Scripting Reference

for

BrainVoyager 22.2.1

Rainer Goebel

Contents

Chapter 1

Introduction

Welcome to the BrainVoyager Scripting Reference! In addition to the new data management tools, scripting can be an option to automate data analysis processes.
This reference can be used for scripting in BrainVoyager 20.x (BrainVoyager QX 3.0) and higher, but most commands can also be used in BrainVoyager QX 2.8, 2.6, 2.4 and 2.3, some even in BrainVoyager QX 2.2, and some in BrainVoyager QX 2.1. This manual contains the BrainVoyager functions (Application Programmer's Interface (API), see Chapter 3), some introduction to the JavaScript language (Chapter 6) and a short section on writing scripts in BrainVoyager with a user interface (see Chapter 5).
When starting with scripting, in BrainVoyager or in general, please also check out our Getting Started with Scripting in BrainVoyager Guide. This can be found in the "GettingStartedGuides" folder of the BrainVoyager directory or online at the BrainVoyager support site.
Please note that scripting with Python, including workflows, is also available in BrainVoyager.

Latest update: 14-10-2021

1.1 Location of script files

Scripts (*.js) for BrainVoyager can be stored in the folder
`(My) Documents/BrainVoyager/Extensions/Scripts/`, so that they will be present in the BrainVoyager "Scripts" menu. An option to specify a custom scripts folder has been added in v2.4.1. Users can switch between standard and custom folder in "Settings" dialog.

1.2 History

1.2.1 Scripting in BrainVoyager

BrainVoyager 20-22.2

Several surface functions have been introduced (such as functions for MTC statistics) and some have been renamed from 'mesh' to 'geometry' (new function names are SmoothGeometrySimple(), SmoothGeometry(), InflateGeometry(), InflateGeometryToSphere(), CorrectInflatedSphereDistortions() and SimplifyGeometry()). The function CorrectSliceTiming() and CorrectSliceTimingWithSliceOrder() received a third argument.

BrainVoyager 21.4.5

When applying MTC preprocessing commands, the `FileNameOfPreprocessdMTC` string property always returned an empty string. This issue has been fixed.

BrainVoyager 21.4

It is now possible to load and save NIfTI files via Python (and JavaScript) scripts. There is no new command - the respective functionality is invoked in case that the file name parameter of the `OpenDocument()` command of the global `BrainVoyager` object contains the file extension ".nii" or ".nii.gz". The NIfTI file extension also triggers exporting VMR anatomical and FMR functional documents as NIfTI files when using the `SaveAs()` command of a `Document` object. Note that the earlier deprecated opening and creation document command names using `Project` (e.g. `CreateProjectVMR()`) have now been removed, i.e. scripts need to use the corresponding `Document` commands (e.g. `CreateDocumentVMR()`).

BrainVoyager 21

Replaced `GetMeshScene()` by CreateMeshScene(). The 'BrainVoyagerQX' object cannot be used anymore, please replace by 'BrainVoyager'.

BrainVoyager 20.6/BrainVoyager QX 3.6

Some commands have been added to BrainVoyager related to surface maps: SmoothMap(), SmoothMapLags(), mesh.CreateSurfaceMapFromVolumeMap() and mesh.CreateSurfaceMapFromVolumeMapDepth(). Some documentation has been added for map functions: LoadVolumeMaps(), ShowVolumeMap(), mesh.ShowSurfaceMap(), mesh.LoadSurfaceMaps().

BrainVoyager 20.4/BrainVoyager QX 3.4

The `BrainVoyagerQX` object can now be replaced by `BrainVoyager`, but it will remain backwards compatible (until version 21.0). A new function for obtaining an isovoxel anatomical image, TransformToIsoVoxel(), has been added. Further new functions are: CoregisterFMRToVMR(), CoregisterFMRToVMRUsingBBR(), NormalizeToMNISpace(), CreateVTCInMNISpace() and property `Path`. Note also that the "CreateProjectFMR/DMR/VMR/AMR" commands have been renamed to "CreateDocumentFMR/DMR/VMR/AMR" to be consistent with the terminology used in the data analysis manager and now generally in BrainVoyager. For the 20.x releases, the "CreateProject..." names can still be used, but they will be removed in version 21.

BrainVoyager 20/BrainVoyager QX 3.0

A large part of the scripting for analysis in volume space can now be performed via the new data management tools, which may replace scripting. There is one new scripting function `GetMeshScene()`, which replaces the VMR property `CurrentMeshScene` (see section 3.13). In addition to JavaScript, it is now possible to use the Python scripting language.

BrainVoyager QX 2.8

There are a lot of new functions for surface processing in this version, for example for inhomogeneity correction, transformation of anatomical data to AC-PC and Talairach space, morphing meshes, running single-study GLMs and cortex-based alignment. See sections 3.12 (transformations and normalisation) and 3.13 (surface functions) of the API for more information. Also, there are now mesh and mesh scene objects, which can be obtained via the anatomical object (see example scripts).

v2.8.2 Two new functions and a property, concerning correction of slice timing and mean intensity adjustment, have been introduced.

v2.8.4 Added new script commands to handle VOIs, including loading .voi files, selecting and displaying VOIs in VMR windows; furthermore, a single-study VOI-GLM command has been added that is useful for batch analyzing VOI data from multiple subjects, e.g. in the context of analyzing neurofeedback modulation results; new VOI-GLM contrast commands allow to specify contrasts and to access resulting contrast statistics. For details, see section 3.11 including the example script "`NewCommands_v284.js`".

BrainVoyager QX 2.6

The property `UseBoundingBoxForVTCCreation` is now default set to false, so one only needs to mention the property in a script (set to 'true') in case one wishes to use this numerical bounding box creation.

BrainVoyager QX 2.4.1

It is now possible to perform *temporal high-pass filtering (drift removal) using the GLM approach* using Fourier or discrete cosine transform (DCT) basis functions. In previous versions, only the FFT-based high-pass filtering was available. The new commands are "TemporalHighPassFilterGLMFourier()" and "TemporalHighPassFilterGLMDCT()" with one parameter that specifies the number of cycles (pairs of two basis functions) used to build an appropriate design matrix. The installed script "HighPassFilterUsingGLM.js" shows how to use the new commands. Other new scripting commands allow to interrogate information about the running BrainVoyager version including the build number and whether the program is running in 32 or 64 bit mode. The installed script "VersionScript.js" shows how these commands can be used.
Scripts are installed in a standard location within the user's "Documents" folder ("BVQXExtensions/Scripts"). For some scenarios, it would be beneficial if scripts could be accessed from a custom folder as default, i.e. when written scripts are made available to members of a research group in a shared network folder. For such scenarios it is now possible to change the default scripts folder in the "Scripts" tab of the "Global Preferences" dialog.
There are two new possibilities for manipulating VTCs. The first is that the manual specification of bounding boxes is enabled. This works for any target reference space. Use the property 'UseBoundingBoxForVTCCreation' in combination with TargetVTCBoundingBoxXStart (or Y, or Z) and TargetVTCBoundingBoxXEnd (or Y, or Z).
It is now possible to save the VTC after a protocol has been linked, using the command SaveVTC().

BrainVoyager QX 2.3

BrainVoyager QX can now be used in combination with AppleScript on Mac OS X. For more information, see the 'BVQXAppleScripting.pdf' guide.
For Qt Script there are the following additions. When creating VMR projects, the internally created V16 data set is now stored to disk. When saving the VMR data with a new name ("save as" command used usually after VMR creation), both files will be renamed as long as they have the default "untitled.vmr/.v16" file name.
The new command "CorrectSliceTimingWithSliceOrder" allows to run slice scan correction with a custom slice order (see "Preprocessing.js" script). The "getCurrentDirectory()" function is now a property, i.e. you can use "BrainVoyagerQX.CurrentDirectory" to read and set its value.
There are also new properties pointing to common locations:

The "PathToData" property points as default to the "BVQXData" folder in the user's "(My)Documents" folder and the "PathToSampleData" points as default to the "BVQXSampleData" folder.
Please note that the `FileNameOfPreprocessedFMR` now provides the filename and the path, not just the filename.

BrainVoyager QX 2.2

There are now reading and writing (File I/O) possibilities. Also, the COM (component object model) functionality has been implemented, which means that communication between COM-enabled applications on Windows is possible, for example scripting BrainVoyager from Matlab.
New BrainVoyager script functions are available for preprocessing VTC files, creation of MTC from VTC and create function handles.

BrainVoyager QX 2.1

The language is now fully ECMA-script compliant, which means it is basically JavaScript. Most of the language features are the same; the graphical user interface (GUI) widgets can be made using external user interface files (*.ui)(see section 5).
The parameter `dataType` has been added for creating VTC and VDW files. Two properties for changing the confound in SDM files have been added.

BrainVoyager QX 2.0

No changes.

BrainVoyager QX 1.10

In BrainVoyager QX 1.10.4, it is also possible to create VDW files via scripting.
In BrainVoyager QX 1.10.3, the types of interpolation that can be selected have been extended. For more information, please consult the Interpolation in motion correction page.

BrainVoyager QX 1.9

This version is updated for BrainVoyager QX 1.9. Two important new changes in BrainVoyager QX 1.9 are the DTI analysis functionality and scripting via the component object model (COM)(this works on Windows platforms). Concerning COM, a separate guide appeared called ScriptingBrainVoyagerQXfromMatlab.pdf. For a short introduction, please see the topic Using BrainVoyager via COM.
Also, there are 5 new scripting functions:
RenameDicomFilesInDirectory(), BrowseFile(), BrowseDirectory(), CreateProjectDMR() and CreateProjectMosaicDMR().

For details on creating diffusion weighted projects (DMR), see the topic Creating DMR projects.

For the function to rename DICOM files and the use of BrowseDirectory(), please see the new rename DICOM files page.

The new BrainVoyager Getting Scripted Guide can be consulted for a step-by-step approach into scripting.

In 1.9.10, the number of interpolation options has increased for slice scan time correction* and VTC creation. For details, see the BrainVoyager QX 1.9.10 Release Notes.
Changes in the programming language itself concern, for example, the `undefined` which is in BrainVoyager QX 1.9 an object (so no double quotes are needed). Also, the arguments for the getOpenFileName(s) functions have changed. The language specification for Qt Script 1.2.2 by Trolltech can be found also in this guide.

1.2.2 Documentation history

13-10-2021

Added third argument to CorrectSliceTiming() and CorrectSliceTimingWithSliceOrder() . With thanks to Dr. Bick.

29-09-2021

Several surface functions have been added and some renamed (like SmoothGeometry() instead of `SmoothMesh()`). Updated the CBA flowchart.

28-06-2021

Added argument to the UpdateAppearance() function.

30-04-2020

Added script example to open NIfTI files to the OpenDocument() function. Updated scripting history.

19-06-2019, 20-08-2019

Updated scripts to BrainVoyager 21; the script to create documents has been updated using the new Getting Started Guide data. Added some information about the MessageBox() function, changed `Document` functions to `Project` functions and added NIfTI extension. Updated some screencaptures to "dark mode". Removed section about Visual Basic Script. Add descriptions for the four *.mtc preprocessing functions. Some updating of text and references.

15/16-08-2017

Note about CorrectIntensityInhomogeneities().

23-05-2017

Changed filter for BrowseFile().

09-05-2017, 11-04-17

Changed description of standard parameters for CorrectMotion() (interpolation). Added remark about resulting filename with `_SCCTBL_` for CorrectSliceTimingUsingTimeTable().

27-02-2017

Added description of Remove(), which deserved more attention. Added 'image registration in BrainVoyager'-diagram.

31-01-2017

Corrected CreateVTCInMNISpace() description. Added AddPredictor(), SaveAs(), which were somehow omitted before.

24-01-2017

Added new scripting functions for BrainVoyager 20.4.

v2.1

Added remark about `BrainVoyager` object.

v2.0

A fix in the VOI-GLM script (thanks to David Mehler). Further added some VOI functions that were not yet documented. Changed the naming in the text from 'BrainVoyager QX' to 'BrainVoyager'. Added diagram 3.4 for contrasts. Finally, fixing links and omissions in the text.

v1.9

Added a diagram and examples to the description of regular expressions (section 6.3.4). Also, diagrams for statistical procedures were added (figures 3.2, 3.3).

v1.8

Removed the section 'Switching from BrainVoyager QX 2.0 to 2.1', because it is outdated. Elaborated the File I/O chapter and added descriptions per widget in the chapter about GUI scripts. Added filter options for `BrowseFile()`.

v1.7

Added the ROI/VOI functions and properties to list of functions and as section 3.11 *24-11-14*.

v1.6

Added information about new slice scan time correction and mean intensity adjustment functions and paragraph 2.5.1 about setting breakpoints during debugging. Also, paid some more attention to the BrainVoyagerQX object functions BrowseFile() and BrowseDirectory().

v1.5

Added new VMR functions and mesh scene object functions to documentation.

v1.4

Added mesh object functions.

v1.3

Added descriptions of String functions (section 6.3.2).

v1.2

Corrected description of `SliceScanTimeCorrection()`.

v1.1

03-04-2013 Corrected the property `CorrectForSerialCorrelations`: value is integer so that both AR(1) and AR(2) can be applied.

v1.0

08-06-2012 Added a section on design matrix properties (in particular, setting confounds and constant predictors) and brief information about setting bounding boxes for VTCs and saving VTCs.

v0.9

08-03-2012 Added short section about the script object, added the new functions from BrainVoyager QX 2.4.1 to the API documentation.

v0.8

07-02-2012 Added a section about using Qt Objects in BrainVoyager scripting (section 6.4.2) and added some information to the File I/O chapter (Chapter 4). Improved layout and markup of the document.

v0.7

21-11-2011 Added the GUI scripts example, moved the "Switching from BrainVoyager QX 2.0 to 2.1 scripting" to the Appendix.

Chapter 2

The BrainVoyager Script Editor

2.1 General properties of the BrainVoyager Script Editor

In the BrainVoyager Script Editor it is possible to perform all scripting operations, namely creating a script, debugging a script en running a script.

2.1.1 The user interface

The Script Editor will be displayed in 3/4 window. To see the script editor buttons “Load”, “Save”, “Save As”, “Close”, “Debug” and “Run” on Mac OS X, press the green circle in the left upper corner of the window. The line numbers of the script are provided on the left side of the screen. The currently selected line in the script will be colored light yellow.

2.2 Creating scripts

A script can be created in the BrainVoyager script editor (see figure 2.2). The script editor can be opened via the “Scripting...” menu option in BrainVoyager (see figure 2.1).

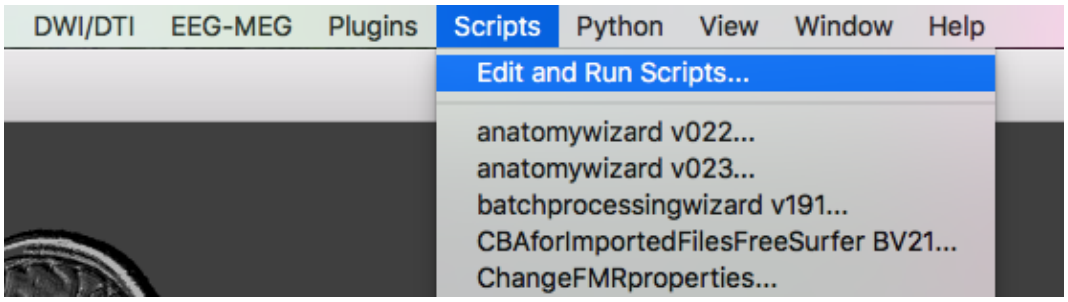


Figure 2.1: Open the Scripting Editor

2.3 Running scripts

There are two ways to run a script. If the script has no user interface, the script can be run directly from the BrainVoyager "Scripts" menu (see the Example scripts being listed in the "Scripts" menu of figure 2.1), provided that the scripts are present in the folder `Documents/BVExtensions/Scripts/`. It is also possible to run the script by clicking the "Run" button in the BrainVoyager Script Editor. Scripts that come with a user interface, so that it has a dialog(s) which is defined in an accompanying *.ui file, can be started by clicking the "Run" button in the BrainVoyager Script Editor.
For running a script, load the script if it is not opened yet via the button "Load" in the lower left corner of the Scripting Editor. Then, click "Run" in the lower right corner of the Scripting Editor window. All command lines that are not embedded within a `function ... { }` block will be executed (of course these lines can invoke the functions).

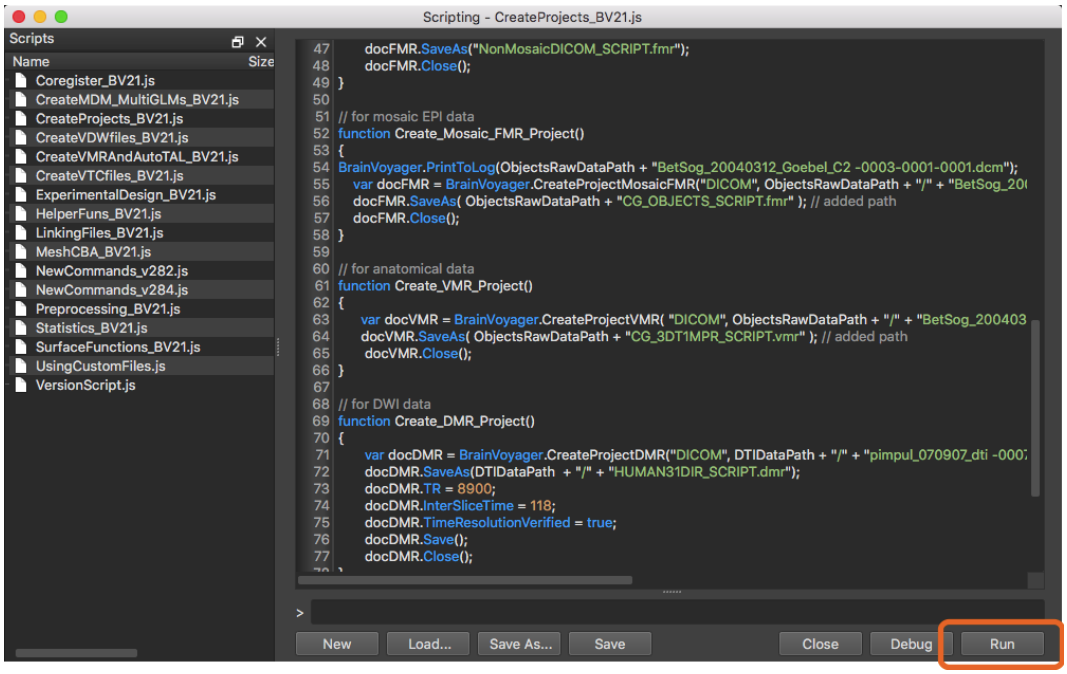


Figure 2.2: Running a script

The script will run; effects may be visible via BrainVoyager dialogs (see figure 2.2) or messages printed to the BrainVoyager Log tab (see figure 2.3).

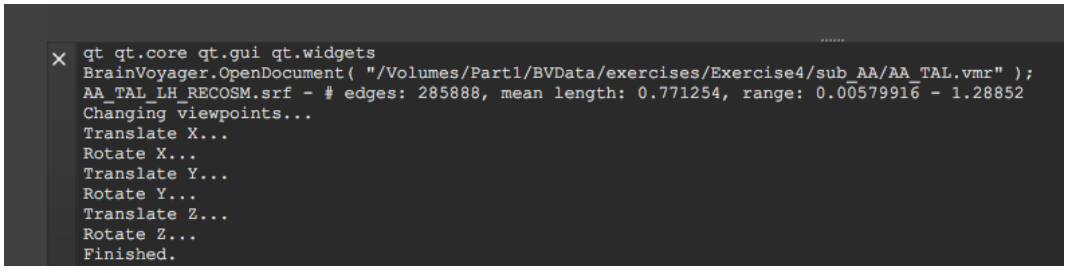


Figure 2.3: Messages printed to the BrainVoyager Log tab while running a script

2.4 Using the interpreter

In the BrainVoyager Script Editor, also a direct interpreter is available. Type the command to be evaluated in the command line field, which is depicted in figure 2.4, then press "Enter".

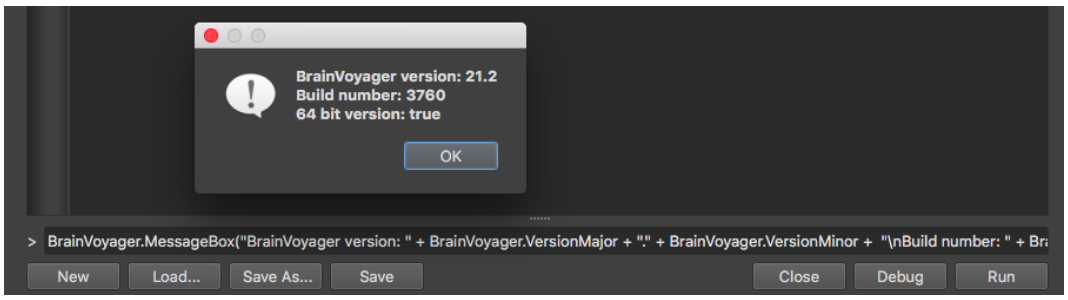


Figure 2.4: Commands typed in this field will be evaluated directly

The command and the output of the command will be visible in the command history window directly above the command line (see figure 2.5). If the command does not produce output, the command history window will print `undefined`, otherwise the output is placed under the command in the history window.

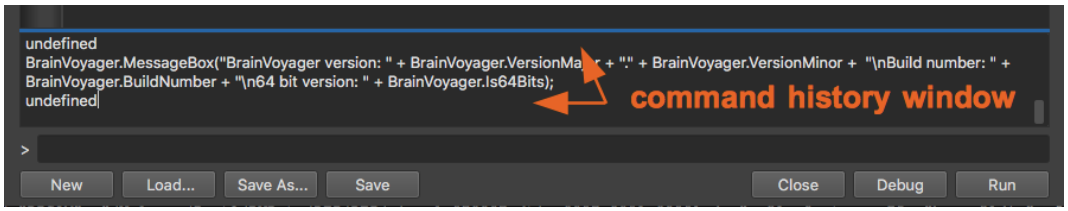


Figure 2.5: Commands typed in the command line are printed in the command history window

2.5 Debugging

When an error cannot be found, it can be useful to try the debugging functionality. Click the "Debug" button and the window as depicted in figure 2.6 will appear. This will locate the error. If the script does not contain bugs after pressing 'Run' (green arrow) in the Debugger, the Debugger will disappear after running the script and the Script Editor will be shown again.

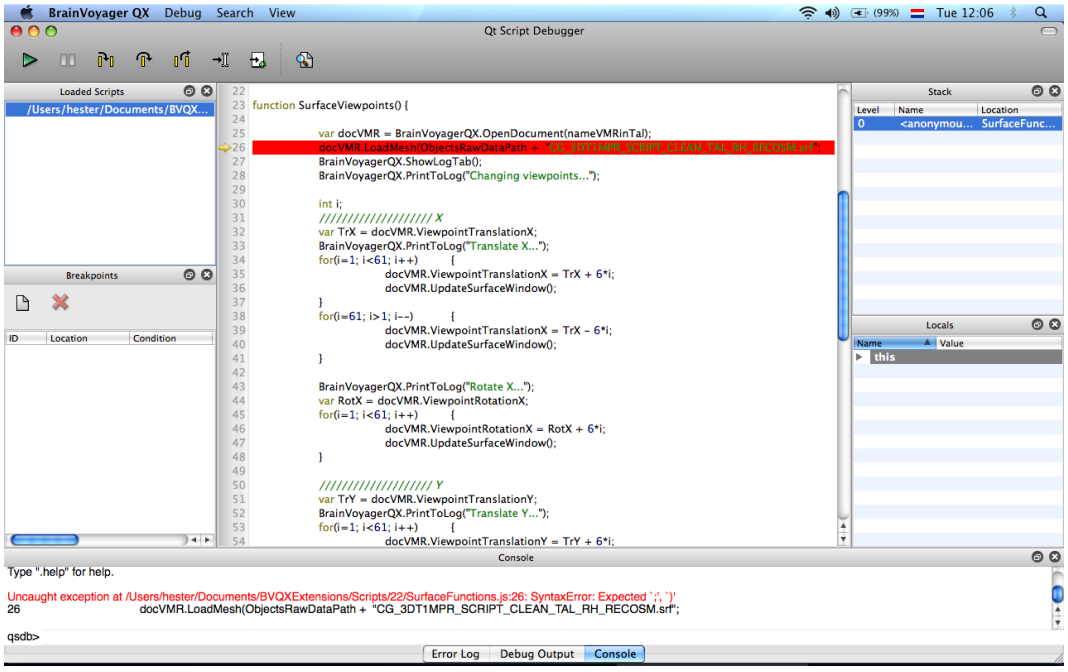


Figure 2.6: Debugging in the Scripting Editor

2.5.1 Setting breakpoints

A breakpoint can also be used go stepwise through the script while it is running, in order to inspect the values of the variables in scope. Click left of the line to set a breakpoint (a round, red dot); click again to remove the breakpoint. A list of all breakpoints with line numbers can be found on the left hand side of the Debugger dialog. When the script runs until the breakpoint, all values that have been assigned to variables in scope, will be visible under 'Scope' on the right hand side of the Debugger dialog.

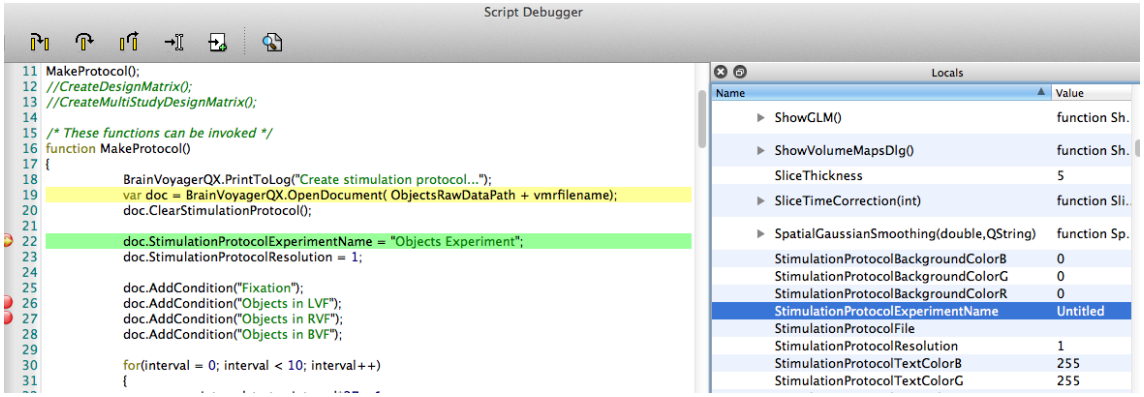


Figure 2.7: Using run to cursor

In figure 2.8, we see for example that the project property `StimulationProtocolExperimentName` has now been filled (starting with 'Objects Exp ...'); in figure 2.7 we see that this was still 'Untitled'.

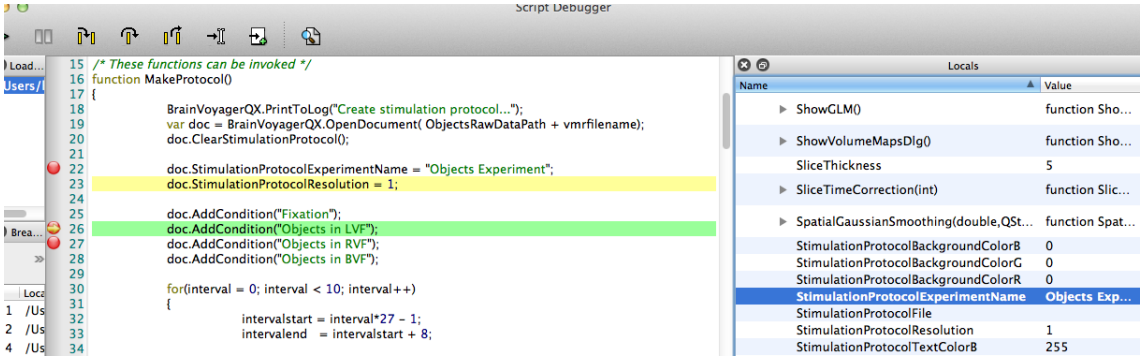


Figure 2.8: Setting breakpoints

Using the button 'Step into' (figure 2.9), the Debugger will run the script line by line.

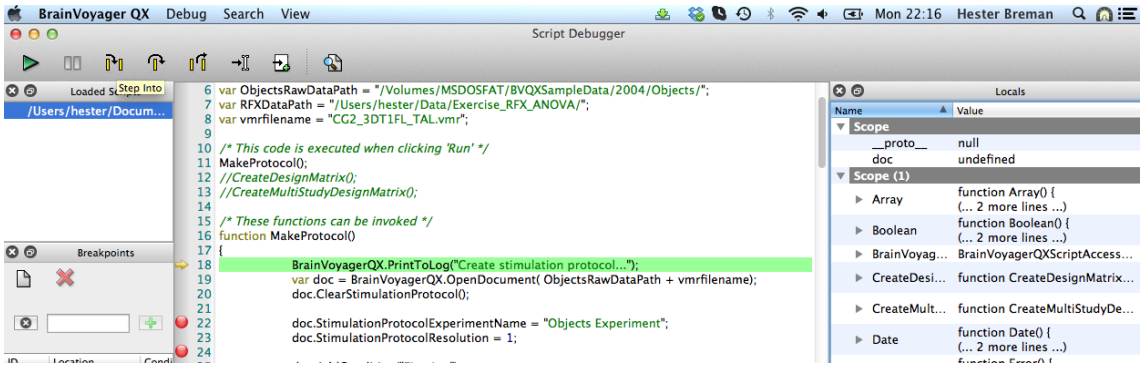


Figure 2.9: The 'Step into' button runs the script line by line (upper left hand side of figure). Note that the doc (VMR) object is still 'undefined' at line 18 (upper right hand side).

The yellow arrow will indicate at which line the Debugger is. In the Scope section, we will see which values have been assigned to our object `doc` (see figure 2.10).

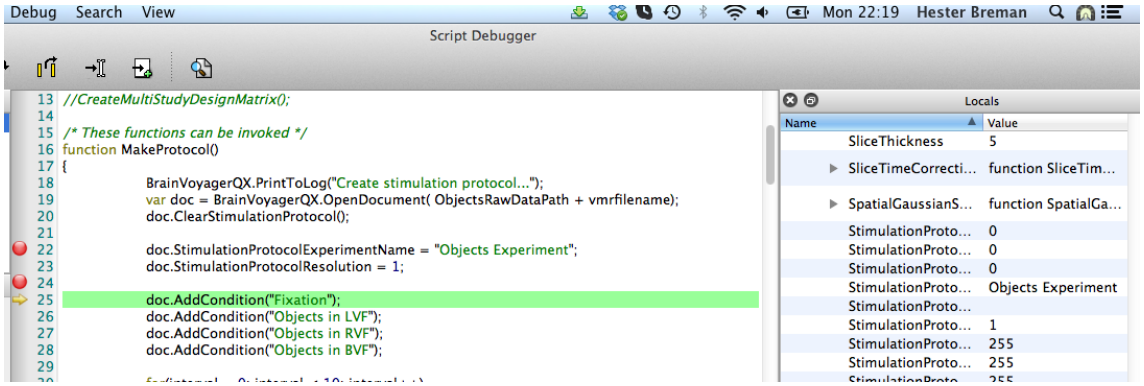


Figure 2.10: Now the Debugger has run the script until line 25, the doc object now has values assigned; these can be inspected under Locals: Scope on the right hand side.

Chapter 3

BrainVoyager Scripting Commands (API)

3.1 Introduction

In this chapter is documented which BrainVoyager scripting methods are available. Within BrainVoyager, one can run JavaScript (this reference) or Python (see Developer's Guide in the /BrainVoyager/UsersGuide/ directory). On Windows it is also possible to run BrainVoyager from Matlab using COM (see the PDF in /BrainVoyager/GettingStartedGuides/ directory).
On Mac, AppleScript can be used (see the guide in the /BrainVoyager/UsersGuide/ directory).

3.2 List of all methods

AddCondition()
AddContrast()
AddCurvatureFileForGroupCBA()
AddPredictor()
AddInterval()
AddMesh()
AddROIContrast()
AddStudyAndDesignMatrix()
AdjustMeanIntensity()
AnonymizeDicomFilesInDirectory()
ApplyHemodynamicResponseFunctionToPredictor()
AutoACPCAndTALTransformation()
AutoTransformToIsoVoxel()
AutoTransformToSAG()
BrowseDirectory()
BrowseFile()
ClearContrasts()
ClearDesignMatrix()
ClearGroupCBACurvatureFiles()
ClearMultiStudyGLMDefinition()
ClearStimulationProtocol()
Close()
ComputeMultiStudyGLM()
ComputeRFXGLM()
ComputeSingleStudyGLM()
ComputeSingleStudyGLMForVOI()
CoregisterFMRToVMR()
CoregisterFMRToVMRUsingBBR()
CorrectIntensityInhomogeneities()
CorrectMotion()
CorrectMotionEx()
CorrectMotionTargetVolumeInOtherRun()
CorrectMotionTargetVolumeInOtherRunEx()
CorrectSliceTiming()
CorrectSliceTimingUsingTimeTable()
CorrectSliceTimingWithSliceOrder()
CreateAverageCurvatureGroupMap()
CreateAverageFoldedGroupMesh()
CreateMTCFromVTC()
CreateDocumentAMR()
CreateDocumentDMR()
CreateDocumentFMR()
CreateDocumentFMRSlicesTimeLooping()
CreateDocumentMosaicDMR()
CreateDocumentMosaicFMR()
CreateDocumentVMR()
CreateStandardSphereMesh()
CreateVDWinACPCSpace()
CreateVDWInTALSpace()
CreateVDWInVMRSpace()
CreateVTCInACPCSpace()
CreateVTCInMNISpace()
CreateVTCInTALSpace()
CreateVTCInVMRSpace()
GetBetaNameOfROIGLM()
GetBetaValueOfROIGLM()

GetMeshScene()
GetNameOfROIContrast
GetNameOfVOI()
GetPValueOfROIContrast
GetTValueOfROIContrast
GetVoxelIntensity(x, y, z)
HideAllVOIs()
LoadMultiStudyGLMDefinitionFile()
LinearTrendRemoval()
LinearTrendRemoval() (VTC)
LinkAMR()
LinkMTC()
LinkStimulationProtocol()
LinkVTC()
LoadGLM()
LoadMesh()
LoadSingleStudyGLMDesignMatrix()
LoadVOIFile()
LoadVolumeMaps()
MapSphereMeshFromStandardSphere()
mesh.AddStudyAndDesignMatrixAndCortexMapping()
mesh.CalculateCurvature()
mesh.CalculateCurvatureCBA()
mesh.ClearDesignMatrix()
mesh.ClearMultiStudyGLMDefinition()
mesh.ComputeMultiStudyGLM()
mesh.ComputeRFXGLM()
mesh.ComputeSingleStudyGLM()
mesh.CorrectInflatedSphereMesh()
mesh.CorrectInflatedSphereDistortions()
mesh.CreateMTCFromVTC()
mesh.CreateMultiScaleCurvatureMap()
mesh.CreateSphericalCoordinatesMapFromSMP()
mesh.CreateSurfaceMapFromVolumeMap()
mesh.CreateSurfaceMapFromVolumeMapDepth()
mesh.InflateGeometry(
mesh.InflateMeshToSphere()
mesh.InflateGeometryToSphere()
mesh.InflateGeometryToSphereExt()
mesh.InflateMesh()
mesh.LoadGLM()
mesh.LinearTrendRemoval()
mesh.LinkMTC()
mesh.LoadMultiStudyGLMDefinitionFile()
mesh.LoadSingleStudyGLMDesignMatrix()
mesh.MeshScene.UpdateSurfaceWindow()
mesh.SaveAs()
mesh.SaveMultiStudyGLMDefinitionFile()
mesh.ShowGLM()
mesh.ShowSurfaceMap()
mesh.SimplifyGeometry()
mesh.SaveSurfaceMaps()
mesh.GetNameOfSurfaceMap()
mesh.SmoothGeometry()
mesh.SmoothGeometrySimple()
mesh.SmoothMesh()
mesh.SmoothRecoMesh()
mesh.SmoothCurrentMap()

mesh.SpatialSmoothing()
mesh.SaveSingleStudyGLMDesignMatrix
mesh.SaveGLM()
mesh.SaveMTC()
mesh.TemporalHighPassFilterFFT()
mesh.TemporalGaussianSmoothing()
mesh.UpdateAppearance()
mesh.Update3DViewer()
meshScene.MergeMeshesInScene()
MessageBox()
MoveWindow()
NormalizeToMNISpace()
OpenDocument()
PrepareROIContrasts()
PrintToLog()
Remove()
RenameDicomFilesInDirectory()
RunCBA()
RunRigidCBA()
SaveAs()
SaveGLM()
SaveMesh()
SaveMultiStudyGLMDefinitionFile()
SaveSingleStudyGLMDesignMatrix()
SaveSnapshotOfSurfaceWindow()
SaveSnapshotOf3DViewer()
SaveStimulationProtocol()
SaveVTC()
ScalePredictorValues()
SelectVOI()
SetConditionColor()
SetContrastString()
SetContrastValue()
SetContrastValueAtIndex()
SetCurrentContrast()
SetCurrentContrastAtIndex()
SetPredictorValues()
SetPredictorValuesFromCondition()
SetStandardSphereToFoldedMesh()
SetVoxelIntensity(x, y, z, intensity)
ShowGLM()
ShowLogTab()
ShowSelectedVOIs()
ShowVolumeMap()
SmoothMap()
SmoothMapLags()
SpatialGaussianSmoothing()
SpatialGaussianSmoothing() (VTC)
TemporalGaussianSmoothing()
TemporalGaussianSmoothing() (VTC)
TemporalHighPassFilter()
TemporalHighPassFilterGLMDCT()
TemporalHighPassFilterGLMFourier()
TemporalHighPassFilter() (VTC)
TransformToIsoVoxel()
UpdateSurfaceWindow()

3.3 BrainVoyager-specific classes

Since "BrainVoyager QX" has evolved into "BrainVoyager", the main application object can be called `BrainVoyagerQX` or `BrainVoyager` (from version 20.4).
A BrainVoyager document in the scripting module is an FMR, VMR, DMR, AMR project. To retrieve a pointer to a BrainVoyager document object, use the property `ActiveDocument`:
```
var doc = BrainVoyager.ActiveDocument;
```
when the project is currently open in BrainVoyager. If it needs to be opened first, provide the document name and use the function `OpenDocument()`:
```
var doc = BrainVoyager.OpenDocument(docname);.
```

3.4 Functions of the BrainVoyager object

3.4.1 List of methods

AnonymizeDicomFilesInDirectory()
RenameDicomFilesInDirectory()
PrintToLog()
ShowLogTab()
MessageBox()
MoveWindow()
BrowseFile()
BrowseDirectory()
OpenDocument()
Also the `CreateDocument()` functions below (see section 3.5) are functions of the BrainVoyager object. All other functions in BrainVoyager's API are functions of the Document object (AMR/FMR/DMR/VMR projects).

