</tr>
<tr>
<td>setName</td>
<td>33</td>
</tr>
<tr>
<td>Class DomText</td>
<td>33</td>
</tr>
<tr>
<td>getNextSibling</td>
<td>33</td>
</tr>
<tr>
<td>getPrevSibling</td>
<td>33</td>
</tr>
<tr>
<td>getText</td>
<td>33</td>
</tr>
<tr>
<td>isElement</td>
<td>33</td>
</tr>
<tr>
<td>isText</td>
<td>34</td>
</tr>
<tr>
<td>removeThis</td>
<td>34</td>
</tr>
<tr>
<td>setText</td>
<td>34</td>
</tr>
<tr>
<td>Class Experiment</td>
<td>34</td>
</tr>
<tr>
<td>getCount</td>
<td>34</td>
</tr>
<tr>
<td>getPath</td>
<td>34</td>
</tr>
<tr>
<td>getResidueType</td>
<td>34</td>
</tr>
<tr>
<td>getSpectrumType</td>
<td>34</td>
</tr>
<tr>
<td>setResidueType</td>
<td>34</td>
</tr>
<tr>
<td>setSpectrumType</td>
<td>35</td>
</tr>
<tr>
<td>toString</td>
<td>35</td>
</tr>
<tr>
<td>Class</td>
<td>Members</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Frame</td>
<td>getContentsRect, getLineWidth, getMargin, getMidLineWidth, setFrameStyle, setLineWidth, setMargin, setMidLineWidth</td>
</tr>
<tr>
<td>GroupBox</td>
<td>setSpacing</td>
</tr>
<tr>
<td>Grid</td>
<td>setSpacing</td>
</tr>
<tr>
<td>Icon</td>
<td>setSize, getSize</td>
</tr>
<tr>
<td>Image</td>
<td>setSize, getSize</td>
</tr>
<tr>
<td>Label</td>
<td>setAlignment, setTitle, setBuddy, setIndent, setText</td>
</tr>
<tr>
<td>LineEdit</td>
<td>getText, isEdited, setCallback, setEdited, setReadOnly, setText</td>
</tr>
<tr>
<td>ListItem</td>
<td>createItem, destroy, getFirstChild, getListView, getNextSibling, getParent, isOpen, isSelected, seticon, setOpen, setSelected, setText</td>
</tr>
<tr>
<td>ListView</td>
<td>addColumn, clear, clearSelection, createItem, ensureVisible, getColumnCount, getMaxCount, getFirstChild, getFirstChild, getSelected, removeColumn, selectAll, setAllColsMarked, setCallback, setColumnTitle</td>
</tr>
<tr>
<td>Class</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------</td>
</tr>
<tr>
<td>MultiLineEdit</td>
<td>44</td>
</tr>
<tr>
<td>MenuBar</td>
<td>44</td>
</tr>
<tr>
<td>MainWindow</td>
<td>43</td>
</tr>
<tr>
<td>getAttr</td>
<td>49</td>
</tr>
<tr>
<td>getVol</td>
<td>49</td>
</tr>
<tr>
<td>getLabel</td>
<td>49</td>
</tr>
<tr>
<td>getId</td>
<td>49</td>
</tr>
<tr>
<td>getColor</td>
<td>49</td>
</tr>
<tr>
<td>getAttr</td>
<td>49</td>
</tr>
<tr>
<td>getAssig</td>
<td>48</td>
</tr>
<tr>
<td>setPen</td>
<td>48</td>
</tr>
<tr>
<td>setFont</td>
<td>48</td>
</tr>
<tr>
<td>moveTo</td>
<td>48</td>
</tr>
<tr>
<td>getBounding</td>
<td>48</td>
</tr>
<tr>
<td>setMultiSelection</td>
<td>42</td>
</tr>
<tr>
<td>setRootDecorated</td>
<td>43</td>
</tr>
<tr>
<td>setSorting</td>
<td>43</td>
</tr>
<tr>
<td>setSortIndicated</td>
<td>43</td>
</tr>
<tr>
<td>setStepSize</td>
<td>43</td>
</tr>
<tr>
<td>sort</td>
<td>43</td>
</tr>
<tr>
<td>MenuBar</td>
<td>44</td>
</tr>
<tr>
<td>clear</td>
<td>44</td>
</tr>
<tr>
<td>insertMenu</td>
<td>44</td>
</tr>
<tr>
<td>isEnabled</td>
<td>44</td>
</tr>
<tr>
<td>removeMenu</td>
<td>44</td>
</tr>
<tr>
<td>setCallback</td>
<td>44</td>
</tr>
<tr>
<td>setEnabled</td>
<td>44</td>
</tr>
<tr>
<td>MultiLineEdit</td>
<td>44</td>
</tr>
<tr>
<td>getLine</td>
<td>45</td>
</tr>
<tr>
<td>getLineCount</td>
<td>45</td>
</tr>
<tr>
<td>getText</td>
<td>45</td>
</tr>
<tr>
<td>insertLine</td>
<td>45</td>
</tr>
<tr>
<td>isEdited</td>
<td>45</td>
</tr>
<tr>
<td>setAlignment</td>
<td>45</td>
</tr>
<tr>
<td>setCallback</td>
<td>45</td>
</tr>
<tr>
<td>setEdited</td>
<td>45</td>
</tr>
<tr>
<td>setReadOnly</td>
<td>45</td>
</tr>
<tr>
<td>setReadonly</td>
<td>45</td>
</tr>
<tr>
<td>setText</td>
<td>45</td>
</tr>
<tr>
<td>setWordWrap</td>
<td>45</td>
</tr>
<tr>
<td>MyWidget</td>
<td>46</td>
</tr>
<tr>
<td>setAcceptFocus</td>
<td>46</td>
</tr>
<tr>
<td>setAutoBackground</td>
<td>46</td>
</tr>
<tr>
<td>setCallback</td>
<td>46</td>
</tr>
<tr>
<td>setMouseTracking</td>
<td>47</td>
</tr>
<tr>
<td>Painter</td>
<td>47</td>
</tr>
<tr>
<td>drawContour</td>
<td>47</td>
</tr>
<tr>
<td>drawEllipse</td>
<td>47</td>
</tr>
<tr>
<td>drawIcon</td>
<td>47</td>
</tr>
<tr>
<td>drawImage</td>
<td>47</td>
</tr>
<tr>
<td>drawLine</td>
<td>47</td>
</tr>
<tr>
<td>drawPoint</td>
<td>47</td>
</tr>
<tr>
<td>drawRect</td>
<td>47</td>
</tr>
<tr>
<td>drawSlice</td>
<td>47</td>
</tr>
<tr>
<td>drawText</td>
<td>47</td>
</tr>
<tr>
<td>fillRect</td>
<td>48</td>
</tr>
<tr>
<td>getBounding</td>
<td>48</td>
</tr>
<tr>
<td>lineTo</td>
<td>48</td>
</tr>
<tr>
<td>moveTo</td>
<td>48</td>
</tr>
<tr>
<td>setBrush</td>
<td>48</td>
</tr>
<tr>
<td>setFont</td>
<td>48</td>
</tr>
<tr>
<td>setPen</td>
<td>48</td>
</tr>
<tr>
<td>Peak</td>
<td>48</td>
</tr>
<tr>
<td>getAmp</td>
<td>48</td>
</tr>
<tr>
<td>getAssig</td>
<td>48</td>
</tr>
<tr>
<td>getAttr</td>
<td>49</td>
</tr>
<tr>
<td>getColor</td>
<td>49</td>
</tr>
<tr>
<td>getld</td>
<td>49</td>
</tr>
<tr>
<td>getLabel</td>
<td>49</td>
</tr>
<tr>
<td>getPos</td>
<td>49</td>
</tr>
<tr>
<td>getVol</td>
<td>49</td>
</tr>
<tr>
<td>setAttr</td>
<td>50</td>
</tr>
</tbody>
</table>
### Class PeakList
- `createPeak`
- `getAtomType`
- `getAttr`
- `getBatchList`
- `getDimCount`
- `getId`
- `getName`
- `getPeak`
- `getPeaks`
- `removePeak`
- `saveToFile`
- `setAmp`
- `setAssig`
- `setAttr`
- `setBatchList`
- `setColor`
- `setLabel`
- `setName`
- `setPos`
- `setVol`
- `getSequence`
- `getPeakLists`
- `getPeakList`
- `getFragment`
- `getCombinedFragment`
- `getAttr`
- `addCandidate`
- `setText`
- `setIcon`
- `setCallback`
- `removeItem`
- `setVol`
- `setName`
- `setLabel`
- `setBatchList`
- `setAttr`
- `setAmp`
- `saveToFile`
- `removePeak`
- `getPeaks`
- `getPeak`
- `getName`
- `getId`
- `getDimCount`
- `createPeak`

### Class PopupMenu
- `clear`
- `getText`
- `insertItem`
- `insertSeparator`
- `insertSubmenu`
- `isChecked`
- `isEnabled`
- `popup`
- `removeItem`
- `setAccel`
- `setCallback`
- `setChecked`
- `setEnabled`
- `setIcon`
- `setText`
- `setWhatsThis`

### Class Project
- `addCandidate`
- `assignSpin`
- `assignSystem`
- `createSpin`
- `createSystem`
- `getAttr`
- `getCombinedFragment`
- `getFragment`
- `getName`
- `getPeakList`
- `getPeakLists`
- `getResidue`
- `getSequence`
- `getSpectra`
- `getSpectrum`
- `getSpin`
- `getSpinLinks`
- `getSpins`
- `getSystem`
- `getSystems`
- `getTolerance`
- `linkSpins`
- `linkSystems`
<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResidueType</td>
<td>67</td>
</tr>
<tr>
<td>Residue</td>
<td>66</td>
</tr>
<tr>
<td>RadioButton</td>
<td>62</td>
</tr>
<tr>
<td>ProtonList</td>
<td>61</td>
</tr>
<tr>
<td>getSucc</td>
<td>66</td>
</tr>
<tr>
<td>getPred</td>
<td>66</td>
</tr>
<tr>
<td>getId</td>
<td>66</td>
</tr>
<tr>
<td>setAttr</td>
<td>65</td>
</tr>
<tr>
<td>removeRecord</td>
<td>65</td>
</tr>
<tr>
<td>getSpectrumTypes</td>
<td>65</td>
</tr>
<tr>
<td>getResidueTypes</td>
<td>65</td>
</tr>
<tr>
<td>getResidueType</td>
<td>65</td>
</tr>
<tr>
<td>getRecord</td>
<td>64</td>
</tr>
<tr>
<td>getProjects</td>
<td>64</td>
</tr>
<tr>
<td>getProject</td>
<td>64</td>
</tr>
<tr>
<td>getAuthor</td>
<td>64</td>
</tr>
<tr>
<td>getAttr</td>
<td>64</td>
</tr>
<tr>
<td>setAttr</td>
<td>63</td>
</tr>
<tr>
<td>setAttrRecord</td>
<td>63</td>
</tr>
<tr>
<td>matchResidue</td>
<td>58</td>
</tr>
<tr>
<td>matchSpin</td>
<td>58</td>
</tr>
<tr>
<td>matchSystems</td>
<td>59</td>
</tr>
<tr>
<td>removeCandidate</td>
<td>59</td>
</tr>
<tr>
<td>removeSpin</td>
<td>59</td>
</tr>
<tr>
<td>removeSystem</td>
<td>59</td>
</tr>
<tr>
<td>setAttr</td>
<td>59</td>
</tr>
<tr>
<td>setLabel</td>
<td>59</td>
</tr>
<tr>
<td>setShift</td>
<td>60</td>
</tr>
<tr>
<td>setSystemType</td>
<td>60</td>
</tr>
<tr>
<td>setTolerance</td>
<td>60</td>
</tr>
<tr>
<td>setValue</td>
<td>60</td>
</tr>
<tr>
<td>unassignSpin</td>
<td>60</td>
</tr>
<tr>
<td>unassignSystem</td>
<td>60</td>
</tr>
<tr>
<td>unlinkSpins</td>
<td>60</td>
</tr>
<tr>
<td>unlinkSystems</td>
<td>60</td>
</tr>
<tr>
<td>ProtonList</td>
<td>61</td>
</tr>
<tr>
<td>setDefault</td>
<td>61</td>
</tr>
<tr>
<td>setFlat</td>
<td>62</td>
</tr>
<tr>
<td>setOn</td>
<td>62</td>
</tr>
<tr>
<td>setPopup</td>
<td>62</td>
</tr>
<tr>
<td>setToggleButton</td>
<td>62</td>
</tr>
<tr>
<td>RadioButton</td>
<td>62</td>
</tr>
<tr>
<td>isChecked</td>
<td>62</td>
</tr>
<tr>
<td>setChecked</td>
<td>62</td>
</tr>
<tr>
<td>Record</td>
<td>62</td>
</tr>
<tr>
<td>getAttr</td>
<td>63</td>
</tr>
<tr>
<td>getAttrs</td>
<td>63</td>
</tr>
<tr>
<td>getId</td>
<td>63</td>
</tr>
<tr>
<td>setAttr</td>
<td>63</td>
</tr>
<tr>
<td>Repository</td>
<td>63</td>
</tr>
<tr>
<td>createRecord</td>
<td>64</td>
</tr>
<tr>
<td>getAttr</td>
<td>64</td>
</tr>
<tr>
<td>getAuthor</td>
<td>64</td>
</tr>
<tr>
<td>getProject</td>
<td>64</td>
</tr>
<tr>
<td>getProjects</td>
<td>64</td>
</tr>
<tr>
<td>getRecord</td>
<td>64</td>
</tr>
<tr>
<td>getRecords</td>
<td>65</td>
</tr>
<tr>
<td>getResidueType</td>
<td>65</td>
</tr>
<tr>
<td>getResidueTypes</td>
<td>65</td>
</tr>
<tr>
<td>get SpectrumType</td>
<td>65</td>
</tr>
<tr>
<td>get SpectrumTypes</td>
<td>65</td>
</tr>
<tr>
<td>get System Types</td>
<td>65</td>
</tr>
<tr>
<td>remove Record</td>
<td>65</td>
</tr>
<tr>
<td>Residue</td>
<td>66</td>
</tr>
<tr>
<td>getAttr</td>
<td>66</td>
</tr>
<tr>
<td>getId</td>
<td>66</td>
</tr>
<tr>
<td>getPred</td>
<td>66</td>
</tr>
<tr>
<td>getSucc</td>
<td>66</td>
</tr>
<tr>
<td>getSystem</td>
<td>66</td>
</tr>
<tr>
<td>getType</td>
<td>66</td>
</tr>
<tr>
<td>getValue</td>
<td>66</td>
</tr>
<tr>
<td>setAttr</td>
<td>67</td>
</tr>
<tr>
<td>ResidueType</td>
<td>67</td>
</tr>
<tr>
<td>getAtom</td>
<td>67</td>
</tr>
</tbody>
</table>

