Scripting on aRTist

March 20, 2023

1 Introduction

The aRTist GUI is developed in Tcl/Tk (www.tcl.tk) and comes with an internal command line console. The console is primarily meant for providing the user with operational output, execution history, and debugging information. Additionally it can be used to enter commands.

2 aRTist Console

The console (a.k.a. aRTist console/Tcl console) can be opened from the Tools menu (*Tools⇒Show Console*). It can be used to execute all the commands embody aRTist's functionality, but also general Tcl commands, and commands of the underlying operating system:

`% ::aRTist::GetVersion` *aRTist command to show the version number.* More information about available aRTist commands can be found in the API description.

`% glob *` *Tcl command to show all contents of a directory.* For Tcl/Tk commands see `https://www.tcl.tk/man/tcl8.6/TclCmd/contents.html` and `https://wiki.tcl-lang.org/`.

`% dir` *Command (Windows Command Prompt) to list the contents of a directory.*

With the command `::aRTist::dock .console`, the console window can be docked to the main window like a module window.

3 TCL scripts

Console commands can be entered manually and will interpreted line by line. To prepare a set of commands or whole algorithms they can be saved in files. So called Tcl scripts (text files of Tcl syntax) are supported by the general file handling in aRTist. Consequently, script files can be opened directly from the File menu (*File⇒Open ...*), or by Drag & Drop on the aRTist GUI. The scripts content will be encapsulated in a Namespace "External" to prevent unintentional interference with the aRTist code (a script's procedure/variable `dummy` has to be addressed `::External::dummy` in aRTist).

4 Remote Control

The command `::RemoteControl::Enable` (*Tools⇒Enable remote access*) starts offering a TCP socket (default: *localhost:3658*) for remote access to the aRTist console. The command `::RemoteControl::CloseServer` shuts down this server socket. The command `::RemoteControl::Status` shows the actual connectivity. The socket can be configured from the Advanced tab of the settings (*Tools⇒Settings...*).

Remote access

The aRTist console can be accessed by a telnet client application connecting to the activated socked, e.g. command line '`# telnet localhost:3658`'. While Windows Command Prompt telnet client seams not to work, CYGWIN/Linux telnet or PuTTY are nice clients for testing.

There is also a single-file executable of a Tcl console: tkcon-2.5.exe[1]. This provides the look & feel of the aRTist console in a separated application (the aRTist console is a integrated Tkcon[2]). With Tkcon the connection to aRTist can be established from the Console menu (*Console⇒Attach To ...⇒Socket⇒Create Connection*).

[1] available at http://wintcltk.sourceforge.net/tkwrap.html#tkcon
[2] wiki.tcl.tk/1878

5 Selected commands

5.1 Arranging the virtual setup

The virtual setup is defined by an assembly list of geometrical objects (source, detector, parts) referenced by part IDs. A

- Positioning

  The position of parts (objects of the assembly list) refers to the center of the bounding box in world coordinates. The center of the bounding box is as well the origin of the part's local coordinate system. The parts have an additional reference position acting as rotation center.

  - Get position of part by `::PartList::Invoke $PartID GetPosition`
  - Set position of part by `::PartList::Invoke $PartID SetPosition $X $Y $Z`, where `$X/Y/Z` are the X, Y, and Z values in millimeters.
  - Get reference position of part by `::PartList::Invoke $PartID GetRefPos`
  - Set reference position of part by `::PartList::Invoke $PartID SetRefPos $X $Y $Z`, where `$X/Y/Z` are the X, Y, and Z values in millimeters.
  - Shift a part by `::PartList::Invoke $PartID Translate $System $X $Y $Z`, where `$System` is the coordinate system (`world`, or `object`). The reference position is shifted as well.

- Orientation

  The orientation values define the extrinsically rotations about axes at center of bounding box (Position). The rotations are performed in the order: Y, X, Z. The rotations are right-handed.