BrainVoyager properties

x: Get or set the position on the x-axis of the BrainVoyager window.
y: Get or set the position on the x-axis of the BrainVoyager window.
ActiveDocument: The currently opened document (a BrainVoyager AMR/FMR/DMR/VMR project).
CurrentDirectory: You can use "BrainVoyagerQX.CurrentDirectory" to read and set its value (changed from function to property in QX 2.3).
MessageBox: Display message via a text box. *Path:* This readonly string property provides the path of any loaded document. The returned string contains the full path to the directory where the document is stored on disk. The path component delimiter is the "/" character, which is also the trailing element of the returned path. The returned value can also be derived from the PathFileName property by removing the last (i.e. file name) component. *PathToData:* This property points as default to the "BrainVoyagerData" folder in the user's "(My)Documents" folder (added in BrainVoyager QX 2.3).
PathToSampleData: This property points as default to the "BVSampleData" folder (added in QX 2.3).
VersionMajor: Get version number of BrainVoyager.
VersionMinor: Get subversion number of BrainVoyager.
BuildNumber: Get build number of BrainVoyager.
Is64Bits: If BrainVoyager version is 64 bits, value is true; otherwise the version is 32 bits.

3.4.2 Detailed description of methods

AnonymizeDicomFilesInDirectory()

AnonymizeDicomFilesInDirectory()
Description: This function renames DICOM files to a standard format that is easier to process by functions reading raw data files operating in the same way as the "RenameDicomFilesInDirectory" command (see above). The patient's name is replaced by the provided new patient's name to anonymize the file resulting file names. The provided new name is also used to replace the value of the patient's name entry in the DICOM file (0010,0010) providing basic anonymization (other patient data such as date of birth is not changed). Note that all DICOM files found in the specified folder receive the same new patient's name, i.e. all files in the directory should be from the same participant.
Parameter 1: String specifying the path (folder) containing the files to process. If a relative path is provided, make sure that the current directory is set appropriately.
Parameter 2: String specifying the new patient name replacing the original name found in the DICOM file (0010,0010). The name is only replaced if the existing size of the text entry is large enough to hold the new patient's name, i.e. the length field of the tag is not adjusted at present and in that case a message is written to the Log pane; it is thus advised to use short names, such as "P24".

RenameDicomFilesInDirectory()

RenameDicomFilesInDirectory()
Description: Rename all DICOM files in the current directory.
Parameter 1: Name of directory.

PrintToLog()

PrintToLog()
Description: Print text to the BrainVoyager Log tab.
Parameter 1: Text to print.

ShowLogTab()

ShowLogTab()
Description: Show the BrainVoyager Log tab.

MoveWindow()

MoveWindow()
Description: Move the BrainVoyager window to a new position on the screen.
Parameter 1: New position on x-axis.
Parameter 2: New position on y-axis.

MessageBox()

MessageBox()
Description: Show a text via a message box. User has to click 'OK' to make active box disappear.
Parameter 1: Text to display on message box.

BrowseFile()

See BrowseFile() in File I/O.

BrowseDirectory()

See BrowseDirectory() in File I/O.

OpenDocument()

OpenDocument()
Description: Open a BrainVoyager project.
Parameter 1: Name of the FMR/AMR/VMR/DMR project to open. This can also be a NIfTI file (*.nii, *.nii.gz).
Returns: The project. If the file is a NIfTI file, it will be opened in a native BrainVoyager format.

3.4.3 Example script

```
// "OpenNIfTIfiles.js"

var filename = BrainVoyager.BrowseFile("Please select a NIfTI file", "*.nii.gz");
BrainVoyager.OpenDocument(filename);
```

3.5 Create documents

3.5.1 List of Methods

CreateDocumentFMR()
LinkAMR()
CreateDocumentMosaicFMR()
CreateDocumentFMRSlicesTimeLooping()
CreateDocumentDMR()
CreateDocumentMosaicDMR()
CreateDocumentVMR()
LinkStimulationProtocol()
LinkVTC()
CreateDocumentAMR()
SaveAs()
Close()

3.5.2 Create FMR document

CreateDocumentFMR()

CreateDocumentFMR() *Function:* CreateDocumentFMR(FileType, FirstFileName, NrOfVolumes, nrOfVolumesToSkip, createAMR, nrOfSlices, prefixSTCs, swapBytes, resX, resY, nrBytes, savingDir).
Description: FMR documents consist of a set of functional data in the original "slice space".
Member of class: BrainVoyager.
Parameter 1: FileType (string): file type of original data. One of `"DICOM"`, `"SIEMENS"`, `"GE_I"` (no parameters for +20 logic like in GUI),`"GE_MR"`, `"PHILIPS_REC"` or `"ANALYZE"`.
Parameter 2: FirstFileName: name of the first "raw" data file.
Parameter 3: NrOfVolumes: number of volumes for this document.
Parameter 4: nrOfVolumesToSkip: number of volumes that should be skipped at the beginning, so number of volumes in FMR document will be (NrOfVolumes - nrOfVolumesToSkip).
Parameter 5: createAMR: boolean (true or false): create AMR from first EPI volume or not.
Parameter 6: nrOfSlices: number of slices per volume.
Parameter 7: prefixSTCs: name for the stc file.
Parameter 8: swapBytes: swap bytes (true or false).
Parameter 9: resX - dimension of image along x-axis
Parameter 10: resY - dimension of image along y-axis
Parameter 11: nrBytes - number of bytes per pixel, usually 2.
Parameter 12: savingDir - directory for saving the FMR document.
Returns: Document

CreateDocumentMosaicFMR()

Function: CreateDocumentMosaicFMR()
Description: FMR documents consist of a set of functional data in the original "slice space". The "Mosaic" version of FMR creation is necessary when reading Siemens files from scanning sequences which store several slices within a single image. The format of such "mosaic-images" is not a stack of slices (i.e. as in ANALYZE files), therefore special treatment is required.
Member of class: BrainVoyager.
Parameter 1: FileType (string): file type of original data. One of `"DICOM"`, `"SIEMENS"`, `"GE_I"` (no parameters for +20 logic like in GUI),`"GE_MR"`, `"PHILIPS_REC"` or `"ANALYZE"`.
Parameter 2: FirstFileName
Parameter 3: NrOfVolumes
Parameter 4: nrOfVolumesToSkip
Parameter 5: createAMR
Parameter 6: nrOfSlices
Parameter 7: prefixSTCs
Parameter 8: swapBytes
Parameter 9: mosaicResX - mosaic size: dimension of images in volume along x-axis
Parameter 10: mosaicResY - mosaic size: dimension of images in volume along x-axis
Parameter 11: nrBytes

Parameter 12: savingDir
Parameter 13: volsInImg - number of volumes per file
Parameter 14: resX - dimension of image along x-axis
Parameter 15: resY - dimension of image along y-axis
Returns: Document
Example:

```
 var docFMR = BrainVoyager.CreateDocumentMosaicFMR("DICOM",
ObjectsRawDataPath + "BetSog_20040312_Goebel_C2 -0003-0001-0001.dcm", 252, 2, true,
 25, "untitled-", false, 320, 320, 2, ObjectsRawDataPath, 1, 64, 64 );
```

CreateDocumentFMRSlicesTimeLooping()

Function: CreateDocumentFMRSlicesTimeLooping()
Description: See CreateDocumentFMR(). The current function is similar to using the "Slices x time" checkbox via the BrainVoyager Create Document dialog.
Member of class: BrainVoyager.
Parameter 1: FileType (string): file type of original data. One of `"DICOM"`, `"SIEMENS"`, `"GE_I"` (no parameters for +20 logic like in GUI),`"GE_MR"`, `"PHILIPS_REC"` or `"ANALYZE"`.
Parameter 2: FirstFileName: name of the first "raw" data file.
Parameter 3: NrOfVolumes: number of volumes for this document.
Parameter 4: nrOfVolumesToSkip: number of volumes that should be skipped at the beginning, so number of volumes in FMR document will be (NrOfVolumes - nrOfVolumesToSkip).
Parameter 5: createAMR: boolean (true or false): create AMR from first EPI volume or not.
Parameter 6: nrOfSlices: number of slices per volume.
Parameter 7: prefixSTCs: name for the stc file.
Parameter 8: swapBytes: swap bytes (true or false).
Parameter 9: resX - dimension of image along x-axis
Parameter 10: resY - dimension of image along y-axis
Parameter 11: nrBytes - number of bytes per pixel, usually 2.
Parameter 12: savingDir - directory for saving the FMR document.
Returns: Document

LinkAMR()

Function: LinkAMR()
Description: Link the provided AMR to the currently opened FMR.
Parameter 1: Name of the AMR file.

FMR document properties

TR: Repetition time in milliseconds. For example '2000'.
InterSliceTime: Time in milliseconds between acquisition of two adjacent slices. Example value: 80;
TimeResolutionVerified: Property to assert that the time resolution is correction. Values either true or false (boolean).
PixelSizeOfSliceDimX: Size of a pixel in millimeters in x-dimension. Example value: 3.5.
PixelSizeOfSliceDimY: Size of a pixel in millimeters in y-dimension. Example value: 3.5.
SliceThickness: Thickness of a slice in millimeters. For example '3'.
GapThickness: Space between slices, measured in millimeters. Example value: 0.99.
VoxelResolutionVerified: Property ensuring that the voxel resolution of the FMR document has been set properly. Value is either true or false (boolean).
HasSliceTimeTable: Indicates whether the data contain a time table for multiband data (MB-EPI). Use with CorrectSliceTimingUsingTimeTable().

3.5.3 Create DMR documents

CreateDocumentDMR()

Function: CreateDocumentDMR()
Parameter 1: Filetype (string): file type of original data. One of `"DICOM"`, `"PHILIPS_REC"` or `"ANALYZE"`.
Parameter 2: firstFile (string): the filename and path of the first file of the data.
Parameter 3: Number of directions (integer): the number of directions (=volumes).
Parameter 4: Number of directions to skip (integer): the number of directions (=volumes) to skip.
Parameter 5: Create AMR (boolean): should an AMR document be created: true or false. This is for visualization purposes.
Parameter 6: nrOfSlices (integer): the number of slices in a volume.
Parameter 7: Prefix (string): Name for DWI file.
Parameter 8: isLittleEndian (boolean): Is 'true' when the byte order is little endian; otherwise 'false'.
Parameter 9: nrOfBytes (integer): Number of bytes for each pixel. 1 byte is 8 bits. Example value: 2.
Parameter 10: xSize (integer): Size of image along x-axis. Example value: 128.
Parameter 11: ySize (integer): Size of image along y-axis. Example value: 128.
Parameter 12: Number of bytes per pixel (integer): Precision of intensity value. Usually 2 byte (16 bits).
Parameter 13: Saving directory (string): Name of path where DMR document should be saved.
Returns: Document

CreateDocumentMosaicDMR()

Function: CreateDocumentMosaicDMR()
Member of class: BrainVoyager.
Parameter 1: FileType (string): file type of original data. One of `"DICOM"`, `"SIEMENS"`, `"GE_I"` (no parameters for +20 logic like in GUI),`"GE_MR"`, `"PHILIPS_REC"` or `"ANALYZE"`.
Parameter 2: FirstFileName
Parameter 3: NrOfDirections (number of volumes)
Parameter 4: nrOfVolumesToSkip
Parameter 5: createAMR
Parameter 6: nrOfSlices
Parameter 7: prefixSTCs
Parameter 8: swapBytes
Parameter 9: mosaicResX - mosaic size: dimension of images in volume along x-axis
Parameter 10: mosaicResY - mosaic size: dimension of images in volume along x-axis
Parameter 11: nrBytes
Parameter 12: savingDir
Parameter 13: volsInImg - number of volumes per file
Parameter 14: resX - dimension of image along x-axis
Parameter 15: resY - dimension of image along y-axis
Returns: Document

3.5.4 Create VMR documents

CreateDocumentVMR()

Function: CreateDocumentVMR()
Creates an anatomical document in 8-bit (.vmr) and 16-bit (*.v16).*
Parameter 1: Filetype (string): file type of original data. One of `"DICOM"`, `"SIEMENS"`, `"GE_I"` (no parameters for +20 logic like in GUI),`"GE_MR"`, `"PHILIPS_REC"` or `"ANALYZE"`.
Parameter 2: firstFile (string): the filename and path of the first file of the data.
Parameter 3: nrOfSlices (integer): the number of slices in a volume.
Parameter 4: isLittleEndian (boolean): Is 'true' when the byte order is little endian; otherwise 'false'.
Parameter 5: xSize (integer): Size of image along x-axis. Example value: 256.
Parameter 6: ySize (integer): Size of image along y-axis. Example value: 256.
Parameter 7: nrOfBytes (integer): Number of bytes for each pixel. 1 byte is 8 bits. Most often used value: 2 bytes.
Returns: Document

LinkStimulationProtocol()

Function: LinkStimulationProtocol()
Description: Valid only if the document is a) of type FMR or b) of type VMR and if a VTC file has been linked. In the latter case, the specified stimulation protocol is linked to the VTC file. To establish a permanent link, save the FMR document or the VTC file.
Parameter 1: Name of the stimulation protocol file.

LinkVTC()

Function: LinkVTC()
Description: Link the provided VTC file to the currently opened VMR.
Parameter 1: Name of the normalized functional data file (*.vtc).

List of VMR properties

VMRVoxelResolutionX: Size of a pixel along x-dimension.
VMRVoxelResolutionY: Size of a pixel along y-dimension.
VMRVoxelResolutionZ: Size of a pixel along z-dimension.
ExtendedTALSpaceForVTCCreation: Necessary to set explicitly before creating a VTC in any space. When this property is true, it will create a VTC where the Talairach bounding box includes the cerebellum.
FileNameOfCurrentVTC Retrieve file name of attached functional (VTC) file.
UseBoundingBoxForVTCCreation: True if one would like to create a bounding box of a different size than the default, using the offset into the VMR. Set this property to true of false before starting to create any VTC. (Since 2.6: this property is default set to false, so it does not need to be set before creation of any VTC.)
TargetVTCBoundingBoxXStart: Start of the bounding box in VMR on x-axis.
TargetVTCBoundingBoxYStart: Start of the bounding box in VMR on y-axis.
TargetVTCBoundingBoxZStart: Start of the bounding box in VMR on z-axis.
TargetVTCBoundingBoxXEnd: End of the bounding box in VMR on x-axis.
TargetVTCBoundingBoxYEnd: End of the bounding box in VMR on y-axis.
TargetVTCBoundingBoxZEnd: End of the bounding box in VMR on z-axis.
MeshScene: Obtain mesh scene object. See MeshScene in surface section.

3.5.5 Create AMR documents

CreateDocumentAMR()

Function: CreateDocumentAMR()
Description: Creates an AMR document file. AMR documents consist of a set of two-dimensional anatomical scans used to overlay statistical maps in the original "slice space". If successful, an AMR document is returned. Use this object to access document methods. The name of the first file must contain the full path information. You may want to check proper reading of your data using the New Document Wizard or Create Document dialog before using this command in your scripts. *Parameter 1:* Filetype (string): file type of original data. One of `"DICOM"`, `"SIEMENS"`, `"GE_I"` (no parameters for +20 logic like in GUI),`"GE_MR"`, `"PHILIPS_REC"` or `"ANALYZE"`.
Parameter 2: firstFile (string): the filename and path of the first file of the data.
Parameter 3: nrOfSlices (integer): the number of slices in a volume.
Parameter 4: isLittleEndian (boolean): Is 'true' when the byte order is little endian; otherwise 'false'. Is usually 'true'.
Parameter 5: xSize (integer): Size of image along x-axis. Example value: 256.
Parameter 6: ySize (integer): Size of image along y-axis. Example value: 256.
Parameter 7: nrOfBytes (integer): Number of bytes for each pixel. 1 byte is 8 bits. Example value: 2.
Returns: Document

3.5.6 For all documents

Close()

Description: Close(): Close the current FMR/VMR/AMR/DMR document.

SaveAs()

Description: SaveAs(): Save the FMR/VMR/AMR/DMR document.
Parameter 1: Name for the document (string). The NIfTI file extension triggers exporting VMR anataomical and FMR functional documents as NIfTI files when using the "SaveAs" command of a "Document" object.

3.5.7 Example scripts

```
/*  CreateDocuments_BV21_NewGSGdata.js
Script for BrainVoyager QX 2.2
By Rainer Goebel 1995-2009
Modified for BVQX 2.1 (2009), for BrainVoyager 21 (07-2019) with new GSG data (08-2019).
*/

var ObjectsRawDataPath = BrainVoyager.BrowseDirectory("Please select the path to the BrainVoyager sample data");
var DTIDataPath = BrainVoyager.BrowseDirectory("Please select the path to the BrainVoyager DWI sample data");
//"/Users/BVuser/Data/Human31dir/";

BrainVoyager.PrintToLog("Create FMR document...");
Create_FMR_Document();
BrainVoyager.PrintToLog("Create mosaic FMR document...");
Create_Mosaic_FMR_Document();
BrainVoyager.PrintToLog("Create VMR document...");
Create_VMR_Document();
BrainVoyager.PrintToLog("Create DMR document...");
Create_DMR_Document();
BrainVoyager.PrintToLog("Create AMR document...");
Create_AMR_Document();


// for mosaic EPI data
function Create_Mosaic_FMR_Document()
{
    BrainVoyager.PrintToLog(ObjectsRawDataPath + "/02-03BlockedRun1/dcm/" +
     "JudEck_20181128_BI_Exercises_sess4-0002-0001-00001.dcm");
    var docFMR = BrainVoyager.CreateDocumentMosaicFMR("DICOM",
     ObjectsRawDataPath + "/02-03BlockedRun1/dcm/" + "JudEck_20181128_BI_Exercises_sess4-0002-0001-00001.dcm",
291, 2, true, 64, "untitled-", false, 800, 800, 2, ObjectsRawDataPath, 1, 100, 100);
    docFMR.SaveAs( ObjectsRawDataPath + "/02-03BlockedRun1/" + "sub-01_ses-04_task-blocked_run- 1_bold.fmr" );
    docFMR.Close();
}

// for non-mosaic EPI data
function Create_FMR_Document()
{
    var firstfilename = BrainVoyager.BrowseFile("Please select the first file", "*.dcm");
     // "JudEck_20181128_BI_Exercises_sess4-0010-0001-00001.dcm";
  var docFMR = BrainVoyager.CreateDocumentFMR( "DICOM", firstfilename, 1, 0, true, 36, "untitled", false, 64, 64,
2, ObjectsRawDataPath + "/10-11GrefieldMapping");
docFMR.SaveAs("NonMosaicDICOM_SCRIPT.fmr");
docFMR.Close();
}

// for anatomical data
function Create_VMR_Document()
{
    BrainVoyager.PrintToLog(ObjectsRawDataPath + "/08MPRAGE_1mm_Nondistorted/" +
     "JudEck_20181128_BI_Exercises_sess4-0008-0001-00001.dcm");
      var docVMR = BrainVoyager.CreateDocumentVMR( "DICOM",
       ObjectsRawDataPath + "/08MPRAGE_1mm_Nondistorted/" +
"JudEck_20181128_BI_Exercises_sess4-0008-0001-00001.dcm", 192, false, 256, 256, 2 );
      docVMR.SaveAs( ObjectsRawDataPath + "/08MPRAGE_1mm_Nondistorted/"
       + "sub-01_ses-04_acq-nondistorted_T1w.vmr" );
    //docVMR.SaveAs( ObjectsRawDataPath + "/" + "sub-01_ses-04_acq-nondistorted_T1w.nii" );
     // uncomment to save as NIfTI instead of VMR
docVMR.CorrectIntensityInhomogeneities(); // IIHC can only be performed directly after VMR creation
docVMR.Close();
}

// for DWI data
function Create_DMR_Document()
{
var docDMR = BrainVoyager.CreateDocumentDMR("DICOM", DTIDataPath + "/" + "pimpul_070907_dti -0007-0001-00001.dcm",
31, 0, true, 75, "human31dir", false, 128, 128, 2, DTIDataPath);
docDMR.SaveAs(DTIDataPath  + "/" + "HUMAN31DIR_SCRIPT.dmr");
docDMR.TR = 8900;
docDMR.InterSliceTime = 118;
docDMR.TimeResolutionVerified = true;
docDMR.Save();
docDMR.Close();
}

// for localizers, etc.
function Create_AMR_Document()
{
    var docAMR = BrainVoyager.CreateDocumentAMR( "DICOM", ObjectsRawDataPath  + "/01Loc/"
     + "JudEck_20181128_BI_Exercises_sess4-0001-0001-00001.dcm", 9, false, 512, 512, 2 );
    docAMR.SaveAs(ObjectsRawDataPath  + "/" + "CG_SLICELOCALIZER_SCRIPT.amr");
    docAMR.Close();
}

/* Document properties:
TR (property)
```

```
set via: docFMR.TR = 2000;
InterSliceTime (property)
NrOfVolumes (property)
PixelSizeOfSliceDimX (property)
PixelSizeOfSliceDimY (property)
InterSliceTime (property)
SliceThickness  (property)
GapThickness  (property)
VoxelResolutionVerified (property)
TimeResolutionVerified (property)
FileNameOfPreprocessdFMR (property)
obtain via: var newname = docFMR.FileNameOfPreprocessdFMR;
*/
```

3.6 BV document functions: Preprocessing of functional data (FMR)

3.6.1 List of methods

AdjustMeanIntensity()
CorrectSliceTiming()
CorrectSliceTimingWithSliceOrder()
CorrectSliceTimingUsingTimeTable()
CorrectMotion()
CorrectMotionEx()
CorrectMotionTargetVolumeInOtherRun()
CorrectMotionTargetVolumeInOtherRunEx()
Remove() TemporalHighPassFilterGLMFourier()
TemporalHighPassFilterGLMDCT()
TemporalHighPassFilter()
LinearTrendRemoval()
TemporalGaussianSmoothing()
SpatialGaussianSmoothing()

3.6.2 Detailed description of methods

AdjustMeanIntensity()

Function: AdjustMeanIntensity()
Description: Volume-based adjustment of mean intensity (MIA). During operation, the program plots two curves, one showing the measured mean intensity of volumes over time and the other showing the mean level of each volume after correction (a straight line). Furthermore, a zero-mean predictor of the global fluctuations is automatically stored to disk allowing to add it as a confound predictor in the design matrix. If the MIA preprocessed FMR is used for further processing (e.g., VTC creation), adding of the MIA confound predictor is not necessary. If one wants to perform the correction as part of the GLM, however, one should use the non-MIA FMR for further processing adding the MIA confound predictor to the design matrix.
Example: The following example applies mean intensity adjustment to a FMR document:

```
docFMR.AdjustMeanIntensity();
```

CorrectSliceTiming()

Function: CorrectSliceTiming()
Description: Most EPI sequences measure the slices of a functional volume in succession, but one often would like to treat the data of one volume as if it were acquired at the same time, particularly in the context of event-related studies. Using linear interpolation, the present method resamples the time series for the different slices in such a way that the resulting slice time courses can be treated as if they were obtained simultaneously. Valid only if document is of type FMR. Slice time correction can only be done in FMR documents because these documents contain the time series data separated with respect to the individually measured slices (STC files); this information is lost in VTC files after spatial transformation. Slices typically are scanned ascending (i.e., slice 1, slice 2, slice 3 ...) or interleaved (i.e., slice 1, slice 3, slice 5 .. slice 2, slice 4, slice 6 ...). While the TR value should be easily obtained from the scanner protocol or from a file header, the inter slice time might be more difficult to get. If scanning ran continously, i..e. if there is no pause between scanning the last slice of volume N and the first slice of volume $N + 1$, then you can simply divide the TR value by the number of slices per volume to get the inter slice time. If scanning ran not continously, you must either get the information about the duration of acquiring all slices or the duration of the pause between volumes. An alternative possibility (used by our group) is to read slice trigger pulses from the scanner measuring the time point when each slice is scanned; by subtracting, for example the time point of slice 1 from the time point of slice 2 results in the inter slice time. The resulting corrected data is automatically saved to disk. The names for the new FMR document and the new STC prefix is determined as in the GUI version, i.e., if the FMR document "`cg_objects.fmr`" is used, the resulting new file on disk will be "`cg_objects_SCCAI.fmr`". In addition, a set of new STC file, actuallly containing the time series data, is stored to disk.
Parameter 1: Scan order: 0: Ascending, 1: Ascending-interleaved, 2: Ascending-interleaved 2 (Siemens only), 10: Descending, 11: Descending-interleaved, 12: Descending-interleaved 2.
Parameter 2: Interpolation method: 0: trilinear, 1: cubic spline, 2: windowed SINC. For ascending interleaved slice order, this results in the following filenames: trilinear: `*_SCLAI.fmr`; cubic spline: `*_SCCAI.fmr`, windowed SINC: `*_SCSAI.fmr`.
Parameter 3: Multiband factor (new since $\pm$ BrainVoyager 22)
Returns: Success (boolean).

CorrectSliceTimingWithSliceOrder()

Function CorrectSliceTimingWithSliceOrder():
Description: In case default options for the scan order parameter do not apply, you can also use a free slice order (as a string param) to specify slice time correction (new in BVQX 2.3).
Parameter 1: Scan order with slice numbers specified in a text string.
Parameter 2: Interpolation method: 0: trilinear, 1: cubic spline, 2: windowed SINC. For ascending interleaved slice order, this results in the following filenames: trilinear: `*_SCLAI.fmr`; cubic spline: `*_SCCAI.fmr`, windowed SINC: `*_SCSAI.fmr`.
Parameter 3: Multiband factor (new since $\pm$ BrainVoyager 22)
Returns: Success (boolean).

Example: The following string specifies the same order as the "ascending interleaved" option using cubic spline interpolation and multiband 2:

```
var ok = FMR.CorrectSliceTimingWithSliceOrder("1 14 2 15 3 16 4 17 5 18 6 19 7 20 8 21 9 22 10 23 11 24 12 25 13", 1, 2);
```

CorrectSliceTimingUsingTimeTable()

Function CorrectSliceTimingUsingTimeTable():
Description: The slice-scan time correction for multi-band sequences that acquire 2 or more slices in a single shot. For Siemens Mosaic data files, slice timing information is now directly extracted from the DICOM header; since this information (time of acquisition for each slice with respect to the begin of a volume) can be used to correct slice timing differences for all 2D EPI sequences (e.g. with ascending, descending, interleaved slice order with or without multi-band with or without extra (silent) gaps between TRs), a new "slice time table" option is now used as default if the respective data is available. In order to be available for preprocessing, the extracted slice timing data (one value per slice) is permanently stored in created .FMR files. The availability of time table can be interrogated using the `'HasSliceTimeTable'` property)(function and property new in BVQX 2.8.2). The resulting file contains `*_SCCTBL*.fmr`
Parameter 1: Interpolation method (1 = cubic spline).
Example: The following example applies slice scan time correction using cubic spline interpolation, for multi-band and single-band data:

```
if (docFMR.HasSliceTimeTable)
docFMR.CorrectSliceTimingUsingTimeTable(1); // 1: cubic spline interpolation
else
docFMR.CorrectSliceTiming(2, 1, 2); // 2: ascending interleaved 2, 1: cubic spline interpolation, 2: multiband factor 2
```

CorrectMotion()

Function: CorrectMotion()
Description: Detects and corrects rigid-body motion within an FMR file. The target volume provided by the user serves as the reference to which all other volumes are aligned. The new file name is based on the name of the FMR file prior to starting the filter and adds an abbreviation describing the preprocessing performed. If, for example, the name of the FMR file was "cg_objects_SCCAI.fmr", the new name will be "cg_objects_SCCAI_3DMCS.fmr". The added infix "_3DMCS" describes that motion correction (MC) has been performed in 3D (3DMC), i.e. fitting 3 translation and 3 rotation parameters. All 3D preprocessing steps add such descriptive naming abbreviations which makes it easy to get the information about the sequence of steps which has been performed to produce a particular FMR file.
Interpolation: SINC estimation and resampling (BrainVoyager 20.2), resulting filename `*_3DMCS.fmr`; trilinear estimation and SINC resampling (BrainVoyager 20.4), resulting filename `*_3DMCTS.fmr`. *Parameter 1:* Target volume.
Returns: True or false (boolean).

CorrectMotionEx()

Function: CorrectMotionEx()
Description: Detects and corrects rigid-body motion within an FMR file. The target volume provided by the user serves as the reference to which all other volumes are aligned. In this version, the default settings (described in CorrectMotion()) can be modified.
The new file name is based on the name of the FMR file prior to starting the filter and adds a suing describing the preprocessing performed. If, for example, the name of the FMR file was "cg_objects_SCCAI.fmr", the new name will be "cg_objects_SCCAI_3DMC.fmr". The added infix "_3DMC" describes that motion correction (MC) has been performed in 3D (3DMC), i.e. fitting 3 translation and 3 rotation parameters. All 3D preprocessing steps add such descriptive naming abbreviations which makes it easy to get the information about the sequence of steps which has been performed to produce a particular FMR file.
Parameter 1: Target volume: number of the volume to which other volumes should be aligned.
Parameter 2: Interpolation method: 0 and 1: trilinear detection and trilinear interpolation, 2: trilinear detection and sinc interpolation or 3: sinc detection of motion and sinc interpolation.
Parameter 3: Use full data set: true if yes, false if one would like to use the reduced dataset.
Parameter 4: Maximum number of iterations: defines for how many iterations the parameters should be

fitted. Value in GUI is default '100'.
Parameter 5: Generate movies: true if yes, false if no. Creates an *.avi movie on Windows and *.mov on Mac OS X.
Parameter 6: Generate extended log file: true if one would like the motion estimation parameters in a text file, false otherwise.
Returns: True or false (boolean).

CorrectMotionTargetVolumeInOtherRun()

Function: CorrectMotionTargetVolumeInOtherRun()
Description: Detects and corrects rigid-body motion within several runs. This intra-session alignment method makes it possible to align all volumes of all runs in a session to the same targetvolume. This version uses the default settings as shown in the GUI version (FMR Data Preprocessing): trilinear interpolation to perform the rigid-body translation/rotation, a reduced data set (every second voxel in each dimension = one eighth of a full volume = 12.5%), a maximum of 100 iterations to fit a volume to the reference, creation of pre- and post movie files and a standard and extended log file. The new file name is based on the name of the FMR file prior to starting the filter and adds a suing describing the preprocessing performed. If, for example, the name of the FMR file was "cg_objects_SCCAI.fmr", the new name will be "cg_objects_SCCAI_3DMC.fmr". The added infix "_3DMC" describes that motion correction (MC) has been performed in 3D (3DMC), i.e. fitting 3 translation and 3 rotation parameters. All 3D preprocessing steps add such descriptive abbreviations to the name which makes it easy to get the information about the sequence of steps which has been performed to produce a particular FMR file. The intra-session alignment is integrated in the 3D motion correction step by specifying to which target volume and target run the data should be aligned. The target run should be the one, which is closest in time to the recorded 3D data set to minimize the effect of motion across scans. If a session, for example, started with a 3D data set followed by three runs, run 1, run 2 and run 3, 3D motion correction in the first run would proceed as in the CorrectMotion() method by selecting a target volume i.e. volume 1 (default). For run 2, the same target volume in run 1 is specified aligning th data of run 2 directly with run 1. The same is specified for run 3, i.e. the data are directly aligned to the target volume in run 1. This procedure ensures that all volumes in all runs are aligned to the very same target volume. Note that the described strategy works only if all runs have been recorded with the same nominal slice positions. If slice positions have been changed across runs, intra-session alignment can be achieved by using coregistration.
Parameter 1: Target FMR name: The name of the FMR document to which the current document should be aligned.
Parameter 2: Target volume number: The number of the volume within that run to which the document should be aligned.

CorrectMotionTargetVolumeInOtherRunEx()

Function: CorrectMotionTargetVolumeInOtherRunEx()
Description: Perform combined intra-session alignment and motion correction.
Parameter 1: Target FMR name: name of the run to which the current FMR document should be aligned.
Parameter 2: Target volume: number of the volume to which other volumes should be aligned.
Parameter 3: Interpolation method: 0 and 1: trilinear detection and trilinear interpolation, 2: trilinear detection and sinc interpolation or 3: sinc detection of motion and sinc interpolation.
Parameter 4: Use full data set: true if yes, false if one would like to use the reduced dataset (default in GUI).
Parameter 5: Maximum number of iterations: defines for how many iterations the parameters should be fitted. Value in GUI is default '100'.
Parameter 6: Generate movies: true if yes, false if no. This feature has been disabled for some time.
Parameter 7: Generate extended log file: true if one would like the motion estimation parameters in a text file, false otherwise.
Returns: True or false (boolean).