© 2003 by DATONAL AG All rights reserved
getAtomGroup
getAtomGroups
getAtoms
getAttr
getId
getLetter
getName
getShort
getSystemType
setAttr

Class ScrollView
addChild
center
guaranteedVisible
moveChild
removeChild
resizeChild
scrollBy
scrollTo
setCallback

getDimension
getMinMax
recalculateMinMax
setAutoScale
setBuffer
setColor
setMinMax
toPoint
toPpm

getAt
getAtomType
getAtPoint
getAtPpm
getAttr
getDimCount
getAddress
getDouble
getFolding
getFreq
getId
ggetIndex
getLabel
getName
getPlanePpm
getPlaneRange
getPpmRange
getRFFreq
getSampleCount
getSlicePpm
getSliceRange
ggetType
setAttr

Class Spectrum

getAt
getAtomType
getAtPoint
getAtPpm
getAttr
getDimCount
getAddress
getAddress
getDouble
getFolding
getFreq
getId
ggetIndex
getLabel
getName
getPlanePpm
getPlaneRange
getPpmRange
getRFFreq
getSampleCount
getSlicePpm
getSliceRange
getype
setAttr

Class SpectrumType

getAtomType
getAttr
getDimCount
getAddress
getAddress
getDouble
getLabels
getName
getStep
getStepCount ................................................................. 77
isNoesy ...................................................................... 77
setAttr ..................................................................... 77

Class Spin .............................................................. 78
getAtomType............................................................. 78
getAttr ................................................................. 78
getId ........................................................................ 78
getLabel ................................................................... 78
getLink ....................................................................... 78
getLinks ................................................................. 79
getShift ...................................................................... 79
getShifts .................................................................... 79
getSystem .............................................................. 79
setAttr ..................................................................... 79

Class SpinLink ............................................................. 79
getAmp ...................................................................... 79
getAttr ................................................................. 80
gerLeft ....................................................................... 80
gerRight ...................................................................... 80
isVisible ................................................................ 80
setAttr ..................................................................... 80

Class SpinSystem ............................................................. 80
getAttr ................................................................. 81
gerCandidates ........................................................... 81
getId ........................................................................ 81
getPred ...................................................................... 81
getResidue ............................................................ 81
gerSpins ..................................................................... 81
gerSucc ....................................................................... 81
gerSystemType ............................................................... 81
isAcceptable ........................................................... 82
setAttr ..................................................................... 82

Class Splitter ................................................................... 82
setOrientation ............................................................. 82

Class SystemType ............................................................. 81
gerAmp ...................................................................... 81
gerAttr ................................................................. 82
gerCandidates ........................................................... 82
gerId ........................................................................ 82
gerName ................................................................. 82
setAttr ..................................................................... 82

Class TabWidget .......................................................... 83
addTab ....................................................................... 83
gerCurrentPage .......................................................... 83
setCallback ............................................................. 83
setMargin .................................................................. 83
setTabEnabled ........................................................... 83
setTabPosition ........................................................... 83
setTitle ..................................................................... 83
showPage ................................................................. 83

gClass TextView ........................................................... 84
gerText ...................................................................... 84
setText ..................................................................... 85

Class VBox ............................................................... 85
Class Widget .............................................................. 85
clearFocus .............................................................. 85
close ........................................................................ 85
destroy ................................................................. 85
gerBgColor .............................................................. 85
gerCaption .............................................................. 85
gerData ...................................................................... 86
gerFgColor .............................................................. 86
gerFont ...................................................................... 86
gerFrameGeometry .......................................................... 86
getGeometry ................................................................. 86
getMaximumSize ...................................................... 86
getMinimumSize ......................................................... 86
getParentWidget ....................................................... 86
getSize ................................................................. 86
getTopLevelWidget .................................................. 86
hasFocus ........................................................... 87
hide ................................................................. 87
isActiveWindow ..................................................... 87
isDesktop .......................................................... 87
isEnabled .......................................................... 87
isFocusEnabled ..................................................... 87
isHidden .......................................................... 87
isMaximized ......................................................... 87
isMinimized ........................................................ 87
isModal ............................................................ 88
isPopup ............................................................ 88
isTopLevel .......................................................... 88
isVisible ........................................................... 88
lower ............................................................... 88
mapFromGlobal ..................................................... 88
mapFromParent ..................................................... 88
mapToGlobal ........................................................ 89
mapToParent ........................................................ 89
move .............................................................. 89
raise .............................................................. 89
resize .............................................................. 89
setActiveWindow ................................................... 89
setBgColor .......................................................... 89
setCallback ........................................................ 89
setCaption .......................................................... 90
setEnabled .......................................................... 90
setFixedSize .......................................................... 90
setFocus ............................................................ 90
setFont ............................................................. 90
setMaximumSize ................................................... 90
setMinimumSize ..................................................... 90
setUpdatesEnabled .................................................. 91
show ............................................................... 91
showFullScreen ..................................................... 91
showMaximized ..................................................... 91
showMinimized ..................................................... 91
showNormal ....................................................... 91
update ............................................................ 91
updateAll .......................................................... 92
updateGeometry .................................................... 92
Class WidgetStack .................................................. 92
addWidget .......................................................... 92
getTopWidget ......................................................... 92
setTopWidget ........................................................ 92
Library dlg ............................................................ 92
beginProgress ....................................................... 92
createCanvas ....................................................... 93
createContour ....................................................... 93
createlcon .......................................................... 93
createSlice .......................................................... 94
doscript ............................................................ 94
endProgress .......................................................... 95
currentDir ........................................................... 95
decimal .............................................................. 95
getInteger ........................................................... 95
genOpenFileName .................................................. 96
getSaveFileName ................................................................. 36
getSymbol ................................................................. 37
getText ................................................................. 37
loadIcon ................................................................. 37
loadImage ................................................................. 37
logError ................................................................. 37
logInfo ................................................................. 38
logWarning ................................................................. 38
require ................................................................. 38
selectColor ................................................................. 39
selectFont ................................................................. 39
setCurrentDir ................................................................. 39
showError ................................................................. 39
showInfo ................................................................. 40
showWarning ................................................................. 40
updateProgress ................................................................. 40
Library gui ................................................................. 41
createButtonGroup ................................................................. 41
createCheckBox ................................................................. 41
createComboBox ................................................................. 41
createDialog ................................................................. 41
createFrame ................................................................. 41
createGrid ................................................................. 41
createGroupBox ................................................................. 41
createHBox ................................................................. 41
createLabel ................................................................. 41
createLineEdit ................................................................. 41
createMainView ................................................................. 42
createMultiLineEdit ................................................................. 42
createPopupMenu ................................................................. 42
createPushButton ................................................................. 42
createRadioButton ................................................................. 42
createScrollView ................................................................. 42
createSplitter ................................................................. 42
createTabWidget ................................................................. 42
createTextView ................................................................. 42
createVBox ................................................................. 42
createWidget ................................................................. 42
createWidgetStack ................................................................. 43
cursorPos ................................................................. 43
Library spec ................................................................. 43
composeLabel ................................................................. 43
createExperiment ................................................................. 43
createPeakList ................................................................. 43
createProtonList ................................................................. 43
decomposeLabel ................................................................. 43
openPeakList ................................................................. 43
openProtonList ................................................................. 43
openSpectrum ................................................................. 44
Library xml ................................................................. 44
createDocument ................................................................. 44
openDocument ................................................................. 44
parseDocument ................................................................. 44
Introduction

CARA (Computer Aided Resonance Assignment) is the application for the analysis of NMR spectra and computer aided resonance assignment developed at and used by 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. In this manual we will focus on the CARA/Lua Application Programming Interface which allows custom scripts to access the CARA Object Model.

For a detailed description of the Lua programming language used by CARA, please consult the "Lua 5.0 Reference Manual" (April 11, 2003) by Roberto Ierusalimschy, Luiz Henrique de Figueiredo and Waldemar Celes. It can be found at http://www.lua.org/docs.html or http://www.lua.org/ftp/refman-5.0.pdf. This manual will reference the Lua reference manual as "LRM".

The user interface of CARA is documented by means of a Wiki, which can be found at http://www.mol.biol.ethz.ch:8060/CARA/Wiki. There will be other documentation in near future about the use cases and concepts of CARA. A paper about the novel integration algorithm built into CARA is about to be published.

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.

Lua has straight forward syntax and semantics. It is a language with dynamic typing, which means you don't have to specify a data type for your variable as you might be used to in Pascal or Fortran. You create a variable by just assigning a value to a name.

```lua
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. 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:

```lua
--[[ 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 build 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 table to represent both records and arrays. Contrary 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"

-- index by value of a variable (=3)

valueA = 3

-- index by string, like a record

tableA[ "abc" ] = 3e-10

-- short form looks even more record like

tableA.abc = "hello"

-- a constructor with initialization


-- tables can be nested

tableA.sub = tableB

-- 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 render 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:

```lua
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:

```lua
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 automatic 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 heavy 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 thousands of elements) to a global variable. The garbage collector of Lua will keep this huge table in memory as long as a variable is referencing 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
eamples.