  - Get orientation of part by `::PartList::Invoke $PartID GetOrientation`
  - Set orientation of part by `::PartList::Invoke $PartID SetOrientation $X $Y $Z`, where `$X/Y/Z` are angular values around X, Y, and Z axis in degrees.
  - Rotate a part by `::PartList::Invoke $PartID Rotate $System $Alpha $X $Y $Z`, where `$System` is the coordinate system (`world`, or `object`). `$Alpha` is the rotation angle in degrees. The rotation axis is defined by the part's reference position and direction `$X $Y $Z`.
  - Rotate a part with special rotation center by `::PartList::Invoke $PartID Rotate global $Alpha $X $Y $Z $CX $CY $CZ`, where `$CX/CY/CZ` is the rotation center. *Hint: This command is only defined for rotations on global coordinate axes.*

- Sizeing

  - Get size of part by `::PartList::Invoke $PartID GetSize`
  - Set size of part by `::PartList::Invoke $PartID SetSize $Sx $Sy $Sz`, where `$Sx/y/z` are the X, Y, and Z values in Millimeters.
  - Scale part by `::PartList::Invoke $PartID Scale $System $X $Y $Z`, where `$System` is the coordinate system (`world`, or `object`), and `$X/Y/Z` are the scaling factors.
  - Get the Cartesian min/max values (bounding box) of part by `::PartList::Invoke $PartID GetBounds`

- Group transformations

  If `::PartList::Invoke` command is called with a list of part IDs, it will treat all this parts in the same way as separated calls with all the single IDs. The parameter for the part ID can consists of one or more part IDs (list of real part IDs), or the words 'parts' (all parts in the assembly list), 'all' (all parts including source and detector), and 'selection' (all selected parts). *Hint: To treat a group of pats with one `Invoke` command may not be useful in all cases. Different reference positions or local coordinates may block to treat parts as a group.*

  Examples (Only "world" transformations are used, because the parts could have differing local coordinate directions.):

  - Translate all parts by *command*`::PartList::Invoke parts Translate world $X $Y $Z`
  - Rotate tow parts (with ID 1 and 4) for 45° in X around the global origin by *command*`::PartList::Invoke {1 4} Rotate world 45 1 0 0 {0 0 0}`

- Material

- Set material of part by *command*`::PartList::Set $PartID Material $Mat`, where `$Mat` is an valid material name.

- Visibility

  - Activate/deactivate a part by *command* `::PartList::Set $PartID Visible on`, or `::PartList::Set $PartID Visible off`

5.2 Source properties

- Source in virtual scene:

  - Get/set the position of the source in Millimeter of world coordinates:
    command `Source GetPosition`
    command `Source SetPosition <X> <Y> <Z>`
    Hint: This equal (see 5.1) to: *command*`::PartList::Invoke S GetPosition`
  - Get/set the orientation:
    command `Source GetOrientation`
    command `Source SetOrientation <X> <Y> <Z>`
  - Get/set the size, the extent of the detector plane in Millimeter. The Z component is mandatory but without effect. If the size does not fit in the pixel grid, it will be reduzed to the next pixel bounary.
    command `Source GetSize`
    command `Source SetSize <X> <Y> <Z>`

- Source spectrum:

  - A saved spectra can be loaded by *command*`::FileIO::OpenAny [PathToSpectrumFile]`
  - A tube spectra can be generated by *command*`::XSource::ComputeSpectrum` , where the parameters come from the following variables:
    * Xsource(AngleIn)
    * Xsource(FilterMaterial)
    * Xsource(FilterThickness)
    * Xsource(Resolution)
    * Xsource(TargetAngle)
    * Xsource(TargetMaterial)
    * Xsource(TargetThickness)
    * Xsource(Transmission)
    * Xsource(Tube)
    * Xsource(Voltage)
    * Xsource(WindowMaterial)
    * Xsource(WindowThickness)

    Call *command*`::XSource::CheckSpectrum` afterwarts, to potentially reduce the number of bins in the spectra to a maximum of 128 bins.