Remove()

Function: Remove()
Description: Deletes the currently open document from disk.

TemporalHighPassFilterGLMFourier()

Function: TemporalHighPassFilterGLMFourier()
Parameter 1: A parameter that specifies the number of cycles (pairs of two basis functions) used to build an appropriate design matrix.

TemporalHighPassFilterGLMDCT()

Function: TemporalHighPassFilterGLMDCT()
Parameter 1: A parameter that specifies the number of cycles (pairs of two basis functions) used to build an appropriate design matrix.

TemporalHighPassFilter()

Function: TemporalHighPassFilter()
Description: Apply a high-pass filter to the functional data ('classical approach', FFT).
Parameter 1: Cut-off value.
Parameter 2: Units: `"cycles"` or `"Hz"` (string).
Note: Includes linear trend removal.

LinearTrendRemoval()

Function: LinearTrendRemoval()
Description: Apply a linear high-pass filter to the functional data.
Note: Is not necessary when using `TemporalHighPassFilter()`.

TemporalGaussianSmoothing()

Function: TemporalGaussianSmoothing() *Description:* Since temporal gaussian smoothing blurs timing information across neighboring data points, it is not recommended as default. Temporal smoothing improves, however, the signal-to-noise ratio by removing high frequency fluctuations. The width of the kernel can now be specified in seconds. Note that the specification in seconds is only correct if the TR value has been specified correctly. Example value for kernel width: "2.8" seconds. If you want to specify the width of the kernel in units of data points (TR's), set the data points parameter instead of the secs parameter.
Parameter 1: Width of kernel.
Parameter 2: Units: `"s"` or `"TR"` (string).

SpatialGaussianSmoothing()

Function: SpatialGaussianSmoothing()
Description: Apply a spatial low-pass filter to the functional data.
Parameter 1: Width of kernel (FWHM).
Parameter 2: Units: `"mm"` or `"px"` (string).

3.6.3 Example script

Please note that the document (project) property `FileNameOfPreprocessdFMR` returns just the name of the preprocessed FMR before BrainVoyager QX 2.3, and the name including the path from BrainVoyager QX 2.3 onwards (2011).

```
// Example script for preprocessing a FMR document (single run)
// Feel free to adapt this script to your needs
//
//  Created by Rainer Goebel, last modified November 03 2005
//  Modified for QX 2.1, August 31 2009, February 3 2010, for BrainVoyager 21.4, Summer 2019 HB

//var ObjectsRawDataPath = "/Users/BVuser/Data/ObjectsDicomGSG/";
//var MocoISAPath = "/Users/BVuser/Data/testdata/moco_isa/";

Preprocess_FMR();
//MotionCorrectionISA();

function Preprocess_FMR()
{
    var ret = BrainVoyager.TimeOutMessageBox( "This script function will run standard FMR preprocessing steps."+
                                              "\n\nYou can cancel this script by pressing the 'ESCAPE' key.", 8);
    if (!ret) return;

    var newgsgdata = true;
    // Create a new FMR or open a previously created one.
    // Here we open the "sub-01_ses-04_task-blocked_run-1_bold.fmr" file
    var docname = BrainVoyager.BrowseFile("Please select the FMR file", "*.fmr");
    var docFMR = BrainVoyager.OpenDocument(docname); // the file path and name are saved in the variable "docname"
//    var docFMR = BrainVoyager.OpenDocument(ObjectsRawDataPath  + "sub-01_ses-04_task-blocked_run-1_bold.fmr");

    // Set spatial and temporal parameters relevant for preprocessing
    // You can skip this, if you have checked that these values are set when reading the data
    // To check whether these values have been set already (i.e. from header), use the "VoxelResolutionVerified"
    // and "TimeResolutionVerified" properties
    if( !docFMR.TimeResolutionVerified )
    {
docFMR.TR = 2000;
if (newgsgdata) {
docFMR.InterSliceTime = 31;
} else {
docFMR.InterSliceTime = 80;
}
docFMR.TimeResolutionVerified = true;
    }
    if( !docFMR.VoxelResolutionVerified )
    {
if (newgsgdata) {
docFMR.PixelSizeOfSliceDimX = 2;
docFMR.PixelSizeOfSliceDimY = 2;
docFMR.SliceThickness = 2;
docFMR.GapThickness = 0;
} else {
docFMR.PixelSizeOfSliceDimX = 3.5;
docFMR.PixelSizeOfSliceDimY = 3.5;
docFMR.SliceThickness = 3;
docFMR.GapThickness = 0.99;
}
docFMR.VoxelResolutionVerified = true;
    }

    // We also link the PRT file, if available (if no path is specified, the program looks in folder of document)
    docFMR.LinkStimulationProtocol("sub-01_ses-04_task-blocked_run-1_events.prt");

    // We save the new settings into the FMR file
    docFMR.Save();

    //
    // Preprocessing step 1: Slice time correction
    //
    ret = BrainVoyager.TimeOutMessageBox("Preprocessing step 1: Slice time correction.\n\n" +
    "To skip this step, press the 'ESCAPE' key.", 5);
    if(ret)
    {
        if (docFMR.HasSliceTimeTable)
            docFMR.CorrectSliceTimingUsingTimeTable(1); // 1: cubic spline interpolation
        else
            docFMR.CorrectSliceTiming(2, 1, 2);
            // 2: ascending interleaved 2, 1: cubic spline interpolation, 2: multiband factor
            // First param: Scan order 0  -> Ascending,  1  -> Asc-Interleaved,  2  -> Asc-Int2,
            // 10 -> Descending, 11 -> Desc-Int,  12 -> Desc-Int2
            // Second param: Interpolation method: 0 -> trilinear, 1 -> cubic spline, 2 -> sinc

    ResultFileName = docFMR.FileNameOfPreprocessdFMR;
    docFMR.Close();          // close input FMR
```

```
    docFMR = BrainVoyager.OpenDocument( ResultFileName );
 }
 //
 // Preprocessing step 2: 3D motion correction
 ret = BrainVoyager.TimeOutMessageBox("Preprocessing step 2: 3D motion correction.\n\n" +
  "To skip this step, press the 'ESCAPE' key.", 5);
 if(ret)
 {
  docFMR.CorrectMotion(1); // 1 is target volume. For more parameters, use CorrectMotionEx();
  ResultFileName = docFMR.FileNameOfPreprocessdFMR;
  docFMR.Close();            // close input FMR
  docFMR = BrainVoyager.OpenDocument( ResultFileName );
 }
 //
 // Preprocessing step 3: Spatial Gaussian Smoothing  (not recommended for individual analysis with a 64x64 matrix)
 /*ret = BrainVoyager.TimeOutMessageBox("Preprocessing step 3: Spatial gaussian smoothing.\n\n" +
  "To skip this step, press the 'ESCAPE' key.", 5);
 if(ret)  {
   docFMR.SpatialGaussianSmoothing( 4, "mm" );    // FWHM value and unit
   ResultFileName = docFMR.FileNameOfPreprocessdFMR;
   docFMR.Close();          // close input FMR
   docFMR = BrainVoyager.OpenDocument( ResultFileName );
 }*/

 // Preprocessing step 4: Temporal High Pass Filter, includes Linear Trend Removal
 ret = BrainVoyager.TimeOutMessageBox("Preprocessing step 4: Temporal high-pass filter.\n\n" +
  "To skip this step, press the 'ESCAPE' key.", 5);
 if(ret) {
   docFMR.TemporalHighPassFilterGLMFourier(2);
   ResultFileName = docFMR.FileNameOfPreprocessdFMR;
   docFMR.Close();           // close input FMR
   docFMR = BrainVoyager.OpenDocument( ResultFileName );
 }

 // Preprocessing step 5: Temporal Gaussian Smoothing  (not recommended for event-related data)
/*ret = BrainVoyager.TimeOutMessageBox("Preprocessing step 5: Temporal gaussian smoothing.\n\n" +
"To skip this step, press the 'ESCAPE' key.", 5);
 if (ret) {
  docFMR.TemporalGaussianSmoothing( 10, "s" );
  ResultFileName = docFMR.FileNameOfPreprocessdFMR;
  docFMR.Close();           // close input FMR
  docFMR = BrainVoyager.OpenDocument( ResultFileName );
 }*/

 // docFMR.Close() // you may want to close the final document, i..e to preprocess another run
}


//~~~~~~~~~~~ extra motion correction example ~~~~~~~~~~~

function MotionCorrectionISA()
{
    var docname = BrainVoyager.BrowseFile("Please select the FMR file", "*.fmr");
    var docFMR = BrainVoyager.OpenDocument(docname); // file path and name are saved in the variable "docname"
    var targetdocname = BrainVoyager.BrowseFile("Please select the target FMR file", "*.fmr");
    docFMR.CorrectMotionTargetVolumeInOtherRunEx(targetdocname, 1, 1, true, 100, false, true); // target FMR,
    // target volume, interpolation method (0 and 1: trilinear detection and trilinear interpolation,
    // 2: trilinear detection and sinc interpolation or 3: sinc detection of motion and sinc interpolation),
    // use full data set, maximum number of iterations, generate movies, extended log file
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

3.6.4 Example script 'HighPassFilterUsingGLM.js'

Example script for high-pass temporal filtering using GLM-Fourier basis set of a FMR document (single run).

```
// Example script for high-pass temporal filtering using GLM-Fourier basis set of a FMR document (single run).
// Feel free to adapt this script to your needs
//
//  Created by Rainer Goebel February 2012, updated June 2019

// you can now use and set the properties "PathToData" and "PathToSampleData" (
var ObjectsRawDataPath = BrainVoyager.BrowseDirectory("Please select the directory of the CG Objects experiment") + "/";
// BrainVoyager.PathToSampleData +  "ObjectsDicomGSG/";

function HighpassFilterWithGLM()
{
    // Create a new FMR or open a previously created one.
    // Here we open the "CG_OBJECTS_SCRIPT.fmr" file created previously via script
    var docFMR = BrainVoyager.OpenDocument(ObjectsRawDataPath  + "CG_OBJECTS_SCRIPT.fmr");
    if (docFMR == undefined)
        return;

    docFMR.TemporalHighPassFilterGLMFourier( 2 );
    // you can also use GLM with Discrete Cosine Transform (DCT) basis functions: docFMR.TemporalHighPassFilterGLMDCT(2);
    ResultFileName = docFMR.FileNameOfPreprocessdFMR;
    docFMR.Close();           // docFMR.Remove();  // close or remove input FMR
    docFMR = BrainVoyager.OpenDocument( ResultFileName );
}

HighpassFilterWithGLM();
```

3.7 BV document functions: Preprocessing of functional data (VTC)

3.7.1 List of methods

3.7.2 Detailed description of methods

SpatialGaussianSmoothing()

Function: SpatialGaussianSmoothing() (VTC)
Description: Spatial low-pass filter for VTC file; removes spatial high-frequency elements in VTC file, like sharp edges.
Parameter 1: FWHM value (number)
Parameter 2: FWHM unit: “mm” or “vx”

LinearTrendRemoval()

Function: LinearTrendRemoval() (VTC)
Description: Removes temporal linear trends in VTC file. A linear trend is estimated by fitting a line through the data estimating its slope and intercept.

TemporalHighPassFilter()

Function: TemporalHighPassFilter() (VTC)
Description: Removes low-frequency noise in VTC file, for example physiological noise.
Parameter 1: High-pass filter value
Parameter 2: High-pass filter unit: “cycles” or “Hz”

TemporalGaussianSmoothing()

Function: TemporalGaussianSmoothing() (VTC)
Description: Removes high-frequency noise in VTC file, like sharp peaks in the timecourse.
Parameter 1: FWHM value (number)
Parameter 2: FWHM unit: “d” or “dps” (data points) or “s” or “secs” (seconds)

3.7.3 Example script

```
// This simple script shows how VTC files linked ot a VMR document can be preprocessed
// This is especially helpful if VTCs are created with no (or modest, i.e. 4mm) spatial smoothing
// but when spatial smoothing (e.g. with FWHM of 8-10mm) is desired for group studies.
// In this case one could smooth the original FMR and create a second VTC file. It is, however,
// much more efficient to smooth the VTC file directly with an appropriate kernel as shown here
// While spatial smoothing is probably the most useful scenario of VTC smoothing, the
// code below shows how to call all available preprocessing options.
//
// To prepare this script, load a VMR and link a VTC - or add the appropriate script commands

var docVMR = BrainVoyager.ActiveDocument;

BrainVoyager.PrintToLog("Current VTC file: " + docVMR.FileNameOfCurrentVTC);

// now smooth VTC with a large kernel of 10 mm:
//
docVMR.SpatialGaussianSmoothing( 10, "mm" );    // FWHM value and unit ("mm" or "vx")

BrainVoyager.PrintToLog("Name of spatially smoothed VTC file: " + docVMR.FileNameOfCurrentVTC);

// now we could do a linear trend removal (see code in comments)
// since high-pass temporal filter (see below) includes LTR, we skip this here
//docVMR.LinearTrendRemoval();    // FWHM value and unit ("mm" or "vx")
//BrainVoyager.PrintToLog("Name of VTC file without linear trends: " + docVMR.FileNameOfCurrentVTC);

// now perform temporal high-pass filter
//
docVMR.TemporalHighPassFilter(3, "cycles");    // HP alue and unit ("cycles" or "Hz")
BrainVoyagerQX.PrintToLog("Name of VTC file without linear trends: " + docVMR.FileNameOfCurrentVTC);

// now perform Gaussian temporal smoothing
//
docVMR.TemporalGaussianSmoothing(3, "dps"); // FWHM value and unit: "d" or "dps" (data points) or "s" or "secs" (seconds)
BrainVoyager.MessageBox("Name of temporally smoothed VTC file: " + docVMR.FileNameOfCurrentVTC);

// Note that all intermediate VTC files are kept on disk. In order to remove no longer needed files, use
// the file access script routines (see "UsingCustomFiles.js" script)
```

3.8 BV document (mesh) functions: Preprocessing of functional data (MTC)

3.8.1 List of methods

SpatialSmoothing()
LinearTrendRemoval()
TemporalHighPassFilterFFT()
TemporalGaussianSmoothing()

3.8.2 Detailed description of methods

SpatialSmoothing()

Function: SpatialSmoothing()
Description: Spatial low-pass filter for MTC file; removes spatial high-frequency elements in MTC file, like sharp edges.
Parameter 1: FWHM value (number)

LinearTrendRemoval()

Function: LinearTrendRemoval()
Description: Apply a linear high-pass filter to the functional data.

TemporalHighPassFilterFFT()

Function: TemporalHighPassFilterFFT()
Description: Apply a high-pass filter to the functional data ('classical approach', FFT).
Parameter 1: Cut-off value.

TemporalGaussianSmoothing()

Function: TemporalGaussianSmoothing()
Description: Since temporal gaussian smoothing blurs timing information across neighboring data points, it is not recommended as default. Temporal smoothing improves, however, the signal-to-noise ratio by removing high frequency fluctuations. The width of the kernel can now be specified in seconds. Note that the specification in seconds is only correct if the TR value has been specified correctly. Example value for kernel width: "2.8" seconds. If you want to specify the width of the kernel in units of data points (TR's), set the data points parameter instead of the secs parameter.
Parameter 1: Width of kernel.
Parameter 2: Units: `"Seconds"`

3.8.3 Example script to preprocess an MTC

```
//
// Example script showing how to apply preprocessing with the new "mesh" object
// Feel free to adapt this script to your needs
//
//  Created by Rainer Goebel, May 2013; updated for BV 21+ June 2019
//

//var ObjectsRawDataPath = BrainVoyager.BrowseDirectory("Please select the directory with example data") +"/";
// BrainVoyager.PathToSampleData + "ObjectsDicomGSG/";
var mtcname = BrainVoyager.BrowseFile("Please select the functional surface file (MTC)", "*.mtc");
BrainVoyager.PrintToLog("MTC name: " + mtcname);

// when calling this function, we assume a VMR and the desired mesh
// have been loaded already (see also other "Mesh..." scripts)
function PreprocessMTC(mtcname)
{
    docVMR = BrainVoyager.ActiveDocument;
    if(docVMR == undefined) return;

    var meshscene = docVMR.CreateMeshScene();
    var mesh = meshscene.CurrentMesh;
    if (mesh == undefined) {
BrainVoyager.PrintToLog("Could not access mesh");
    return;
    } else {
BrainVoyager.PrintToLog("Mesh name: " + mesh.FileName);
  }

    var ok = mesh.LinkMTC(mtcname);
    if (!ok) {
BrainVoyager.PrintToLog("Could not link MTC");
return;
    }
    //ok = mesh.LinkMTC(ObjectsRawDataPath +
      "CG_Objects_Float_SCCAI_3DMCT_THPGLMF2c_TAL_NGF_LH.mtc" ); if(!ok) return;

    mesh.SpatialSmoothing(3);
    mesh.LinearTrendRemoval();
    mesh.TemporalHighPassFilterFFT(3);
    mesh.TemporalGaussianSmoothing(5, "Seconds");

    var PreprocessedMTCFileName = mesh.FileNameOfPreprocessdMTC;
    BrainVoyager.PrintToLog("Preprocessed .MTC: " + PreprocessedMTCFileName);
    mesh.SaveMTC(PreprocessedMTCFileName);
}

PreprocessMTC(mtcname);
```

3.9 BV document functions: Experimental design

3.9.1 List of methods for stimulation protocols

ClearStimulationProtocol()
LinkStimulationProtocol()
AddCondition()
SetConditionColor()
AddInterval()
SaveStimulationProtocol()
SaveVTC()

3.9.2 List of properties for stimulation protocols

StimulationProtocolFile: String
StimulationProtocolExperimentName: String
StimulationProtocolResolution: Number (1=volumes, 2=msec)
StimulationProtocolBackgroundColorR: Number
StimulationProtocolBackgroundColorG: Number
StimulationProtocolBackgroundColorB: Number
StimulationProtocolTimeCourseColorR: Number
StimulationProtocolTimeCourseColorG: Number
StimulationProtocolTimeCourseColorB: Number
StimulationProtocolTextColorR: Number
StimulationProtocolTextColorG: Number
StimulationProtocolTextColorB: Number
StimulationProtocolTimeCourseThickness: Number

3.9.3 Detailed description of methods

ClearStimulationProtocol()

Function: ClearStimulationProtocol()
Description: Function to start a new stimulation protocol.

LinkStimulationProtocol()

Function: LinkStimulationProtocol()
Description: Link the stimulation protocol with the provided name to the currently opened VMR file.
Parameter 1: Name protocol: name of the stimulation protocol (*.prt).

AddCondition()

Function: AddCondition() *Description:* Add a condition for the current stimulation protocol.
Parameter 1: Name for the condition (string).

SetConditionColor()

Function: SetConditionColor()
Description: To discriminate the different conditions, different colors can be used. The colors in BrainVoyager are specified as a combination of red, green and blue components, in this order. To lowest value for each component is 0 and the highest value is 255. When one of the components is set to 255 and the other two to 0, a primary color is obtained. When each of the components red, green and blue is set to 0, the result will be black in absence of all colors. Example colors are displayed in figure 3.1.
Parameter 1: Name of condition (string)
Parameter 2: Red color component (0-255)
Parameter 3: Green color component (0-255)
Parameter 4: Blue color component (0-255)

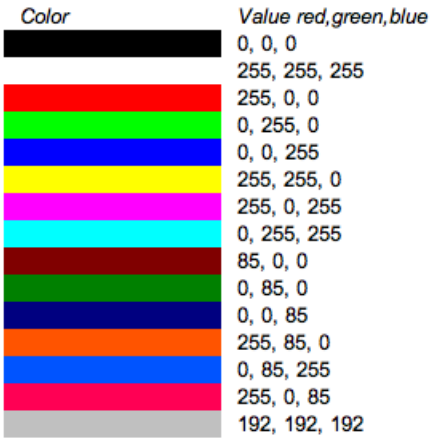


Figure 3.1: Example colors for protocol conditions

AddInterval()

Function: AddInterval()
Description: Add an interval for a condition.
Example: `fmr.AddInterval("Images in RVF", 1, 2);.`
Parameter 1: Name of condition (string)
Parameter 2: Start of interval in milliseconds or volumes
Parameter 3: End of interval in milliseconds or volumes

SaveStimulationProtocol()

Function: SaveStimulationProtocol()
Description: Save the newly created stimulation protocol with the provided name.
Parameter 1: Name for the protocol file.

SaveVTC()

Function: SaveVTC()
Description: Save the VTC, which will also save the name of the stimulation protocol in a VTC. New in BVQX 2.4.1.
Parameter 1: Name for current VTC. When using an empty string (””), the current VTC file name will be used.

3.9.4 Example scripts

Creating a stimulation protocol

To use the code, select the text below and save with the JavaScript extension ".js".

```
var FMR = BrainVoyager.ActiveDocument;
FMR.ClearStimulationProtocol();
FMR.StimulationProtocolExperimentName = "Faces Houses in LVF, CVF, RVF";
FMR.StimulationProtocolResolution = 1;
FMR.AddCondition("Faces_LVF");
FMR.AddCondition("Houses_RVF");
FMR.AddCondition("Faces_CVF");
FMR.AddCondition("Houses_LVF");
FMR.AddCondition("Faces_RVF");
FMR.AddCondition("Houses_CVF");

FMR.AddInterval("Faces_LVF", 4, 11);
FMR.AddInterval("Houses_RVF", 20, 27);
FMR.AddInterval("Faces_CVF", 36, 43);
FMR.AddInterval("Houses_LVF", 52, 59);
FMR.AddInterval("Faces_RVF", 68, 75);
FMR.AddInterval("Houses_CVF", 84, 91);
FMR.AddInterval("Faces_LVF", 100, 107);
FMR.AddInterval("Houses_RVF", 116, 123);
FMR.AddInterval("Faces_CVF", 132, 139);
FMR.AddInterval("Houses_LVF", 148, 155);
FMR.AddInterval("Faces_RVF", 164, 171);
FMR.AddInterval("Houses_CVF", 180, 187);
FMR.AddInterval("Faces_LVF", 196, 203);
FMR.AddInterval("Houses_RVF", 212, 219);
FMR.AddInterval("Faces_CVF", 228, 235);
FMR.AddInterval("Houses_LVF", 244, 251);
FMR.AddInterval("Faces_RVF", 260, 267);
FMR.AddInterval("Houses_CVF", 276, 283);

FMR.SetConditionColor("Faces_LVF", 200, 43, 43);
FMR.SetConditionColor("Houses_RVF", 200, 200, 43);
FMR.SetConditionColor("Faces_CVF", 43, 200, 43);
FMR.SetConditionColor("Houses_LVF", 43, 200, 200);
FMR.SetConditionColor("Faces_RVF", 43, 43, 200);
FMR.SetConditionColor("Houses_CVF", 200, 43, 200);

FMR.StimulationProtocolBackgroundColorR = 0;
FMR.StimulationProtocolBackgroundColorG = 0;
FMR.StimulationProtocolBackgroundColorB = 0;
FMR.StimulationProtocolTimeCourseColorR = 255;
FMR.StimulationProtocolTimeCourseColorG = 255;
FMR.StimulationProtocolTimeCourseColorB = 255;
FMR.StimulationProtocolTimeCourseThickness = 2;
FMR.SaveStimulationProtocol(SavingPath + "/s01_sess4_blocked_vol.prt");
FMR.Save(); // to save link to protocol permanently
```

Save VTC (including protocol name)

```
var docVMR = BrainVoyager.ActiveDocument;

BrainVoyager.PrintToLog("Current VTC file: " + docVMR.FileNameOfCurrentVTC);  // show name of current VTC

docVMR.LinkStimulationProtocol( "CG_Objects.prt" );
docVMR.SaveVTC("test_script.vtc"); // when using an empty string (""), the current VTC file name will be used
```

3.9.5 List of Methods for design matrices

ClearDesignMatrix()
AddPredictor()
SetPredictorValues()
SetPredictorValuesFromCondition()
ApplyHemodynamicResponseFunctionToPredictor()
ScalePredictorValues()
SaveSingleStudyGLMDesignMatrix()
ClearMultiStudyGLMDefinition()
AddStudyAndDesignMatrix()
SaveMultiStudyGLMDefinitionFile()

3.9.6 List of Properties for design matrices

FirstConfoundPredictorOfSDM: Integer. Provides possibility to indicate when, after columns of predictors of interest, the confound predictor columns start.
SDMContainsConstantPredictor: Boolean: true or false.

3.9.7 Detailed description of methods

ClearDesignMatrix()

Function: ClearDesignMatrix()
Description: Removes any current design matrices.

AddPredictor()

Function: AddPredictor()
Description: Add new regressor to design matrix.
Parameter 1: Name of regressor (string)

SetPredictorValues()

Function: SetPredictorValues()
Description: Specify the predictor value from a certain time point to the specified end time point in a specific condition.
Parameter 1: Name of condition (string)
Parameter 2: Timepoint from (integer)
Parameter 3: Timepoint to (integer)
Parameter 4: Value for the predictor

SetPredictorValuesFromCondition()

Function: SetPredictorValuesFromCondition()
Description: Set the predictor values for the provided condition using the information from the stimulation protocol.
Parameter 1: Name of predictor in design matrix (string)
Parameter 2: Name of condition in stimulation protocol (string)
Parameter 3: Maximum value for predictor. Default: '1.0'

ApplyHemodynamicResponseFunctionToPredictor()

Function: ApplyHemodynamicResponseFunctionToPredictor()
Description: Apply the hemodynamic response function (HRF) to the provided predictor (Boynton). To use the 2-gamma HRF, BrainVoyager QX plugins can be used (see the 'Design Matrix Access Functions' topic in the 'Plugins' chapter of the BrainVoyager User's Guide).
Parameter 1: Name of the condition that should be convolved with the HRF.

ScalePredictorValues()

Function: ScalePredictorValues()
Description: Scale the values of the provided predictor.
Parameter 1: Name of the condition that should be scaled.
Parameter 2: Maximum value for scale, for example 1.0.
Parameter 3: Boolean (true or false).

SaveSingleStudyGLMDesignMatrix()

Function: SaveSingleStudyGLMDesignMatrix()
Description: Save the single study design matrix as *.rtc or *.sdm file. *Parameter 1:* Name for the design matrix file.

ClearMultiStudyGLMDefinition()

Function: ClearMultiStudyGLMDefinition()
Description: Remove any present *.mdm file.

AddStudyAndDesignMatrix()

Function: AddStudyAndDesignMatrix()
Description: Add a combination of functional data (*.vtc) and design matrix (*.sdm).
Parameter 1: Name of the functional run (*.vtc)
Parameter 2: Name of the design matrix (*.sdm)

SaveMultiStudyGLMDefinitionFile()

Function: SaveMultiStudyGLMDefinitionFile()
Description: Save the newly created multi-study design matrix (*.mdm).
Parameter 1: Name for the multi-study design matrix file.

3.9.8 Some elaboration on design matrix properties

The scriptable design matrix properties display the behavior specified below; some situations that can arise are described far below.
SDMContainsConstantPredictor is by default false (situation 1). To create a constant predictor, not only the property 'SDMContainsConstantPredictor' needs to be set to true, also the predictor needs to be added and given a value (situation 2).
If any confound is defined and the variable 'FirstConfoundPredictorOfSDM' is set, BrainVoyager will assume that this is the constant (situation 3) and set the field 'FirstConfoundPredictor' in the *.sdm file to 1. So this would mean that it would not be possible to define any other confound without also defining a constant. Any constant predictor should be manually defined in the script, and the SDMContainsConstantPredictor should then be set to 'true' (situation 4).
FirstConfoundPredictor should be set after the predictor is actually defined, otherwise it will be set to 1 in the *.sdm file (situation 5).
Possible situations concerning the specification of confound predictors including constants in BrainVoyager via scripting:

1. If nothing is specified, the 'SDMContainsConstantPredictor' will be false. The field 'IncludesConstant' will then be 0, there will be no column with a constant, and the field FirstConfoundPredictor in the *.sdm file will point at the first column after the predictors of interest (which will not be present).

2. If the field 'SDMContainsConstantPredictor' is set to true in the script, but no actual constant predictor is provided, the 'IncludesConstant' will still be 0 and there will be no column with a constant.

3. If a confound predictor is manually specified, the FirstConfoundPredictorOfSDM is manually set to that column in the script, but the variable SDMContainsConstantPredictor is not set in the script, BrainVoyager will assume that this first confound predictor is the constant, and the field IncludesConstant in the *.sdm file will be set to 1 by BrainVoyager.

4. If neither the variable 'SDMContainsConstantPredictor' nor the variable 'FirstConfoundPredictorOfSDM' is set, the field 'FirstConfoundPredictor' in the *.sdm file will automatically point to the column after the last predictor, even if this last predictor is a manually specified constant.

5. If the variable 'FirstConfoundPredictorOfSDM' is set to a column - say column 4- before any predictors are specified, the field 'FirstConfoundPredictor' in the *.sdm file will point to column 1, even if later in the script three predictors of interest and one confound predictor would be specified.

3.9.9 Example scripts

To use the code, select the text below and save with the JavaScript extension ".js".

```
/*  ExperimentalDesign_BV21_newGSGdata.js
Script for BrainVoyager QX 2.1, adapted for BrainVoyager 21+ (June 2019) and new GSG data (24-08-19)
*/
/* Declarations */
var vmrname = BrainVoyager.BrowseFile("Please select the VMR file", "*.vmr");
// for all functions (*.sdm and *.mdm creation)
var ObjectsRawDataPath = BrainVoyager.BrowseDirectory("Please select the directory of the data") + "/";
var vtcname = BrainVoyager.BrowseFile("Please select the VTC file", "*.vtc"); // for single subject (*.prt and *.sdm)
var RFXDataPath = BrainVoyager.BrowseDirectory("Please select the directory of the random effects data") + "/";
//Users/BVuser/Data/Exercise_RFX_ANOVA/";

CreateDesignMatrix();
CreateMultiStudyDesignMatrix();

function CreateDesignMatrix()
{
BrainVoyager.PrintToLog("Create single subject design matrix...");
var doc = BrainVoyager.OpenDocument( vmrname ) ;
//ObjectsRawDataPath + "sub-01_ses-04_acq-nondistorted_T1w.vmr" );
doc.LinkVTC( vtcname );
doc.LinkStimulationProtocol(ObjectsRawDataPath + "s01_sess4_blocked_vol.prt");
doc.ClearDesignMatrix();
// doc.AutoAddConstantPredictor();

doc.AddPredictor("Faces_LVF");
doc.SetPredictorValuesFromCondition("Faces_LVF", "Faces_LVF", 1.0);
doc.ApplyHemodynamicResponseFunctionToPredictor("Faces_LVF");

doc.AddPredictor("Houses_RVF");
doc.SetPredictorValuesFromCondition("Houses_RVF", "Houses_RVF", 1.0);
doc.ApplyHemodynamicResponseFunctionToPredictor("Houses_RVF");

doc.AddPredictor("Faces_CVF");
doc.SetPredictorValuesFromCondition("Faces_CVF", "Faces_CVF", 1.0);
doc.ApplyHemodynamicResponseFunctionToPredictor("Faces_CVF");

doc.AddPredictor("Houses_LVF");
doc.SetPredictorValuesFromCondition("Houses_LVF", "Houses_LVF", 1.0);
doc.ApplyHemodynamicResponseFunctionToPredictor("Houses_LVF");

doc.AddPredictor("Faces_RVF");
doc.SetPredictorValuesFromCondition("Faces_RVF", "Faces_RVF", 1.0);
doc.ApplyHemodynamicResponseFunctionToPredictor("Faces_RVF");

doc.AddPredictor("Houses_CVF");
doc.SetPredictorValuesFromCondition("Houses_CVF", "Houses_CVF", 1.0);
doc.ApplyHemodynamicResponseFunctionToPredictor("Houses_CVF");