-- 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 CARA Object Model (CARM)

The CARA Object Model (CARM) contains all entities CARA has to manage together 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 be productive). The rest of the model and an adapter to all CARA scope windows will be available in near future.

The following diagram (Figure 1) 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 themselves 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 1: CARM Core Classes](image)

Additionally CARA/Lua offer 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 near future. With the current release, you can already display planes and slices within your windows anyway, but you have to take care of interactivity by yourself (shouldn't be a problem to a hard core scripting crack).
Figure 2: User Interface Controls

Figure 2 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 3: 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 4: 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 above). 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

ca. 500 functions.

Global Variable 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.

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

Class Atom

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 neighbours (this relation represents covalent bonds) and can be part of an AtomGroup.

getAtomType

Parameters

None.

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>Atom symbol according to periodic system of elements.</td>
</tr>
</tbody>
</table>
getGroup

Parameters
None.

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AtomGroup or nil</td>
<td></td>
<td>The group this atom belongs to if any, e.g. QB for HB2 or HB3</td>
</tr>
</tbody>
</table>

getMagnitude

Parameters
None.

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The number of real atoms represented by this object, i.e. number 3 for the H in CH3.</td>
</tr>
</tbody>
</table>

getName

Parameters
None.

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the atom, unique within its molecule, z.B. CA</td>
</tr>
</tbody>
</table>

getNeighbours

Parameters
None.

Return Values

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

getValue

Parameters
None.

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number or nil</td>
<td></td>
<td>Mean value, if available, otherwise nil.</td>
</tr>
<tr>
<td>2</td>
<td>number or nil</td>
<td></td>
<td>Standard deviation, if value is available</td>
</tr>
</tbody>
</table>
Class AtomGroup

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 recognized by their name (i.e. QB), which must be unique within a ResidueType.

getAtoms

**Parameters**

None.

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Table[ string, Atom ]</td>
<td></td>
<td>A table with all atoms belonging to this group. The key of the table is the name of the atom in the value.</td>
</tr>
</tbody>
</table>

getName

**Parameters**

None.

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the atom group, unique within its molecule, z.B. QB</td>
</tr>
</tbody>
</table>

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.

clone

**Parameters**

None.

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Buffer</td>
<td></td>
<td>Creates a new Buffer as a complete copy of this Buffer.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..d</td>
<td>number</td>
<td></td>
<td>Sample index 1..n for each of the d dimensions of the Buffer.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
</table>
### Nr. Type Opt. Description

| 1 | number | Amplitude at the given sample index. |

#### See Also

Spectrum:getPlanePpm, Spectrum:getSlicePpm, Buffer:getSampleCount

---

### getAtomType

#### Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Dimension 1 to 2, depending on getDimCount</td>
</tr>
</tbody>
</table>

#### Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>Atom symbol according to periodic system of elements.</td>
</tr>
</tbody>
</table>

#### See Also

Buffer:getDimCount

---

### 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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..n</td>
<td>number</td>
<td></td>
<td>PPM value for each dimension of the Buffer.</td>
</tr>
</tbody>
</table>

#### Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Amplitude at the given PPM point.</td>
</tr>
</tbody>
</table>

#### See Also

Spectrum:getPlanePpm, Spectrum:getSlicePpm

---

### getDimCount

#### Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Number of dimensions of this Buffer: 1 or 2</td>
</tr>
</tbody>
</table>

### getFreq

#### Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Dimension 1 to 2, depending on getDimCount</td>
</tr>
<tr>
<td>2</td>
<td>number</td>
<td></td>
<td>Sample index 1..n</td>
</tr>
</tbody>
</table>

#### Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>PPM position corresponding to the given sample index</td>
</tr>
</tbody>
</table>
See Also
Buffer:getDimCount

getIndex

Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Dimension 1 to 2, depending on getDimCount</td>
</tr>
<tr>
<td>2</td>
<td>number</td>
<td></td>
<td>PPM position</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Sample index corresponding to the given PPM position</td>
</tr>
</tbody>
</table>

See Also
Buffer:getDimCount

getPpmRange

Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Dimension 1 to 2, depending on getDimCount</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>PPM value at sample index 0 (middle of sample)</td>
</tr>
<tr>
<td>2</td>
<td>number</td>
<td></td>
<td>PPM value at sample index n-1 (middle of sample)</td>
</tr>
</tbody>
</table>

See Also
Buffer:getDimCount, Buffer:getSampleCount

getSampleCount

Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Dimension 1 to 2, depending on getDimCount</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Number of samples in the given dimensions.</td>
</tr>
</tbody>
</table>

See Also
Buffer:getDimCount

resample

Params: number (width), number (height)
Returns: none.
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 original image processing algorithm (scale reduction).
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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..d</td>
<td>number</td>
<td></td>
<td>Sample index 1..n for dimension 1..d.</td>
</tr>
<tr>
<td>d+1</td>
<td>number</td>
<td></td>
<td>Amplitude to be set at the sample point</td>
</tr>
</tbody>
</table>

Return Values
None.

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

Class Button
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.

getText
Params: none
Returns: string

isOn
Params: none
Returns: boolean

isToggleButton
Params: none
Returns: boolean

setAccel
Params: string (acceleration code, e.g. "Ctrl+S")
Returns: none

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

<table>
<thead>
<tr>
<th>event</th>
<th>function( self )</th>
<th>Params: Button</th>
<th>Returns: none</th>
</tr>
</thead>
<tbody>
<tr>
<td>gui.event.Clicked</td>
<td>function( self )</td>
<td>Params: Button</td>
<td>Returns: none</td>
</tr>
<tr>
<td>gui.event.Toggled</td>
<td>function( self, state )</td>
<td>Params: Button, boolean</td>
<td>Returns: none</td>
</tr>
</tbody>
</table>

setIcon
Params: Icon
Returns: none

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

```
  Button group
  └ First
  └ Second
  └ Third
```

**addButton**

Params: Button, number (optional id)
Returns: number (id)

**setCallback**

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

- `gui.event.Clicked`  
  function( self, buttonId )  
  Params: ButtonGroup, number
  Returns: none

**setExclusive**

Params: boolean (default true)
Returns: none

**Class Canvas**
A Canvas is an independant 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 or 20 TWIPs per point). The coordinate system has its origin in
the upper left corner of the canvas. On screen a point corresponds to one pixel in general.
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 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 )
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 ] ) -- draw curve
end
canvas:commit()

begin
No params. 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().

commit
No params. Ends a painting transaction started by begin(). Only now the drawing becomes visible. It is an error to call this method without a previous call to begin().

drawContour
Params: ContourPlot, x, y, w, h
Draws the given contour plot at the given point position. The plot scales into the area given by w, h.

drawEllipse
Parameter: x, y, w, h. Draws an ellipse with the current pen and brush at center point x/y with extension w, h.

drawIcon
Params: Icon, number (pos. x), number (pos. y)
Returns: none

drawImage
Parameters: Image, x, y, w (opt.), h (opt.)
Draws the given image at the given point position. If w and h are left out, the image is painted with its original dimension. You can stretch the image to w and h if needed.

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

drawPoint
Parameter: x, y. Draws a single point at x/y using the current pen.
drawRect

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

drawSlice

Params: SlicePlot, x, y, w, h
Draws the given slice plot at the given point position. The plot scales into the area given by w, h.

drawText

Parameter: x, y, string. 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.

fillRect

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

getBounding

Parameter: string. Returns w, h. This function accepts a text string and returns its width and height it will have when drawn on screen with the current font.

lineTo

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

moveTo

Parameter: x, y. This function sets the current position to x/y without drawing anything. This is useful when calling lineTo afterwards.

setBgColor

Parameter: string. 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.).

setBrush

Parameter: string or 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.

setCaption

Parameter: string. This function sets the caption of the canvas window to string.

setFont

Parameter: string (font name), number (point size), boolean (bold), boolean (italic). This function sets the current font, which affects the following calls to drawText.
setPen

Parameter: string (color), number (width). 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.

setSize

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

**Class CheckBox**

CheckBox inherits all features of Button.

```
☑ First
☐ Second
☐ Third
```

isChecked

Params: none
Returns: boolean

setChecked

Params: boolean (default true)
Returns: none

**Class ComboBox**

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

```
Choice 1
```

addItem

Params: string (label text), Icon (optional)
Returns: number (index of new item)

clear

Params: none
Returns: none

gGetCurrentItem

Params: none
Returns: number (index of selected item, 1..n)

gGetCurrentText

Params: none
Returns: string (text of selected item)
setCallback

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

- gui.event.Changed function( self, text )
  Params: ComboBox, string
  Returns: none

- gui.event.Activated function( self, index )
  Params: ComboBox, number
  Returns: none

setCurrentItem

Params: number (index to select, 1..n)
Returns: none

setEditable

Params: boolean (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.

setEditText

Params: string
Returns: none
Use this function to preset the text of the edit part of the control.

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 committed drawings (see Canvas:commit).

```lua
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()
```

setBuffer

Params: Buffer
Returns: none.
This function sets the sample buffer to be contoured. The buffer must be two dimensional.

setLineWidth

Params: number (line width in points)
Returns: none.
Set the line width of the pen used to draw the contour lines.
setNegColor

Params: 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.).

setParams

Params: number (factor), number (threshold), string (optional, "+" or ")
Sets the contour parameters for this object. To see the effect, the plot has to be redrawn with Canvas:drawContour. Leave out the third parameter, if you want to draw pos. and neg. amplitudes.

setPosColor

Params: 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.).

toPoint

Params: number (dim. 1 or 2), number (origin on canvas), number (span on canvas), number (ppm)
Returns: number (point position of given ppm value)

toPpm

Params: number (dim. 1 or 2), number (origin on canvas), number (span on canvas), number (point)
Returns: number (ppm value of given point position)

Class Dialog

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.

accept

Params: none.
Returns: none.
Use this function to close the modal dialog, returning exec with a true value. The function has no effect when the dialog is not shown.

exec

Params: none.
Returns: boolean
This function 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 functions when the user presses them. In these functions you can call accept() (in which case exec returns true) or reject() (in which case exec returns false). The function also returns false if the user closes the dialog by other means (e.g. the "x" button in the title bar on windows).

reject

Params: none.
Returns: none.
Use this function to close the modal dialog, returning exec with a false value. The function has no effect when the dialog is not shown.
**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 diagram shows the objects with their relations to each other.

![Diagram of DomDocument, DomElement, and DomText 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.

```lua
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
```

**getDocumentElement**

*Params: 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.

**getXml**

*Params: 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.

**saveToFile**

*Params: 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 being DomElements and DomTexts polymorphically. The attribute names are unique within a DomElement, whereas the child DomElements can have an arbitrary name (because their order is relevant).

**createElement**

*Params:* 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.

**createText**

*Params:* 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 succeeding text elements to one text using the function `normalize`.

**getAttribute**

*Params:* string (name)
*Returns:* string or nil
Use this function to access an attribute of this element.

**getAttributes**

*Params:* none.
*Returns:* Table [ string (name), string (value) ]
This function returns the list of all attributes of this element.

**getChildren**

*Params:* none.
*Returns:* Table [ number (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.

**getFirstChild**

*Params:* none.
*Returns:* DomElement or DomText or nil
This function returns the first child of this element (equivalent to `DomElement:getChildren()[1]`).

**getLastChild**

*Params:* 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).

**getName**

*Params:* none
*Returns:* string (the tag name of this element)

**getNextSibling**

*Params:* 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.

getPrevSibling

Params: 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.

hasAttribute

Params: string (attribute name)
Returns: boolean
This function returns true, if an attribute with the given name is defined in this element.

isElement

Params: 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.

isText

Params: 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.

normalize

Params: none
Returns: none
Use this function to combine all successive DomText elements of the children list of this element into one DomText.

removeAttribute

Params: 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.

removeThis

Params: 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.

setAttribute

Params: string (attribute name), string (attribute value)
Returns: none.
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.
setName

Params: 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.

![Diagram showing instance relationship]

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".

getNextSibling

Params: 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.

getPrevSibling

Params: 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.

getText

Params: none
Returns: string (the text)

isElement

Params: 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.
isText

Params: 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.

removeThis

Params: 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.

setText

Params: string
Returns: none.