- Exposure:

  - The variable for the exposure value (mA or GBq) is *variable*`::Xsource(Exposure)`

- Focal Spot:

  - Spot type can be set by *variable*`::Xsetup(SourceSampling)`,where valid values are:
    * "point" for single point source,
    * "<n>" (single number) for a unregular grid of <n> of source points Poison Disk distributed over the spot extent.
    * "<m>x<n>" (tow numbers deliminated by 'x') for a regular grid of <m> by <n> source points.
  - The spot size variables are *variable*`::Xsetup_private(SGSx)` and *variable*`::Xsetup_private(SGSy)`
  - A spot profile image can be provided with `::XSouce::LoadSpot <file name>` and deleted with `::XSouce::ClearSpot` .

5.3 Scattering

- Parameters
  - Mode *variable*`::Xscattering(Mode)` ... Scattering mode. Possible values are: `off`, `build-up factor`, `external file`, `McRay`.
  - reference point *variable*`::Xscattering(AutoBase)` ... This defines the automatic reference point. Possible values are: `off min max center picked`.
  - picked position *variable*`::Xscattering(PickedPosX)`and *variable*`::Xscattering(PickedPosY)` ... Define the reference pixel.
  - build up *variable*`::Xscattering(Buildup)`... Build-up factor.
- McRay
  - Load results as external file *variable*`::Xscattering(McRayInitFile)`... Use Monte Carlo result to set up scatter image for subsequent runs (on: 1, off: 0). If activated, it will switch to scattering mode "`external file`" for next simulation run.
  - Use original image size *variable*`::Xscattering(ResampleMC)`... Don't reduce image size for Monte Carlo calculation (on: 1, off: 0). It is recommended to deactivate this option (set to "0")..
  - Set defaults *variable*`::Xscattering(ResetMcRay)`... Ignore manual settings in McRay module (on: 1, off: 0). It is recommended to activate this option (set to "1").
  - # of photons *variable*`::Xscattering(nPhotons)`... Number of photon trajectories to start.

5.4 Run a simulation

- Use *command*`::Engine::StartStopCmd` to start a simulation run and display the resulting images (same as Run button).
- Calling *command*`::Engine::Go <material>` will just return a list of resulting images. The optional paramiter `<material>` will be ignored in radiography mode. In thickness-map mode, the command will return a simgle image of the thickness map for the requested material `<material>`.

5.5 Detector properties

- The detector in the virtual scene:
  - Get/set the position of the detector in Millimeter of world coordinates:
    command `Detector GetPosition`
    command `Detector SetPosition <X> <Y> <Z>`
    Hint: This equal (see 5.1) to: *command*`::PartList::Invoke D GetPosition`
  - Get/set the orientation:
    command `Detector GetOrientation`
    command `Detector SetOrientation <X> <Y> <Z>`
  - Get/set the size, the extent of the detector plane in Millimeter. The Z component is mandatory but without effect. If the size does not fit in the pixel grid, it will be reduzed to the next pixel bounary.
    command `Detector GetSize`
    command `Detector SetSize <X> <Y> <Z>`
- Geometry:

  "Geometry" here refers to the size, number of pixels, and resolution of the detector image. Since this is an over-determined system, if one parameter is changed, others have to be updated as well. To do this in a defined way, one of the parameters "Size", "Pixels" or "Resolution" must be selected as the parameter to be updated automatically:
  - *variable* ::Xsetup_private(DGauto) ... values: "Size", "Pixel", "Resolution", where a quantity of this type will be adjusted, if a quantity of another type is changed.