// You can also set any value at any time point (interval), here we define a linear trend predictor
/* doc.AddPredictor("Linear Trend");
for(i = 1; i<= doc.NrOfVolumes; i++)
{
value = 0.1*i;
doc.SetPredictorValues("Linear Trend", i, i, value);
}
doc.ScalePredictorValues("Linear Trend", 1.0, false);
*/
//doc.AutoAddConstantPredictor();
BrainVoyager.CurrentDirectory = ObjectsRawDataPath; //"/Users/BVuser/Data/ObjectsDicomGSG/");
doc.SaveSingleStudyGLMDesignMatrix("FacesHousesDesignMatrix.sdm");
}

function CreateMultiStudyDesignMatrix()
{
BrainVoyager.PrintToLog("Create multi subject design matrix...");
var doc = BrainVoyager.OpenDocument(RFXDataPath + "Average_Tal.vmr");
doc.ClearMultiStudyGLMDefinition();
var subjName;
var nrOfSubjects = 17;
for (i=0; i<nrOfSubjects; i++) {
subjName = RFXDataPath + "Sub" + (i+1) + "_run1_SCSAI2_3DMCTS_LTR_THPFFT3c_TAL.vtc"; // create subject name
doc.AddStudyAndDesignMatrix(subjName, RFXDataPath + "run1_SCSAI2_3DMCTS_LTR_THPFFT3c_TAL.sdm");
// now use subject name and always same protocol
}
doc.SaveMultiStudyGLMDefinitionFile(RFXDataPath + "MultiStudy_FROMSCRIPT.mdm");
}
}
```

3.10 BV document functions: Statistics

3.10.1 List of Methods

LoadSingleStudyGLMDesignMatrix()
LoadMultiStudyGLMDefinitionFile()
ComputeSingleStudyGLM()
ComputeMultiStudyGLM()
ComputeRFXGLM()
LoadGLM()
ShowGLM()
SaveGLM()
ClearContrasts()
SetCurrentContrast()
SetCurrentContrastAtIndex()
AddContrast()
SetContrastValue()
SetContrastString()
SetContrastValueAtIndex()
LoadVolumeMaps()
ShowVolumeMap()

3.10.2 List of Properties

CorrectForSerialCorrelations: Integer. If set to ”1”, AR(1) is used, if set to“2”, AR(2) is used.
SeparationOfSubjectPredictors: Boolean (true or false). Create one beta value for condition in each subject (concatenated runs).
SeparationOfStudyPredictors: Boolean (true or false). Create one beta value for condition in each run.
ZTransformStudies: Boolean (true or false). Use z-transform for data.
PSCTransformStudies: Boolean (true or false). Use percent-signal-change (PSC) transformation for data.
ZTransformStudiesBaselineOnly: Boolean (true or false).

3.10.3 Detailed description of methods

Computing the general linear model (GLM)

LoadSingleStudyGLMDesignMatrix()

Function: LoadSingleStudyGLMDesignMatrix()
Description: Load a design matrix file (*.rtc, *.sdm).
Parameter 1: Name of the design matrix file (string).

LoadMultiStudyGLMDefinitionFile()

Function: LoadMultiStudyGLMDefinitionFile()
Description: Load a design matrix file (*.rtc, *.sdm).
Parameter 1: Name of the design matrix file (string).

ComputeSingleStudyGLM()

Function: ComputeSingleStudyGLM()
Description: Compute a fixed-effects general linear model for a single run.

ComputeMultiStudyGLM()

Function: ComputeMultiStudyGLM()
Description: Compute a fixed-effects general linear model for a group of studies.

ComputeRFXGLM()

Function: ComputeRFXGLM()
Description: Compute a random-effects general linear model.

LoadGLM()

Function: LoadGLM()
Description: Load a general linear model file (*.glm) from harddisk.
Parameter 1: Name of the *.glm file.

ShowGLM()

Function: ShowGLM()
Description: Show the GLM that just has been computed.

SaveGLM()

Function: SaveGLM()
Description: Save the GLM that just has been computed.
Parameter 1: Name for the GLM (with *.glm extension).

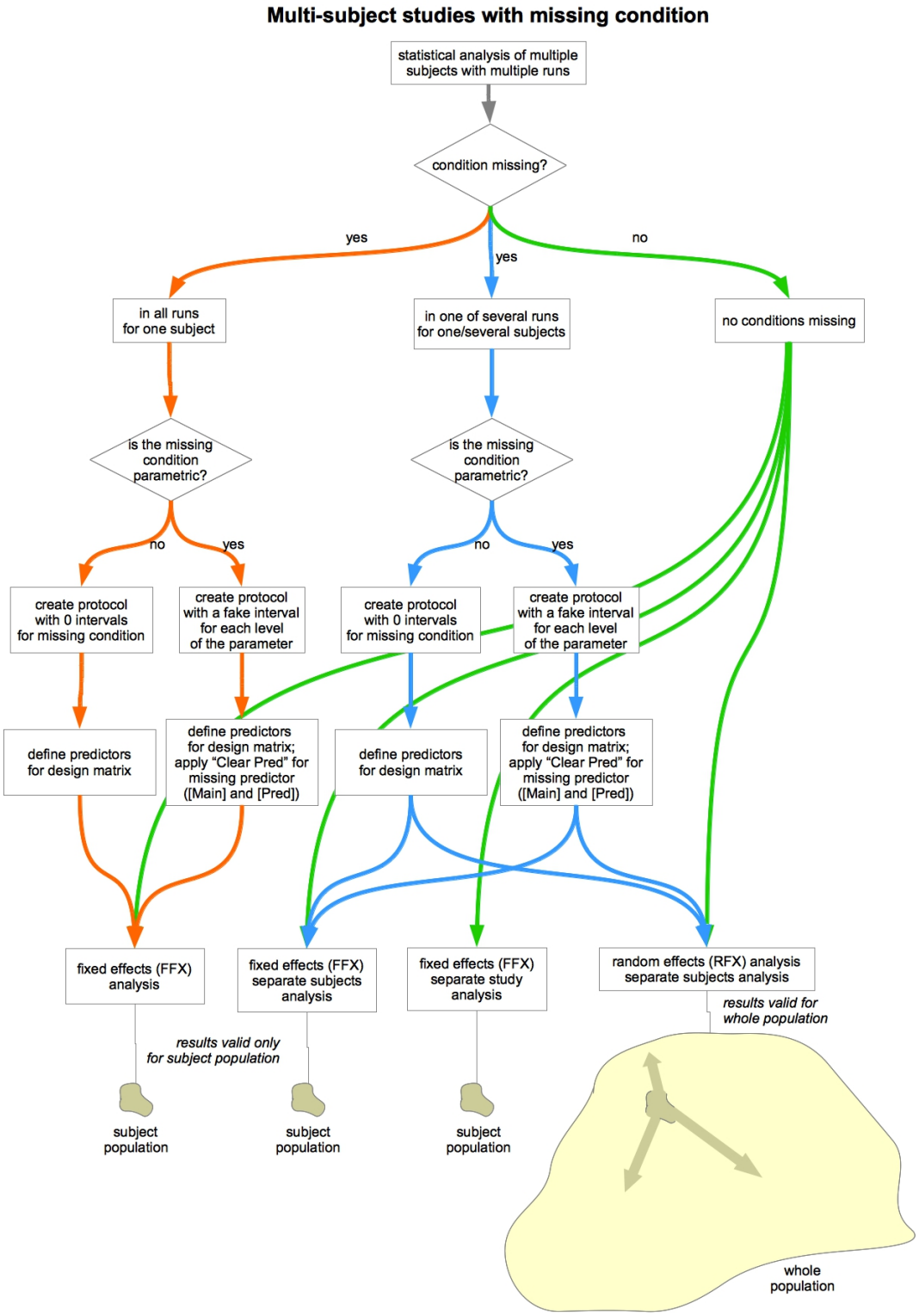


Figure 3.2: Different types of GLMs (part I)

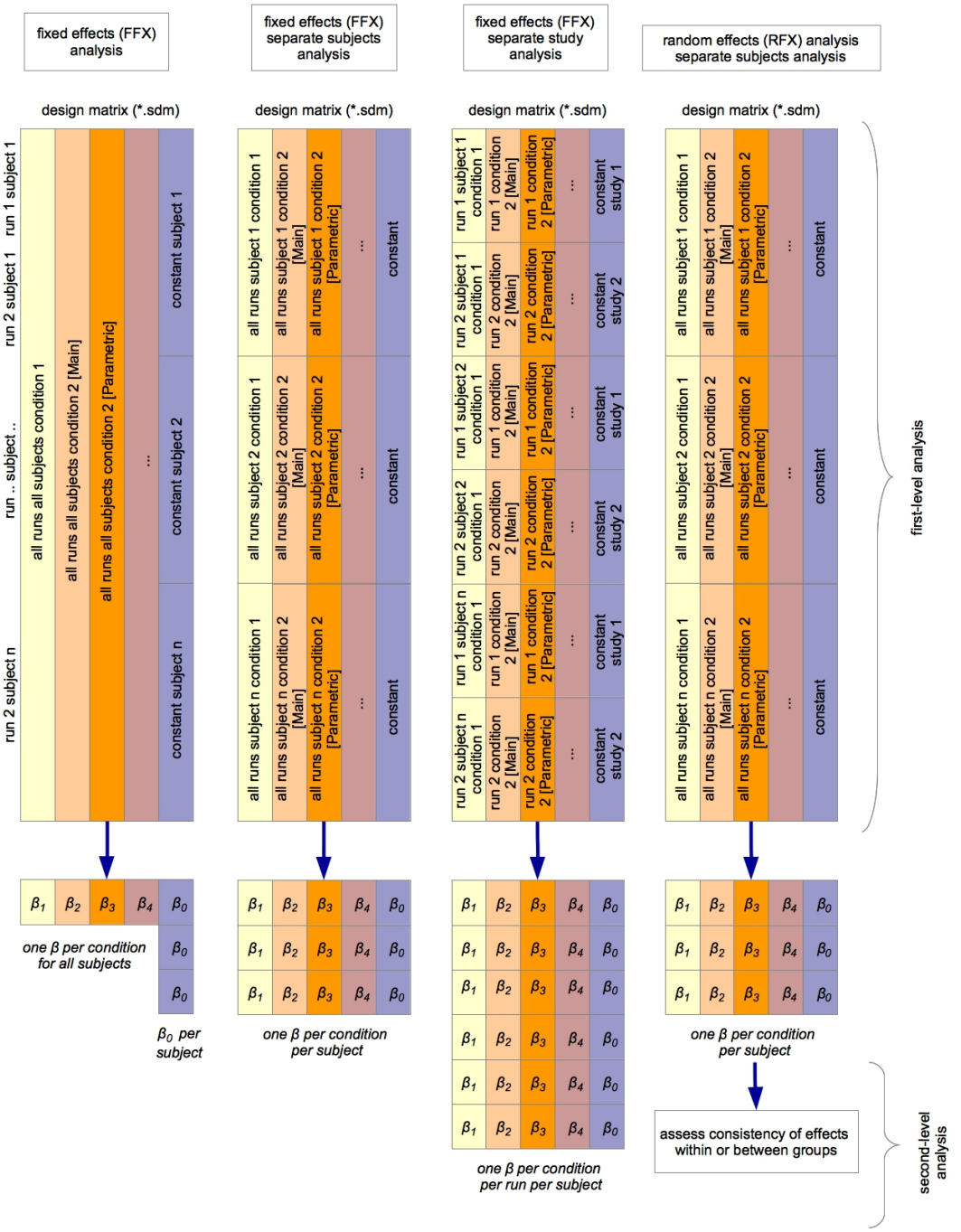


Figure 3.3: Different types of GLMs (part II)

Setting contrasts

A hypothesis can be tested using contrasts. There are several functions to set contrasts in BrainVoyager (see functions below and figure 3.4).

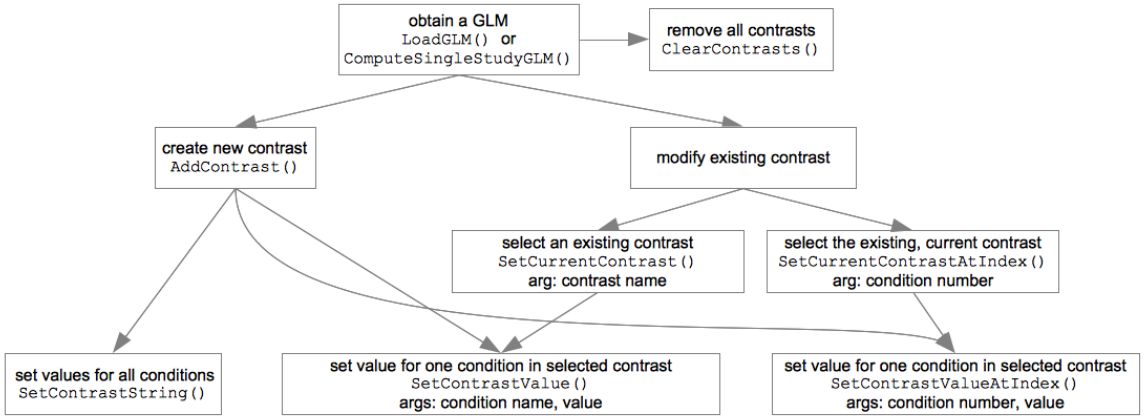


Figure 3.4: Setting contrasts

ClearContrasts()

Function: ClearContrasts()
Description: Remove any current contrasts.

AddContrast()

Function: AddContrast()
Description: Add a contrast.
Parameter 1: Name for the contrast, for example `"LVF > RVF"`.

SetContrastString()

Function: SetContrastString()
Description: Set the contrast by entering a sequence of parameters (one number for each condition) in a string. Only works if `AddContrast()` has been invoked first.
Parameter 1: parameters in string, for example: ”1 -1 0 0”

SetCurrentContrastAtIndex()

Function: SetCurrentContrastAtIndex()
Description: The effect of this function is to set the internal “contrast pointer” to one of the existing contrasts. To have an effect, the command has to be followed by “SetContrastValueAtIndex()” which set the value of the respective vector element of that contrast that is chosen by the “SetCurrentContrastAtIndex” command.
Parameter 1: Index of predictor, starts counting at 1.

SetContrastValueAtIndex()

Function: SetContrastValueAtIndex()
Description: Set the contrast by entering a parameter for one condition. Only works if `AddContrast()` has been invoked first.

Parameter 1: Index of predictor, starts counting at 1.
Parameter 2: Value for predictor, for example -1.

SetCurrentContrast()

Function: SetCurrentContrast()
Description: The effect of this function is to set the internal "contrast pointer" to one of the existing contrasts. To have an effect, the command has to be followed by "setContrastValue()" which set the value of the respective vector element of that contrast that is chosen by the "SetCurrentContrast" command.
Parameter 1: Name of the contrast.

SetContrastValue()

Function: SetContrastValue()
Description: Set a contrast value for a condition.
Parameter 1: Name of the condition.
Parameter 2: Value (integer), for example $+1$ or -1.

Volume maps (*.vmp)

LoadVolumeMaps()

Function: LoadVolumeMaps()
Description: Load a volume map (*.vmp) on an anatomical file (*.vmr).
Parameter 1: Name of the map.
Returns: True or false (boolean).

ShowVolumeMap()

Function: ShowVolumeMap()
Description: Show one of the loaded maps.
Parameter 1: The index of the map to show (starting at 1).

3.10.4 Example scripts

To use the code, select the text below and save with the JavaScript extension ".js"; then, load in the Script Editor and click"Run". Or copy-paste the text directly in the Script Editor.

```
/*  Statistics.js
Script for BrainVoyager QX 2.1, adapted for BrainVoyager 21 (11-04-19)
*/
/* Declarations */
var ObjectsRawDataPath = "/Users/BVuser/Data/ObjectsDicomGSG/";
var RFXDataPath = "/Users/BVuser/Data/Exercise_RFX_ANOVA/";

/* This code is executed when clicking the 'Run' button */
try {

RunSingleSubjectGLM(); // invoke function below
OverlayContrasts();
RunMultiSubjectGLM();
RunRandomEffectsGLM();

} catch (error) {
BrainVoyager.PrintToLog("Error: " + error);
}

/* These functions can be invoked */
function RunSingleSubjectGLM()
{
    try {
        var ObjectsRawDataPath = "/Users/BVuser/Data/ObjectsDicomGSG/";
        var doc = BrainVoyager.OpenDocument(ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_TAL.vmr");
        doc.LinkVTC(ObjectsRawDataPath + "CG_OBJECTS_SCRIPT_TAL.vtc");
        doc.ClearDesignMatrix();
        doc.LoadSingleStudyGLMDesignMatrix(ObjectsRawDataPath + "CG_OBJECTS_FROMSCRIPT.sdm");
        doc.LinkStimulationProtocol(ObjectsRawDataPath + "CG_OBJECTS_FROMSCRIPT.prt");
        doc.CorrectForSerialCorrelations = 2; // If set to 1, AR(1) is used, if set to2, AR(2) is used.
        doc.ComputeSingleStudyGLM();
        doc.ShowGLM();
        doc.SaveGLM(ObjectsRawDataPath + "CG_OBJECTS_FROMSCRIPT.glm");
} catch (error) {
BrainVoyager.PrintToLog(error);
}
}

function RunMultiSubjectGLM()
{
var doc = BrainVoyager.OpenDocument(RFXDataPath + "Average_Tal.vmr");
doc.CorrectForSerialCorrelations = 2; // If set to 1, AR(1) is used, if set to2, AR(2) is used.
doc.SeparationOfSubjectPredictors = true;
doc.ZTransformStudies = true;
doc.LoadMultiStudyGLMDefinitionFile(RFXDataPath + "MultiStudy_FROMSCRIPT.mdm");
doc.ComputeMultiStudyGLM();
doc.ShowGLM();
doc.SaveGLM(RFXDataPath + "MultiStudy_FROMSCRIPT.glm");
}

function RunRandomEffectsGLM()
{
var doc = BrainVoyager.OpenDocument(RFXDataPath + "Average_Tal.vmr");
doc.LoadMultiStudyGLMDefinitionFile(RFXDataPath + "MultiStudy_FROMSCRIPT.mdm");
doc.PSCTransformStudies = true;
doc.ComputeRFXGLM();
doc.SaveGLM(RFXDataPath + "RFXGLM_FROMSCRIPT.glm");
}

function OverlayContrasts()
{
var doc = BrainVoyager.OpenDocument(ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_CLEAN_TAL.vmr");
doc.LoadGLM("CG_OBJECTS_FROMSCRIPT.glm");
doc.ClearContrasts();
doc.AddContrast("LVF > RVF");
doc.SetContrastValue("LVF", +1);
doc.SetContrastValue("RVF", -1);
// alternatively: doc.SetContrastString("1 -1 0 0");
// or ("Index" starts with 1)
// doc.SetContrastValueAtIndex(1, +1);
//  doc.SetContrastValueAtIndex(2, -1);

doc.ShowGLM();
}
```

A script to load a volume map:

```
var vmrname = BrainVoyager.BrowseFile("Please select the anatomy", "*.vmr");
```

```
var vmr = BrainVoyager.OpenDocument(vmrname);
var vmpname = BrainVoyager.BrowseFile("Please select the map", "*.vmp");
var success = vmr.LoadVolumeMaps(vmpname);
vmr.ShowVolumeMap(1);
```

For a script to load a surface map, see page 103.

3.11 BV document functions: Statistics on regions-of-interest (ROI/VOI)

3.11.1 List of Methods

LoadVOIFile()
GetNameOfVOI()
HideAllVOIs()
SelectVOI()
ShowSelectedVOIs()
PrepareROIContrasts()
AddROIContrast()
ComputeSingleStudyGLMForVOI()
GetNameOfROIContrast()
GetTValueOfROIContrast()
GetPValueOfROIContrast()
GetBetaNameOfROIGLM()
GetBetaValueOfROIGLM()

3.11.2 List of Properties

NrOfVOIs: Returns the number of volumes-of-interest (VOI).
NrOfPredictorsInSingleStudyDM: Returns the number of predictors in a linked single study design matrix (*.sdm/*.rtc).
NrOfTimePointsInSingleStudyDM: Returns the number of time points (volumes) in a linked single study design matrix (*.sdm/*.rtc).
NrOfROIContrasts: Returns the number of VOI contrasts for the VOI.

3.11.3 Detailed description of methods

LoadVOIFile()

Function: LoadVOIFile()
Description: Load *.voi file to an anatomical file (*.vmr).
Parameter 1: Name of the *.voi file, including the path.

HideAllVOIs()

Function: HideAllVOIs()
Description: Do not display any activation from VOIs on anatomical data (*.vmr).

SelectVOI()

Function: SelectVOI()
Description: Select the volume-of-interest (*.voi) that will be displayed on the anatomical data (*.vmr).
Parameter 1: The number of the volume-of-interest. Please note that the VOI-index is 0-based, so the first volume-of-interest can be selected as `docVMR.SelectVOI(0);`.

GetNameOfVOI()

Function: GetNameOfVOI()
Description: Returns the name of the linked VOI.
Parameter 1: Index of the VOI (start counting at 0).

ShowSelectedVOIs()

Function: ShowSelectedVOIs()
Description: All volumes-of-interest that have been selected via `SelectVOI()` are now being displayed on the anatomical data (*.vmr).

PrepareROIContrasts()

Function: PrepareROIContrasts()
Description: Prepare contrasts for the regions-of-interest.
Parameter 1: Number of predictors (obtain number of predictors via property, see `NrOfPredictorsInSingleStudyDM` in section 3.11.2.

AddROIContrast()

Function: AddROIContrast()
Description: Add a contrast for the region-of-interest.
Parameter 1: Name for the contrast.
Parameter 2: Values for the contrast (within quotes, as string).

ComputeSingleStudyGLMForVOI()

Function: ComputeSingleStudyGLMForVOI()
Description: Run a single study GLM for a region-of-interest. First load a single study design matrix file (*.sdm), functional data (*.vtc) and a regions-of-interest file (*.voi) (see example script below).
Parameter 1: VOI index (number, 0-based).
Parameter 2: Time course normalization (number); 0 = none, 1 = percent signal change, 2 = z-transformation, 3 = z-transformation in baseline periods.
Parameter 3: Serial correlation correction (number); 0 = none, 1 = with AR(1) model, 2 = with AR(2) model.

GetNameOfROIContrast()

Function: GetNameOfROIContrast()
Description: Get the name of a specific region-of-interest contrast.
Parameter 1: Number of region-of-interest (ROI) contrast; first ROI has number 0.

GetTValueOfROIContrast()

Function: GetTValueOfROIContrast()
Description: VOI-GLM result, expressed as t-value.
Parameter 1: Number of region-of-interest (ROI) contrast; first ROI has number 0.

GetPValueOfROIContrast()

Function: GetPValueOfROIContrast()
Description: VOI-GLM result, expressed as p-value.
Parameter 1: Number of region-of-interest (ROI) contrast; first ROI has number 0.

3.11.4 Example script

To use the code, select the text below and save with the JavaScript extension ".js"; then, load in the Script Editor and click"Run". Or copy-paste the text directly in the Script Editor.

```
// NewCommands_v284.js
// New script commands in v2.8.4 support VOI processing including running single-study GLM analyses
// Rainer Goebel 2014, adapted for BV21 June 2019

BrainVoyager.ShowLogTab();

var vmrname = BrainVoyager.BrowseFile("Please select the anatomical file", "*.vmr");
// BrainVoyager.ActiveDocument;
var docVMR = BrainVoyager.OpenDocument(vmrname);
//var curFolder = BrainVoyager.CurrentDirectory;
var VOIFile = BrainVoyager.BrowseFile("Please select the VOI file", "*.voi");
//BrainVoyager.PathToSampleData + "ObjectsDicomGSG/Visual.voi";
BrainVoyager.PrintToLog("VOI file: " + VOIFile);
var ok = docVMR.LoadVOIFile(VOIFile);
if(!ok) {
BrainVoyager.PrintToLog("Could not open .VOI file");
return;
}

var n_vois = docVMR.NrOfVOIs;
BrainVoyager.PrintToLog("No. of VOIs: " + n_vois);
for(i=0; i<n_vois; i++)
{
var voi_name = docVMR.GetNameOfVOI(i);
BrainVoyager.PrintToLog("Name of VOI " + (i+1) + ": " + voi_name);
}

docVMR.HideAllVOIs();
// select all vois for display
for(i=0; i<n_vois; i++)
docVMR.SelectVOI(i); // VOI index is 0-based!

docVMR.ShowSelectedVOIs();
var vtcname = BrainVoyager.BrowseFile("Please select the design matrix file", "*.vtc");
docVMR.LinkVTC(vtcname);
var sdmname = BrainVoyager.BrowseFile("Please select the design matrix file", "*.sdm");
// BrainVoyager.PathToSampleData + "ObjectsDicomGSG/SingleStudy.sdm"
docVMR.LoadSingleStudyGLMDesignMatrix(sdmname);
var n_preds = docVMR.NrOfPredictorsInSingleStudyDM;
var n_points = docVMR.NrOfTimePointsInSingleStudyDM;

docVMR.PrepareROIContrasts(n_preds); // param: number of predictors
docVMR.AddROIContrast("Left vs Baseline", "1 0 0 0");
docVMR.AddROIContrast("Right vs Baseline", "0 1 0 0");
docVMR.AddROIContrast("Left vs Right", "1 -1 0 0");

// param 1: VOI index (0-based)
// param 2: time course normalization, 0 -> none, 1 -> percent change, 2 -> z, 3 -> z in baseline periods
// param 3: serial correlation correction, 0 -> none, 1 -> with AR(1) model, 2 -> with AR(2) model
docVMR.ComputeSingleStudyGLMForVOI(0, 1, 2);

// Main stats for contrasts are accessible (full GLM/contrast results are printed in Log)
for(i=0; i<docVMR.NrOfROIContrasts; i++) {
var c_name = docVMR.GetNameOfROIContrast(i);
var c_t = docVMR.GetTValueOfROIContrast(i);
var c_p = docVMR.GetPValueOfROIContrast(i);

BrainVoyager.PrintToLog("VOI-GLM result for contrast " + (i+1) + " (" + c_name + "): t = " + c_t + "  p = " + c_p);
}

// The GLM betas can also be retrieved
for(i=0; i<n_preds; i++) {
var b_name = docVMR.GetBetaNameOfROIGLM(i);
var b_val = docVMR.GetBetaValueOfROIGLM(i);
BrainVoyager.PrintToLog("VOI-GLM beta " + (i+1) + " (" + b_name + "): " + b_val);
}
```

3.12 BV document functions: Transformations and Normalization

3.12.1 List of methods

AutoTransformToIsoVoxel()
TransformToIsoVoxel()
AutoTransformToSAG()
SetVoxelIntensity(x, y, z, intensity)
GetVoxelIntensity(x, y, z)
CorrectIntensityInhomogeneities()
CoregisterFMRToVMRUsingBBR()
CoregisterFMRToVMR()
AutoACPCAndTALTransformation()
CreateVTCInVMRSpace()
CreateVTCInACPCSpace()
CreateVTCInTALSpace()
CreateVTCInMNISpace()
CreateVDWInVMRSpace()
CreateVDWinACPCSpace()
CreateVDWInTALSpace()
NormalizeToMNISpace()

3.12.2 Detailed description of methods

Transforming VMR files

AutoTransformToIsoVoxel()

Function: AutoTransformToIsoVoxel()
Description: Transforms a non-isovoxel VMR file to isovoxel (same voxel sizes in x-dimension, y-dimension and z-dimension, in this case 1x1x1mm). The result is saved to disk with the new name.
Parameter 1: Interpolation method (integer): 1 - trilinear, 2 - cubic spline (recommended), 3 - sinc
Parameter 2: New VMR name (string): name for transformed VMR.
Returns: boolean: success.

TransformToIsoVoxel()

*Function:*TransformToIsoVoxel()
Description: Transforms a non-isovoxel VMR file to an image with same resolution in all dimensions with variable framing cube dimension. The result is saved to disk with the new name.
Parameter 1: Interpolation method (integer): 1 - trilinear, 2 - cubic spline (recommended), 3 - sinc
Parameter 2: New VMR name (string): name for transformed VMR.
Parameter 3: Target resolution: float (for all three dimensions)
Parameter 4: Framing cube dimension: one of the dimensions: 256, 384, 512, 768 (for all three dimensions) *Returns:* boolean: success.

AutoTransformToSAG()

Function: AutoTransformToSAG()
Description: Transforms an isovoxel VMR file to sagittal orientation. The result is saved to disk with the new name. *Note:* does not work if the voxel sizes of the VMR are not equal, therefore isovoxelation might need to be applied on beforehand.
Parameter 1: newname (string): name for transformed VMR.
Returns: boolean

SetVoxelIntensity(x, y, z, intensity)

Function: SetVoxelIntensity(x, y, z, intensity)
Description: Give the voxel at position (x,y,z) a new intensity value. This values between 0 and 225 are gray scale; the values between 225 and 255 are color values (see figure 3.5).
Parameter 1: x: position of voxel on x-axis.
Parameter 2: y: position of voxel on y-axis.
Parameter 3: z: position of voxel on z-axis.
Parameter 4: intensity: new intensity value.

GetVoxelIntensity(x, y, z)

GetVoxelIntensity(x, y, z)
Description: Obtain the intensity value at position (x,y,z).
Returns: Intensity value: integer between 0 and 255.

CorrectIntensityInhomogeneities()

CorrectIntensityInhomogeneities()
Description: Perform non-uniformity (B_1) correction with default parameters. This will save a corrected, brain-peeled anatomical image with `*_IIHC` in the name to disk. Please note that this only works directly after running `CreateDocumentVMR()` or when a *.v16 file with the same name is available in the same folder as the *.vmr.

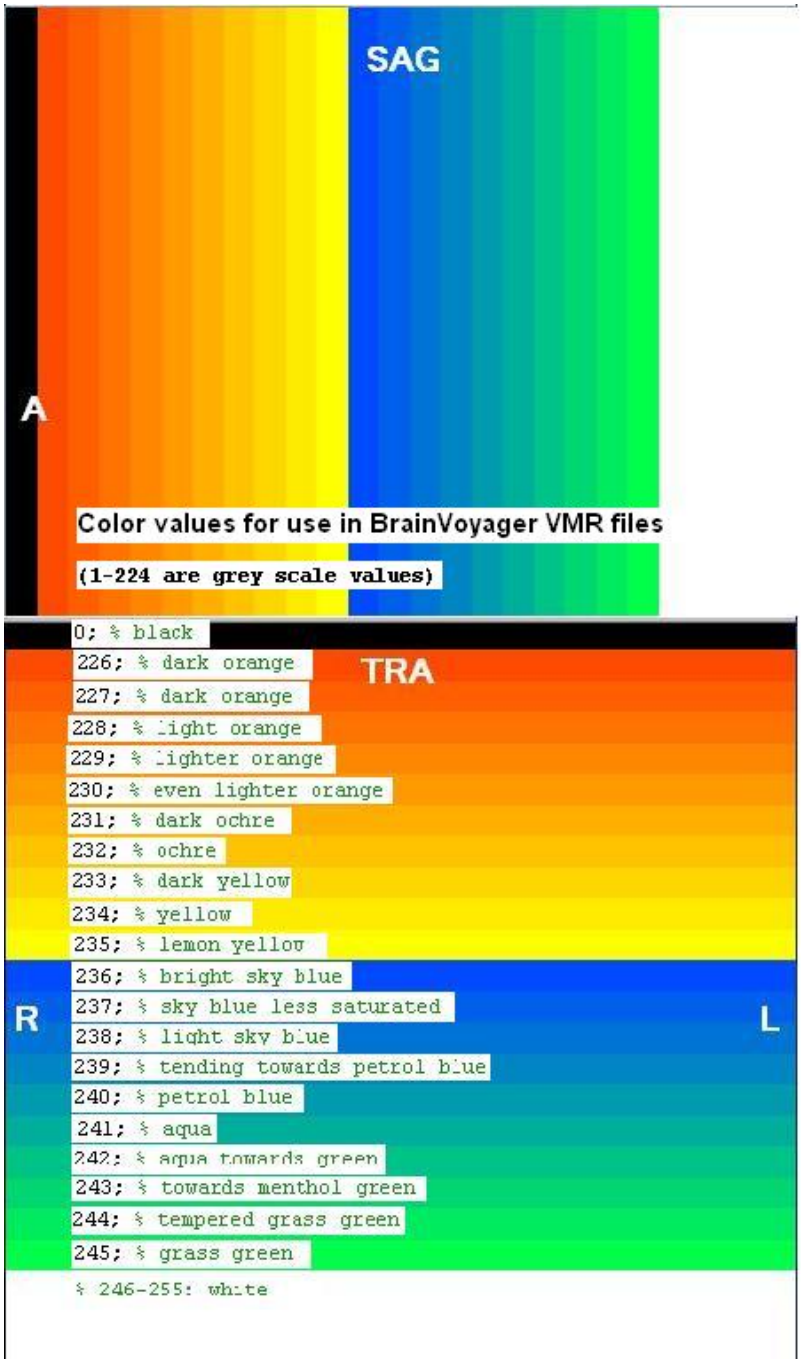


Figure 3.5: Intensity values for an anatomical file (*.vmr)

AutoACPCAndTALTransformation()

Function: AutoACPCAndTALTransformation()
Description: Transform the VMR object to AC-PC (`*_aACPC.vmr`) and Talairach space. The resulting files ((`*_aACPC.vmr`, `*_TAL.vmr`, `*_aACPC.tal`) will be saved to disk. The input *.vmr needs to be $1 \times 1 \times 1mm$ (if necessary, AutoTransformToIsoVoxel() can be applied first).

CoregisterFMRToVMR()

Function: CoregisterFMRToVMR()
Description: Calculates a robust alignment of the specified FMR with the anatomical data of the calling VMR document object using alignment information from the headers of (DICOM) images as well as an iterative intensity gradient-based matching procedure. The boolean return value indicates whether the command could be executed successfully. The command selects a representative volume of the FMR data (see useAttachedAMR parameter), which is aligned to the volume data of the hosting VMR to calculate two transformation matrices, a headerbased initial alignment (IA) TRF file and a matching-based finetuning adjustment (FA) TRF file. The two generated files can be used subsequently to actually transform all volumes of the FMR-STC data in the space of the hosting VMR; the calculation of coregistration files is separated from the actual application since it is usually desired to transform the FMR-STC data into a normalized space beyond the space of the hosting VMR (see VTC creation commands below).
Parameter 1: FMRFileName (str): String specifying the name of the input FMR file, which will be coregistered to the hosting VMR. The string should contain the full path file name. The representative functional volume of the FMR that will be used to calculate the IA and FA coregistration matrices is determined by the useAttachedAMR parameter.
Parameter 2: useAttachedAMR (int): Boolean specifying whether the original FMR volume should be used (value: 0) or whether the volume in an attached AMR should be used as input (value: 1, recommended). If requested but no AMR is available, the first FMR volume will be used instead. The linked AMR is usually only a resampled (higher resolution) version of the first volume of the original functional data (thus also called a "pseudo" AMR). Using the linked (pseudo) AMR is recommended since this volume will not be modified by FMR preprocessing operations such as spatial smoothing (it will also be properly modified in case of across-run motion correction). In case that the linked AMR is a true T1 weighted anatomical coplanar set of slices from an extra scan, a value of 2 should be used for this parameter, which will not invert voxel intensities, which is otherwise performed to match the functional data volume to the anatomical VMR data volume.
Returns: Boolean: true (success) or false.

CoregisterFMRToVMRUsingBBR()

Function: CoregisterFMRToVMRUsingBBR()
Description: Calculates a robust alignment of the specified FMR with the anatomical data of the calling VMR document object using alignment information from the headers of (DICOM) images as well as an iterative boundary-based registration (BBR) matching procedure. The boolean return value indicates whether the command could be executed successfully. This BBR-based version of the FMR-VMR coregistration command takes more calculation time than the CoregisterFMRToVMR() command (see above) since several intermediate calculations have to be performed, including a whilte/grey matter segmentation, reconstruction and smoothing of a white/grey matter mesh; if the mesh for the VMR document is already available (from a previous run), the segmentation and mesh processing steps are skipped and only the BBR algorithm is performed. The command uses the first volume of the FMR data, which is aligned to the volume data of the hosting VMR to calculate two transformation matrices, a header-based initial alignment (IA) TRF file and a BBR based fine-tuning adjustment (`BBR_FA`) TRF file. The two generated files can be used subsequently to actually transform all volumes of the FMR-STC data in the space of the hosting VMR; the calculation of coregistration files is separated from the actuall application since it is usually desired to transform the FMR-STC data into a normalized space beyond the space of the hosting VMR (see VTC creation commands below).
Parameter 1: FMRFileName (str) String specifying the name of the input FMR file, which will be coregistered to the hosting VMR. The string should contain the full path file name.
Returns: Boolean: true (success) or false.

The BBR registration script function produces the following files:

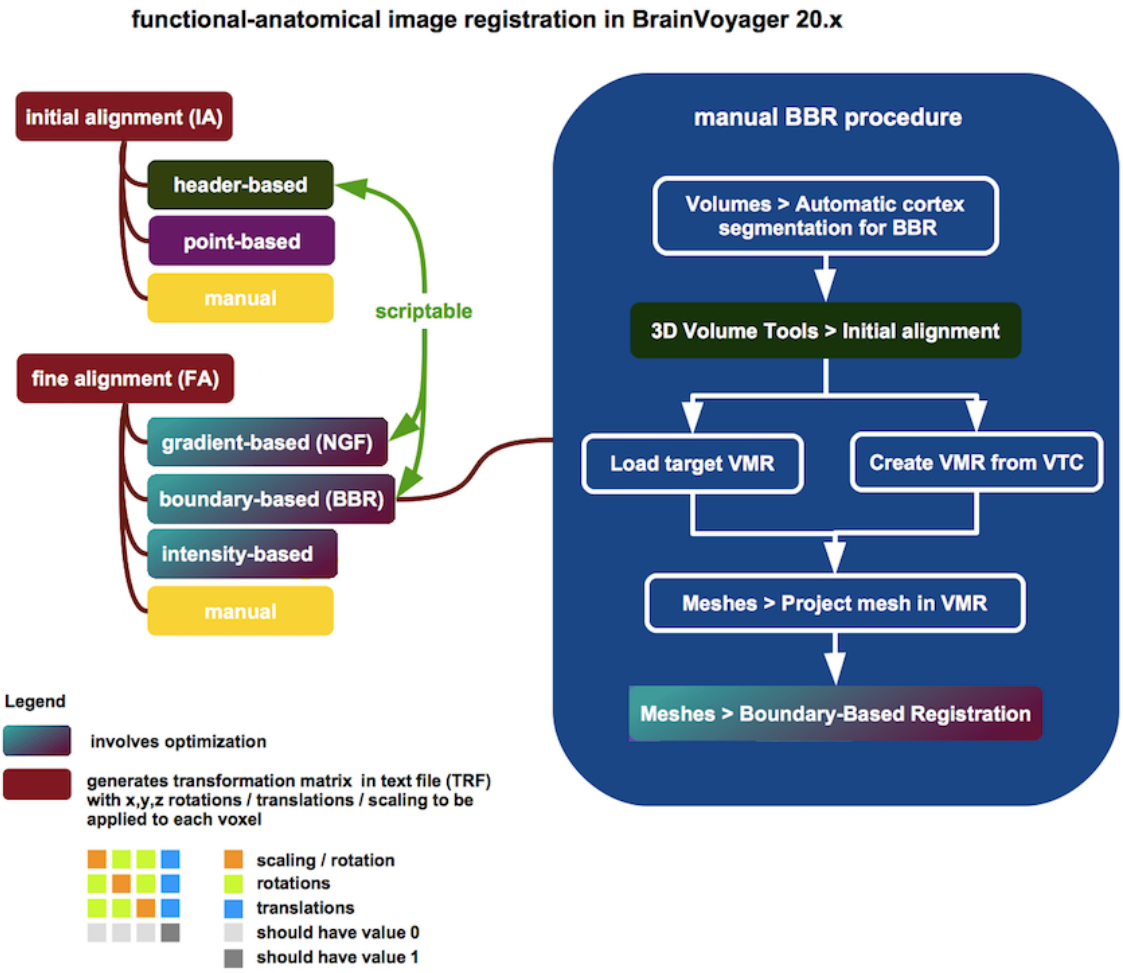
SRF (3x): `ETC-7x-R5_WM_RECO`, `ETC-7x-R5_WM_RECOSM`, `ETC-7x-R5_WM_RECOSM_D300k`

TRF (3x): `BBR_3D3D.trf`, `BBR_FA.trf`, `IA.trf`

VMR (3x): `ETC-7x-R5_WM`, `ETC-7x-R5`, `For-BBR`

VWP (2x): `ETC-7x-R5_WM_RECO`, `ETC-7x-R5_WM_RECOSM`

FLT (1x): `TempCSFDistMap`



3.12.3 Example script to perform image registration of functional (FMR) to anatomical (VMR) data

This is the example script `Coregister_BV21.js`.