**Class Experiment**

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.

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

getCode

Params: none
Returns: number (number of rows in the path table)

getPath

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

getResidueType

Params: none
Returns: ResidueType or nil

getSpectrumType

Params: none
Returns: SpectrumType or nil

setResidueType

Params: ResidueType
Returns: none.
setSpectrumType  
Params: SpectrumType  
Returns: none.

toString  
Params: none  
Returns: string (a pretty printed text of the path table)

**Class Frame**
This class inherits all the features of 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.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>lineWidth()</th>
<th>midLineWidth()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box + Plain</td>
<td>Box + Raised</td>
<td>Box + Sunked</td>
<td>Panel + Plain</td>
<td>Panel + Raised</td>
<td>Panel + Sunked</td>
</tr>
<tr>
<td>WinPanel + Plain</td>
<td>WinPanel + Raised</td>
<td>WinPanel + Sunked</td>
<td>HLine + Plain</td>
<td>HLine + Raised</td>
<td>HLine + Sunked</td>
</tr>
<tr>
<td>VLine + Plain</td>
<td>VLine + Raised</td>
<td>VLine + Sunked</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

getContentsRect  
Params: none.  
Returns: number (x), number (y), number (width), number (height)  
This is the rectangle within the frame where the contents resides.

gGetLineWidth  
Params: none.  
Returns: number (points)

gGetMargin  
Params: none.  
Returns: number (points)
getMidLineWidth

Params: none.
Returns: number (points)

setFrameStyle

Params: number (frame style code), number (optional, shadow style code, default Plain)
Returns: none.
The following shadow style codes are available: Frame.style.Plane, Frame.style.Raised and Frame.style.Sunken (see the diagram above).

setLineWidth

Params: number (points)
Returns: none.

setMargin

Params: number (points)
Returns: none.

setMidLineWidth

Params: number (points)
Returns: none.

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

<table>
<thead>
<tr>
<th>One</th>
<th>Two</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three</td>
<td>Four</td>
</tr>
<tr>
<td>Five</td>
<td></td>
</tr>
</tbody>
</table>

setSpacing

Params: number (points)
Returns: none

Class GroupBox
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

Params: number (points)
Returns: none

setAlignment

Params: number (alignment code)
Returns: none
Sets the alignment of the title. Use the values gui.align.Left, gui.align.HCenter or gui.align.Right.

setColumns

Params: 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.

setOrientation

Params: number (gui.Horizontal or gui.Vertical)
Returns: none

setTitle

Params: string (new title)
Returns: none

**Class HBox**
This class inherits all features of Frame. Use it to horizontally align its child widgets.

```
One Two Three Four Five
```

setSpacing

Params: number (points)
Returns: none

**Class Icon**
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.

getSize

Params: none
Returns: number (width in points), number (height in points)

**Class Image**
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.
```lua
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()
```

**getSize**

**Params:** none  
**Returns:** number (width in points), number (height in points)

---

**Class Label**

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

```
This is a Label
it spans multiple lines
```

**setAlignment**

**Params:** number (horiz. alignment), number (vertic. alignment), boolean (expand tabs, default false), boolean (word break, default false)  
**Returns:** none  
For horizontal alignment use gui.align.Left, gui.align.Right and gui.align.HCenter  
For vertical alignment use gui.align.Top, gui.align.Bottom and gui.align.VCenter

**setBuddy**

**Params:** Widget  
**Returns:** none  
Use this function to associate a label with another widget. If you activate a shortcut of the label text, the focus goes to the buddy widget.

**setIndent**

**Params:** number (points)  
**Returns:** none

**setText**

**Params:** 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**

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

```
Hello
```
getText

Params: none
Returns: string

isEdited

Params: none
Returns: boolean
This flag becomes true when the text is changed either by the user or by setText. See also setEdited().

setCallback

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

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

setEdited

Params: boolean
Returns: none
Sets the edited flag of the widget.

setReadOnly

Params: boolean
Returns: none

setText

Params: string
Returns: none

Class ListItem

A ListItem is an item of a ListView. It is created by ListView:createItem() or ListItem:createItem().

createItem

Params: 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.

destroy

Params: none
Returns: none
Remove this item from the list. The item vanishes immediately from screen. The Lua reference becomes useless.

getFirstChild

Params: none
Returns: ListItem or nil
Use this function to iterate over the child items.
**getListView**

Params: none
Returns: ListView (the ListView this item belongs to)

**getNextSibling**

Params: none
Returns: ListItem or nil
Use this function to iterate over all items of a hierarchy level.

**getParent**

Params: none
Returns: ListItem or nil

**isOpen**

Params: none
Returns: boolean

**isSelected**

Params: none
Returns: boolean

**setIcon**

Params: number (column id, 1..n), Icon
Returns: none

**setOpen**

Params: boolean (default true)
Returns: none

**setSelected**

Params: boolean (default true)
Returns: none

**setText**

Params: number (column id, 1..n), string
Returns: none

**Class ListView**

This class inherits all features of ScrollView. You can use ListView to present tables, lists or trees to the user.
addColumn

Params: string (column title)
Returns: number (index of new column, 1..n)

clear

Params: none
Returns: none
Remove all items from the list.

clearSelection

Params: none
Returns: none
Deselect all items.

createItem

Params: 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.

ensureVisible

Params: ListItem
Returns: none
Scroll the list so the given item becomes visible.

columnCount

Params: none
Returns: number

getFirstChild

Params: none
Returns: ListItem or nil

getSelected

Params: none
Returns: ListItem or nil
This function returns the currently selected item. Works only in single-selection-mode.

removeColumn

Params: number (column index, 1..n)
Returns: none

selectAll

Params: boolean (default true)
Returns: none
Use this function to either select (true) or unselect (false) all items of the list.

setAllColsMarked

Params: boolean (default true)
Returns: none
If true, the selection mark covers all columns. If false, only the first column is covered.

setCallback

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

gui.event.RightPressed function( self, item, x, y, column )
Params: ListView, ListItem or nil, number, number, number
Returns: none

gui.event.Selection function( self )
Params: ListView
Returns: none

gui.event.Clicked function( self, item, x, y, column )
Params: ListView, ListItem or nil, number, number, number
Returns: none

gui.event.DblClicked function( self, item )
Params: ListView, ListItem or nil
Returns: none

gui.event.Return function( self, item )
Params: ListView, ListItem or nil
Returns: none

gui.event.Expanded function( self, item )
Params: ListView, ListItem or nil
Returns: none

gui.event.Collapsed function( self, item )
Params: ListView, ListItem or nil
Returns: none

setColumnTitle

Params: number (column index, 1..n), string (new title)
Returns: none

setItemMargin

Params: number (points)
Returns: none

setMultiSelection

Params: boolean (default true)
Returns: none
If true, the user is allowed to select more than one item in the list.

**setRootDecorated**

Params: boolean (default true)
Returns: none
If true, the open/closed handles are also shown for the top-level items.

**setSorting**

Params: number (column index, 1..n), boolean (ascending, default true)
Returns: none
Explicitly sort the given column.

**setSortIndicated**

Params: boolean (default true)
Returns: none
If true, a little triangle next to the column title shows the sort direction.

**setStepSize**

Params: number (points)
Returns: none

**sort**

Params: none
Returns: none
Re-sort the whole list.

**Class MainWindow**

This class 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.

```plaintext
MainWindow

File Edit Options Help

Central
Widget

Ready
```

**getMenuBar**

Params: none
Returns: MenuBar

**setCentralWidget**

Params: Widget
Returns: none
showStatusText

Params: string (text), number (optional duration in milli seconds)
Returns: none
Displays the given text on the status line. When the optional duration is given, the text automatically disappears after the timeout.

**Class MenuBar**

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.

clear

Params: none
Returns: none
Remove all menues from the menu bar.

insertMenu

Params: PopupMenu, number (optional menu id), number (optional index)
Returns: number (menu id)

isEnabled

Params: number (menu id)
Returns: boolean

removeMenu

Params: number (menu id)
Returns: none

setCallback

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

```plaintext
gui.event.Activated function( self, menuId ) Params: MenuBar, number
Returns: none
```

setEnabled

Params: number (menu id), boolean (default true)
Returns: none

**Class MultiLineEdit**

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

![Multiline Editor Example](image)
getLine

Params: number (line number, 1..n)
Returns: string

getLineCount

Params: none
Returns: number

getText

Params: none
Returns: string (the whole text)

insertLine

Params: string (the text line), number (optional line number, 1..n, default at the end of the text)
Returns: none

isEdited

Params: none
Returns: boolean
This flag becomes true, if the text is changed by the user or by a call to setText() or insertLine().

setAlignment

Params: number (alignment code)
Returns: none
Use one of the following values: gui.align.Left, gui.align.Right or gui.align.HCenter

setCallback

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

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

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

setEdited

Params: boolean (default true)
Returns: none

setReadOnly

Params: boolean (default true)
Returns: none

setText

Params: string
Returns: none

setWordWrap

Params: boolean (default true)
Returns: none
If true, the widget automatically breaks text lines, so the text fits into the width of the widget.

**Class MyWidget**

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

**setAcceptFocus**

Params: 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).

**setAutoBackground**

Params: 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.

**setCallback**

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

- **gui.event.SizeHint**
  function( self )
  return width, height
  Params: MyWidget
  Returns: number, number

- **gui.event.MousePressed**
  function( self, x, y, left, mid, right, shift, ctrl, alt )
  Params: MyWidget, number, number, 6 x boolean
  Returns: none

- **gui.event.MouseReleased**
  function( self, x, y, left, mid, right, shift, ctrl, alt )
  Params: MyWidget, number, number, 6 x boolean
  Returns: none

- **gui.event.MouseMoved**
  function( self, x, y, left, mid, right, shift, ctrl, alt )
  Params: MyWidget, number, number, 6 x boolean
  Returns: none

- **gui.event.DblClicked**
  function( self, x, y, left, mid, right, shift, ctrl, alt )
  Params: MyWidget, number, number, 6 x boolean
  Returns: none

- **gui.event.KeyPressed**
  function( self, keyCode, text, shift, ctrl, alt ) return accepted
  Params: MyWidget, number, string or nil, 3 x boolean
  Returns: boolean

- **gui.event.KeyReleased**
  function( self, keyCode, text, shift, ctrl, alt ) return accepted
  Params: MyWidget, number, string or nil, 3 x boolean
  Returns: boolean

- **gui.event.FocusIn**
  function( self )
  Params: MyWidget
  Returns: none

- **gui.event.FocusOut**
  function( self )
  Params: MyWidget
  Returns: none

- **gui.event.Paint**
  function( self, painter )
  Params: MyWidget, Painter
  Returns: none

- **gui.event.Resized**
  function( self, w, h, oldW, oldH )
  Params: MyWidget, number, number, number, number
  Returns: none

- **gui.event.Closing**
  function( self ) return accept
  Params: MyWidget
  Returns: boolean

- **gui.event.Showing**
  function( self )
  Params: MyWidget
  Returns: none

- **gui.event.Hiding**
  function( self )
  Params: MyWidget
  Returns: none
setMouseTracking

Params: 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 Painter

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

drawContour

Params: ContourPlot, x, y, w, h
Draws the given contour plot at the given point position. The plot scales into the area given by w, h.

drawEllipse

Parameter: x, y, w, h. Draws an ellipse with the current pen and brush at center point x/y with extension w, h.

drawIcon

Params: Icon, number (pos. x), number (pos. y)
Returns: none

drawImage

Parameters: Image, x, y, w (opt.), h (opt.)
Draws the given image at the given point position. If w and h are left out, the image is painted with its original dimension. You can stretch the image to w and h if needed.

drawLine

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

drawPoint

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

drawRect

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

drawSlice

Params: SlicePlot, x, y, w, h
Draws the given slice plot at the given point position. The plot scales into the area given by w, h.

drawText

Parameter: x, y, string. 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.
fillRect

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

getBounding

Parameter: string. Returns w, h. This function accepts a text string and returns its width and height it will have when drawn on screen with the current font.

lineTo

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

moveTo

Parameter: x, y. This function sets the current position to x/y without drawing anything. This is useful when calling lineTo afterwards.

setBrush

Parameter: string or 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.

setFont

Parameter: string (font name), number (point size), boolean (bold), boolean (italic). This function sets the current font, which affects the following calls to drawText.

setPen

Parameter: string (color), number (width). 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.