  The core geometry parameters are listed here:
  - *variable* `::Xsetup_private(DGSx)` ... detector size in X, widget name: XSize
  - *variable* `::Xsetup_private(DGSy)` ... detector size in Y, widget name: YSize

- *variable* `::Xsetup(DetectorPixelX)` ... number of pixels in X, widget name: XPixel
- *variable* `::Xsetup(DetectorPixelY)` ... number of pixels in Y, widget name: YPixel
- *variable* `::Xsetup_private(DGdx)` ... pixel size in X, widget name: XResolution
- *variable* `::Xsetup_private(DGdy)` ... pixel size in Y, widget name: YResolution
- *variable* `::Xsetup(SquarePixel)` ... force equel pixel size in X and Y, widget name: not needed

After changing one of the listed parameters the *command* `::XDetector::UpdateGeometry` needs to be called with the respective widget name, e.g.:

```
set ::Xsetup(DetectorPixelX) 100; ::XDetector::UpdateGeometry ::XDetector::widget(XPixel)
```

Hint: If you chang one quantity, *variable* ::Xsetup_private(DGauto) must be set to a different type of quantity. Atherwise `::XDetector::UpdateGeometry` will not work properly.

An additional parameter controls the sampling per pixel as an anti-aliasing measure:

- *variable* `::Xsetup(DetectorSampling)` ... values: "source dependent" (same sampling as for focal spot), "X" (Poisson disk sampling with X points), "XxY" (regular grid)
- Exposure

  Some of the parameters may are deactivated depending on the exposure/"reference point" mode. Deactivated parameters do not influence the simulation, or will be overwritten at an simulation run. To keep the GUI self-consistent it is needed to call *[command]* `::XDetector::ExposureModeChange` after changing *[variable]* `::Xdetector(AutoD)`. But the functionality is not depending on it.

  - reference point *[variable]* `::Xdetector(AutoD)` ... This defines the automatic exposure mode. Possible values are: `off min max center picked`. If not "off", the exposure time is determined by the program to reach a pixel value of "set to" at "picked position". To enable a explicitly defined exposure time use "off" here.
  - picked position *[variable]* `::Xdetector(AutoDPosX)`and *[variable]* `::Xdetector(AutoDPosY)` ... Define the pixel position for automatic exposure.
  - set to *[variable]* `::Xdetector(RefGV)` ... Optical density value or gray value desired at reference point.
  - exposure time *[variable]* `::Xdetector(Scale)` ... Exposure time in seconds.
  - number of frames *[variable]* `::Xdetector(NrOfFrames)` ... Number of frames to average.
- Parameter Override

  - unsharpness *[variable]* `::Xdetector(Unsharpness)` ... Detector unsharpness in Millimeter. Active if *[variable]* `::Xdetector(UnsharpnessOn)` is "1", only.
  - long range unsharpness *[variable]* `::Xdetector(LRUnsharpness)` and *{variable]* `::Xdetector(LRRatio)` ... Detector long range unsharpness in Millimeter and ratio in %. Active if *[variable]* `::Xdetector(UnsharpnessOn)` is "1", only.
  - noise factor *[variable]* `::Xdetector(NoiseFactor)` ... Nonlinear multiplier to adjust the noise model of the chosen detector characteristic. Active if *[variable]* `::Xdetector(NoiseFactorOn)` is "1", only.

5.6 Modules

Modules are extentions of aRTist using a common infrastructure. Commands for module managemant are:

`::Mouldes::GetAll` Returns a list of all available modules.

`::Modules::Available <module>` Checks if the module is present. Returns 1 or 0.

`::Modules::Run <module>` Starts a module. This typically opens a GUI of the module.

`::Modules::Invoke <module> <command> [<arguments>]` Calls a subroutine of the module.

`::Modules::Set <module> <variable> [<value>]` Access a variable of module namespace. If no value is specified, the actual value is returned.

5.6.1 CtScan

Subroutines

`::Modules::Run CtScan` Starts/initializes the module.

`::Modules::Invoke CtScan Execute` Starts the CT scan simulation.

Variables

`CtScan` Array variable with all scan parameters. Use `::Modules::Set CtScan CtScan` to show all array members.