```
/* Coregister_BV21.js

Resulting files:
<FMRname>-TO-<VMRname>_IIHC_IA.trf
<FMRname>-TO-<VMRname>_IIHC_FA.trf
or
<FMRname>-TO-<VMRname>_IIHC_IA.trf
<FMRname>-TO-<VMRname>_BBR_FA.trf
<FMRname>-TO-<VMRname>_BBR_3D3D.trf
<FMRname>-TO-<VMRname>_For-BBR.vmr
```

```
in FMR file directory
*/

var vmrname = BrainVoyager.BrowseFile("Please select the VMR file", "*.vmr");
var vmr = BrainVoyager.OpenDocument(vmrname);
var fmrname = BrainVoyager.BrowseFile("Please select the FMR file", "*.fmr");
//var success = vmr.CoregisterFMRToVMR(fmrname, false);
var success = vmr.CoregisterFMRToVMRUsingBBR(fmrname);
```

CreateVTCInVMRSpace()

Function: CreateVTCInVMRSpace()
Description: Transforms the time course data of an FMR document into a defined 3D space, in this case native space. The result of this transformation is a VTC file. Please note that the VMR properties 'UseBoundingBoxForVTCCreation' and 'ExtendedTALSpaceForVTCCreation' should be set to true or false (this applies to all spaces) before starting to create a VTC. (Since 2.6: the 'UseBoundingBoxForVTCCreation' property is default set to false, so it does not need to be set before creation of any VTC.)
Parameter 1: Name FMR: Name of the functional data file (*.fmr) which should be transformed to the space of the VMR.
Parameter 2: Name IA file: Name of the initial alignment transformation file (`*_IA.trf`).
Parameter 3: Name FA file: Name of the fine alignment transformation file (`*_FA.trf`).
Parameter 4: Name VTC: Name for the new VTC file.
Parameter 5: Datatype: Create the VTC in integer 2-byte format: 1 or in float format: 2.
Parameter 6: Resolution: Target resolution, either '1' (1x1x1mm), '2' or '3'.
Parameter 7: Interpolation: '0' for nearest neighbor interpolation, '1' for trilinear interpolation, '2' for sinc interpolation.
Parameter 8: Threshold: intensity threshold for bounding box (is not relevant for Talairach space, but should be provided). Default value: '100'.
Returns: True (success) or false.

CreateVTCInACPCSpace()

Function: CreateVTCInACPCSpace()
Description: Create a volume-time-course (VTC) file in AC-PC space. Please note that the VMR properties 'UseBoundingBoxForVTCCreation' and 'ExtendedTALSpaceForVTCCreation' should be set to true or false (this applies to all spaces) before starting to create a VTC. (Since 2.6: the 'UseBoundingBoxForVTCCreation' property is default set to false, so it does not need to be set before creation of any VTC.)
Parameter 1: Name FMR: Name of the functional data file (*.fmr) which should be transformed to AC-PC space.
Parameter 2: Name IA file: Name of the initial alignment transformation file (`*_IA.trf`).
Parameter 3: Name FA file: Name of the fine alignment transformation file (`*_FA.trf`).
Parameter 4: Name ACPC file: Name of the transformation file of the VMR to AC-PC space (`*_ACPC.trf`).
Parameter 5: Name VTC: Name for the new VTC file.
Parameter 6: Datatype: Create the VTC in integer 2-byte format: 1 or in float format: 2.
Parameter 7: Resolution: Target resolution, either '1' (1x1x1mm), '2' or '3'.
Parameter 8: Interpolation: '0' for nearest neighbor interpolation, '1' for trilinear interpolation, '2' for sinc interpolation.
Parameter 9: Threshold: intensity threshold for bounding box (is not relevant for Talairach space, but should be provided). Default value: '100'.
Returns: True (success) or false.

CreateVTCInTALSpace()

Function: CreateVTCInTALSpace()
Description: Valid only if the opened document is of type VMR. FMR documents contain functional data in the originally recorded slices without any knowledge about where these slices are located with respect to a 3D reference frame, i.e. Talairach space. Transforming the functional data in Talairach space

allows to analyze data from the same subject across different scanner sessions as well as to analyze data coming from different subjects. The resulting 4D VTC file consists of a series of 3D volumes aligned in stereotactic space. The file is saved to disk under the name for the new VTC file. The first parameter, the name of the FMR, specifies the FMR document whose functional data should be transformed in 3D space (the functional data actually resides in STC files which are referenced by the FMR document). The spatial transformation into Talairach space is controlled by three files which must exist prior to calling this method. The initial and fine alignment files are responsible to align the stack of 2D slices at the correct position of a 3D VMR data set which is typically recorded in the same session as the functional data. If the functional data has been registered with a 3D VMR data set, the further alignment information can be obtained from anatomical transformations. The 3D data set to which the functional data has been aligned can be rotated into the AC-PC plane (see the BrainVoyager User's Guide). A transformation file (*.trf) is produced which transforms the source 3D data set into the AC-PC plane. Since the functional data should undergo exactly the same transformation, you must enter the obtained file as the name of the ACPC file for the present method. The AC-PC plane space is not Talairach space. The final step is to apply a non-linear scaling operation to bring the data in stereotactic space. This is done for the 3D VMR data set in AC-PC space and results in a TAL file. In order to apply the same transformatio to the functional data, enter the obtained file as the name of the TAL file.
Please note that the VMR properties 'UseBoundingBoxForVTCCreation' and 'ExtendedTALSpaceForVTC-Creation' should be set to true or false (this applies to all spaces) before starting to create a VTC. (Since BrainVoyager QX 2.6: 'UseBoundingBoxForVTCCreation' is default set to false, so does not need to be set before each VTC any more.)
Parameter 1: Name FMR: Name of the functional data file (*.fmr) which should be transformed to Talairach space.
Parameter 2: Name IA file: Name of the initial alignment transformation file (`*_IA.trf`).
Parameter 3: Name FA file: Name of the fine alignment transformation file (`*_FA.trf`).
Parameter 4: Name ACPC file: Name of the transformation file of the VMR to AC-PC space (`*_ACPC.trf`).
Parameter 5: Name TAL file: Name of the file containing 12 landmarks used to transform the VMR to Talairach space (`*.tal`).
Parameter 6: Name VTC: Name for the new VTC file.
Parameter 7: Datatype: Create the VTC in integer 2-byte format: 1 or in float format: 2.
Parameter 8: Resolution: Target resolution, either '1' (1x1x1mm), '2' or '3'.
Parameter 8: Interpolation: '0' for nearest neighbor interpolation, '1' for trilinear interpolation, '2' for sinc interpolation.
Parameter 9: Threshold: intensity threshold for bounding box (is not relevant for Talairach space, but should be provided). Default value: '100'.
Returns: True (success) or false.
Note: Before using this function, please set explicitly the parameter
`ExtendedTALSpaceForVTCCreation`, for example:
`docVMR.ExtendedTALSpaceForVTCCreation = false;`. This is necessary to prevent an arbitrary size of the Talairach bounding box, with or without cerebellum.

CreateVTCInMNISpace()

Function: CreateVTCInMNISpace()
Description: Transforms the specified FMRSTC data into MNI space producing a VTC file as output. The boolean return value indicates whether the command could be executed successfully. The IA, FA and MNI transformation files will be combined into one transformation matrix. The resulting resolution will be determined by the spatial resolution (millimeters) of the calling VMR document object but it can be reduced by an integral value using the resolution parameter.
Parameter 1: FMRFileName (str): String specifying the name of the source FMR file; the voxel time course data in the STC file that is referenced by the FMR file will be used as input of the VTC creation process.
Parameter 2: IACoregTRFFileName (str): String specifying the name of the initial alignment (IA) transformation (TRF) file resulting from the preparatory FMRVMR alignment step. Specification of a full path file name is recommended and required if the TRF file is not located in the same directory as the FMR input file.
Parameter 3: FACoregTRFFileName (str): String specifying the name of the finetuning adjustment (FA) transformation (TRF) file resulting from the preparatory FMRVMR alignment step. Specification of a

full path file name is recommended and required if the TRF file is not located in the same directory as the FMR input file.
Parameter 4: MNITRFFileName (str): String specifying the name of the MNI transformation (TRF) file resulting from previously transforming the hosting VMR document into MNI space (e.g. for the default MNI152 space, the file will end with `_TO_MNI_a12.trf` after standard MNI transformation, see NormalizeToMNISpace() command).
Parameter 5: VTCFileName (str): String specifying the name of the resulting volume time course (VTC) file in ACPC space. Specification of a full path file name is recommended and required if the resulting file should be stored in another directory than the one of the FMR input file.
Parameter 6: dataType (int): Integer in the range [1, 2] specifying the data type for the values in the resulting VTC output file; value"1" will produce integer values (2 byte), value "2" (recommended) will produce float32 (4 byte float) values.
Parameter 7: resolution (int): Integer in the range [1, 3] specifying the spatial resolution of the voxels of the resulting VTC data relative to the resolution of the calling VMR document object. A value of "1" indicates that the resolution of the VTC data will be the same as the resolution of the hosting VMR, value "2" indicates that 1 VTC voxel will correspond to $2 \times 2 \times 2$ (8) VMR voxels and value "3" indicates that 1 VTC voxel will correspond to $3 \times 3 \times 3$ (27) VMR voxels.
Parameter 8: interpolationMethod (int): Integer in the range [0, 2] specifying the interpolation method used for resampling the STC input data. Providing value "0" will select nearest neighbor interpolation, value "1" trilinear interpolation and value "2" will select sinc interpolation. Sinc interpolation will be very slow if no GPU is available and enabled for sinc interpolation.
Parameter 9: intensityThreshold (int): A value that is used to separate background voxels (low intensity) from brain voxels (high intensity). Note that for MNI transformation this value is not used since the bounding box of the resulting VTC data is set to the standard dimensions of the resulting MNI space.
Returns: True (success) or false (boolean).

NormalizeToMNISpace()

Function: NormalizeToMNISpace()
Description: Performs automatic brain normalization on an anatomical file (VMR) to MNI space using template matching. To work successfully, the calling VMR document should contain brain extracted and inhomogeneity corrected data (see CorrectIntensityInhomogeneities() command above). The boolean return value indicates whether the command could be executed successfully. If successful, a new VMR file with the data in MNI152 space are stored to disk under a name that is based on the input VMR document:
"`[inputname]_MNI.vmr''`. The used MNI transformation parameters are stored to disk in the TRF file
"`[inputname]_TO_MNI_a12.trf`"; furthermore the files
"`[input name]_TO_MNIColin27_a12.trf`" and
"`[inputname]_TO_ICBM452_a12.trf`" are stored that can be used to transform the native space VMR also to related spaces, namely to the Colin27 and ICBM 452 MNI space. These TRF files can also be used to apply the MNI transformation to other VMRs in the same native space and as part of the transformation of functional data into MNI space (see CreateVTCInMNISpace() command). The original VMR document is closed and the resulting MNI file is loaded into the BrainVoyager workspace. The loaded MNI file is made the current document and can be accessed using the `ActiveDocument` command of the BrainVoyager application object (section 3.4.1).
Returns: True (success) or false.

3.12.4 Example script to create a VMR and transform to Talairach space

Below is included an example script to create a VMR, perform inhomogeneity correction and transform to AC-PC and Talairach space.

```
// CreateVMRAndAutoTAL_BV21.js (modified 16-12-16, 12-06-19, 22-08-19)
//var ObjectsRawDataPath = BrainVoyager.BrowseDirectory("Please select the path to the BrainVoyager sample data") + "/";
var filename = BrainVoyager.BrowseFile("Please select the first anatomical file", "*.dcm");

function CreateVMRandCorrectForInhomogeneities()
{
   var docVMR = BrainVoyager.CreateDocumentVMR( "DICOM", filename, 192, false, 256, 256, 2 );
   // var docVMR = BrainVoyager.CreateDocumentVMR( "DICOM",
   // ObjectsRawDataPath + "/08MPRAGE_1mm_Nondistorted/" + "JudEck_20181128_BI_Exercises_sess4-0008-0001-00001.dcm",
   // 192, false, 256, 256, 2 );
    docVMR.SaveAs( ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT.vmr" );
    docVMR.CorrectIntensityInhomogeneities(); // this function works only directly after VMR creation
}

function AutoACPCAndTAL()
{
var ok;
var docVMR = BrainVoyager.ActiveDocument;
docVMR.AutoACPCAndTALTransformation();
}

CreateVMRandCorrectForInhomogeneities();
AutoACPCAndTAL();
```

3.12.5 Example script to create VTC files

```
/*  CreateVTCfiles_BV21.qs
Script to create VTC files
By Rainer Goebel 2005
Modified for QX 2.1 2009, for BV21 030419, 060619, 220819 by HB
*/

/* This information is used in the functions */
// Setting hardcoded paths:
/* var ObjectsRawDataPath = "/Users/BVuser/Data/ObjectsDicomGSG/";
var nameFMR = ObjectsRawDataPath + "sub-01_ses- 04_task-blocked_run-1_bold_SCCTBL_3DMCTS_THPGLMF2c.fmr";
var nameVMRinNative = ObjectsRawDataPath +"sub-01_ses-04_acq-nondistorted_ T1w_IIHC.vmr";
etc...
 */

    // Setting paths via file selection dialog:
    var DTIDataPath  = BrainVoyager.BrowseDirectory("Please select the directory with DWI data");
    var nameVMRinNative  = BrainVoyager.BrowseFile("Please select the native VMR file", "*.vmr");
    var nameVMRinAcpc = BrainVoyager.BrowseFile("Please select the VMR file in AC-PC space", "*ACPC.vmr");
    var nameVMRinTal  = BrainVoyager.BrowseFile("Please select the VMR file in Talairach space", "*TAL.vmr");
    var nameVMRinMNI  = BrainVoyager.BrowseFile("Please select the VMR file in MNI space", "*MNI.vmr");
    var nameFMR  = BrainVoyager.BrowseFile("Please select the FMR file", "*.fmr");
    var nameIAfile = BrainVoyager.BrowseFile("Please select the initial alignment file", "*IA.trf");
    var nameFAfile  = BrainVoyager.BrowseFile("Please select the fine alignment file", "*FA.trf");
    var nameACPCfile  = BrainVoyager.BrowseFile("Please select the AC-PC transformation file", "*ACPC.trf");
    var nameTALfile  = BrainVoyager.BrowseFile("Please select the Talairach landmarks file", "*.tal");
    var nameMNIfile = BrainVoyager.BrowseFile("Please select the MNI transformation file", "*a12.trf");

//var today = new Date();
var dataType = 2; // 1: int16, 2: float32
var resolution = 2; // one of 1, 2 or 3 mm^2
var interpolation = 1; // 0 for nearest neighbor interpolation, 1 for trilinear interpolation, 2 for sinc interpolation
var threshold = 100; // intensity threshold for bounding box (is not relevant for Talairach space, still required).
var extendedBoundingBox = false;

/*  This code is executed when clicking 'Run'
If an error occurs, it is printed to the BrainVoyager Log tab
*/
BrainVoyager.ShowLogTab();
BrainVoyager.PrintToLog("Start creating VTCs...");
try {
CreateVTCinNativeSpace();
CreateVTCinAcpcSpace();
CreateVTCinTalairachSpace(false); // make a VTC in Talairach space with extended bounding box = false
CreateVTCinTalairachSpace(true); // make a VTC in Talairach space with extended bounding box (in z-dir)
CreateVTCinMNISpace();
} catch (e) {
BrainVoyager.PrintToLog("Error: " + e);
}

/* These functions are invoked from the section above */
function CreateVTCinNativeSpace()
{
```

```
var docVMR = BrainVoyager.OpenDocument(nameVMRinNative);
docVMR.ExtendedTALSpaceForVTCCreation = false;
var nameVTCinNative = getNewName(nameFMR, "_NATIVE", ".vtc");
var success = docVMR.CreateVTCInVMRSpace(nameFMR, nameIAfile, nameFAfile, nameVTCinNative,
                                         dataType, resolution, interpolation, threshold);
if (success) BrainVoyager.PrintToLog("Created VTC file: " + nameVTCinNative);
docVMR.Close();
}

function CreateVTCinAcpcSpace()
{
var docVMR = BrainVoyager.OpenDocument(nameVMRinAcpc);
var nameVTCinACPC = getNewName(nameFMR, "_ACPC", ".vtc");
docVMR.ExtendedTALSpaceForVTCCreation = false;
var success = docVMR.CreateVTCInACPCSpace(nameFMR, nameIAfile, nameFAfile, nameACPCfile,
                              nameVTCinACPC, dataType, resolution, interpolation, threshold);
docVMR.Close();
}

function CreateVTCinTalairachSpace(useExtendedBoundingBox)
{
var docVMR = BrainVoyager.OpenDocument(nameVMRinTal);
var nameVTCinTAL = getNewName(nameFMR, "_TAL", ".vtc");
docVMR.ExtendedTALSpaceForVTCCreation = useExtendedBoundingBox; // this is true or false
var success = docVMR.CreateVTCInTALSpace(nameFMR, nameIAfile, nameFAfile, nameACPCfile, nameTALfile,
                                         nameVTCinTAL, dataType, resolution, interpolation, threshold);
docVMR.Close();
}

function CreateVTCinMNISpace()
{
var docVMR = BrainVoyager.OpenDocument(nameVMRinMNI);
var nameVTCinMNI = getNewName(nameFMR, "_MNI", ".vtc");
docVMR.ExtendedTALSpaceForVTCCreation = false;
var success = docVMR.CreateVTCInMNISpace(nameFMR, nameIAfile, nameFAfile, nameMNIfile,
                                         nameVTCinMNI, dataType, resolution, interpolation, threshold);
docVMR.Close();
}

function getNewName(name, infix, suffix) {
var fi = new QFileInfo(name);
var newname = fi.path() + "/" + fi.baseName() + infix + suffix;
return newname;
}
```

3.12.6 Creating diffusion weighted (VDW) files

CreateVDWInVMRSpace()

Function: CreateVDWInVMRSpace()
Description: Create a normalized diffusion weighted (VDW) file in native space (directly from scanner).
Parameter 1: Name DMR: Name of the diffusion-weighted data file (*.dmr) which should be transformed.
Parameter 2: Name IA file: Name of the initial alignment transformation file (`*_IA.trf`).
Parameter 3: Name FA file: Name of the fine alignment transformation file (`*_FA.trf`).
Parameter 4: Name VDW: Name for the new VDW file.
Parameter 5: Datatype: Create the VDW in integer 2-byte format: 1 or in float format: 2.
Parameter 6: Resolution: Target resolution, either '1' (1x1x1mm), '2' or '3'.
Parameter 7: Interpolation: '0' for nearest neighbor interpolation, '1' for trilinear interpolation, '2' for sinc interpolation.
Parameter 8: Threshold: intensity threshold for bounding box. Default value: '100'.
Returns: True (success) or false.

CreateVDWInACPCSpace()

Function: CreateVDWInACPCSpace()
Description: Create a normalized diffusion weighted (VDW) file in AC-PC space.
Parameter 1: Name DMR: Name of the diffusion-weighted data file (*.dmr) which should be transformed to AC-PC space.
Parameter 2: Name IA file: Name of the initial alignment transformation file (`*_IA.trf`).
Parameter 3: Name FA file: Name of the fine alignment transformation file (`*_FA.trf`).
Parameter 4: Name ACPC file: Name of the transformation file of the VMR to AC-PC space (`*_ACPC.trf`).
Parameter 5: Name VDW: Name for the new VDW file.
Parameter 6: Datatype: Create the VDW in integer 2-byte format: 1 or in float format: 2.
Parameter 7: Resolution: Target resolution, either '1' (1x1x1mm), '2' or '3'.
Parameter 8: Interpolation: '0' for nearest neighbor interpolation, '1' for trilinear interpolation, '2' for sinc interpolation.
Parameter 9: Threshold: intensity threshold for bounding box. Default value: '100'.
Returns: True (success) or false.

CreateVDWInTALSpace()

Function: CreateVDWInTALSpace()
Description: Create a normalized diffusion weighted (VDW) file in Talairach space.
Parameter 1: Name DMR: Name of the diffusion-weighted data file (*.dmr) which should be transformed to Talairach space.
Parameter 2: Name IA file: Name of the initial alignment transformation file (`*_IA.trf`).
Parameter 3: Name FA file: Name of the fine alignment transformation file (`*_FA.trf`).
Parameter 4: Name ACPC file: Name of the transformation file of the VMR to AC-PC space (`*_ACPC.trf`).
Parameter 5: Name TAL file: Name of the file containing 12 landmarks used to transform the VMR to Talairach space (`*.tal`).
Parameter 6: Name VDW: Name for the new VDW file.
Parameter 7: Datatype: Create the VDW in integer 2-byte format: 1 or in float format: 2.
Parameter 8: Resolution: Target resolution, either '1' (1x1x1mm), '2' or '3'.
Parameter 9: Interpolation: '0' for nearest neighbor interpolation, '1' for trilinear interpolation, '2' for sinc interpolation.
Parameter 10: Threshold: intensity threshold for bounding box (is not relevant for Talairach space, but should be provided). Default value: '100'.
Returns: True (success) or false.

3.12.7 Example script to create VDW files

```
/*  CreateVDWfiles.qs
Script to create VDW files (normalised diffusion weighted)
*/

/* This information is used in the functions */
// Setting hardcoded paths:
/*    var DTIDataPath  = "/Users/BVuser/Data/Human31dir/";
var nameDMR  = DTIDataPath + "HUMAN31DIR_SCRIPT.dmr";
var nameVMRinNative  = DTIDataPath +"human.vmr";
var nameVMRinAcpc = DTIDataPath +"human_ACPC.vmr";
var nameVMRinTal  = DTIDataPath +"human_TAL.vmr";
var nameIAfile  = DTIDataPath + "HUMAN31DIR_SCRIPT-TO-human_IA.trf";
var nameFAfile  = DTIDataPath + "HUMAN31DIR_SCRIPT-TO-human_FA.trf";
var nameACPCfile  = DTIDataPath + "human_ACPC.trf";
var nameTALfile  = DTIDataPath + "human_ACPC.tal";
 */
    // Setting paths via file selection dialog:
    var DTIDataPath  = BrainVoyager.BrowseDirectory("Please select the directory with DWI data") + "/";
    var nameDMR  = BrainVoyager.BrowseFile("Please select the DMR file", "*.dmr");
    var nameVMRinNative  = BrainVoyager.BrowseFile("Please select the native VMR file", "*.vmr");
    var nameVMRinAcpc = BrainVoyager.BrowseFile("Please select the VMR file in AC-PC space", "*ACPC.vmr");
    var nameVMRinTal  = BrainVoyager.BrowseFile("Please select the VMR file in Talairach space", "*TAL.vmr");
    var nameIAfile  = BrainVoyager.BrowseFile("Please select the initial alignment file", "*IA.trf");
    var nameFAfile  = BrainVoyager.BrowseFile("Please select the fine alignment file", "*FA.trf");
    var nameACPCfile  = BrainVoyager.BrowseFile("Please select the AC-PC transformation file", "*ACPC.trf");
    var nameTALfile  = BrainVoyager.BrowseFile("Please select the Talairach landmarks file", "*.tal");

var today  = new Date();
var nameVDWinNative = DTIDataPath + "HUMAN31DIR_SCRIPT_NATIVE.vdw";
var nameVDWinAcpc = DTIDataPath + "HUMAN31DIR_SCRIPT_ACPC.vdw";
var nameVDWinTal = DTIDataPath + "HUMAN31DIR_SCRIPT_TAL.vdw";
var dataType                = 2; // integer 2-byte format: 1 or in float format: 2.
var resolution  = 3; // one of 1, 2 or 3 mm^2
var interpolation  = 1;
var threshold  = 100;

/*  This code is executed when clicking 'Run'
If an error occurs, it is printed to the BrainVoyager QX Log tab
*/
BrainVoyager.ShowLogTab();
BrainVoyager.PrintToLog("Start creating VDWs...");
try {
CreateVDWinNativeSpace();
CreateVDWinAcpcSpace();
CreateVDWinTalSpace();
} catch (e) {
BrainVoyager.PrintToLog("Error: " + e);
}

/* These functions are invoked from the section above */
function CreateVDWinNativeSpace() {

BrainVoyager.TimeOutMessageBox("This function creates a VDW file in native space...", 5);
var vmr = BrainVoyager.OpenDocument(nameVMRinNative);
BrainVoyager.PrintToLog("Start creating VDW in native space...");
vmr.ExtendedTALSpaceForVTCCreation = false; // this is true or false
var success = vmr.CreateVDWInVMRSpace(nameDMR, nameIAfile, nameFAfile, nameVDWinNative,
dataType, resolution, interpolation, threshold);
if (success) {
BrainVoyager.PrintToLog("Created VDW: " + nameVDWinNative);
} else {
BrainVoyager.PrintToLog("VDW creation did not succeed.");
}
  vmr.Close();
}

function CreateVDWinAcpcSpace() {

BrainVoyager.TimeOutMessageBox("This function creates a VDW file in AC-PC space...", 5);
var vmr = BrainVoyager.OpenDocument(nameVMRinAcpc);
BrainVoyager.PrintToLog("Start creating VDW in AC-PC space...");
vmr.ExtendedTALSpaceForVTCCreation = false; // this is true or false
var success = vmr.CreateVDWInACPCSpace(nameDMR, nameIAfile, nameFAfile, nameACPCfile,
nameVDWinAcpc, dataType, resolution, interpolation, threshold);
if (success) {
BrainVoyager.PrintToLog("Created VDW: " + nameVDWinAcpc);
} else {
BrainVoyager.PrintToLog("VDW creation did not succeed.");
}
  vmr.Close();
}

function CreateVDWinTalSpace() {

BrainVoyager.TimeOutMessageBox("This function creates a VDW file in Talairach space...", 5);
```