Class Peak

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 scope windows 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).

getAmp

Params: 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.

getAssig

Params: none.
Returns: number or nil (dim. 1), ..., number or nil (dim. d)
Returns the assignment in all dimensions or nil, when it’s not defined.
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**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td></td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

**See Also**

setText, getColor

**getColor**

Params: none.
Returns: number
This is the color also associated with XEASY peaks. Values 1 to 6 are privileged in XEASY.

**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.

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The ID number of the given object, unique within class/repository</td>
</tr>
</tbody>
</table>

**getLabel**

Param: 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.

**getPos**

Param: Spectrum (optional)
Returns: number (dim. 1), ..., number (dim. d)
Gives the PPM position associated with a peak. When a spectrum is specified, the function returns the alias position (if available) or the default position.

**getVol**

Params: Spectrum (optional)
Returns: Amplitude
Get the Volume of the peak, optionally aliased to a certain spectrum. If the spectrum has no alias, the default is used.
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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute.</td>
</tr>
</tbody>
</table>

Return Values

None.

See Also

getAttr

Class PeakList

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).

createPeak

Params: number (PPM dim. 1), ..., number (PPM dim. d)
Returns: Peak
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.

getAtomType

Param: number (dimension 1..d)
Returns: string (atom type symbol)

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td></td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

See Also

setAttr

getBatchList

Params: 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:

```lua
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
```

getDimCount

Params: none.
Returns: number (dimension count)

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.

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The ID number of the given object, unique within class/repository</td>
</tr>
</tbody>
</table>

ggetName

Params: none
Returns: string (the name associated with the peaklist)

ggetPeak

Params: number (the peak ID number)
Returns: Peak or nil
If there is no peak known with the given ID, the function returns nil.

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

ggetPeaks

Params: none.
Returns: Table[ number (peak ID), Peak ]
This function returns all peaks contained in the peaklist as a table, indexed by the peak ID number.

`removePeak`

**Param:** Peak  
**Returns:** none.  
This function removes the peak passed as parameter from the peaklist. The peak is still alive afterwards, but no longer useful.

`saveToFile`

**Param:** 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.

`setAmp`

**Params:** Peak, number (optional, amplitude), Spectrum (optional)  
**Returns:** none.  
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.

`setAssign`

**Params:** Peak, number (assignment dim. 1), ..., number (assignment dim. d)  
**Returns:** none.

`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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute.</td>
</tr>
</tbody>
</table>

**Return Values**  
None.

**See Also**  
`getAttr`

`setBatchList`

**Param:** Table[ number (index 1..n), 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.

`setColor`

**Params:** Peak, number (color value)  
**Returns:** none.
setLabel

Params: Peak, string (optional, the label)
Returns: none.

setName

Params: string (the new name of the peaklist)
Returns: none

setPos

Params: Peak, number (PPM dim. 1), ..., number (PPM dim. d), Spectrum (optional)
Returns: none.
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.

setVol

Params: Peak, number (optional, the new volume), Spectrum (optional)
Returns: none.
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.

Class PopupMenu

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.

clear

Params: none
Returns: none
Remove all menu items and sub-menus from this menu.

getText

Params: number (menu id)
Returns: string

insertItem

Params: string (menu text), number (optional menu id), number (optional index, 1..n)
Returns: number (menu id)
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.
insertSeparator

Params: none
Returns: none
Insert a menu separator line at the end of the menu.

insertSubmenu

Params: PopupMenu, string (menu text), number (optional menu id), number (optional index, 1..n)
Returns: number (menu id)
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.

isChecked

Params: number (menu id)
Returns: boolean

isEnabled

Params: number (menu id)
Returns: boolean

popup

Params: number (pos. x), number (pos. y)
Returns: none
Use this function to open the popup menu at the given position (e.g. as reaction on a right-click on the mouse button).

removeItem

Params: number (menu id)
Returns: none

setAccel

Params: number (menu id), string (acceleration string, e.g. "Ctrl+S")
Returns: none

setCallback

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

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

setChecked

Params: number (menu id), boolean (default true)
Returns: none

setEnabled

Params: number (menu id), boolean (default true)
Returns: none
setIcon

Params: number (menu id), Icon
Returns: none

setText

Params: number (menu id), string (text)
Returns: none
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.

setWhatsThis

Params: number (menu id), string (text)
Returns: none

**Class Project**

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.

addCandidate

Params: SpinSystem, ResidueType
Returns: none.
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.

assignSpin

Params: Spin, SpinSystem
Returns: none.
This function tries to assign the given Spin to the given SpinSystem. The function aborted the script with an error, if it would break consistency of the model.

assignSystem

Params: SpinSystem, Residue
Returns: none.
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.

createSpin

Params: string (atom type symbol), number (PPM position), Spectrum (optional)
Returns: Spin
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.

createSystem

Params: none.
Returns: SpinSystem.
This function creates a new SpinSystem and automatically gives it a unique ID number.
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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td></td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

See Also

setAttr

getCombinedFragment

Params: SpinSystem (predecessor), SpinSystem (successor)
Returns: Table[ number (index), SpinSystem ]
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.

getFragment

Params: SpinSystem
Returns: Table[ number (index), SpinSystem ]
This function simply returns an array representing the spin system fragment, to which the given spin system belongs.

getName

Params: none.
Returns: string (the name of the project)

getPeakList

Params: number (peaklist ID)
Returns: PeakList or nil
This function tries to find a peaklist with the given ID number. If it finds one, it will be returned.

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

getPeakLists

Params: 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.
getResidue

Params: 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. Contrary 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 getPred and getSucc instead.

getSequence

Params: 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 for all of the Lua library or a simple for loop (see example).

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

getSpectra

Params: none.
Returns: Table[ number (spectrum ID), Spectrum ]
This function returns all spectra of the project in an array indexed by the spectrum ID.

getSpectrum

Params: number (spectrum ID)
Returns: Spectrum or nil
This function tries to find a spectrum with the given ID number. If it finds one, it will be returned.

getSpin

Params: number (spin ID)
Returns: Spin or nil
This function tries to find a spin with the given ID number. If it finds one, it will be returned.

getSpinLinks

Params: none.
Returns: Table[ number (index 1..n), Spectrum ]
This function returns all spin links of the project in an array.

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

getSpins

Params: none.
Returns: Table[ number (spin ID), Spin ]
This function returns all spins of the project in an array indexed by the spin ID.
```lua
s = cara:getProject():getSpins()
for a,b in pairs(s) do print( a.." ",b:getLabel() ) end
```

**getSystem**

*Params:* number (spin system ID)
*Returns:* SpinSystem or nil

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

**getSystems**

*Params:* none.
*Returns:* Table[number (spin system ID), SpinSystem ]

This function returns all spin systems of the project in an array indexed by the spin system ID.

**getTolerance**

*Params:* string (atom type symbol)
*Returns:* number (matching tolerance in PPM)

**linkSpins**

*Params:* Spin, Spin
*Returns:* SpinLink

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

**linkSystems**

*Params:* SpinSystem, SpinSystem
*Returns:* none.

Use this function to combine spin systems to fragments. The function maintains and checks the consistency of the model. If both spin systems 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.

**matchResidue**

*Params:* SpinSystem, Residue, number (optional offset), Spectrum (optional)
*Returns:* number (rating), number (weight), number (zero count), bool (excluded), bool (assigned)

This function calculates a match of the given spin system on the given residue. 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, the default positions otherwise.

The returned rating represents a "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 being out of its deviation. Excluded is true, if the spin system has candidates set and residue is not part of them. Assigned is true, if the residue already is assigned to a spin system.

**matchSpin**

*Params:* Spin, SpinSystem, Spectrum (optional)
*Returns:* number (rating), number (weight), number (zero count)

This function calculates a match of the given spin (PPM position) to the given spin system. Only spins with the same atom type are matched. If a spectrum is provided, its alias spin positions are used, the default positions otherwise. The match is influenced by the tolerance value for the atom type. 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 being out of the tolerance (and thus not contributing to weight or rating).

**matchSystems**

Params: SpinSystem, SpinSystem, bool (ignore labels), Spectrum (optional)
Returns: number (rating), number (weight), number (zero count)
This function does a complete cross correlation of the spins of both spin systems. If "ignore labels" is true, all spins with the same atom type are correlated. If it is false, only spins with equal labels are correlated (considering 0 and -1 offsets). The match is rated using the tolerance value of the atom types of the compared spins. If a spectrum is provided, its alias spin positions are used, the 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 being out of the tolerance (and thus not contributing to weight or rating).

**removeCandidate**

Params: SpinSystem, ResidueType
Returns: none.
This function removes the given residue type from the assignment candidate list of the given spin system. See also addCandidate.

**removeSpin**

Params: 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 (spin system, etc.). Otherwise the function is aborted with an error. The removed spin is still alive but useless.

**removeSystem**

Params: SpinSystem
Returns: none
This function tries to remove the spin system from the project. Consistency rules demand that the spin system is not part of an assignment (spin, residue, etc.). Otherwise the function is aborted with an error. The removed spin system is still alive but useless.

**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**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute</td>
</tr>
</tbody>
</table>

**Return Values**

None.

**See Also**

getAttr

setLabel

Params: Spin, string (optional, spin label)
Returns: none.
This function tries to set the label of the given spin. The label must obey the CARA spin label syntax. When left out, it defaults to the empty label. If the spin belongs to a spin system, the label must be acceptable by the spin system. In case of a consistency violation the execution is aborted with an error.

```lua
setShift
```

Params: Spin, number (PPM shift), Spectrum (optional)
Returns: none.
This function sets the default or alias (if spectrum provided) shift of the given spin.

```lua
setSystemType
```

Params: SpinSystem, SystemType
Returns: none.
This function sets the spin system type of the given spin system. This is useful for homonuclear, sequential assignment strategy.

```lua
setTolerance
```

Params: string (atom type symbol), number (PPM tolerance)
Returns: none.
Sets the matching tolerance of the given atom type. See also getTolerance.

```lua
setValue
```

Params: Residue, string (atom label), number (mean value, optional), number (dev. value, optional)
Returns: none.
Use this function to set the (random coil) PPM values associated with the given residue. These values are used by 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.

```lua
unassignSpin
```

Params: Spin
Returns: none.
Removes the given spin from its previously assigned spin system, if it has one. Otherwise the execution is aborted with an error.

```lua
unassignSystem
```

Param: SpinSystem
Returns: none.
Unassigns the given spin system from the previously assigned residue. For consistency reasons, the whole fragment, to which this spin system belongs, is unassigned. If there was no assignment, the execution is aborted with an error.

```lua
unlinkSpins
```

Param: Spin, Spin
Returns: none
If the two spins were linked, the function removes the corresponding SpinLink from the project. Nothing happens, when the spins were not linked.

```lua
unlinkSystems
```

Param: SpinSystem, SpinSystem
Returns: none.
This function unlinks the two spin systems, i.e. the fragment is cut between the systems. The residue assignments of the spin systems remain the same after the cut. If the systems were not linked, execution is aborted with an error.

**Class ProtonList**

This class can be used to load and save XEASY proton list files. It is like this possible to program custom selections of the spins to be written (e.g. using spins with -1 offset from neighbour spin systems 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.

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

**getAtom**

*Params:* number (index)
*Returns:* number (spin ID), number (shift), string (atom label), number (system ID)

**getCount**

*Params:* none.
*Returns:* number (the number of atoms in the list)

**isValid**

*Params:* number (index)
*Returns:* boolean

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

**saveToFile**

*Params:* string (file path)
*Returns:* none.

This function tries to write the proton list to the file with the given path. If the file cannot be written, the function is aborted with an error. Keep in mind that existing files with the same path will be overwritten.

**setAtom**

*Params:* number (index), number (spin ID), number (shift), string (atom label), number (system ID)
*Returns:* none

**Class PushButton**

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

```lua
PL = PushButton()
PL:setLabel( 'Press Me' )
```

**setDefault**

*Params:* 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.

**setFlat**

Params: boolean (default true)
Returns: none
If true, the button is drawn without a border.

**setOn**

Params: boolean (default true)
Returns: none
Set the button to the "pressed" state. Only useful for toggle buttons (see setToggleButton).

**setPopup**

Params: PopupMenu
Returns: none
Set the popup menu to be opened, when the user presses this button.

**setToggleButton**

Params: boolean (default true)
Returns: none
If true, the button has two 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**

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

```plaintext
First
Second
Third
```

**isChecked**

Params: none
Returns: boolean

**setChecked**

Params: boolean (default true)
Returns: none
If this button is part of a ButtonGroup, the other buttons of the group are automatically unchecked.

**Class Record**

A record is a flexible, persistent data structure 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.

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td></td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

See Also
setAttr
getAttrs

Params: 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.

getId

Params: none.
Returns: number (OID)

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute.</td>
</tr>
</tbody>
</table>

Return Values
None.

See Also
getAttr

Class Repository

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")

createRecord

Params: none
Returns: Record
This function creates a new persistent Record within the current repository. You can set or read attributes of this records. They are automatically saved with the repository and available next time you open it. CARA manages record identity using Object IDentities (OID), which are positive numbers starting from one. Whenever you "store" a Record into an attribute of an object, 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.

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td></td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

See Also
setAttr

getAuthor

Params: none.
Returns: string
This function returns the author attribute of the repository (identical to getAttr("Author"));

getProject

Params: string (optional project name)
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.

getProjects

Params: none
Returns: Table{ string (project name), Project }
Returns an array containing all projects of the repository, indexed by project name.

getRecord

Params: 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).
getRecords

Params: none.
Returns: Table[ number (OID), Record ]
Returns a list of all records of the repository.

getResidueType

Params: string (short name)
Returns: ResidueType or nil

getResidueTypes

Params: none
Returns: Table[ string (name), ResidueType ]
Returns an array containing all residue types of the repository, indexed by name.

getSpectrumType

Params: string (name)
Returns: SpectrumType or nil

getSpectrumTypes

Params: none
Returns: Table[ string (name), SpectrumType ]
Returns an array containing all spectrum types of the repository, indexed by name.

getSystemTypes

Params: none
Returns: Table[ number (ID), SystemType ]
Returns an array containing all system types of the repository, indexed by ID number.

removeRecord

Params: Record
Returns: none
Removes the given Record from the repository. Nothing happens, if the Record was not part of the repository. As soon as a Record is removed, it is still alive but useless.

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute.</td>
</tr>
</tbody>
</table>

Return Values

None.

See Also

getAttr
**Class 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**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td></td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

**See Also**

setAttr

**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.

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The ID number of the given object, unique within class/repository</td>
</tr>
</tbody>
</table>

**getPred**

Params: none.
Returns: Residue or nil.
This function returns the predecessor of the residue in the sequence (to N-terminus), if it has one.

**getSucc**

Params: none.
Returns: Residue or nil.
This function returns the successor of the residue in the sequence (to C-terminus), if it has one.

**getSystem**

Params: none.
Returns: SpinSystem or nil.
This function returns the assigned SpinSystem (see Project:assignSystem), if it has one.

**getType**

Params: none.
Returns: ResidueType (never nil)

**getValue**

Params: string (atom label)
Returns: number (mean value), number (deviation value)

**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**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute.</td>
</tr>
</tbody>
</table>

**Return Values**

None.

**See Also**

getAttr

**Class ResidueType**

**getAtom**

Params: string (atom name)
Returns: Atom or nil.

getAtomGroup

Params: string (atom group name)
Returns: AtomGroup or nil

getAtomGroups

Params: none.
Returns: Table[ string (atom group name), AtomGroup ]

getAtoms

Params: none
Returns: Table[ string (atom name), Atom ]

**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**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td></td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

**See Also**

getAttr
**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.

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The ID number of the given object, unique within class/repository</td>
</tr>
</tbody>
</table>

**getLetter**

Param: none.
Returns: string (the single letter name of the residue type)

**getName**

Param: none.
Returns: string (the full name of the residue type)

**getShort**

Param: none.
Returns: string (the three letter code of the residue type, also used as unique ID)

**getSystemType**

Param: none.
Returns: SystemType or nil.

**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**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute.</td>
</tr>
</tbody>
</table>

**Return Values**
None.

**See Also**
getAttr

**ScrollView**

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 control the visible detail by operating the scroll bars.
addChild

Params: Widget, number (pos. x, defaul 0), number (pos. y, default 0)
Returns: none

center

Params: number (pos. x), number (pos. y)
Returns: none
Adjusts the scroll bars so the given position becomes the new center of the ScrollView (if possible).

ensureVisible

Params: number (pos. x), number (pos. y)
Returns: none
Ensures, that the given position is part of the visible area. Otherwise the scroll bars are appropriately adjusted.

moveChild

Params: Widget, number (pos. x), number (pos. y)
Returns: none
Move the given child widget to the new position.

removeChild

Params: Widget
Returns: none

resizeContents

Params: number (width), number (height)
Returns: none
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).

scrollBy

Params: number (delta x), number (delta y)
Returns: none

scrollTo

Params: number (pos. x), number (pos. y)
Returns: none
setCallback

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

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

### Class SlicePlot

A SlicePlot is used to draw slices according 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 committed drawings (see Canvas:commit).

```lua
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()
```

getDimension

```lua
Params: none
Returns: number (dim. 1 or 2)
```

getMinMax

```lua
Params: none
Returns: number (minimal amplitude), number (maximal amplitude)
```

recalcMinMax

```lua
Params: none
Returns: none
```

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

setAutoScale

```lua
Params: boolean (auto-scale, default true), boolean (auto-center, default false)
Returns: none
```

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.

setBuffer

```lua
Params: Buffer
Returns: none
```

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

setColor

```lua
Params: string (color code)
Returns: none.
Set the color used to draw the diagram. The code has the usual format ("#RRGGBB", etc.).
```
setMinMax

Params: number (minimum amplitude), number (maximum amplitude)
Returns: none

toPoint

Params: number (origin on canvas), number (span on canvas), number (ppm)
Returns: number (point position of given ppm value)

toPpm

Params: number (origin on canvas), number (span on canvas), number (point)
Returns: number (ppm value of given point position)

Class Spectrum

This class represents an NMR spectrum living in a file of the file system. It is an abstract concept, independant of the concrete 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 always is optimally efficient, regardless of the direction and orientation choosen.

getAt

This function returns the amplitude at the given index position. The spectrum can have d dimensions. When the position is out of valid range, the script is aborted with an error.

Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..d</td>
<td>number</td>
<td></td>
<td>Sample index 1..n for each dimension of the spectrum.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Amplitude at the given sample index.</td>
</tr>
</tbody>
</table>

See Also

Spectrum:getDimCount, Spectrum:getSampleCount

ggetAtomType

Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Dimension d, depending on getDimCount</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>Atom symbol according to periodic system of elements.</td>
</tr>
</tbody>
</table>

See Also

Spectrum:getDimCount

ggetAtPoint

Params: Table (array of ppm values, one per dimension), boolean (folded, default false)
Returns: number (amplitude)
getAtPpm

This function returns the amplitude at the given PPM position. The spectrum can have d dimensions. The position can be out of the spectral width. In this case the folded value is returned.

**Parameters**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..d</td>
<td>number</td>
<td></td>
<td>PPM value for each dimension of the spectrum.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Amplitude at the given PPM point.</td>
</tr>
</tbody>
</table>

**See Also**

Spectrum:getFolding

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**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td></td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

**See Also**

setAttr

getDimCount

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Number of dimensions of this spectrum</td>
</tr>
</tbody>
</table>

getFilePath

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The path in the file system where the spectrum (i.e. the param file) resides</td>
</tr>
</tbody>
</table>

getFolding

**Parameters**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Dimension 1..d</td>
</tr>
</tbody>
</table>
Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The symbol representing the folding type of the dimension (empty, ”RSH” or ”TPPI”).</td>
</tr>
</tbody>
</table>

getFreq

Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Dimension 1 to d, depending on getDimCount</td>
</tr>
<tr>
<td>2</td>
<td>number</td>
<td></td>
<td>Sample index 1..n</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>PPM position corresponding to the given sample index</td>
</tr>
</tbody>
</table>

See Also
Spectrum:getDimCount

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.

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The ID number of the given object, unique within class/repository</td>
</tr>
</tbody>
</table>

ggetIndex

Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Dimension 1..d</td>
</tr>
<tr>
<td>2</td>
<td>number</td>
<td></td>
<td>PPM position</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Sample index corresponding to the given PPM position</td>
</tr>
</tbody>
</table>

See Also
Spectrum:getDimCount

getAddress

Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Dimension 1..d</td>
</tr>
</tbody>
</table>
Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The label associated with the spectrum dimension</td>
</tr>
</tbody>
</table>

getName

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of this spectrum (normally unique within project)</td>
</tr>
</tbody>
</table>

getPlanePpm

Use this function to cut out a plane from a spectrum with d 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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Spectrum dim. 1..d mapped to Buffer dim. 1</td>
</tr>
<tr>
<td>2</td>
<td>number</td>
<td></td>
<td>Spectrum dim. 1..d mapped to Buffer dim. 2</td>
</tr>
<tr>
<td>3,4</td>
<td>number</td>
<td></td>
<td>PPM range of the plane in spectrum dimension 1</td>
</tr>
<tr>
<td>...</td>
<td>number</td>
<td></td>
<td>PPM ranges of the plane for the other spectrum dimensions</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Buffer</td>
<td></td>
<td>The 2D buffer containing the resulting plane</td>
</tr>
</tbody>
</table>

See Also

Spectrum:getDimCount

getPlaneRange

getPpmRange

Params: number (dim. x), number (dim. y), Table (from), Table (to), boolean (folded, default false)

Returns: Buffer

This function is similar to getPlanePpm(), but accepts two table containing the ppm ranges. Each table contains a ppm value for each dimension of the spectrum (represented by index 1..d). The first table contains the from-values, the second the to-values (e.g. the range of dim. 2 is given by from[2] and to[2]). If folded=false, the parts out of spectrum are filled with zeroes.
getRfFreq

Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Dimension 1..d, depending on getDimCount</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The spectrometer frequency in MHz for the given dimension.</td>
</tr>
</tbody>
</table>

getSampleCount

Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Dimension 1..d, depending on getDimCount</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Number of samples n in the given dimensions.</td>
</tr>
</tbody>
</table>

getSlicePpm

Use this function to cut out a slice from a spectrum with d 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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>Spectrum dim. 1..d mapped to Buffer dim. 1</td>
</tr>
<tr>
<td>2,3</td>
<td>number</td>
<td></td>
<td>PPM range of the slice in spectrum dimension 1</td>
</tr>
<tr>
<td></td>
<td>number</td>
<td></td>
<td>PPM ranges of the slice for the other spectrum dimensions</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Buffer</td>
<td></td>
<td>The 2D buffer containing the resulting plane</td>
</tr>
</tbody>
</table>

See Also

Spectrum:getDimCount

getSliceRange

Params: number (dim. x), Table (from), Table (to), boolean (folded, default false)

Returns: Buffer

This function is similar to getSlicePpm(), but accepts two table containing the ppm ranges. Each table contains a ppm value for each dimension of the spectrum (represented by index 1..d). The first table contains the from-values, the second the to-values (e.g. the range of dim. 2 is given by from[2] and to[2]). If folded=false, the parts out of spectrum are filled with zeroes.
**getOfType**

### Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SpectrumType or nil</td>
<td></td>
<td>The associated spectrum type. Not nil for all spectra belonging to a project (gotten by Project:getSpectrum)</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute.</td>
</tr>
</tbody>
</table>

### Return Values

None.