```
var vmr = BrainVoyager.OpenDocument(nameVMRinTal);
BrainVoyager.PrintToLog("Start creating VDW in Talairach space...");
vmr.ExtendedTALSpaceForVTCCreation = false; // always set this property
var success = vmr.CreateVDWInTALSpace(nameDMR, nameIAfile, nameFAfile, nameACPCfile,
nameTALfile, nameVDWinTal, dataType, resolution, interpolation, threshold);
if (success) {
BrainVoyager.PrintToLog("Created VDW: " + nameVDWinTal);
} else {
BrainVoyager.PrintToLog("VDW creation did not succeed.");
}
  vmr.Close();
}
```

3.13 BV document functions: Surface functions

3.13.1 List of Methods

These functions can be invoked using an anatomical volume (VMR):
LoadMesh()
AddMesh()
SaveMesh()
UpdateAppearance()
SaveSnapshotOf3DViewer(); LinkMTC()
CreateMTCFromVTC()
CreateMeshScene()

The following functions can be invoking with the mesh object (SRF):
mesh.SaveAs()
mesh.CalculateCurvature()
mesh.CalculateCurvatureCBA()
mesh.ClearMultiStudyGLMDefinition()
mesh.CreateSurfaceMapFromVolumeMap()
mesh.CreateSurfaceMapFromVolumeMapDepth()
mesh.SmoothMesh()
mesh.SmoothGeometry()
mesh.SmoothGeometrySimple()
mesh.SmoothRecoMesh()
mesh.SmoothCurrentMap()
mesh.InflateMeshToSphere()
mesh.InflateGeometryToSphere()
mesh.InflateGeometryToSphereExt()
mesh.InflateMesh()
mesh.InflateGeometry(
mesh.SimplifyGeometry()
mesh.CorrectInflatedSphereMesh()
mesh.CorrectInflatedSphereDistortions()
mesh.CreateMultiScaleCurvatureMap()
mesh.SpatialSmoothing()
mesh.LinearTrendRemoval()
mesh.TemporalHighPassFilterFFT()
mesh.TemporalGaussianSmoothing()
mesh.CreateMTCFromVTC()
mesh.LinkMTC()
mesh.SaveMTC()
mesh.ClearDesignMatrix()
mesh.LoadSingleStudyGLMDesignMatrix()
mesh.SaveSingleStudyGLMDesignMatrix
mesh.LoadMultiStudyGLMDefinitionFile()
mesh.AddStudyAndDesignMatrixAndCortexMapping()
mesh.SaveMultiStudyGLMDefinitionFile()
mesh.ComputeSingleStudyGLM()
mesh.ComputeMultiStudyGLM()
mesh.ComputeRFXGLM()
mesh.LoadGLM()
mesh.ShowGLM()
mesh.SaveGLM()
mesh.LoadSurfaceMaps()
mesh.ShowSurfaceMap()
mesh.SaveSurfaceMaps()
mesh.GetNameOfSurfaceMap()
mesh.CreateSphericalCoordinatesMapFromSMP()

mesh.Update3DViewer()

The following functions can be invoking with the mesh scene object:
mesh.MeshScene.UpdateSurfaceWindow()
meshScene.LoadMesh()
MapSphereMeshFromStandardSphere()
SetStandardSphereToFoldedMesh()
CreateStandardSphereMesh()
ClearGroupCBACurvatureFiles()
AddCurvatureFileForGroupCBA()
RunRigidCBA()
RunCBA()
CreateAverageCurvatureGroupMap()
CreateAverageFoldedGroupMesh()
SaveSnapshotOf3DViewer()
meshScene.MergeMeshesInScene()

3.13.2 List of Properties

The properties can be used with an anatomical volume (VMR): MeshScene

The mesh (SRF) object has the following properties:
mesh.FileName
mesh.NrOfVertices
mesh.MorphingUpdateInterval
mesh.FileNameOfPreprocessdMTC
mesh.CorrectForSerialCorrelations
mesh.NrOfMTCVolumes
mesh.NrOfMTCTimePoints
mesh,NrOfSurfaceMaps

The mesh scene object has the following properties:
meshScene.CurrentMesh
meshScene.ViewpointPositionX
meshScene.ViewpointPositionY
meshScene.ViewpointPositionZ
meshScene.ViewpointRotationX
meshScene.ViewpointRotationY
meshScene.ViewpointRotationZ
meshScene.SphereResolutionCBA

3.13.3 Description of VMR object methods

LoadMesh()

Function: LoadMesh()
Description: Load a mesh into the current scene. This requires a *.vmr to be open. See also meshScene.LoadMesh().
Parameter 1: Name of the polygon mesh (*.srf).

AddMesh()

Function: AddMesh()
Description: Add a mesh to the current scene. This requires a *.vmr to be open.
Parameter 1: Name of the polygon mesh (*.srf).

SaveMesh()

Function: SaveMesh()
Description: Save the current mesh. This is a method of the *.vmr object. Please note that from BrainVoyager QX 2.8 one can use the mesh.SaveAs() function.
Parameter 1: Name for the polygon mesh (*.srf).

SaveSnapshotOfSurfaceWindow() (obsolete)

Function: SaveSnapshotOfSurfaceWindow()
Description: Save a snapshot of the current OpenGL window. This function is obsolete since BrainVoyager 20 or 21. Use SaveSnapshotOf3DViewer() instead.
Parameter 1: Name for the snapshot.
Note: The file type of the snapshot is determined the filetype selected in BrainVoyager → Preferences.

SaveSnapshotOf3DViewer()

Function: SaveSnapshotOf3DViewer()
Description: Save a snapshot of the current OpenGL window. This is a function of meshscene: meshscene.SaveSnapshotOf3DViewer().
Parameter 1: Name for the snapshot.
Returns: Success (boolean).
Note: The file type of the snapshot is determined the filetype selected in BrainVoyager → Preferences.

CreateMeshScene()

Function: CreateMeshScene()
Description: This function opens the 3D Viewer, so that `MeshScene` functions can be used. It gives you an existing MeshView if previously created or creates a new one and returns this one. Since BrainVoyager 20/21.

3.13.4 Description of MeshScene object methods

mesh.MeshScene.UpdateSurfaceWindow() (obsolete)

Function: mesh.MeshScene.UpdateSurfaceWindow()
Description: After any operation on a mesh, invoke this function to see the effect. This function is obsolete since BrainVoyager 20 or 21. Use mesh.UpdateAppearance() instead.

getMeshScene()

Function: getMeshScene()
Description: Get mesh scene object. Might be obsolete from about BrainVoyager 21; could try doc.CreateMeshScene() instead.

Returns: Mesh scene object.

mesh.MeshScene.getCurrentMesh()

Function: mesh.MeshScene.getCurrentMesh()
Description: Obtain mesh in mesh scene, in order to use the mesh functions below.
Returns: Mesh object.

meshScene.LoadMesh()

Function: meshScene.LoadMesh()
Description: Load a mesh from the mesh scene object. The mesh scene object can be obtained from the VMR object. This requires BrainVoyager QX 2.8. For older versions, a mesh can also be loaded via a VMR (see LoadMesh()). To obtain the mesh, use
`mesh = meshScene.CurrentMesh`.
Returns: boolean.

meshScene.MergeMeshesInScene()

Function: meshScene.MergeMeshesInScene()
Description: Merge meshes currently open in 3D Viewer.
Returns: Name of merged mesh

3.13.5 Description of Mesh object methods

mesh.SaveAs()

Function: mesh.SaveAs()
Description: Save the current mesh. To be used in BrainVoyager QX 2.8 or higher.
Parameter 1: Name for the polygon mesh (*.srf).

mesh.UpdateAppearance()

Function: mesh.UpdateAppearance()
Description: Show changes on mesh. Function replacing UpdateSurfaceWindow() from BrainVoyager 20 or 21.

mesh.Update3DViewer()

Function: mesh.Update3DViewer()
Description: Show changes on mesh.
Parameter 1 (optional): Process events (boolean)

mesh.UpdateGeometry()

Function: mesh.UpdateGeometry()
Description:

3.13.6 Mesh morphing

mesh.SmoothMesh() (obsolete)

Function: mesh.SmoothMesh()
Description: This is the standard function for smoothing a mesh (`*_RECO.srf`). As a result, the mesh

will shrink. For a function without shrinking, use mesh.SmoothRecoMesh(). To be used in BrainVoyager QX 2.8 or higher. (See example script "MeshMorphing.js".) From BrainVoyager 20 or 21, use mesh.SmoothGeometry().
Parameter 1: Number of smoothing cycles/iterations (for example 30).
Parameter 2: Smoothing force (f.e. 0.07).

mesh.SmoothGeometry()

Function: mesh.SmoothGeometry()
Description: This is the standard function for smoothing a mesh (`*_RECO.srf`). As a result, the mesh will shrink. For a function without shrinking, use mesh.SmoothRecoMesh(). To be used in BrainVoyager 20 or 21 or higher. (See example script "MeshMorphing.js".)
Parameter 1: Number of smoothing cycles/iterations (for example 30).
Parameter 2: Smoothing force (f.e. 0.07).

mesh.SmoothRecoMesh()

Function: mesh.SmoothRecoMesh()
Description: Special "high-frequency" smoothing removing jags of reconstructed voxel borders without shrinking the mesh (`*_RECO.srf`). For the standard version of the function, see mesh.SmoothMesh(). To be used in BrainVoyager QX 2.8 or higher. (See example script "MeshMorphing.js".)
Parameter 1: Number of smoothing iterations (for example 50).
Parameter 2: Smoothing force (f.e. 0.07).

mesh.CalculateCurvature()

Function: mesh.CalculateCurvature()
Description: After smoothing, calculate the curvature of the mesh via this function. (See example script "MeshMorphing.js".) To be used in BrainVoyager QX 2.8 or higher.

mesh.InflateMesh() (obsolete)

Function: mesh.InflateMesh()
Description: Inflate the mesh (`*_RECO.srf`) after smoothing. To be used in BrainVoyager QX 2.8 or higher. (See example script "MeshMorphing.js".) From BrainVoyager 20 or 21 use mesh.InflateGeometry().
Parameter 1: Number of morphing iterations (for example 500).
Parameter 2: Smoothing force (for example 0.8).
Parameter 3: (" "). If " " used for 3rd param (area reference mesh), the current mesh at the moment the function is called is used to calculate the area of the reference mesh.

mesh.InflateGeometry()

Function: mesh.InflateGeometry()
Description: Inflate the mesh (`*_RECO.srf`) after smoothing. To be used in BrainVoyager 20 or 21 or higher. (See example script "MeshMorphing.js".)
Parameter 1: Number of morphing iterations/cycles (for example 500).
Parameter 2: Smoothing force (for example 0.8).
Parameter 3: (" "). If " " used for 3rd param (area reference mesh), the current mesh at the moment the function is called is used to calculate the area of the reference mesh.

mesh.InflateGeometryToSphere()

Function: mesh.InflateGeometryToSphere()
Description: Inflate the mesh to a sphere. From BrainVoyager 20 or 21.
Parameter 1: Number of morphing iterations/cycles (for example 500).
Returns: Success (boolean)

mesh.InflateGeometryToSphereExt()

Function: mesh.InflateGeometryToSphereExt()
Description: Inflate the mesh to a sphere. From BrainVoyager 20 or 21.
Parameter 1: Number of morphing iterations/cycles (for example 500).
Parameter 2: Morphing force minimum value.
Parameter 3: Morphing force maximum value.
Parameter 4: Correction force minimum value.
Parameter 5: Correction force maximum value.
Returns: Success (boolean)

mesh.CorrectInflatedSphereDistortions()

Function: mesh.CorrectInflatedSphereDistortions()
Description: Apply corrections for any distortions that were introduced when transforming the mesh to a sphere
Parameter 1: Number of morphing iterations/cycles.
Returns: Success (boolean)

mesh.SimplifyGeometry()

Function: mesh.SimplifyGeometry()
Description: Reduce number of vertices in the mesh
Parameter 1: Number of target vertices.
Returns: String

mesh.SmoothGeometrySimple()

Function: mesh.SmoothGeometrySimple()
Description: Smooth the mesh
Parameter 1: Number of morphing iterations/cycles.
Parameter 2: Smoothing force.
Returns: Success (boolean)

mesh.SmoothGeometry()

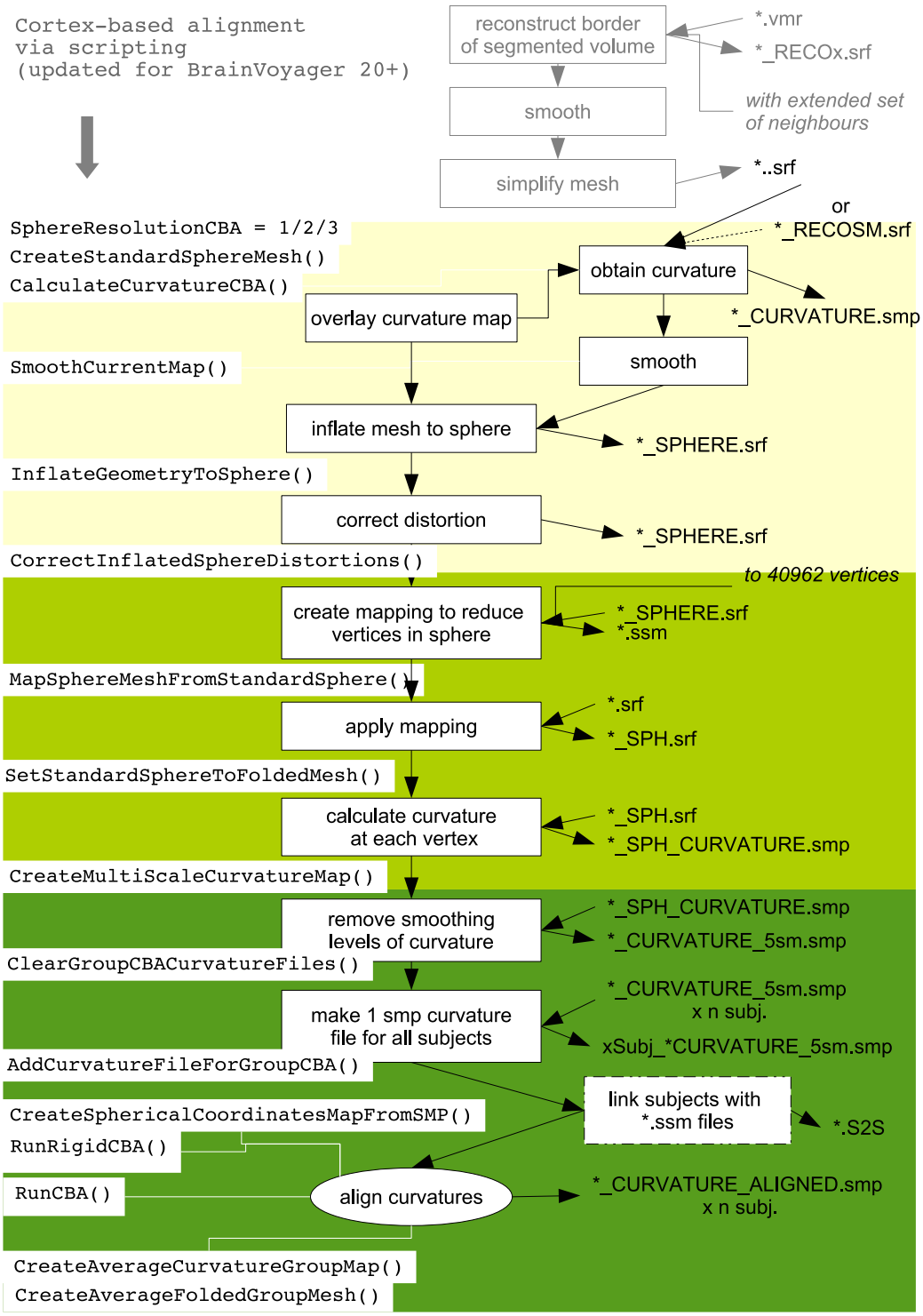
Function: mesh.SmoothGeometry()
Description: Smooth the mesh
Parameter 1: Number of morphing iterations/cycles.
Parameter 2: Smoothing force.
Returns: Success (boolean)

mesh.getMorphingUpdateInterval()

Function: mesh.getMorphingUpdateInterval()
Description:
Parameter 1: Number of iterations/cycles before updating.

mesh.setMorphingUpdateInterval()

Function: mesh.setMorphingUpdateInterval()
Description:
Parameter 1: Number of iterations/cycles before updating.


Cortex-based alignment
via scripting
(updated for BrainVoyager 20+)
reconstruct border of segmented volume
*.vmr
*_RECOx.srf
with extended set of neighbours
smooth
simplify mesh
*..srf
or
*_RECOSM.srf
SphereResolutionCBA = 1/2/3
CreateStandardSphereMesh()
CalculateCurvatureCBA()
obtain curvature
overlay curvature map
*_CURVATURE.smp
smooth
SmoothCurrentMap()
inflate mesh to sphere
*_SPHERE.srf
InflateGeometryToSphere()
correct distortion
*_SPHERE.srf
CorrectInflatedSphereDistortions()
to 40962 vertices
create mapping to reduce vertices in sphere
*_SPHERE.srf
*.ssm
MapSphereMeshFromStandardSphere()
apply mapping
*.srf
*_SPH.srf
SetStandardSphereToFoldedMesh()
calculate curvature at each vertex
*_SPH.srf
*_SPH_CURVATURE.smp
CreateMultiScaleCurvatureMap()
remove smoothing levels of curvature
*_SPH_CURVATURE.smp
*_CURVATURE_5sm.smp
ClearGroupCBACurvatureFiles()
make 1 smp curvature file for all subjects
*_CURVATURE_5sm.smp
x n subj.
xSubj_*CURVATURE_5sm.smp
AddCurvatureFileForGroupCBA()
link subjects with *.ssm files
*.S2S
CreateSphericalCoordinatesMapFromSMP()
RunRigidCBA()
RunCBA()
align curvatures
*_CURVATURE_ALIGNED.smp
x n subj.
CreateAverageCurvatureGroupMap()
CreateAverageFoldedGroupMesh()

3.13.7 Cortex-based alignment (CBA)

mesh.CalculateCurvatureCBA()

Function: mesh.CalculateCurvatureCBA()
Description: Calculate the curvature of a mesh. The mesh object should refer to a smoothed mesh (`*_RECOSM.srf`) or preferably a simplified mesh (`* _D80k.srf`). While not strictly necessary, it is recommended to compute and overlay a curvature map on the folded cortex mesh, since it allows to identify sulci and gyri when the mesh is morphed to a sphere in the succeeding step.To be used in BrainVoyager QX 2.8 or higher.

mesh.SmoothCurrentMap()

Function: mesh.SmoothCurrentMap()
Description: Smooths the curvature map (`*_CURVATURE.smp`). To be used in BrainVoyager QX 2.8 or higher.
Parameter 1: Number of smoothing steps (for example 5).

mesh.InflateMeshToSphere() (obsolete)

Function: mesh.InflateMeshToSphere()
Description: Inflates the simplified mesh (`*_D80k.srf`) to a sphere (`*_SPHERE.srf`). This will start an iterative morphing process that uses several forces with values that will be changed gradually over time. A *smoothing force* is used to remove the gyri and sulci from the mesh, while a *to-sphere force* attempts to push the mesh vertices outward towards points on a sphere. To be used in BrainVoyager QX 2.8 or higher. Apply CorrectInflatedSphereMesh() afterwards; then save the sphere using `mesh.SaveAs()` when the result is acceptable.
Parameter 1: Number of inflation steps (for example 800).

mesh.CorrectInflatedSphereMesh() (obsolete)

Function: mesh.CorrectInflatedSphereMesh()
Description: The distortion correction process uses a "push-relax" approach on the spherical mesh (`*_SPHERE.srf`), to correct for some parts, which are expanded - especially mid-lateral and mid-medial parts - and other parts, which are squeezed - especially frontal and occipital regions. To be used in BrainVoyager QX 2.8 or higher. From BrainVoyager 20 or 21, use mesh.CorrectInflatedSphereDistortions().
Parameter 1: Number of correction steps (3000 should be sufficient).

MapSphereMeshFromStandardSphere()

Function: MapSphereMeshFromStandardSphere()
Description: Function of `MeshScene` object: `meshscene.MapSphereMeshFromStandardSphere()`. After the mesh has been inflated to a sphere (`*_SPHERE.srf`), this function can be used to calculate a mapping to a standard sphere with 40962 vertices to reduce the number of vertices. To be used in BrainVoyager QX 2.8 or higher.
Returns 1: A mapping between meshes file (*.ssm).

SetStandardSphereToFoldedMesh()

Function: SetStandardSphereToFoldedMesh()
Description: Apply the mapping (*.ssm) obtained from mesh.MapSphereMeshFromStandardSphere(). This results in a standard "folded sphere" (`*_SPH.srf`). To be used in BrainVoyager QX 2.8 or higher.
Parameter 1: Name of folded mesh (`*_RECOSM.srf`). *Returns:* Name of "folded sphere" (`*_SPH.srf`).

mesh.CreateMultiScaleCurvatureMap()

Function: mesh.CreateMultiScaleCurvatureMap()
Description: Since the standard sphere is now folded as the original sphere, the curvature can be calculated directly at each vertex. This can be performed with the current function. Since the alignment

procedure uses a coarse-to-fine approach, several different curvature maps will be calculated with various levels of smoothing. The resulting SMP file will contain four sub-maps with the smoothing levels that are provided, for example 2, 7, 20 and 40. To be used in BrainVoyager QX 2.8 or higher.
Parameter 1: Smoothing level 1 (should be most coarse, for example 0 or 2).
Parameter 2: Smoothing level 2 (can be quite coarse, for example 7).
Parameter 3: Smoothing level 3 (should be quite fine, for example 20).
Parameter 4: Smoothing level 4 (should be most fine, for example 40).
Returns: Name of curvature file (`*_CURVATURE.smp`).

CreateStandardSphereMesh()

Function: CreateStandardSphereMesh()
Description: Create a standard sphere mesh via the mesh scene object. To be used in BrainVoyager QX 2.8 or higher.

ClearGroupCBACurvatureFiles()

Function: ClearGroupCBACurvatureFiles()
Description: Remove any present information about curvature files via the mesh scene object. To be used in BrainVoyager QX 2.8 or higher.

AddCurvatureFileForGroupCBA()

Function: AddCurvatureFileForGroupCBA()
Description: Inform BrainVoyager about the curvature files that are to be used in the cortex-based alignment. This function of the mesh scene object has to be invoked repeatedly until the files of all subjects are declared (see script example "MeshCBA.js"). In order to be consistent with the file names used during the CBA procedure, the same subject identifiers should be used for the subject mesh names and surface maps. The program uses the initial part of a map's name to identify a subject; this can be a number (e.g. "362"), text (e.g. "XY") or a combination of both (e.g. "S16"). To be used in BrainVoyager QX 2.8 or higher. Then, proceed to next step RunRigidCBA() or RunCBA().
Parameter 1: Name of curvature file (`*_SPH_CURVATURE.smp`).
Returns: Success or no success (boolean).

mesh.CreateSphericalCoordinatesMapFromSMP()

Function: mesh.CreateSphericalCoordinatesMapFromSMP()
Description: Create a spherical target curvature file for RunRigidCBA(). From BrainVoyager QX 2.8.
Parameter 1: Name of target curvature file (`*_SPH_CURVATURE.smp`), any should be fine.
Returns: Name of spherical target curvature file (we get an empty string if it did not work).

RunRigidCBA()

Function: RunRigidCBA()
Description: This is a function of the mesh scene object. After all files have been added via AddCurvatureFileForGroupCBA(), the surface maps of the subjects can be transformed into group-aligned space by this function. The resulting aligned surface maps are automatically saved to disk using the name of the original SMP with an added ``_ALIGNED'' string (`*_CURVATURE_ALIGNED_RIGIDONLY.smp`). The result of rigid alignment is automatically saved in a ".rga" file that will be used by subsequent CBA when using same input curvature SMP files. Then, proceed to next step RunCBA(). To be used in BrainVoyager QX 2.8 or higher.
Parameter 1: Name of target curvature file, which can be created via the function mesh.CreateSphericalCoordinatesMapFromSMP().
Returns: Success or no success (boolean).

RunCBA()

Function: RunCBA()
Description: This is a function of the mesh scene object. After all files have been added via AddCurvatureFileForGroupCBA(), the surface maps of the subjects can be transformed into group-aligned space by this function. The resulting aligned surface maps are automatically saved to disk using the name of the original SMP with an added ``_ALIGNED'' string (`*_CURVATURE_ALIGNED.smp`). To be used in BrainVoyager QX 2.8 or higher. *Returns:* Success or no success (boolean).

CreateAverageCurvatureGroupMap()

Function: CreateAverageCurvatureGroupMap()
Description: This convenience function can only be called after RunCBA() (with internally stored set of .SRF, .SSM, .SMP file names): it creates a group curvature map indicating the quality of alignment; also, it saves a ”.cal” file that can be used in the Cortex-Based Alignment dialog. To be used in BrainVoyager QX 2.8 or higher. *Returns:* Success or no success (boolean).

CreateAverageFoldedGroupMesh()

Function: CreateAverageFoldedGroupMesh()
Description: This convenience function can only be called after RunCBA() (with internally stored set of .SRF, .SSM, .SMP file names): create folded group average cortex using created alignment (SSM) files. The function also saves a “.sal” file that can be used in the Cortex-Based Alignment dialog; the function only works if names of folded SPH meshes can be derived from curvature names by replacing trailing substring with `_RECOSM_SPH.srf`. To be used in BrainVoyager QX 2.8 or higher. *Returns:* Success or no success (boolean).

3.13.8 Mesh time course (MTC) creation, preprocessing and statistics

LinkMTC()

Function: LinkMTC()
Description: Link a mesh time series file to a surface file (*.srf). This is a function of the *.vmr object.
Parameter 1: Name of the mesh time course file (*.mtc) including a path.

CreateMTCFromVTC()

Function: CreateMTCFromVTC()
Description: Create a mesh time series to be displayed on a surface file (*.srf). This is a function of the *.vmr object. First open an anatomical file (*.vmr) and link functional normalised data (*.vtc) and a surface file (*.srf).
Parameter 1: Sampling depth inside white matter, for example -1.0.
Parameter 2: Sampling depth inside gray matter, for example 3.
Parameter 3: New name for the mesh time course file, for example “mesh.mtc”.

mesh.CreateMTCFromVTC()

Function: mesh.CreateMTCFromVTC()
Description: Project the functional data (*.vtc) on the surface mesh (*.srf). It is assumed that a *.vmr with a linked *.vtc as well as a mesh is available (e.g. loaded). To be used in BrainVoyager QX 2.8 or higher. See script “MeshMTCFromVTC.js”.
Parameter 1: Sampling VTC along mesh vertex normals (0 = at vertex) from (for example -1.0).
Parameter 2: Sampling VTC along mesh vertex normals (0 = at vertex) to (for example 2.0).
Parameter 3: Name for the new *.mtc.
Returns: Success or no success (boolean).

mesh.LinkMTC()

Function: mesh.LinkMTC()
Description: Load a mesh time course file (*.mtc). To be used in BrainVoyager QX 2.8 or higher. See script "MeshMTCSingleStudyGLM.js".
Parameter 1: Name of *.mtc file.
Returns: Success or no success (boolean).

mesh.getNrOfMTCTimePoints()

Function: mesh.getNrOfMTCTimePoints()
Description: Get number of volumes in mesh time course file (*.mtc).
Returns: Number of volumes/time points.

mesh.SpatialSmoothing()

Function: mesh.SpatialSmoothing()
Description: This function applies nearest neighbor interpolation to the MTC file (approximating Gaussian; see figure 3.13.8). See script "MeshMTCPreprocessing.js".
Parameter 1: Repetition value (number).

mesh.LinearTrendRemoval()

Function: mesh.LinearTrendRemoval()
Description: This function applies linear trend removal to the functional data (*.mtc) on the mesh. See script "MeshMTCPreprocessing.js".

mesh.TemporalHighPassFilterFFT()

Function: mesh.TemporalHighPassFilterFFT()
Description: Apply a temporal high pass filter to the functional data (*.mtc). See script "MeshMTCPreprocessing.js".
Parameter 1: Cut-off value for the frequency-space filter in cycles (number; default: 3).

mesh.TemporalGaussianSmoothing()

Function: mesh.TemporalGaussianSmoothing()
Description: Temporal low-pass filter to be applied to the functional data (*.mtc) on the mesh. See script "MeshMTCPreprocessing.js".
Parameter 1: Size of filter at FWHM
Parameter 2: Filter unit, "Seconds" (string)

mesh.SaveMTC()

Function: mesh.SaveMTC()
Description: Save the *.mtc after preprocessing. To be used in BrainVoyager QX 2.8 or higher. See script "MeshMTCPreprocessing.js". *Parameter 1:* Name for *.mtc file.

mesh.ClearDesignMatrix()

Function: mesh.ClearDesignMatrix()
Description: Load a anatomical file (*.vmr), mesh (*.srf) and link functional data (*.mtc). Then it is time to remove any remaining design matrices via this function. To be used in BrainVoyager QX 2.8 or higher. See script "MeshMTCSingleStudyGLM.js".

mesh.LoadSingleStudyGLMDesignMatrix()

Function: mesh.LoadSingleStudyGLMDesignMatrix()
Description: Function to load a single-subject or run design matrix (*.sdm). To be used in BrainVoyager QX 2.8 or higher.
Parameter 1: Name of design matrix file (*.sdm).
Returns: Success or no success (boolean).

mesh.ComputeSingleStudyGLM()

Function: mesh.ComputeSingleStudyGLM()
Description: Load an anatomical file (*.vmr), a mesh (*.srf), obtain a mesh object, link functional data (*.mtc) and load a design matrix file (*.sdm), then this function can be used. To be used in BrainVoyager QX 2.8 or higher. See script "MeshMTCSingleStudyGLM.js".

mesh.ShowGLM()

Function: mesh.ShowGLM()
Description: After running a single study GLM, make the results visible by using this function. To be used in BrainVoyager QX 2.8 or higher.

mesh.LoadGLM()

Function: mesh.LoadGLM()
Description: Load a single study GLM on a mesh. *Parameter 1:* Name of general linear model file (*.glm).
Returns: Success (boolean)

mesh.SaveGLM()

Function: mesh.SaveGLM()
Description: Save a single study GLM. *Parameter 1:* Name for general linear model file (*.glm).

mesh.ClearMultiStudyGLMDefinition()

Function: mesh.ClearMultiStudyGLMDefinition()
Description: Clear before calculating a new multi study GLM.

mesh.AddStudyAndDesignMatrixAndCortexMapping()

Function: mesh.AddStudyAndDesignMatrixAndCortexMapping()
Description: Function for creating a multi study GLM design
Parameter 1: Name of functional data (*.mtc).
Parameter 2: Name of design matrix file (*.sdm).
Parameter 3: Name transformation file (*.ssm).

mesh.SaveMultiStudyGLMDefinitionFile()

Function: mesh.SaveMultiStudyGLMDefinitionFile()
Description: Save a multi study GLM.
Parameter 1: Name for multi study design file (*.mdm).

mesh.LoadMultiStudyGLMDefinitionFile()

Function: mesh.LoadMultiStudyGLMDefinitionFile()
Description: Load a multi study design file (*.mdm). *Parameter 1:* Name of multi study design file (*.mdm).

mesh.getCorrectForSerialCorrelations()

Function: mesh.getCorrectForSerialCorrelations()
Description: Find out the setting for serial correlation correction.
Returns: Level of correction (0 = no correction, 1 = AR(1), 2 = AR(2)).

mesh.setCorrectForSerialCorrelations()

Function: mesh.setCorrectForSerialCorrelations()
Description: Set the level of serial correlation correction.
Parameter 1: Level of correction (0 = no correction, 1 = AR(1), 2 = AR(2)).

mesh.ComputeMultiStudyGLM()

Function: mesh.ComputeMultiStudyGLM()
Description: Compute a multi study general linear model.

mesh.ComputeRFXGLM()

Function: mesh.ComputeRFXGLM()
Description: Compute a random effects multi study general linear model.

mesh.SaveGLM()

Function: mesh.SaveGLM()
Description: Save the GLM that was calculated via `mesh.ComputeSingleStudyGLM()`. To be used in BrainVoyager QX 2.8 or higher. See script "MeshMTCSingleStudyGLM.js".
Parameter 1: Name for general linear model (*.glm) file.

mesh.CreateSurfaceMapFromVolumeMap()

Function: mesh.CreateSurfaceMapFromVolumeMap()
Description: Create a surface map (*.smp) from a volume map (*.vmp). Requires an open anatomical file (*.vmr), surface file (*.srf) and volume map (*.vmp).
Parameter 1: Interpolation method (integer)
Parameter 2: Sample only non zero values (boolean)

3.13.9 Surface maps (SMP)

mesh.CreateSurfaceMapFromVolumeMapDepth()

Function: mesh.CreateSurfaceMapFromVolumeMapDepth()
Description: Create a depth surface map (*.smp) from a cortical thickness map (*.vmp). Requires an open anatomical file (*.vmr), surface file (*.srf) and volume map (*.vmp).
Parameter 1: Interpolation method (integer)
Parameter 2: Sample only non zero values (boolean)
Parameter 3: Sample maximum values (boolean)
Parameter 4: Depth start, relative to mesh vertex positions in mm (float)
Parameter 5: Depth end, relative to mesh vertex positions in mm (float)
Parameter 6: Step size (float)

mesh.LoadSurfaceMaps()

Function: mesh.LoadSurfaceMaps()
Description: Load a surface map (*.smp) on a surface file (*.srf). Requires open anatomical file (*.vmr) and surface mesh (*.srf).
Parameter 1: Name of the surface map including the path.

mesh.ShowSurfaceMap()

Function: mesh.ShowSurfaceMap()
Description: Show the surface map (*.smp) that has been loaded on a surface file (*.srf).
Parameter 1: Map index (starting at 1)

mesh.SmoothMap()

Function: mesh.SmoothMap()
Description: New version of SmoothCurrentMap(); this new version allows to specify map index, e.g. to enumerate and smooth all maps. Tthis version also allows to restrict vertices (neighbors) used for smoothing. Requires an open anatomical file (*.vmr), surface file (*.srf) and volume map (*.vmp).
Parameter 1: index of map to be smoothed (in range: 0 to mesh.NrOfSurfaceMaps - 1)
Parameter 2: number of smooth cycles using values of nearest neighbors
Parameter 3: True: restrict smoothing to vertices with values unequal zero (zero considered as ”missing value”)
Parameter 4: True: restrict inclusion of neighbors to those with values unequal zero
Parameter 5: True: restrict smoothing to vertices with values larger than the map threshold (generalized version of param 3)

mesh.SmoothMapLags()

Function: mesh.SmoothMapLags()
Description: note that circular angle maps are smoothed circular automatically Requires an open anatomical file (*.vmr), surface file (*.srf) and volume map (*.vmp).
Parameter 1: Index of map to be smoothed (in range: 0 to mesh.NrOfSurfaceMaps - 1)
Parameter 2: Number of smooth cycles using values of nearest neighbors
Parameter 3: True: smooth lags circular, False: smooth by simple averaging
Parameter 4: True: restrict smoothing to vertices with values unequal zero (zero considered as ”missing value”)
Parameter 5: True: restrict inclusion of neighbors to those with values unequal zero
Parameter 6: True: restrict smoothing to vertices with values larger than the map threshold (generalized version of param 3)

mesh.SaveSurfaceMaps()

Function: mesh.SaveSurfaceMaps()
Description: Save current maps. To be used in BrainVoyager 20.0 or higher.
Parameter 1: Name for surface map (*.smp) file.

mesh.GetNameOfSurfaceMap()

Function: mesh.GetNameOfSurfaceMap()
Description: Get name of surface map. To be used in BrainVoyager 20.0 or higher.
Parameter 1: Index of required surface map (index starting at 1).
Returns: Name of surface map

mesh.SaveSingleStudyGLMDesignMatrix()

Function: mesh.SaveSingleStudyGLMDesignMatrix()
Description: Save design matrix specified on surface.
Parameter 1: Name for design matrix (*.sdm) file.

3.13.10 Description of properties

MeshScene

Property: MeshScene
Description: VMR property. Obtain mesh scene object from the VMR object.
Use: `var meshScene = docVMR.MeshScene;`. Since BrainVoyager 2.8.

meshScene.CurrentMesh

Property: meshScene.CurrentMesh
Description: meshScene property. Obtain mesh object from the mesh scene object.
Since BrainVoyager 2.8.

meshScene.ViewpointPositionX

Property: meshScene.ViewpointPositionX
Description: Set the x-coordinate for the camera. Since BrainVoyager 2.8. For background information on viewing transformations, see section "Viewing transformations" in the OpenGL guide (an older version is available online at:
http://www.glprogramming.com/red/chapter03.htmlname2.

meshScene.ViewpointPositionY

Property: meshScene.ViewpointPositionY
Description: Set the y-coordinate for the camera. Since BrainVoyager 2.8.

meshScene.ViewpointPositionZ

Property: meshScene.ViewpointPositionZ
Description: Set the z-coordinate for the camera. Since BrainVoyager 2.8.

meshScene.ViewpointRotationX

Property: meshScene.ViewpointRotationX
Description: Rotates the camera on the x-axis with provided number of degrees. Since BrainVoyager 2.8.

meshScene.ViewpointRotationY

Property: meshScene.ViewpointRotationY
Description: Rotates the camera on the y-axis with provided number of degrees. Since BrainVoyager 2.8.

meshScene.ViewpointRotationZ

Property: meshScene.ViewpointRotationZ
Description: Rotates the camera on the z-axis with provided number of degrees. Since BrainVoyager 2.8.

meshScene.SphereResolutionCBA

Property: meshScene.SphereResolutionCBA
Get or set the resolution of the sphere for cortex-based alignment.
1: standard resolution (default)
2: low resolution
3: high resolution

mesh.FileName

Property: mesh.FileName
Description: Obtain the name of the surface file (*.srf).

mesh.NrOfVertices

Property: mesh.NrOfVertices
Description: Obtain the number of vertices of the surface file (*.srf).

mesh.MorphingUpdateInterval

Property: mesh.MorphingUpdateInterval
Description: This sets or gives the number of steps after which the OpenGL window is updated during the morphing process. To be set before `InflateMeshToSphere()` and `CorrectInflatedSphereMesh()`. A default value is 50.

mesh.CorrectForSerialCorrelations

Property: mesh.CorrectForSerialCorrelations
Description: This sets the level of correction for serial correlations in the noise. For AR(1), set to 1, for AR(2), use 2.

mesh.FileNameOfPreprocessdMTC

Property: mesh.FileNameOfPreprocessdMTC
Description: Obtain file name of the functional data (*.mtc) on the mesh after the most recent preprocessing step.

meshScene.SphereResolutionCBA

Property: meshScene.SphereResolutionCBA
Description: Get or set the resolution of the sphere, 1: standard resolution (default), 2: low resolution, 3: high resolution

3.13.11 Example script: visualization

To use the code, select the text below and save with the JavaScript extension ".js". Then, load in the Script Editor and click"Run".

```
/*  SurfaceFunctions_BV22.js
Script to visualize surface files
For BrainVoyager QX 2.1, adapted for BrainVoyager 21 11-04-2019, adapted for BrainVoyager 22 Sept 2021
*/

/* This information is used in the functions */
var ObjectsRawDataPath = BrainVoyager.BrowseDirectory("Please select the BrainVoyager sample experiment directory");
// "/Users/BVuser/Data/ObjectsDicomGSG/";
var nameVMR = BrainVoyager.BrowseFile("Please select the anatomical file", "*.vmr");
// ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_CLEAN_TAL.vmr";
var namesrf = BrainVoyager.BrowseFile("Please select the surface file", "*.srf");

/* The non-commented code will be executed when clicking 'Run' */
LoadVMRandSRF(nameVMR, namesrf);
SurfaceViewpoints(nameVMR, namesrf);
// Merge();


function LoadVMRandSRF(nameVMR, namesrf)
{
var docVMR = BrainVoyager.OpenDocument(nameVMR);
docVMR.LoadMesh(namesrf);// ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_CLEAN_TAL_LH_RECOSM.srf");
var meshscene = docVMR.CreateMeshScene();
var mesh = meshscene.CurrentMesh;
BrainVoyager.PrintToLog("Filename: " + mesh.FileName);
docVMR.AddMesh(ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_CLEAN_TAL_RH_RECOSM.srf");
// mesh.SaveAs(text);
// docVMR.Close();
}

function SurfaceViewpoints(nameVMR, namesrf) {

var docVMR = BrainVoyager.OpenDocument(nameVMR);
var meshscene = docVMR.CreateMeshScene();
meshscene.LoadMesh(namesrf); // ObjectsRawDataPath +  "CG_3DT1MPR_SCRIPT_CLEAN_TAL_RH_RECOSM.srf");
var mesh = meshscene.CurrentMesh;
BrainVoyager.ShowLogTab();
meshscene.SaveSnapshotOf3DViewer(ObjectsRawDataPath + "/" + "Surface.png");
BrainVoyager.PrintToLog("Changing viewpoints...");

var i;
var origX = meshscene.ViewpointTranslationX;
var origY = meshscene.ViewpointTranslationY;
var origZ = meshscene.ViewpointTranslationZ;

var TrX = meshscene.ViewpointTranslationX;//docVMR.ViewpointTranslationX;
BrainVoyager.PrintToLog("Translate X...");
for(i=1; i<10; i++) {
meshscene.ViewpointTranslationX = TrX + 6*i;
mesh.UpdateAppearance();
}
var ret = BrainVoyager.TimeOutMessageBox( "Translated X...", 3);
meshscene.SaveSnapshotOf3DViewer(ObjectsRawDataPath + "/" + "Surface_TranslatedX.png");
meshscene.ViewpointTranslationX = origX;

BrainVoyager.PrintToLog("Rotate X...");
origX = meshscene.ViewpointRotationX;
var RotX = meshscene.ViewpointRotationX;// docVMR.ViewpointRotationX;
var i;
for(i=1; i<10; i++) {
meshscene.ViewpointRotationX = RotX + 6*i;
mesh.UpdateAppearance();//replaced docVMR.UpdateSurfaceWindow();
}
meshscene.SaveSnapshotOf3DViewer(ObjectsRawDataPath + "/" + "Surface_RotatedX.png");
var ret = BrainVoyager.TimeOutMessageBox( "Rotated X...", 3);
meshscene.ViewpointRotationX = origX;

var TrY = meshscene.ViewpointTranslationY;
BrainVoyager.PrintToLog("Translate Y...");
for(i=1; i<10; i++) {
meshscene.ViewpointTranslationY = TrY + 6*i;
mesh.UpdateAppearance();
}
meshscene.SaveSnapshotOf3DViewer(ObjectsRawDataPath + "/" + "Surface_TranslatedY.png");
var ret = BrainVoyager.TimeOutMessageBox( "Translated Y...", 3);
meshscene.ViewpointTranslationY = origY;

BrainVoyager.PrintToLog("Rotate Y...");
var origY = meshscene.ViewpointRotationY;
var RotY = meshscene.ViewpointRotationY;
```

```
for(i=1; i<10; i++) {
meshscene.ViewpointRotationY = RotY + 6*i;
mesh.UpdateAppearance();
}
meshscene.SaveSnapshotOf3DViewer(ObjectsRawDataPath + "/" + "Surface_RotatedY.png");
var ret = BrainVoyager.TimeOutMessageBox( "Rotated Y...", 3);
meshscene.ViewpointRotationY = origY;

var TrZ = meshscene.ViewpointTranslationZ;
BrainVoyager.PrintToLog("Translate Z...");
for(i=1; i<10; i++) {
meshscene.ViewpointTranslationZ = TrZ + 6*i;
mesh.UpdateAppearance();
}

meshscene.SaveSnapshotOf3DViewer(ObjectsRawDataPath + "/" + "Surface_TranslatedZ.png");
var ret = BrainVoyager.TimeOutMessageBox( "Translated Z...", 3);
meshscene.ViewpointTranslationZ = origZ;

BrainVoyager.PrintToLog("Rotate Z...");
var origZ = meshscene.ViewpointRotationZ;
var RotZ = meshscene.ViewpointRotationZ;
for(i=1; i<10; i++) {
meshscene.ViewpointRotationZ = RotZ + 6*i;
mesh.UpdateAppearance();
}
meshscene.SaveSnapshotOf3DViewer(ObjectsRawDataPath + "/" + "Surface_RotatedZ.png");
var ret = BrainVoyager.TimeOutMessageBox( "Rotated Z...", 3);
meshscene.ViewpointRotationZ = origZ;

BrainVoyager.PrintToLog("Finished.");
}

function Merge() {
// assume two meshes in scene
var doc = BrainVoyager.ActiveDocument;
var meshscene = doc.CreateMeshScene();
var txt = meshscene.MergeMeshesInScene();
BrainVoyager.PrintToLog("Name of merged meshes: " + txt);
// var mesh = meshscene.CurrentMesh;
// mesh.SaveAs(text);
}
```

3.13.12 Example script: create MTC (via VMR)

To use the code, select the text below and save with the JavaScript extension ".js". Then, load in the Script Editor and click"Run". Please note that in BrainVoyager QX 2.8, one can use the mesh object to create an MTC (see functions below).