**See Also**

getAttr

**Class SpectrumType**

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 ResidueType) will potentially be visible in the spectrum.

**getAtomType**

Params: number (dimension 1..d)

Returns: string (atom symbol)

**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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

### Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td></td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

**See Also**

setAttr
getDimCount

Params: none.
Returns: number (dimension count d)

dgetDimName

Params: number (dimension 1..d)
Returns: string

ggetKeyLabel

Params: number (dimension 1..d)
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.

ggetLabels

Params: number (dimension 1..d)
Returns: Table[ number (index), string (spin label syntax) ]

ggetName

Params: none
Returns: string

ggetStep

Params: number (step number 1..s)
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:

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

ggetStepCount

Params: none.
Returns: number (number of procedure steps s)

isNoesy

Params: number (dimension 1..d)
Returns: boolean
If the step with the given number selects all spins of its atom type, this function returns true.

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute.</td>
</tr>
</tbody>
</table>

Return Values

None.

See Also

getAttr

**Class Spin**

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.

getParam

| Params: none |
| Returns: string (atom type symbol) |

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

See Also

setAttr

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.

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The ID number of the given object, unique within class/repository</td>
</tr>
</tbody>
</table>

getLabel

| Params: none |

Returns: string (spin label syntax)

getLink

Params: Spin
Returns: SpinLink or nil

getLinks

Params: none.
Returns: Table[ number (index), SpinLink ]

getShift

Params: Spectrum (optional)
Returns: number (PPM shift)

getShifts

Params: none.
Returns: Table[ number (spectrum ID), number (PPM shift) ]

getSystem

Params: none.
Returns: SpinSystem or nil.

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute.</td>
</tr>
</tbody>
</table>

Return Values

None.

See Also

getAttr

**Class SpinLink**

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 by picking a peak in HomoScope and PolyScope.

getAmp

Params: Spectrum (optional)
Returns: number (amplitude)
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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td></td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

See Also

setAttr

getLeft

Params: none.
Returns: number (spin ID)

getRight

Params: none.
Returns: number (spin ID)

isVisible

Params: Spectrum
Returns: boolean

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute.</td>
</tr>
</tbody>
</table>

Return Values

None.

See Also

getAttr

Class SpinSystem

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 a Residue (by first concatenating SpinSystems to fragments).
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**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td></td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

**See Also**

setAttr
getCandidates

getCandidates

Params: none.
Returns: Table [number (index), 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.

**Return Values**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The ID number of the given object, unique within class/repository</td>
</tr>
</tbody>
</table>

getPred

Params: none.
Returns: SpinSystem or nil.

getResidue

Params: none.
Returns: Residue or nil.

getSpins

Params: none.
Returns: Table [number (spin ID), Spin]

g getSucc

Params: none.
Returns: SpinSystem or nil.

g getSystemType

Params: none.
Returns: SystemType or nil.

**isAcceptable**

Params: string (spin label syntax).
Returns: boolean
Use this function to check, whether a spin with the given label would be acceptable within the spin system.

**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**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute.</td>
</tr>
</tbody>
</table>

**Return Values**

None.

**See Also**

getAttr

**Class Splitter**

Splitter inherits all features of Frame. It is used to organize its sub-widgets into panes, whose borders can be interactively moved by the user.

**setOrientation**

Params: number (either gui.Horizontal or gui.Vertical)
Returns: none

**Class SystemType**

A SystemType is used to classify SpinSystems and ResidueTypes. According to the process of sequential assignment (invented by Prof. Kurt Wüthrich), 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.
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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic types, Record</td>
<td></td>
<td>The value of the attribute or nil, if the attribute is not defined.</td>
</tr>
</tbody>
</table>

See Also

setAttr

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.

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The ID number of the given object, unique within class/repository</td>
</tr>
</tbody>
</table>

getName

Params: none
Returns: string (name of system type)

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>2</td>
<td>basic types, Record</td>
<td>x</td>
<td>The value to be assigned to the attribute.</td>
</tr>
</tbody>
</table>

Return Values

None.

See Also

getAttr

Class TabWidget

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 corresponding tab header.
addTab

Params: Widget, string (header title)
Returns: none

getCurrentPage

Params: none
Returns: Widget or nil

setCallback

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

gui.event.Changed function( self, page )  Params: TabWidget, Widget
Returns: none

setMargin

Params: number (points)
Returns: none

setTabEnabled

Params: Widget, boolean (default true)
Returns: none

setTabPosition

Params: number (use TabWidget.Top or TabWidget.Bottom)
Returns: none

setTitle

Params: Widget, string (new title)
Returns: none
The string can contain accelerator keys, e.g. "&Title". The widget switches to this page, when the user presses ALT+T.

showPage

Params: Widget
Returns: none

Class TextView

TextView inherits all features of ScrollView. It is used to display formatted HTML text to the user.
**getText**

Params: none  
Returns: string (HTML text)

**setText**

Params: string (HTML text)  
Returns: none

**Class VBox**
This class inherits all features of HBox. It is used to vertically align its child-widgets.

<table>
<thead>
<tr>
<th>One</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two</td>
</tr>
<tr>
<td>Three</td>
</tr>
<tr>
<td>Four</td>
</tr>
<tr>
<td>Five</td>
</tr>
</tbody>
</table>

**Class Widget**
This is the root class of all GUI components of CARA/Lua. It cannot be instantiated by itself but is implicitly present (by inheritance) whenever you instantiate a derived class.

**clearFocus**

Params: none  
Returns: none  
This function removes the focus from this Widget.

**close**

Params: 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.

**destroy**

Params: 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.

**getBgColor**

Params: none  
Returns: string (color code)

**getCaption**

Params: none  
Returns: string
getData

Params: 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).

defColor

Params: none
Returns: string (color code)

defFont

Params: none
Returns: string (font name), number (point size), boolean (bold), boolean (italic)

defFrameGeometry

Params: none.
Returns: number (pos. x), number (pos. y), number (width), number (height)
This function returns the position and size of this widget relative to its parent. The function considers
the complete widget, including frame and decorations.

defGeometry

Params: none
Returns: number (pos. x), number (pos. y), number (width), number (height)
This function returns the position and size of the contents pane of this widget relative to its parent (not
considering frame and decorations).

defMaximumSize

Params: none
Returns: number (width), number (height)

defMinimumSize

Params: none
Returns: number (width), number (height)

defParentWidget

Params: 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 independant top level window.

defSize

Params: none
Returns: number (width), number (height)

defTopLevelWidget

Params: 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, i.e. a widget tree. When you call this function on a child widget, it returns the root widget of this tree.

**hasFocus**

**Params:** none  
**Returns:** boolean

**hide**

**Params:** none  
**Returns:** none  
Use this function to make a widget disappear from 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.

**isActiveWindow**

**Params:** 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.

**isDesktop**

**Params:** none  
**Returns:** boolean  
Returns true if the widget is a desktop widget, otherwise false. A desktop widget is also a top-level widget.

**isEnabled**

**Params:** none  
**Returns:** boolean  
Returns true if the widget is enabled, or false if it is disabled.

**isFocusEnabled**

**Params:** none  
**Returns:** boolean  
Returns true if the widget accepts keyboard focus, or false if it does not.

**isHidden**

**Params:** 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.

**isMaximized**

**Params:** none  
**Returns:** boolean  
Returns true if this widget is a top-level widget that is maximized, or else false. Note that due to limitations in some window-systems, this does not always report expected results.

**isMinimized**

**Params:** none
Returns: boolean
Returns true if this widget is a top-level widget that is minimized (iconified), or else false.

isModal

Params: none
Returns: boolean
Returns true, if this is a modal widget, e.g. a Dialog.

isPopup

Params: none
Returns: boolean
Returns true, if this is a popup widget, e.g. a PopupMenu.

isTopLevel

Params: none
Returns: boolean
Returns true if the widget is a top-level widget, otherwise false. A top-level widget is a widget which usually has a frame and a caption (title bar). Popup and desktop widgets are also top-level widgets. A 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.

isVisible

Params: 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, as well as having isMinimized() return 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.

lower

Params: 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.

mapFromGlobal

Params: number (pos. x), number (pos. y)
Returns: number (pos. x), number (pos. y)
Translates the global screen coordinates to widget coordinates.

mapFromParent

Params: number (pos. x), number (pos. y)
Returns: number (pos. x), number (pos. y)
Translates the parent coordinates to widget coordinates.
### mapToGlobal

**Params:** number (pos. x), number (pos. y)  
**Returns:** number (pos. x), number (pos. y)  
Translates the widget coordinates to global screen coordinates.

### mapToParent

**Params:** number (pos. x), number (pos. y)  
**Returns:** number (pos. x), number (pos. y)  
Translates the widget coordinates to parent coordinates.

### move

**Params:** number (pos. x), number (pos. y)  
**Returns:** none  
Move this widget to the given position relative to its parent (or the screen, when there is no parent).

### raise

**Params:** 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, this widget will be visually in front of its siblings afterwards.

### resize

**Params:** number (width), number (height)  
**Returns:** none  
Resize the widget to the given size. The size is adjusted if it is outside the minimum or maximum widget size.

### setActiveWindow

**Params:** 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 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.

### setBgColor

**Params:** string (color code)  
**Returns:** none  
Set the background color of this widget.

### setCallback

**Params:** number (event id), function or nil  
**Returns:** none  
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. 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.

```lua
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()
```

**setCaption**

**Params:** string  
**Returns:** none  
Set the window title of this widget. Has no effect, if the widget doesn't display a title bar.

**setEnabled**

**Params:** boolean (default true)  
**Returns:** none  
Enables widget input events if enable is 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 implicitly disables all its children. Enabling respectively enables all child widgets unless they have been explicitly disabled.

**setFixedSize**

**Params:** number (width), number (height)  
**Returns:** none  
Sets this widget to a fixed size, i.e. it will no longer dynamically adapt its size to its parent.

**setFocus**

**Params:** none  
**Returns:** none  
Gives the keyboard input focus to the widget.

**setFont**

**Params:** string (font name), number (point size), boolean (bold), boolean (italic)  
**Returns:** none

**setMaximumSize**

**Params:** number (width), number (height)  
**Return:** none
After calling this function the widget will no longer automatically increase its size above the given values. When it is a top-level widget, the user cannot extend the size of the window above the given values.

**setMinimumSize**

Params: number (width), number (height)
Return: none
After calling this function the widget will no longer automatically decrease its size under the given values. When it is a top-level widget, the user cannot reduce the size of the window below the given values.

**setUpdatesEnabled**

Params: boolean (default true)
Returns: none
Enables widget updates if enable is true, or disables widget updates if enable is false. Calling update() and has no effect if updates are disabled. Paint events from the window system are processed normally even if updates are disabled. This function is normally used to disable updates for a short period of time, for instance to avoid screen flicker during large changes.

**show**

Params: none
Returns: none
Shows the widget and its child widgets.

**showFullScreen**

Params: none
Returns: none
Shows the widget in full-screen mode. Calling this function has no effect for other than top-level widgets. To return from full-screen mode, call showNormal(). Full-screen mode works fine under Windows, but has certain problems under X11.

**showMaximized**

Params: none
Returns: none
Shows the widget maximized. Calling this function has no effect for other than top-level widgets. On X11, this function may not work properly with certain window managers.

**showMinimized**

Params: none
Returns: none
Shows the widget minimized, as an icon. Calling this function has no effect for other than top-level widgets.

**showNormal**

Params: none
Returns: none
Restores the widget after it has been maximized or minimized. Calling this function has no effect for other than top-level widgets.

**update**

Params: number (pos. x), number (pos. y), number (width), number (height), all optional
Returns: none
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.

updateAll

Params: none
Returns: none
This function has the same effect as update() without parameters.

updateGeometry

Params: none.
Returns: none.
Notifies the layout system that this widget has changed and may need to change geometry.

Class WidgetStack
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.

addWidget

Params: Widget
Returns: none
Adds a widget to the stack. It is initially hidden.

getTopWidget

Params: none
Returns: Widget

setTopWidget

Params: Widget
Returns: none
Puts the given widget on top of the stack and thus makes it visible.

Library 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 it 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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td>x</td>
<td>Sets the caption of the progress dialog</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td>x</td>
<td>Sets the label text above the progress bar</td>
</tr>
<tr>
<td>3</td>
<td>string</td>
<td>x</td>
<td>Sets the text of the cancel button</td>
</tr>
</tbody>
</table>

Return Values
None.
Example

dlg.beginProgress( "Testing Progress..." )
max = 1000000
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

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 (nil) as soon as the user closes the canvas window. There is no way to close the canvas window programmatically.

Parameters
None.

Return Values

<table>
<thead>
<tr>
<th>Nr</th>
<th>Type</th>
<th>Opt</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Canvas</td>
<td></td>
<td>A new Canvas ready to be drawn on</td>
</tr>
</tbody>
</table>

See Also
Class Canvas

createContour

Params: Buffer
Returns: ContourPlot
This function aborts with an error, if the given Buffer is not 2D.

createIcon

Params: string (XPM format icon)
Returns: Icon
xpm = [[/* XPM */
  "16 16 7 1"
  "# c #000000"
  "b c #ffffff"
  "e c #000000"
  "d c #404000"
  "c c #c0c000"
  "a c #ffffffc0"
  ". c None"
  "...................."
  "..................."
  .........#..a####....
  .......#b##bba#b####
  ......#b##bba#bbaabb#
  ....#b##bbaa#bbaaabb#
  ..#b##aab#bbaabbbaa#
  #a#aaa#bcbbbbbbbb#
  #c#dc#bcbcbcbcbcb#
  #a#b#bcbbbcabb#b#
  #a#b#bbba#bbaabba#
  #a#bb#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#
  #a#b#bba#bbaabbaa#]]

icon = dlg.createIcon( xpm )
cv = dlg.createCanvas()
cv:begin()
cv:drawIcon( icon, 30, 30 )
cv:commit()

This example script opens a canvas ans displays the following icon: 📚
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.

createSlice

Params: number (dimension 1 or 2), Buffer (optional, 1D required)
Returns: SlicePlot
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. Contrary to require, this function always executes the referenced script (not only once).

Parameters

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>Name of the script to be executed. Must be a script within the repository currently open.</td>
</tr>
</tbody>
</table>

Return Values

None.

See Also

require, dofile in Lua base library
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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>boolean</td>
<td>x</td>
<td>Kills the progress dialog, regardless whether beginProgress was called recursively.</td>
</tr>
</tbody>
</table>

Return Values

None.

See Also

beginProgress, updateProgress

currentDir

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.

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The directory last set by CARA or setCurrentDir</td>
</tr>
</tbody>
</table>

See Also

setCurrentDir

gDecimal

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td>x</td>
<td>The title of the input dialog</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td>x</td>
<td>An optional message text displayed on the dialog</td>
</tr>
<tr>
<td>3</td>
<td>number</td>
<td>x</td>
<td>The default value to be displayed on the dialog</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number or nil</td>
<td></td>
<td>The number entered by the user or nil if he pressed cancel</td>
</tr>
</tbody>
</table>

See Also

gText, gInteger, gSymbol

gInteger

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td>x</td>
<td>The title of the input dialog</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td>x</td>
<td>An optional message text displayed on the dialog</td>
</tr>
<tr>
<td>3</td>
<td>number</td>
<td>x</td>
<td>The default value to be displayed on the dialog</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number or nil</td>
<td></td>
<td>The number entered by the user or nil if he pressed cancel</td>
</tr>
</tbody>
</table>

Example

```lua
code
local id = dlg.getInteger( "Select Peaklist", "Please enter a valid peaklist ID:" )
if id == nil then
  return
end
```

See Also

getText, getDecimal, getSymbol

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td>x</td>
<td>The title of the file dialog</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td>x</td>
<td>The filter pattern to restrict the selection to files of a certain ending (optional all files)</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string or nil</td>
<td></td>
<td>The path to the file the user selected or nil if he pressed cancel</td>
</tr>
</tbody>
</table>

See Also

getText, getDecimal, getSymbol

getOpenFileName

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td>x</td>
<td>The title of the file dialog</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td>x</td>
<td>The filter pattern to restrict the selection to files of a certain ending (optional all files)</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string or nil</td>
<td></td>
<td>The path to the file the user selected or nil if he pressed cancel</td>
</tr>
</tbody>
</table>


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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The title of the input dialog</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td></td>
<td>An optional message text displayed on the dialog</td>
</tr>
<tr>
<td>3..n</td>
<td>string</td>
<td>x</td>
<td>The strings to be displayed in the list, where the user can select from</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string or nil</td>
<td></td>
<td>The string selected by the user or nil if he pressed cancel</td>
</tr>
</tbody>
</table>

See Also

getOpenFileName, getCurrentDir

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td>x</td>
<td>The title of the input dialog</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td>x</td>
<td>An optional message text displayed on the dialog</td>
</tr>
<tr>
<td>3</td>
<td>string</td>
<td>x</td>
<td>The default string value to be displayed on the dialog</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string or nil</td>
<td></td>
<td>The string entered by the user or nil if he pressed cancel</td>
</tr>
</tbody>
</table>

See Also

getInteger, getDecimal, getSymbol

loadIcon

Params: string (file path), string (format, optional)
Returns: Icon

loadImage

Params: string (path to image file), string (optional, format string)
Returns: Image

Use this function to load a bitmap image from a file. The function automatically recognizes 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.

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>Message to be displayed in the log</td>
</tr>
</tbody>
</table>

Return Values

None.

See Also
logInfo, logWarning

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>Message to be displayed in the log</td>
</tr>
</tbody>
</table>

Return Values

None.

See Also
logError, logWarning

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>Message to be displayed in the log</td>
</tr>
</tbody>
</table>

Return Values

None.

See Also
logInfo, logError

require

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. Contrary 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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>Name of the script to be executed. Must be a script within the repository currently open.</td>
</tr>
</tbody>
</table>

Return Values

None.
See Also
doscript, require in Lua base library

selectColor

Params: 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 left by Cancel, the return value is nil.

selectFont

Params: string (name), number (point size), boolean (bold), boolean (italic). Params are optional.
Returns: string (name), number (point size), boolean (bold), boolean (italic), or all nil.
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 left with Cancel, all four return values are nil.

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>A string representing a directory</td>
</tr>
</tbody>
</table>

Return Values

None.

See Also
getCurrentDir

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The title of the dialog</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td></td>
<td>The message text to be displayed on the dialog</td>
</tr>
<tr>
<td>3</td>
<td>string</td>
<td></td>
<td>The label text on button 1 (left)</td>
</tr>
<tr>
<td>4</td>
<td>string</td>
<td></td>
<td>The label text on button 2 (middle)</td>
</tr>
<tr>
<td>5</td>
<td>string</td>
<td></td>
<td>The label text on button 3 (right)</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The button pressed by the user: 1, 2 or 3</td>
</tr>
</tbody>
</table>

Example

dlg.showError( "Select Peaklist", "Unknown peaklist" )

See Also

showInfo, showWarning
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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The title of the dialog</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td></td>
<td>The message text to be displayed on the dialog</td>
</tr>
<tr>
<td>3</td>
<td>string</td>
<td>x</td>
<td>The label text on button 1 (left)</td>
</tr>
<tr>
<td>4</td>
<td>string</td>
<td>x</td>
<td>The label text on button 2 (middle)</td>
</tr>
<tr>
<td>5</td>
<td>string</td>
<td>x</td>
<td>The label text on button 3 (right)</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The button pressed by the user: 1, 2 or 3</td>
</tr>
</tbody>
</table>

See Also
showError, showWarning

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td></td>
<td>The title of the dialog</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td></td>
<td>The message text to be displayed on the dialog</td>
</tr>
<tr>
<td>3</td>
<td>string</td>
<td>x</td>
<td>The label text on button 1 (left)</td>
</tr>
<tr>
<td>4</td>
<td>string</td>
<td>x</td>
<td>The label text on button 2 (middle)</td>
</tr>
<tr>
<td>5</td>
<td>string</td>
<td>x</td>
<td>The label text on button 3 (right)</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td></td>
<td>The button pressed by the user: 1, 2 or 3</td>
</tr>
</tbody>
</table>

See Also
showError, showInfo

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

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Type</th>
<th>Opt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>number</td>
<td>x</td>
<td>A percent value (0..100) expressing the progress of the calculation, i.e. the elapsed time.</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td>x</td>
<td>Sets the label text above the progress bar</td>
</tr>
</tbody>
</table>

Return Values
None.
See Also
beginProgress, endProgress

Library gui

createButtonGroup
Params: Widget (parent), string (opt. title), number (opt. column count, default 1)
Returns: ButtonGroup

createCheckBox
Params: Widget (parent), string (optional, label)
Returns: CheckBox

createComboBox
Params: Widget (parent)
Returns: ComboBox

createDialog
Params: Widget or nil (parent)
Returns: Dialog

createFrame
Params: Widget or nil (parent)
Returns: Frame

createGrid
Params: Widget or nil (parent), number (column count), boolean (vertical, optional, default true)
Returns: Grid

createGroupBox
Params: Widget (parent)
Returns: GroupBox

createHBox
Params: Widget or nil (parent)
Returns: HBox

createLabel
Params: Widget (parent), string (optional text)
Returns: Label

createLineEdit
Params: Widget (parent)
Returns: LineEdit

createListView
Params: Widget (parent) or nil
Returns: ListView

createMainWindow

Params: none
Returns: MainWindow

createMultiLineEdit

Params: Widget (parent)
Returns: MultiLineEdit

createPopupMenu

Params: Widget (parent)
Returns: PopupMenu

createPushButton

Params: Widget (parent), string (caption, optional)
Returns: PushButton

createRadioButton

Params: Widget (parent), string (caption, optional)
Returns: RadioButton

createScrollView

Params: Widget (parent) or nil
Returns: ScrollView

createSplitter

Params: Widget (parent)
Returns: Splitter

createTabWidget

Params: Widget (parent) or nil
Returns: TabWidget

createTextView

Params: Widget (parent)
Returns: TextView

createVBox

Params: Widget (parent) or nil
Returns: VBox

createWidget

Params: Widget (parent) or nil
Returns: MyWidget
createWidgetStack

Params: Widget (parent)
Returns: WidgetStack

cursorPos

Params: none.
Returns: number (pos. x), number (pos. y)
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.

Library spec

composeLabel

Params: string (atom label), number (offset), number (state: Draft..0, Assigned..1, Finalized..2)
Returns: string (spin label syntax)

createExperiment

Params: SpectrumType (optional), ResidueType (optional)
Returns: Experiment
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).

createPeakList

Params: string (atom type dim. 1), ..., string (atom type dim. d )
Returns: PeakList

createProtonList

Params: number (atom count)
Returns: ProtonList
Use this function to create a proton list with a given number of entries. This number cannot be changed afterwards. The atoms are initialized to "invalid" (but legal) values.

decomposeLabel

Params: string (spin label syntax)
Returns: string (atom label), number (offset), number (state: Draft..0, Assigned..1, Finalized..2)

openPeakList

Params: string (file path to XEASY peak list)
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)" ) )
```

openProtonList

Params: string (file path)
Returns: ProtonList
Tries to parse a proton list 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.

openSpectrum

Params: string (file path to spectrum)
Returns: Spectrum
This function aborts with an error, if the spectrum cannot be loaded. The spectrum type is automatically determined from the file ending. CARA 1.0 RC4 supports XEASY and Bruker spectrum formats.

Library xml

createDocument

Params: string (optional)
Returns: DomDocument
Use this function to create a new XML tree as a DOM (Document Object Model, a W3C standard) 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".

openDocument

Params: string
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 formedness" check of the XML syntax. Syntax errors are reported to the shell where CARA was started from.

parseDocument

Params: string (XML syntax)
Returns: DomDocument
This function creates a DomDocument by parsing the XML string given as parameter. If the string contains syntax errors, the function aborts with an error. 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.