```
var path = "C:/Data/bvqxdata/CBA_RFX_June2009/subj_AA/";
var docVMR = BrainVoyager.OpenDocument(path + "AA_TAL.vmr");
docVMR.LinkVTC(path + "AA_Localizer.vtc");
docVMR.LoadMesh(path + "AA_TAL_LH_RECOSM.srf");
docVMR.CreateMTCFromVTC(-1.0, 3.0, "test2.mtc" );
```

3.13.13 Example script: create MTC (via mesh object)

To use the code, select the text below and save with the JavaScript extension "MeshMTCFromVTC.js". Then, load in the Script Editor and click"Run". For use in QX 2.8 or higher.

```
// This script shows how MTC files can be created for a given mesh based on a VMR-VTC file
// it is assumed that a VMR with a linked VTC as well as a mesh is available (e.g. loaded)
//
//  Created by Rainer Goebel, May 2013; checked for BV21 June 2019, BV22 Sept 2021

function ProjectVTCDataOnMesh()
{
    var ok;
    var docVMR = BrainVoyager.ActiveDocument;

    if(docVMR == undefined) return;

    var mesh = docVMR.CurrentMesh;
    if(mesh == undefined) return;

    ok = mesh.CreateMTCFromVTC(-1.0, 2.0, "MeshTimeCourseData.mtc" );
    if(!ok)
        BrainVoyager.PrintToLog("Could not create MTC data from VTC data.");
}

ProjectVTCDataOnMesh();
```

3.13.14 Example script: Cortex-Based Alignment (via mesh scene object)

To use the code, select the text below and save as "MeshCBA.js". Then, load in the Script Editor and click"Run" or place in /Documents/BVExtensions/Scripts/, start BrainVoyager, and select the script from the Script Menu in order to run it. For use in QX 2.8 or higher.

```
// This script shows how to prepare and run (rigid) CBA including group mesh/curvature map visualization.
//
//  Created by Rainer Goebel, May 2013, updated for BV21 June 2019, updated Sept 2021

var CBAfiledir = BrainVoyager.BrowseDirectory("Please select the directory with CBA files");
var docSRFname = BrainVoyager.BrowseFile("Please select a SRF", "*RECOSM.srf");

CreateSphereFromFoldedMesh();
MapSphereGetCurvature();
//RunRigidCBAStep();
//RunFullCBA();


function CreateSphereFromFoldedMesh()
{
var ok;
var docVMRname = BrainVoyager.BrowseFile("Please select a VMR", "*.vmr");
var docVMR = BrainVoyager.OpenDocument(docVMRname);//ActiveDocument;
if (docVMR == undefined) return;
// It is assumed that a folded "*_RECOSM.srf" mesh is available (e.g. loaded)
// var docSRFname = BrainVoyager.BrowseFile("Please select a SRF", "*RECOSM.srf");
     mesh = docVMR.LoadMesh(docSRFname);//
var mesh = docVMR.CurrentMesh;
     if (mesh == undefined) BrainVoyager.PrintToLog("Mesh is not available");
mesh.MorphingUpdateInterval = 50;
BrainVoyager.PrintToLog("Calculate curvature for CBA");
mesh.CalculateCurvatureCBA();
mesh.SaveSurfaceMaps("Curvature.smp");
//BrainVoyager.PrintToLog("Success: " + (ret ? "yes" : "no"));
BrainVoyager.PrintToLog("Name of map: " + mesh.GetNameOfSurfaceMap(1));
BrainVoyager.PrintToLog("Smooth current map...");
mesh.SmoothCurrentMap(5);
BrainVoyager.PrintToLog("Update appearance... ");
    mesh.UpdateAppearance();// new
BrainVoyager.PrintToLog("Inflate mesh to sphere...");
mesh.InflateGeometryToSphere(800);// new
BrainVoyager.PrintToLog("Correct inflated sphere mesh, steps: 3000");
ok = mesh.CorrectInflatedSphereDistortions(3000); // new
BrainVoyager.PrintToLog("mesh.FileName: " + mesh.FileName);
var newName = makeNameWithInfix(mesh.FileName, "_SPHERE");
BrainVoyager.PrintToLog(newName);
mesh.SaveAs(newName);
BrainVoyager.PrintToLog("Saving " + newName + ".");
}

function MapSphereGetCurvature()
{
var ok;
var docVMR = BrainVoyager.ActiveDocument; if(docVMR == undefined) return;
var meshScene = docVMR.MeshScene; if(meshScene == undefined) return;
var mesh = meshScene.CurrentMesh; if(mesh == undefined) return;

// we assume created sphere mesh (see "CreateSphereFromFoldedMesh" above) is available as current mesh
// (created or loaded) AND available on disk (since it is reloaded)
var sphere_mesh_name =  mesh.FileName;
// we need name of folded original mesh below - here we assume that name is same as SPHERE mesh
// but instead of "SPHERE" has "RECOSM" at end of name
var folded_mesh_name = makeNameWithInfix(sphere_mesh_name, "_RECOSM");

meshScene.SphereResolutionCBA = 1; // 1 -> standard resolution (default), 2 -> low resolution, 3 -> high resolution
var saved_ssm_file          = meshScene.MapSphereMeshFromStandardSphere();
var saved_foldedSPH_file = meshScene.SetStandardSphereToFoldedMesh(folded_mesh_name);
mesh = meshScene.CurrentMesh;
BrainVoyager.PrintToLog("Current mesh: " + mesh.FileName);
var saved_curvature_file  = mesh.CreateMultiScaleCurvatureMap(2, 7, 20, 40);
// the obtained curvature file (automatically saved) is needed for CBA - store in array, one for each subject
// (per hemisphere)
BrainVoyager.PrintToLog("Curvature file: " + saved_curvature_file);
}

function RunRigidCBAStep()
{
var ok;
//var docVMR = BrainVoyager.ActiveDocument;
var docVMRname = BrainVoyager.BrowseFile("Please select a VMR", "*.vmr");

BrainVoyager.PrintToLog("Open VMR: " + docVMRname);
var docVMR = BrainVoyager.OpenDocument(docVMRname);
if (docVMR == undefined){
BrainVoyager.PrintToLog("Could not find VMR");
```

```
return;
}
BrainVoyager.PrintToLog("Get mesh scene...");
var meshScene = docVMR.CreateMeshScene();// before BV21: GetMeshScene(); //docVMR.MeshScene;
if (meshScene == undefined) {
BrainVoyager.PrintToLog("Could not get mesh scene");
return;
}
// the mesh for rigid alignment must be a sphere (SPH file)
// to ensure that we have a sphere mesh, we can create one
BrainVoyager.PrintToLog("Create standard sphere mesh... (SPH)");
meshScene.CreateStandardSphereMesh();
var mesh = meshScene.CurrentMesh;

BrainVoyager.PrintToLog("Clear group CBA curvature files....");
meshScene.ClearGroupCBACurvatureFiles();
var NSubjects = 5;
var CurvatureFile, TargetCurvatureFile;
for(i=0; i<NSubjects; i++) // loop over subjects (per hemisphere)
{
// for demonstration, we simply hard-code files here - better to use/prepare array
/* if(i == 0) CurvatureFile = BrainVoyager.PathToSampleData + "CortexBasedAlignment/AA/AA_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if(i == 1) CurvatureFile = BrainVoyager.PathToSampleData + "CortexBasedAlignment/MR/MR_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if(i == 2) CurvatureFile = BrainVoyager.PathToSampleData + "CortexBasedAlignment/PD/PD_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if(i == 3) CurvatureFile = BrainVoyager.PathToSampleData + "CortexBasedAlignment/RM/RM_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if(i == 4) CurvatureFile = BrainVoyager.PathToSampleData + "CortexBasedAlignment/SG/SG_TAL_LH_RECOSM_SPH_CURVATURE.smp";
*/
if(i == 0) CurvatureFile = CBAfiledir + "/sub_AA/AA_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if(i == 1) CurvatureFile = CBAfiledir + "/sub_MR/MR_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if(i == 2) CurvatureFile = CBAfiledir + "/sub_PD/PD_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if(i == 3) CurvatureFile = CBAfiledir + "/sub_RM/RM_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if(i == 4) CurvatureFile = CBAfiledir + "/sub_SG/SG_TAL_LH_RECOSM_SPH_CURVATURE.smp";

BrainVoyager.PrintToLog("Adding curvature files, iteration " + toString(i+1) + ", curvature file: " + CurvatureFile);
ok = meshScene.AddCurvatureFileForGroupCBA(CurvatureFile);
if(!ok) // provided file not found
return;

if(i == 0) TargetCurvatureFile = CurvatureFile; // we here use first case for rigid alignment target (any should be fine)
}

BrainVoyager.PrintToLog("Create spherical coordinates map from curvature file (SMP)...");
var SphericalTargetCurvatureFile = mesh.CreateSphericalCoordinatesMapFromSMP(TargetCurvatureFile);

BrainVoyager.PrintToLog("Spherical target curvature file: " + SphericalTargetCurvatureFile + "...");
if (SphericalTargetCurvatureFile == "") {// we get empty string if it did not work
BrainVoyager.PrintToLog("Rigid CBA did not work with spherical target curvature file " + SphericalTargetCurvatureFile);
return;
}

BrainVoyager.PrintToLog("Run rigid cortex based alignment...");
ok = meshScene.RunRigidCBA(SphericalTargetCurvatureFile);
// the result of rigid alignment is automatically saved in a ".rga" file
// that will be used by subsequent CBA when using same input curvature SMP files
}

function RunFullCBA()
{
var ok;
BrainVoyager.PrintToLog("Run full cortex based alignment...");
var docVMR = BrainVoyager.ActiveDocument; if(docVMR == undefined) return;
var meshScene = docVMR.CreateMeshScene(); if(meshScene == undefined) return; // creates surface window if not present
meshScene.ClearGroupCBACurvatureFiles();
// if this is run after rigid-CBA, we would not strictly fill curvature SMP list again
var NSubjects = 5;
var CurvatureFile;
for(i=0; i<NSubjects; i++) // loop over subjects (per hemisphere)
{
// for demonstration, we simply hard-code files here - better to use/prepare array
/* if(i == 0) CurvatureFile = BrainVoyager.PathToSampleData + "CortexBasedAlignment/AA/AA_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if(i == 1) CurvatureFile = BrainVoyager.PathToSampleData + "CortexBasedAlignment/MR/MR_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if(i == 2) CurvatureFile = BrainVoyager.PathToSampleData + "CortexBasedAlignment/PD/PD_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if(i == 3) CurvatureFile = BrainVoyager.PathToSampleData + "CortexBasedAlignment/RM/RM_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if(i == 4) CurvatureFile = BrainVoyager.PathToSampleData + "CortexBasedAlignment/SG/SG_TAL_LH_RECOSM_SPH_CURVATURE.smp";
*/
if (i == 0) CurvatureFile = CBAfiledir + "/sub_AA/AA_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if (i == 1) CurvatureFile = CBAfiledir + "/sub_MR/MR_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if (i == 2) CurvatureFile = CBAfiledir + "/sub_PD/PD_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if (i == 3) CurvatureFile = CBAfiledir + "/sub_RM/RM_TAL_LH_RECOSM_SPH_CURVATURE.smp";
if (i == 4) CurvatureFile = CBAfiledir + "/sub_SG/SG_TAL_LH_RECOSM_SPH_CURVATURE.smp";
ok = meshScene.AddCurvatureFileForGroupCBA(CurvatureFile);
if (!ok) {// provided file not found
BrainVoyager.PrintToLog("File " + CurvatureFile + " not found");
return;
}
}
ok = meshScene.RunCBA();
```

```
if (!ok) {
BrainVoyager.PrintToLog("Cortex based alignment did not succeed.");
return;
}
// these convenience functions can only be called after CBA (with internally stored set of .SRF, .SSM, .SMP file names):
// create group curvature map indicating quality of alignment, saves also ".cal" file that can be used in CBA dialog
ok = meshScene.CreateAverageCurvatureGroupMap();
// create folded group average cortex using created alignment (SSM) files, saves also ".sal" file that can be used
// in CBA dialog; the function only works if names of folded SPH meshes can be derived from curvature names
// by replacing trailing substring with "_RECOSM_SPH.srf"
ok = meshScene.CreateAverageFoldedGroupMesh();
}

function makeNameWithInfix(name, infix) {
var fi = new QFileInfo(name);
newname = getNewName(name, infix, "." + fi.completeSuffix());
return newname;
}

function getNewName(name, infix, suffix) {
var fi = new QFileInfo(name);
var newname = fi.path() + "/" + fi.baseName() + infix + suffix;
return newname;
}
```

3.13.15 Example script: loading a mesh (via mesh scene object)

To use the code, select the text below and save as "MeshLoading.js". Then, load in the Script Editor and click"Run" or place in /Documents/BVExtensions/Scripts/, start BrainVoyager, and select the script from the Script Menu in order to run it.

```
//
//  Created by Rainer Goebel, May 2013, tested for BV21 June 2019, updated for BV22 Sept 2021
//

//var ObjectsRawDataPath = BrainVoyager.BrowseDirectory("Please select the data directory");
//BrainVoyager.PathToSampleData + "ObjectsDicomGSG/";
//var meshfilename = ObjectsRawDataPath + "/CG_Head.srf";
var meshfilename = BrainVoyager.BrowseFile("Please select the mesh file", "*.srf");
LoadMeshFromMeshScene(meshfilename);
LoadMeshFromVMRDoc(meshfilename);

function LoadMeshFromMeshScene(meshfilename)
{
    var ok;
    var docVMR = BrainVoyager.ActiveDocument; if (docVMR == undefined)  return;
    var meshScene = docVMR.CreateMeshScene(); // before BV20: docVMR.CurrentMeshScene;
    if (meshScene == undefined) return;

    ok = meshScene.LoadMesh(meshfilename); if (!ok) return;
    var mesh = meshScene.CurrentMesh;

    var n_vertices = mesh.NrOfVertices;
    BrainVoyager.PrintToLog("No. of vertices of current mesh: " + n_vertices);
    BrainVoyager.PrintToLog("Viewpoint position, x: " + meshScene.ViewpointPositionX + " y: " +
                            meshScene.ViewpointPositionY + " z: " + meshScene.ViewpointPositionZ);
    meshScene.ViewpointPositionX = -500;
    meshScene.ViewpointRotationX = 0;
    meshScene.ViewpointRotationY = 180;
    meshScene.ViewpointRotationZ = 0;
    for (var i=0; i<90; i++) {
        meshScene.ViewpointPositionX += 2;
        meshScene.ViewpointRotationZ -= 1;
        mesh.UpdateAppearance();
    }
    BrainVoyager.PrintToLog("Viewpoint position, x: " + meshScene.ViewpointPositionX + " y: " +
                            meshScene.ViewpointPositionY + " z: " + meshScene.ViewpointPositionZ);
}

function LoadMeshFromVMRDoc(meshfilename)
{
    var ok;
    var docVMR = BrainVoyager.ActiveDocument;
    if (docVMR == undefined)
        return;

    ok = docVMR.LoadMesh(meshfilename); // LoadMesh(ObjectsRawDataPath + "/CG_Head.srf")
    if (!ok) return;
    var mesh = docVMR.CurrentMesh;

    var n_vertices = mesh.NrOfVertices;
    BrainVoyager.PrintToLog("No. of vertices of current mesh: " + n_vertices);
}
```

3.13.16 Example script: smoothing and inflating a mesh (via mesh object)

To use the code, select the text below and save as "MeshMorphing.js". Then, load in the Script Editor and click"Run" or place in /Documents/BVExtensions/Scripts/, start BrainVoyager, and select the script from the Script Menu in order to run it. For use in QX 2.8 or higher.

```
// MeshMorphing_BV22.js
//
// This script shows how a mesh can be smoothed and inflated.
// It is assumed that a "*_RECO.srf" mesh is available (e.g. loaded)
//
//  Created by Rainer Goebel, May 2013; checked for BV21 June 2019, updated Sept 2021
//

MorphMesh();

function MorphMesh()
{
    var ok;
    var docVMR = BrainVoyager.ActiveDocument;
    if (docVMR == undefined) return;
    var mesh = docVMR.CurrentMesh;
  //var mesh = docVMR.getCurrentMesh();
    if (mesh == undefined) return;

    mesh.MorphingUpdateInterval = 25;
    //mesh.SmoothMesh(30, 0.07); // standard smoothing - mesh will shrink
    ok = mesh.SmoothGeometry(50, 0.07);
//    mesh.SmoothRecoMesh(50, 0.07); // special "high-frequency" smoothing
    //removing jags of reconstructed voxel borders without shrinking mesh
    mesh.CalculateCurvature();
    mesh.SmoothCurrentMap(5);
    mesh.MeshScene.Update3DViewer();//UpdateSurfaceWindow();
    var new_name = makeNameWithInfix(mesh.FileName, "_SM");
    mesh.SaveAs(new_name);
    ok = mesh.InflateGeometry(500, 0.8, "");// if "" used for 3rd param (area reference mesh),
    // the current mesh at the moment the fn is called is
    // used to calculate ref mesh area
    ok =  mesh.InflateGeometryToSphere(800);
    ok = mesh.CorrectInflatedSphereDistortions(500);
    BrainVoyager.PrintToLog("Successfully distortion corrected: " + (ok ? "yes" : "no"));
}

function makeNameWithInfix(name, infix) {
var fi = new QFileInfo(name);
    newname = getNewName(name, infix, "." + fi.completeSuffix());
    return newname;
}

function getNewName(name, infix, suffix) {
    var fi = new QFileInfo(name);
    var newname = fi.path() + "/" + fi.baseName() + infix + suffix;
    return newname;
}
```

3.13.17 Example script: preprocessing an *.mtc file (via mesh object)

To use the code, select the text below and save as "MeshMTCPreprocessing.js". Then, load in the Script Editor and click "Run" or place in /Documents/BVExtensions/Scripts/, start BrainVoyager, and select the script from the Script Menu in order to run it. For use in QX 2.8 or higher.

```
//
// Example script showing how to run statistics with the new "mesh" object
// Feel free to adapt this script to your needs
//
//  Created by Rainer Goebel, May 2013, updated 20-08-2019
//

//var ObjectsRawDataPath = BrainVoyager.BrowseDirectory("Please select the directory with surface files");
var MTCname = BrainVoyager.BrowseFile("Please select the functional data on the mesh", "*.mtc");
BrainVoyager.PrintToLog("MTC name: " + MTCname);

// when calling this function, we assume a VMR and the desired mesh have been loaded already
// (see also other "Mesh..." scripts)
//
function PreprocessMTC()
{

var docVMR = BrainVoyager.ActiveDocument;
if (docVMR == undefined) {
```

```
BrainVoyager.PrintToLog("Could not obtain access to anatomical file");
return;
}

var mesh = docVMR.CurrentMesh;
if (mesh == undefined) {
BrainVoyager.PrintToLog("Could not obtain access to surface file");
return;
}
var ok = mesh.LinkMTC(MTCname);
if (!ok) {
BrainVoyager.PrintToLog("Could not link functional file " + MTCname);
return;
};

BrainVoyager.PrintToLog("Spatial smoothing...");
mesh.SpatialSmoothing(3);
BrainVoyager.PrintToLog("Linear trend removal...");
mesh.LinearTrendRemoval();
BrainVoyager.PrintToLog("Temporal high pass filter...");
mesh.TemporalHighPassFilterFFT(3);
BrainVoyager.PrintToLog("Temporal low pass filter...");
mesh.TemporalGaussianSmoothing(5, "Seconds");

// var PreprocessedMTCFileName = mesh.FileNameOfPreprocessdMTC;
// mesh.SaveMTC(PreprocessedMTCFileName);
BrainVoyager.PrintToLog("Preprocessed .MTC: " + getNewNameWithInfix(MTCname, "_preprocessed"));
mesh.SaveMTC(getNewNameWithInfix(MTCname, "_preprocessed"));
}

function getNewNameWithInfix(name, infix) {
var fi = new QFileInfo(name);
newname = getNewName(name, infix, "." + fi.suffix());//fi.completeSuffix());
return newname;
}

function getNewName(name, infix, suffix) {
var fi = new QFileInfo(name);
var newname = fi.path() + "/" + fi.baseName() + infix + suffix;
return newname;
}

PreprocessMTC();
```

3.13.18 Example script: single study GLM on mesh data (via mesh object)

To use the code, select the text below and save as "MeshMTCSingleStudyGLM.js". Then, load in the Script Editor and click"Run" or place in /Documents/BVExtensions/Scripts/, start BrainVoyager, and select the script from the Script Menu in order to run it. For use in QX 2.8 or higher.

```
//
// Example script showing how to run statistics with the new "mesh" object
// Feel free to adapt this script to your needs
//
//  Created by Rainer Goebel, May 2013; updated to Getting Started Guide 4.0 data, Sept 2021
//

var FacesHousesDataPath = "/Users/BVuser/sub-01/ses-04/";

function RunMeshGLM()
{
var ok;

var docVMR = BrainVoyager.OpenDocument(FacesHousesDataPath+"anat/t1w-nd/sub-01_ses-04_acq-nondistorted_T1w_IIHC_MNI.vmr");
if (docVMR == undefined) return;

ok = docVMR.LoadMesh(FacesHousesDataPath +
"anat/t1w-nd/sub-01_ses-04_acq-nondistorted_T1w_IIHC_MNI_WM_LH_RECOSM.srf");
     if (!ok) return;

var mesh = docVMR.CurrentMesh;
if(mesh == undefined) return;
ok = mesh.LinkMTC(FacesHousesDataPath +
"func/blocked_run1/sub-01_ses-04_task-blocked_run-1_bold_SCCTBL_3DMCTS_THPGLMF3c_IIHC_MNI_WM_LH_RECOSM.mtc");
    if (!ok) return;

mesh.ClearDesignMatrix();
ok = mesh.LoadSingleStudyGLMDesignMatrix(FacesHousesDataPath + "func/blocked_run1/FacesHousesDesignMatrix.sdm");
      if (!ok) return;

mesh.CorrectForSerialCorrelations = 2; // 1 -> AR(1), 2 -> AR(2)
mesh.ComputeSingleStudyGLM();
mesh.ShowGLM();
     mesh.SaveGLM(ObjectsRawDataPath + "FacesHouses_MeshGLM_FROMSCRIPT.glm");
```

```
//docVMR.Close();

//ok = mesh.LinkStimulationProtocolToMTC(FacesHousesDataPath + "func/blocked_run1/sub-01_blocked.prt");
if (!ok) return;
}

RunMeshGLM();
```

Script to create a surface map:

```
var vmrname = BrainVoyager.BrowseFile("Please select the anatomy", "*.vmr");
var vmr = BrainVoyager.OpenDocument(vmrname);
var vmpname = BrainVoyager.BrowseFile("Please select the map", "*.vmp");
var success = vmr.LoadVolumeMaps(vmpname);
vmr.ShowVolumeMap(1);
var srfname = BrainVoyager.BrowseFile("Please select the surface", "*.srf");
var meshScene = vmr.GetMeshScene();
var ok = meshScene.LoadMesh(srfname);
var mesh = meshScene.CurrentMesh;
success = mesh.CreateSurfaceMapFromVolumeMap(1,0);
mesh.ShowSurfaceMap(1);
```

Chapter 4

File input and output (I/O)

4.1 Introduction

4.1.1 Using the BrainVoyager object

From BrainVoyager QX 1.9, the name of a file can be obtained using
`BrainVoyager.BrowseFile(''Please select the file'', ''*.*'').`

BrowseFile()

BrowseFile() *Description:* Get the name of a file on the filesystem via a file dialog.
Parameter 1: Instruction for the type of file.
Parameter 2: Filter (string), either a wildcard for all file types ("*.*") or specific file type (for example "*.vmr"). From BrainVoyager 20.4: For selecting one of several file formats, enter the file format extensions in the filter separated by a space:

```
var name = BrainVoyager.BrowseFile("Select file", "*.fmr *.dmr");
```

Previous versions: enter the file formats in the filter separated by a semicolon:

```
var name = BrainVoyager.BrowseFile("Please select the functional file", "*.vtc; *.fmr");.
```

BrowseDirectory()

BrowseDirectory() *Description:* Get the name of a directory on the filesystem via a dialog; the directory name will not contain a file separation character "/" at the end of the string (for example "/Disk/maindir/subdir").
Parameter 1: Instruction (string).

4.1.2 Using Qt objects

In these versions of BrainVoyager QX and BrainVoyager, file access can be obtained via the `QFile` object. After the `QFile` object has been created, the file object should be opened by providing an open mode, which is read, write or append. The open mode can be specified using the `QIODevice` object. Then, the text can be read or written via the `QTextStream` object. After all reading or writing has been finished, the `QFile` object needs to be closed.
For the use of Qt Objects, see a short discussion in the section 6.4.2.

Class QDir

The class `QDir` can for example be used to obtain a list of files of a specific format in a certain directory, by using its command `entryList()`. An example to print the third element in the file array to the BrainVoyager Log tab has been specified below:

```
    var dir = new QDir("/Users/me/Data/someproject/subsection/");
var list = dir.entryList(["*.*"]);
BrainVoyager.PrintToLog(list[2]);
```

The wildcard "*.*" can of course be replaced by some specific format like "*.dcm".

Another command that can be used with `QDir` is `rename()`:

```
var dir = new QDir();
var success = dir.rename(filename, filenameWithoutSpaces);
```

Also, it is possible to create a directory using the `QDir` object:

```
// This script will create a subdirectory in the selected directory

var mydir = BrainVoyager.BrowseDirectory("Please select a directory");
var dir = new QDir(mydir);
var success = dir.mkdir("dir_created_by_script");
BrainVoyager.PrintToLog("Directory created: " + success);
```

For other functions of `QDir`, please consult the online Qt documentation for QDir.

Class QFile

4.1.3 List of (some) methods

QFile()
open()
close()
exists()
remove()

QFile properties

4.1.4 Detailed description of methods

open()

Function: open()
Description: Open the file object so that it can be read or written.
Parameter: OpenMode flag (an enum of QIODevice, see example script in section 4.3.2).

close()

Function: close()
Description: Close the file object.

exists()

Function: exists()
Description: Check whether the file exists on the hard disk.
Parameter: Filename (string)
Returns: Boolean (true or false)

remove()

Function: remove()
Description: Remove file from the hard disk.
Parameter: Filename (string)
Returns: Boolean (true or false)

4.2 Class QIODevice

4.2.1 List of methods

(Some of) QIODevice properties

WriteOnly: use this object and property for opening a file in write-only mode.
ReadOnly: use this object and property for opening a file in read-only mode.
Append: use this object and property for writing to an existing file, while preserving the contents of the file.

4.2.2 Class QTextStream

List of (some) methods

readLine()
writeString()
writeInt()
writeDouble()

Detailed description of methods

QTextStream()

Function: QTextStream() *Description:* Create a textstream object.
Returns: Textstream object.

readLine()

Function: readLine() *Description:* Reads a line.
Returns: Line (string)

writeString()

Function: writeString() *Description:* Write a string.
Parameter: Some text between double quotes.

writeInt()

Function: writeInt() *Description:* Write an integer.
Parameter: Integer.

writeDouble()

Function: writeDouble() *Description:* Write a number.
Parameter: Floating point number.

4.3 Example scripts

4.3.1 Example scripts: get file name and directory name

```
getFile("*.vmr");
getDir();

function getFile(filter) {

var fileWithPathName = BrainVoyager.BrowseFile("Please select the file", filter);
BrainVoyager.PrintToLog("Filename: " + fileWithPathName);
}

function getDir(filter) {

var dirName = BrainVoyager.BrowseDirectory("Please select the directory");
BrainVoyager.PrintToLog("Directory name: " + dirName);
}
```

4.3.2 Example scripts: write a file I

With the script below, one can write a text file. Please note that for writing a line ending, on Mac and Linux one can use `\n`; on Windows one could try `\r\n`.

```
// A really simple "script"

var f = new QFile("script_generated.txt");
f.open(new QIODevice.OpenMode(QIODevice.WriteOnly));
var ts = new QTextStream(f);

ts.writeString("This file was written by a script using qt_core extension.\n\n");
ts.writeString("This is a decimal value: ");
ts.writeDouble(3.4);
ts.writeString("\n\nThis seem to works fine!\n\n");

f.close();
```

4.3.3 Example script: write a file II

Writes a text file with the number of filenames on the first line, and the filenames on the following lines.

```
var projfilenames = new Array();
projfilenames.push("/Users/me/Data/testdata/subj1.fmr");
projfilenames.push("/Users/me/Data/testdata/subj2.fmr");
var filename = this.writeToTextFile(projfilenames);

function writeToTextFile(projfilenames) {

    var filecounter;
    // write the textfile to the path of the first file
    var name = this.getPath(projfilenames[0]) + "filenames.txt";

     BrainVoyager.PrintToLog("Name of file: " + name);

    var filenamesfile = new QFile(name);
    filenamesfile.open(new QIODevice.OpenMode(QIODevice.WriteOnly));
    var textstr = new QTextStream(filenamesfile);
    textstr.writeDouble(projfilenames.length);
    textstr.writeString("\n");
    for (filecounter = 0; filecounter < projfilenames.length; filecounter++) {
        textstr.writeString(projfilenames[filecounter] + "\n");
    }
    filenamesfile.close();
    BrainVoyager.PrintToLog("Wrote: " + name);
    return name;
}

function getPath(filename) {

var start = 0; // start searching from begin
var last = filename.lastIndexOf("/"); // search for last path separator
var path = filename.substring(start, (last+1)); // now return path without filename
return path;
}
```

4.3.4 Example script: read a file

This script reads a text file that consists of the number of files on the first line, and filenames on all following lines.

```
var filename = String("/Users/me/Data/testdata/filenames.txt");
var projfls = this.readTextFile(filename);


function readTextFile(name) {

    var projfilenames = new Array();
    var filecounter;
    BrainVoyager.PrintToLog("Reading: " + name);

    var filenamesfile = new QFile(name);
    filenamesfile.open(new QIODevice.OpenMode(QIODevice.ReadOnly));
    var textstr = new QTextStream(filenamesfile);
    var nroffiles = parseInt(textstr.readLine());
    for (filecounter = 0; filecounter < nroffiles; filecounter++) {
        var filename = textstr.readLine();
        BrainVoyager.PrintToLog(filename);
        projfilenames.push(filename);
    }
     filenamesfile.close();
     return projfilenames;
}
```

4.3.5 Example: delete a file

This script deletes a file (for example *.fmr), and also extra files of a functional project if they exist (*.stc and *-.stc). (This can be used for BrainVoyager QX 2.1, where the BrainVoyager `Remove()` command does not work.) *Thanks to Dirk Heslenfeld*

```
Remove("/Users/me/Data/Experiment1.fmr");

function Remove(file)
{
  BrainVoyager.PrintToLog("REMOVING: " + file);
  if (QFile.exists(file))
      QFile.remove(file);
  if (QFile.exists(file.substring(0, file.lastIndexOf(".")) + ".stc" ))
      QFile.remove(file.substring(0, file.lastIndexOf(".")) + ".stc" );
  if (QFile.exists(file.substring(0, file.lastIndexOf(".")) + "-.stc" ))
      QFile.remove(file.substring(0, file.lastIndexOf(".")) + "-.stc" );
}
```

Chapter 5

Creating scripts with dialogs

5.1 Introduction

It is possible to connect a graphical user interface (*.ui) to the script, i.e. making a "GUI Script". This makes it possible to re-use the script for another project or by another BrainVoyager user without having to change the underlying script, since the specific parameters are not hardcoded in the script, but provided via the graphical components of the dialog. The user interface is saved in an XML file that can be created by Qt Designer (is included when downloading the Qt Creator package from `http://download.qt.io/official_releases/qt/}.\newline`A nice description how to create a user interface for the script can be found at:
http://www.brainvoyager.com/bvresources/RainersBVBlog/files/guiscriptsandplugins.html.
See also the plugin developer guide "DeveloperGuide.qhc" in the BrainVoyager subfolder /Applications (Program Files)/UsersGuide/, since this also explains the GUI scripts.
An example script is shown in the section below. For more script examples for BrainVoyager, please see http://support.brainvoyager.com/ → "Available Tools → "Available Scripts" → "Scripts for BrainVoyager QX 2.1 and higher".

5.2 Writing GUI scripts

5.2.1 Capturing emitted signals

In graphical user interface (GUI) scripts, user input is collected from the signals emitted from the components on the dialog (*.ui), for example from buttons, radiobuttons, dropdownboxes etc. These graphical components are called 'widgets'. In the script (*.js) these signals can be caught and processed. For example, if the PushButton to start preprocessing has been clicked, invoke in the script a preprocessing function. A dialog can stay active for a long time, so a button could be clicked several times, if this is desirable. For capturing the emitted signals, the following properties can be used:

```
PushButton: clicked
RadioButton: toggled
CheckBox: stateChanged
LineEdit: editingFinished
TextEdit: textChanged (emit signal after each character)
```

In fact, these emitted signals are just notifications that the widget changed in state by action of the user, for example from unchecked to checked. The information itself can be collected via other properties of the widgets, which are described in section 5.2.2. The use of these signals has been illustrated in the extPrintDlg.js/ui script below.

5.2.2 Collecting information in the dialog

For obtaining the information in the widgets, the following properties of the respective widgets can be used:

```
Label: text
RadioButton: checked
CheckBox: checked
LineEdit: text
TextEdit: plainText
ComboBox: currentIndex, currentText (QX 2.2 and higher)
SpinBox: value
DoubleSpinBox: value
TimeEdit: time
DateEdit: date
```

Although it can be useful to capture the information when the signal has been emitted that the state of the widget has changed, it is not strictly necessary to do this directly after the user has entered some information. In the script below, for example, is all information collected via the function `collectInput()` when the user clicks the "Print" button. Say the LineEdit has the name "lePrint", then the text that the user has entered in the LineEdit can be access via `var text = lePrint.text;` if the LineEdit has been registered in the initDlg() function via FindChild() and otherwise via
`var textInLineEdit = this.<dialogname>.lePrint.text;` or, if the LineEdit is placed in a GroupBox:
`var textInLineEdit = this.<dialogname>.<groupboxname>.lePrint.text;`.

5.3 Other functions that can be used with the widgets

5.3.1 ComboBox

To include a ComboBox on the dialog, pull the Combo Box from the menu on the left hand side to the dialog.

Populating the Combo Box

Adding items to the ComboBox is possible in two ways, via the interface, using Qt Creator, or via code.
Via Qt Creator: Double click the box and press the '+' to add items (and '-' to remove them) (see figure 5.1).
*Via code in *.js script:* `addItem(<name>),insertItem(<index starting with 0>, <name>)`

count (property): gives number of items in ComboBox

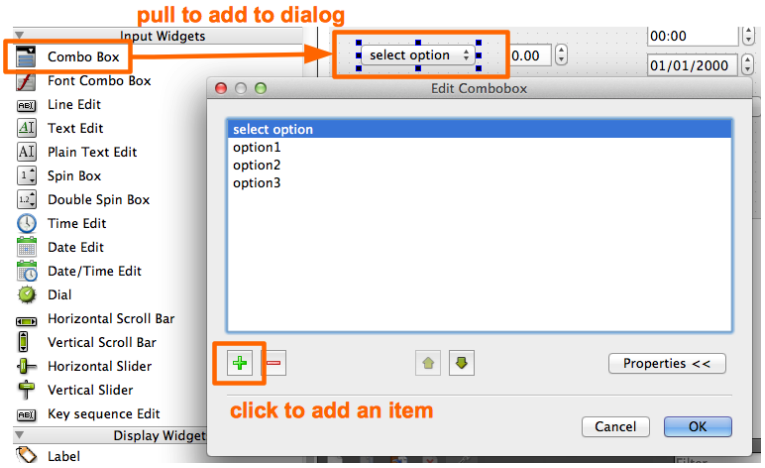


Figure 5.1: ComboBox in Qt Creator

Other commands for the Combo Box
Via the code one can select the index via the function `<name of combobox>.setCurrentIndex(<number>);`. The first item in the ComboBox has index 0.

Retrieving information from the Combo Box Use the following properties: `currentText` property `currentIndex` property

5.3.2 QListWidget

Populating the ListWidget
Adding items to the ListWidget is possible in two ways, via the interface, using Qt Creator, or via code.
Via Qt Creator: Double-click on the widget, the press '+' button (see figure 5.2).
*Via code in *.js script:*
`addItem():...`
`addItems():`

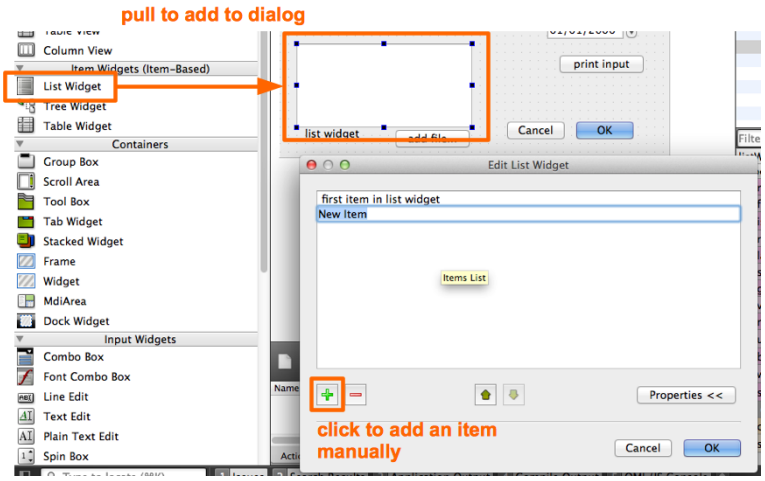


Figure 5.2: ListWidget in Qt Creator

Retrieving information from the List Widget
To retrieve the information the user has entered in the script, give the row number of the line (item), and use the `text` property. `item(<index>):...`
`text()`: returns text from items
example:
`var filename = listWidgetWithFilenames.item(0).text();`

Other commands for List Widget
`clear()`: remove all items in the widget
`count (property)`: gives number of items in ListWidget

5.3.3 Functions for non-GUI scripts vs. functions in GUI scripts: the Script Object

In scripts (*.js) without a dialog, functions have the following form:

```
function <function name>( [arguments] ) {

<instructions>
}
```

while in a script with a dialog or graphical user interface (GUI) one can use the form above or the following form:

```
scriptObj.<function name> = function ( [arguments] ) {

<instructions>
}
```

In the latter case, the function is made a member function of the script object.

What does the script object do?

In scripts with a dialog or graphical user interface (GUI), a script object in the script (*.js) regulates the communication between the "script" and the user input via the GUI. The script object is initialised in the beginning of the script:

```
var scriptObj = new Object;
```

The following properties of the script object can be set:

dialogFileName: To provide the name of the dialog file (*.ui):
```
scriptObj.dialogFileName = "ExampleGUIScript.ui";
```

dialogAccessName: This name can be used when addressing widgets via the widget hierarchy:
```
<this>.<dialog access name>.<widget name>.<emitted signal>.connect()
```

scriptNameForUser: Name of script in BrainVoyager "Scripts" menu.
```
scriptObj.scriptNameForUser = "GUIScriptTest";
```

The following functions always need to be present in the GUI script:

initDlg: This function of the script object contains declarations of the widgets on the *.ui file, and connects signals emitted by the widgets to slots (functions in the script)

returnScriptObj: A function to return the script object (to the script engine in BrainVoyager)

For a basic example GUI script, see section 5.4.

5.3.4 An example script with different widgets

In the script below the behaviour of the different graphical components (widgets) and how to capture their information is demonstrated.

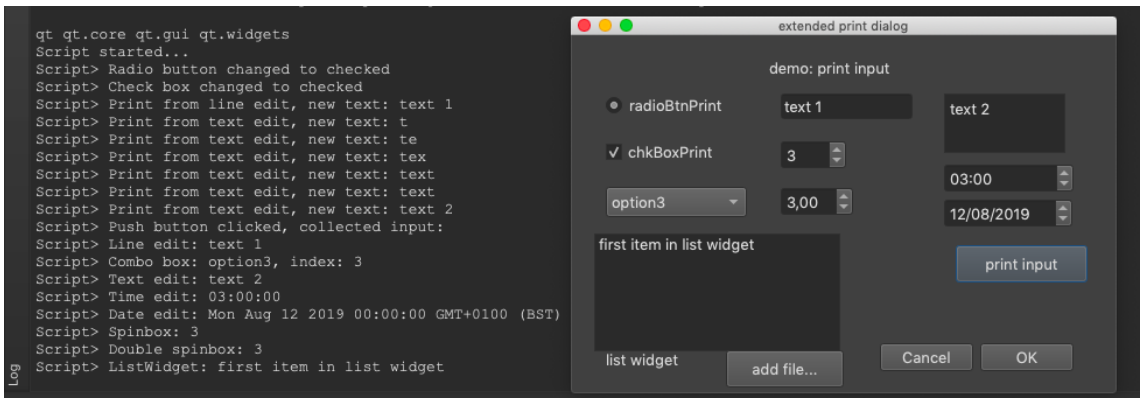


Figure 5.3: Dialog of the example script with different widgets

Behaviour of the script

Two different kinds of information capturing are shown.
At first, each widget *emits a signal* when it has been changed; this is received via the script and a notification is printed to the BrainVoyager Log tab.
Secondly, when the push button has been clicked, the *information* that the user has provided *in the widgets*, is collected and printed to the BrainVoyager Log tab.

Structure of the script

At the complete top and bottom of the script, matters using the script object (`scriptObj`) are defined. The script object is required for communication between BrainVoyager and the script and the script and the dialog.
Then, some custom functions are defined, with the names `collectInput()`,`printFromRadioButton()`, `printFromCheckBox()`,`printFromComboBox()`,`printFromLineEdit()` and `printFromTextEdit()`. In these functions, the information in a widget is collected and printed to the BrainVoyager Log tab.
Finally, in the standard `initDlg()` function, the emission of signals of each widgets is connected to each of these functions via the `connect()` function.

Usage of the script

The script can be used by saving the respective texts below in *.js and *.ui files in the /BVExtensions/Scripts/ directory, and start the BrainVoyager Script Editor. When the script is double clicked, its text will appear in the main screen of the Script Editor. Then the "Run" button can be pressed and the dialog of the script will appear (see figure 5.3).

5.3.5 The JavaScript

To use this script, save the following text with extension *.js in a text editor.

```
// extPrintDlg.js: Example GUI script for BrainVoyager QX 2.1
// This script bundle consists of extPrintDlg.js and extPrintDlg.ui,
place both in /Documents/BVQXExtensions/Scripts/ folder.
// Functionality: The script will print input from the widgets to the BrainVoyager QX log tab
if the button with label 'print input' and name 'btnPrint' is clicked.
// Some widgets will emit signals, and also print their input to the BrainVoyager QX log tab.
// Usage: In the BrainVoyager menu, select 'Scripts' > 'Edit and Run Scripts...' and
click 'Load...' to load this script, then 'Run'.
// HB, BI, 2010.

var scriptObj  = new Object;
scriptObj.scriptNameForUser  = "GUIScriptDemo";
scriptObj.dialogFileName  = "extPrintDlg.ui";
scriptObj.dialogAccessName  = "printDlg"; // the name 'printDlg' is used
// in the rest of the script to address the widgets
BrainVoyagerQX.PrintToLog("Script started...");

// in the initDlg() function below, this function is connected to btnPrint from extPrintDlg.ui
scriptObj.collectInput = function() {

BrainVoyager.PrintToLog("Script> Push button clicked, collected input:");
     BrainVoyager.PrintToLog("Script> Line edit: "  + this.printDlg.lnEditPrint.text);
BrainVoyager.PrintToLog("Script> Combo box: "  + this.printDlg.cmBoxPrint.currentText +
", index: " + this.printDlg.cmBoxPrint.currentIndex);
BrainVoyager.PrintToLog("Script> Text edit: "  + this.printDlg.txtEditPrint.plainText);
BrainVoyager.PrintToLog("Script> Time edit: "  + this.printDlg.tmEditPrint.time);
BrainVoyager.PrintToLog("Script> Date edit: "  + this.printDlg.dtEditPrint.date);
      BrainVoyager.PrintToLog("Script> Spinbox: "  + this.printDlg.spnBoxPrint.value);
      BrainVoyager.PrintToLog("Script> Double spinbox: "  + this.printDlg.dSpnBoxPrint.value);
}

scriptObj.printFromRadioButton = function() {
  if (this.printDlg.radioBtnPrint.checked) {
       BrainVoyager.PrintToLog("Script> Radio button changed to checked");
    } else {
       BrainVoyager.PrintToLog("Script> Radio button changed to unchecked");
    }
}

scriptObj.printFromCheckBox = function() {
    if (this.printDlg.chkBoxPrint.checked) {
        BrainVoyager.PrintToLog("Script> Check box changed to checked");
    } else {
        BrainVoyager.PrintToLog("Script> Check box changed to unchecked");
    }
```

```
}

scriptObj.printFromComboBox = function() {
    var currentItemIndex = this.printDlg.cmBoxPrint.currentIndex;
    var currentItemText = this.printDlg.cmBoxPrint.currentText;
    BrainVoyager.PrintToLog("Script> Print from combobox, current index: " + currentItemIndex
     + ", current text: " + currentItemText);
}

scriptObj.printFromLineEdit = function() {
    var textInLineEdit = this.printDlg.lnEditPrint.text;
    BrainVoyager.PrintToLog("Script> Print from line edit, new text: " + textInLineEdit);
}

scriptObj.printFromTextEdit = function() {
var newText =  this.printDlg.txtEditPrint.plainText;
BrainVoyager.PrintToLog("Script> Print from text edit, new text: " + newText);
}

scriptObj.initDlg = function() {

    this.printDlg.windowTitle = "extended print dialog";

    // connect all the widgets to functions here
    this.printDlg.btnPrint.clicked.connect(this, this.collectInput); // this is the button from the extPrintDlg.ui
    this.printDlg.radioBtnPrint.toggled.connect(this, this.printFromRadioButton);
    this.printDlg.chkBoxPrint.stateChanged.connect(this, this.printFromCheckBox);
    this.printDlg.lnEditPrint.editingFinished.connect(this, this.printFromLineEdit);
    this.printDlg.txtEditPrint.textChanged.connect(this, this.printFromTextEdit);
     // the text edit emits a signal and thus prints after input of each character

}

returnScriptObj = function() {
return scriptObj;
}
returnScriptObj(); // should be after returnScriptOjb = function()
```

5.3.6 The user interface

To use the script, save also the text below with the name "extPrintDlg.ui" (because this is defined in the top of the JavaScript, see above) via a text editor. This text has been automatically generated when the user interface was built with Qt Designer/Creator.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Dialog</class>
 <widget class="QDialog" name="Dialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>468</width>
    <height>307</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Dialog</string>
  </property>
  <widget class="QDialogButtonBox" name="buttonBox">
   <property name="geometry">
    <rect>
     <x>260</x>
     <y>260</y>
     <width>171</width>
     <height>32</height>
    </rect>
   </property>

   Etc...
```

5.4 Procedure to create a GUI script

1. In Qt Creator, select to Create a Qt Designer Form (*.ui)(figure 5.4)
2. In the following dialog, select as type “Dialog” (not Main Window or Widget)
3. In the JavaScript (*.js) in the BrainVoyager script editor, create a script object
4. Set the name of the script object property that specifies the *.ui name
5. Set the name of the script object property that specifies an alias for the dialog
6. In the initDlg() function, connect each graphical component name to a function in the script

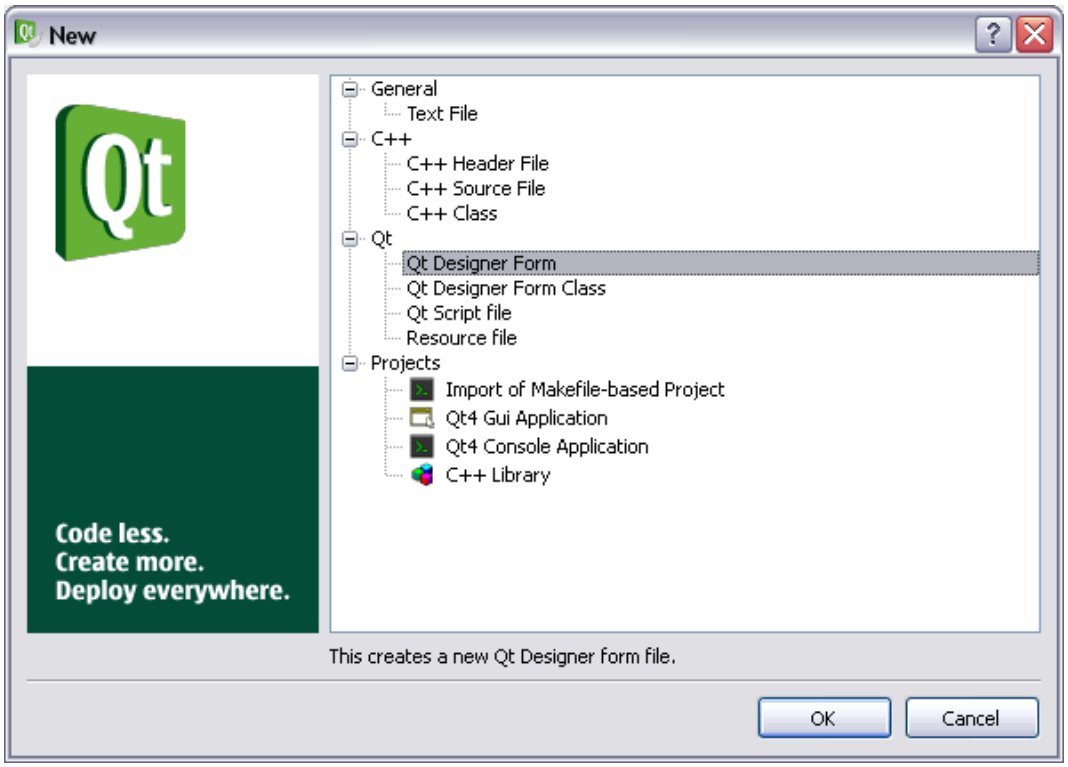


Figure 5.4: Select the Designer Form in Qt Creator

The script (*.js)

```
var scriptObj = new Object;
scriptObj.scriptNameForUser = "GUIScriptTest";
scriptObj.dialogFileName   = "ExampleGUIScript.ui";
scriptObj.dialogAccessName = "ScriptDialog";

scriptObj.initDlg = function() {
this.ScriptDialog.windowTitle = "Example GUI Script";
this.ScriptDialog.printToLogButton.clicked.connect(this, this.printTextToLog);
}

scriptObj.printTextToLog = function() {
BrainVoyager.PrintToLog("Script> Dialog's Print to Log button clicked");
}

returnScriptObj = function() {
return scriptObj;
}
returnScriptObj();
```

5.4.1 The user interface (*.ui)

The user interface can be assembled in Qt Creator (see figure 5.5).

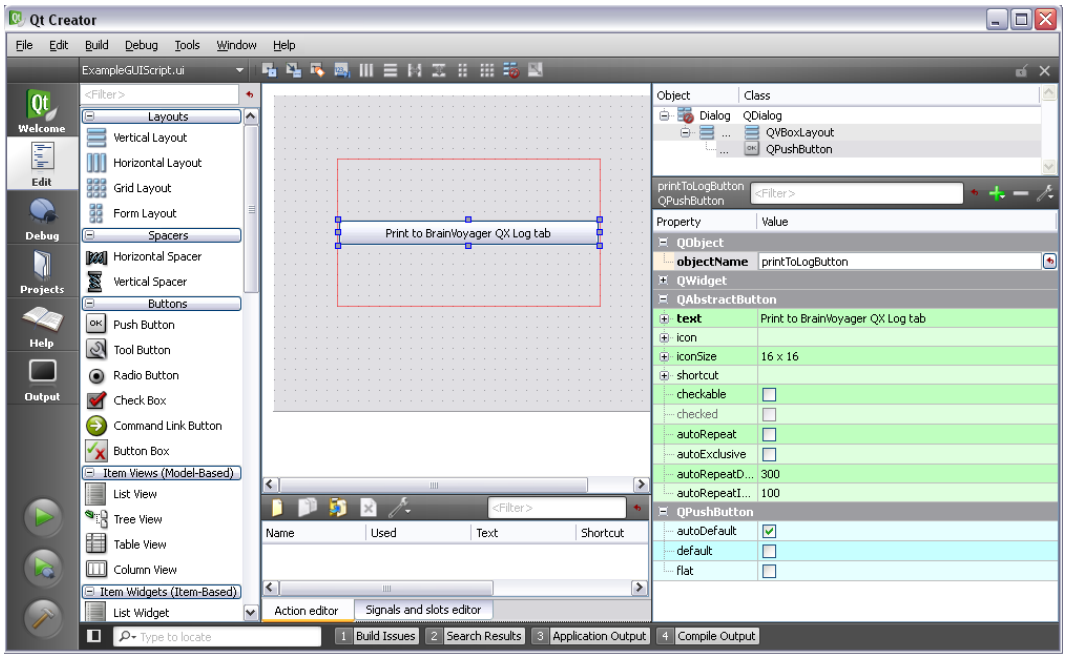


Figure 5.5: Creating the user interface in Qt Creator

When there are many widgets (graphical components), the function `BrainVoyager.FindChild()` can help to address the widget in the script. This is useful when the widget is located in a groupbox on a tab, when it can be a lot of work to find its proper name to address from the script, because it could be:
`<this>.<dialogname>.<tabname>.<groupboxname>.<buttonname>`. For example, when there is a push button on the user interface (*.ui) with the name `btnGetFiles`, one could do the following:

1. Create a global variable `getFileButton`
2. In `initDlg()`, assign this simple name to the push button:
   `getFileButton = BrainVoyager.FindChild("btnGetFiles");`
3. Use the variable `getFileButton` anywhere in the script.

5.4.2 Running a script with user interface

In the BrainVoyager main menu, select "Scripts" → "Edit and Run Scripts...". This will open the script editor. Then, load the script (*.js). When clicking the "Run" button, the dialog will pop up (see figure 5.6). Both script (*.js) and user interface (*.ui) should be placed in the directory `/Documents/BVExtensions/Scripts/`. Please note that it is not possible to run GUI scripts directly from the "Scripts" menu.

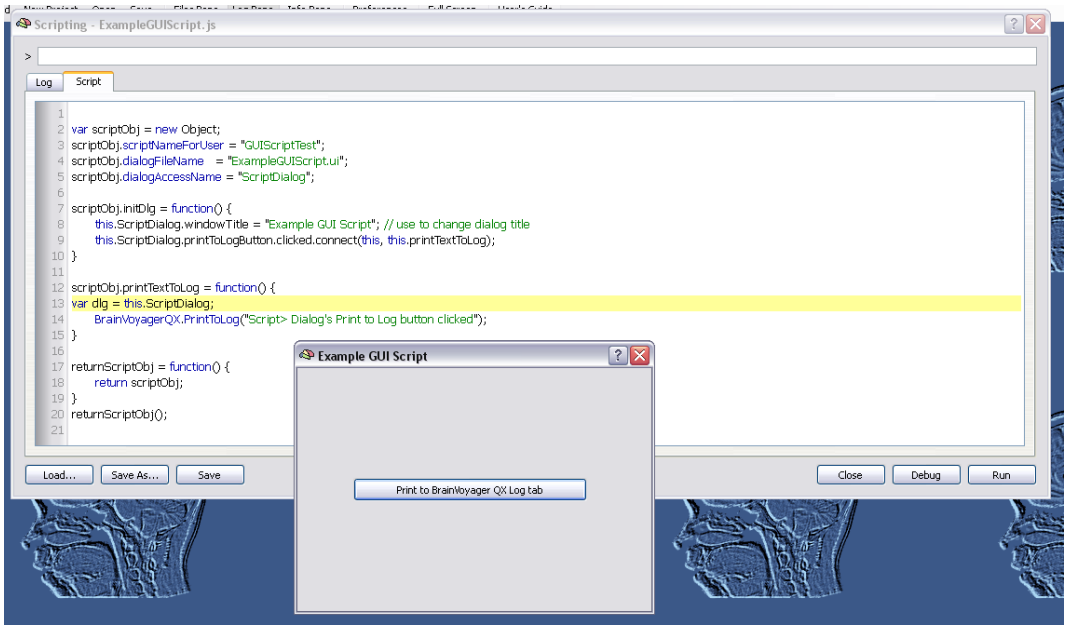


Figure 5.6: Using the user interface in BrainVoyager

5.4.3 Catching errors

In Javascript, one can catch errors using the `try/catch` keywords:

```
  try {
...
(some critical code)
...
} catch (e) {
...
(some measure to take when an error occurs)
...
}
```

The idea is to put the critical code between the brackets of the `try`-block, and to place the measure that will be taken in case the critical code in the `try`-block fails, in the `catch`-block. The error `e` will be text, so one of the measures could be to print the error message to the BrainVoyager Log tab (see figure 5.7), so that the user can do something about the error. Another sensible measure would be of course to cancel any subsequent processes that are dependent of the critical code.

```
BrainVoyagerQX.OpenDocument( "/Users/hester/Data/ObjectsDicomGSG/CG_3DT1MPR_SCRIPT_CLEAN_ACPC.vmr" );
Error: SyntaxError: too few arguments in call to OpenDocument(); candidates are
    OpenDocument(QString)
```

Figure 5.7: Print an error to the BrainVoyager Log tab

Chapter 6

JavaScript language reference

6.1 Introduction

This information about operators and objecta in the JavaScript language in this chapter was described by the WWW consortium; more information about the language can be found at
http://www.w3schools.com/JS/—.
For an excellent tutorial, visit the Mozilla Foundation's web site hosted at
http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide.
Among the many books, David Flanagan's JavaScript: The Definitive Guide (O'Reilly, 2006) is recommended both as a tutorial and as a reference manual. Free examples for the third edition can be found here:
http://examples.oreilly.com/9781565923928/.
Qt Script is supposed to comply to the ECMAScript standard. This standard can be found at:
http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf.

6.2 Keywords

6.2.1 Operators

Assignment operators

Assignment operators are used to assign values to JavaScript variables.
Given that `x=10` and `y=5`, the table below explains the assignment operators:

Table 6.1: Assignment operators

Operator	*Description*	*Same as*	*Result*
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

Arithmetic operators

Arithmetic operators are used to perform arithmetic between variables and/or values.
Given that $y = 5$, the table below explains the arithmetic operators:

Table 6.2: Arithmetic operators

Operator	*Description*	*Example*	*Result*
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2 x=3	
*	Multiplication	x=y*2 x=10	
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1
++	Increment	x=++y	x=6
–	Decrement	x=–y	x=4

Comparison operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.
Given that `x=5`, the table below explains the comparison operators:

Table 6.3: Comparison operators

Operator	*Description*	*Example*
==	is equal to	x==8 is false
===	is exactly equal to (value and type)	x===5 is true, x===“5” is false
! =	is not equal	x!=8 is true
>	is greater than	$x > 8$ is false
<	is less than	$x < 8$ is true
>=	is greater than or equal to	$x >= 8$ is false
<=	is less than or equal to	$x <= 8$ is true

Logical Operators

Logical operators are used to determine the logic between variables or values. The operators are `&&` (and), `II` (or), and `!` (not).

Conditional Operators

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.
Syntax

```
variablename = (condition) ? value1 : value2
```

Example

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

If the variable visitor has the value of “PRES”, then the variable greeting will be assigned the value “Dear President ” else it will be assigned “Dear ”.

Bitwise

```
Xor (^), And (&), Or (|), Not (~), Bitwise left shift (<<)
Bitwise sign propagating right shift (>>)
Bitwise zero-operand right shift (>>>)
```

Special

```
?: (expression ? resultIfTrue : resultIfFalse)
, (evaluation of 1st and 2nd operand, returns 2nd)
function (var variable = function( optArgs ) { Statements } )
```

```
in (property in Object, returns boolean)
instanceof (object instanceof type, returns boolean)
new (var instance = new Type( optArgs )
this (this.property)
typeof (typeof item)
```

6.2.2 Declarations

`var, const, class, this, function`

6.2.3 Control statements

`break, case, catch, continue, default, do, else, for, if, finally, label, return, switch, throw, try, while, with`

6.2.4 Error handling

`try...catch`

6.2.5 Comments

For single line: `//`
Multi-line comments: `/* ... */`

6.3 Data types

The following data types exist in JavaScript:

```
Array, Boolean, Date, Math, Number, String, RegExp
```

It is also possible to create own objects (see paragraph 6.4.1). Some of the data types have been described below: for Array, see section 6.3.1, Math (section 6.3.3), String (section 6.3.2), RegExp (section 6.3.4) and Date (section 6.3.5).

6.3.1 Array

Table 6.4: Array Object properties

Property	*Description*
constructor	Returns a reference to the array function that created the object
index	
input	
length	Sets or returns the number of elements in an array
prototype	Allows you to add properties and methods to the object

Table 6.5: Array Object methods

Method	*Description*
`concat()`	Joins two or more arrays and returns the result
`join()`	Puts all the elements of an array into a string.
...	The elements are separated by a specified delimiter
`pop()`	Removes and returns the last element of an array
`push()`	Adds one or more elements to the end of an array and returns the new length
`reverse()`	Reverses the order of the elements in an array
`shift()`	Removes and returns the first element of an array
`slice()`	Returns selected elements from an existing array
`sort()`	Sorts the elements of an array
`splice()`	Removes and adds new elements to an array
`toSource()`	Represents the source code of an object
`toString()`	Converts an array to a string and returns the result
`unshift()`	Adds one or more elements to the beginning of an array and returns the new length
`valueOf()`	Returns the primitive value of an Array object

Array Properties

Arrays have a single property, `length`, that holds the number of elements in the array.

Array Functions

The Array Functions documentation is available from the Qt Script documentation.

concat()

```
concat( array1, array2, optArray3, ... optArrayN )
```

```
    var x = new Array( "a", "b", "c" );
    var y = concat( a, [ "d", "e" ], [ 90, 100 ] );
    // y == [ "a", "b", "c", "d", "e", 90, 100 ]
```

Concatenates any number of arrays together in the order given, and returns a single array.

join()

```
join( optSeparator )
```

```
    var x = new Array( "a", "b", "c" );
    var y = x.join();         // y == "a,b,c"
    var z = x.join( " * " );  // y == "a * b * c"
```

Joins all the elements of an array together, separated by commas, or the specified optSeparator.

pop()

```
pop()
```

```
    var x = new Array( "a", "b", "c" );
    var y = x.pop(); // y == "c"  x == [ "a", "b" ]
```

Pops the top-most (right-most) element off the array and returns this element.
See also push(), shift() and unshift().

push()

```
push( element1, optElement2, ... optElementN )
```

```
    var x = new Array( "a", "b", "c" );
    x.push( 121 ); // x == [ "a", "b", "c", 121 ]
```

Pushes the given item(s) onto the top (right) end of the array.
See also push(), shift() and unshift().

reverse()

```
reverse()
```

```
    var x = new Array( "a", "b", "c", "d" );
    x.reverse(); // x == [ "d", "c", "b", "a" ]
```

Reverses the elements in the array.

shift()

```
shift()
```

```
    var x = new Array( "a", "b", "c" );
    var y = x.shift(); // y == "a"  x == [ "b", "c" ]
```

Shifts the bottom-most (left-most) element off the array and returns this element.
See also push(), shift() and unshift().

slice()

```
slice( startIndex, optEndIndex )
```

```
    var x = new Array( "a", "b", "c", "d" );
    var y = x.slice( 1, 3 ); // y == [ "b", "c" ]
    var z = x.slice( 2 );    // z == [ "c", "d" ]
```

Copies a slice of the array from the element with the given starting index, startIndex, to the element before the element with the given ending index, optEndIndex. If no ending index is given, all elements from the starting index onward are sliced.

sort()

```
sort( optComparisonFunction )
```

```
    var x = new Array( "d", "x", "a", "c" );
    x.sort(); // x == [ "a", "c", "d", "x" ]
```

Sorts the elements in the array using string comparison. For customized sorting, pass the sort() function a comparison function, optComparisonFunction, that has the following signature and behavior:

```
    function comparisonFunction( a, b ) // signature
```

The function must return an integer as follows:

```
-1 if a < b
0 if a == b
1 if a > b
```

Example:

```
    function numerically( a, b ) { return a < b ? -1 : a > b ? 1 : 0; }
    var x = new Array( 8, 90, 1, 4, 843, 221 );
    x.sort( numerically ); // x == [ 1, 4, 8, 90, 221, 843 ]
```

splice()

```
splice( startIndex, replacementCount, optElement1, ... optElementN )
```

```
    var x = new Array( "a", "b", "c", "d" );

    // 2nd argument 0, plus new elements   ==> insertion
    x.splice( 1, 0, "X", "Y" );
    // x == [ "a", "X", "Y", "b", "c", "d" ]

    // 2nd argument > 0, and no elements   ==> deletion
    x.splice( 2, 1 );
    // x == [ "a", "X", "b", "c", "d" ]

    // 2nd argument > 0, plus new elements ==> replacement
    x.splice( 3, 2, "Z" );
    // x == [ "a", "X", "b", "Z" ]
```

Splices elements into the array and out of the array. The first argument, startIndex, is the start index. The second argument, replacementCount, is the number of elements that are to be replaced. Make the second argument 0 if you are simply inserting elements. The remaining arguments are the elements to be inserted; there can be no elements if you are simply deleting a part of the array, i.e. if the second argument is > 0.

toString()

```
toString()
```

```
    var x = new Array( "a", "b", "c" );
    var y = x.toString();  // y == "a,b,c"
    var z = x.join();      // y == "a,b,c"
```

Joins all the elements of an array together, separated by commas. This function is used when the array is used in the context of a string concatenation or is used as a text value, e.g. for printing. Use join() if you want to use your own separator.

unshift()

```
unshift( expression, optExpression1, ... opExpressionN )
```

```
    var x = new Array( "a", "b", "c" );
    x.unshift( 121 ); // x == [ 121, "a", "b", "c" ]
```

Unshifts the given item(s) onto the bottom (left) end of the array. See also push(), shift() and unshift().

6.3.2 String

The functions that can be used for a string object are shown in the table below.

Table 6.6: String Object methods

Method	*Description*
`anchor("anchorName")`	
`big()`	
`blink()`	
`bold()`	
`charAt(index)`	Returns the character at the specified index
`charCodeAt([i])`	Returns the Unicode of the character at the specified index
`concat(string2)`	Joins two or more strings, and returns a copy of the joined strings
`fixed()`	
`fontcolor(#rrggbb)`	
`fontsize(1to7)`	
`fromCharCode(n1...)*`	Converts Unicode values to characters
`indexOf("str" [,i])`	Returns the position of the first found occurrence of a specified value in a string, otherwise -1
`italics()`	
`lastIndexOf("str" [,i])`	Returns the position of the last found occurrence of a specified value in a string, otherwise -1
`link(url)`	
`localeCompare()`	
`match(regexp)`	Searches for a match between a regular expression and a string, and returns the match
`replace(regexp,str)`	Searches for match between substring (or regexp) and string, replaces matched substr with new substr
`search(regexp)`	Searches for match between a regexp and a string, returns the position of the match
`slice(i,j)`	Extracts a part of a string and returns a new string
`small()`	
`split(char)`	Splits a string into an array of substrings
`strike()`	
`sub()`	
`substring(from, to)`	Return part of the string, starting with position “from”. “to” is optional.
`sup()`	
`toLocaleLowerCase()`	
`toLocaleUpperCase()`	
`toLowerCase()`	Converts a string to lowercase letters
`toString()`	
`toUpperCase()`	Converts a string to uppercase letters
`valueOf()`	Returns the primitive value of a String object

Example of `lastIndexOf()`:

```
"run1.vtc".lastIndexOf(".vtc");
```

returns 4, while

```
"run1.fmr".lastIndexOf(".vtc");
```

returns -1.
The function `lastIndexOf()` returns the position of the substring in string. The string is the object and the substring needs to be passed as argument (between the parentheses) to the function. If substring is not present in the string, returns -1. In the example shown in figure 6.1, the string is ``run1.vtc'' and the substring ``.vtc''.

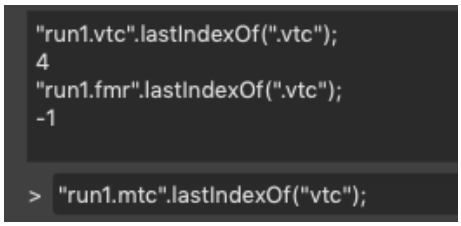


Figure 6.1: lastIndexOf() returns position of substring in string; if substring is not present in the string, returns -1 (using the direct interpreter of the BrainVoyager Script Editor)

6.3.3 Math

The Math object allows you to perform mathematical tasks.
Syntax:

```
var pi_value=Math.PI;
var sqrt_value=Math.sqrt(16);
```

Note: Math is not a constructor. All properties and methods of Math can be called by using Math as an object without creating it.

Table 6.7: Math properties

Property	*Description*
`E`	Returns Euler's number (approx. 2.718)
`LN2`	Returns the natural logarithm of 2 (approx. 0.693)
`LN10`	Returns the natural logarithm of 10 (approx. 2.302)
`LOG2E`	Returns the base-2 logarithm of E (approx. 1.442)
`LOG10E`	Returns the base-10 logarithm of E (approx. 0.434)
`PI`	Returns PI (approx. 3.14159)
`SQRT1_2`	Returns the square root of 1/2 (approx. 0.707)
`SQRT2`	Returns the square root of 2 (approx. 1.414)

Table 6.8: Math methods

Method	*Description*
`abs(x)`	Returns the absolute value of a number
`acos(x)`	Returns the arccosine of a number
`asin(x)`	Returns the arcsine of a number
`atan(x)`	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
`atan2(y,x)`	Returns the angle theta of an (x,y) point as a numeric value between -PI and PI radians
`ceil(x)`	Returns the value of a number rounded upwards to the nearest integer
`cos(x)`	Returns the cosine of a number
`exp(x)`	Returns the value of E^x
`floor(x)`	Returns the value of a number rounded downwards to the nearest integer
`log(x)`	Returns the natural logarithm (base E) of a number
`max(x,y)`	Returns the number with the highest value of x and y
`min(x,y)`	Returns the number with the lowest value of x and y
`pow(x,y)`	Returns the value of x to the power of y
`random()`	Returns a random number between 0 and 1
`round(x)`	Rounds a number to the nearest integer
`sin(x)`	Returns the sine of a number
`sqrt(x)`	Returns the square root of a number
`tan(x)`	Returns the tangent of an angle
`toSource()`	Represents the source code of an object
`valueOf()`	Returns the primitive value of a Math object

6.3.4 RegExp

A regular expression is an object that describes a pattern of characters.
Regular expressions are used to perform powerful pattern-matching and "search-and-replace" functions on text. Syntax:

```
var txt=new RegExp(pattern,modifiers);
```

or more simply:

```
var txt=/pattern/modifiers;
```

`pattern` specifies the pattern of an expression
`modifiers` specify if a search should be global, case-sensitive, etc.

Some concepts used in regular expressions are depicted in figure 6.2.

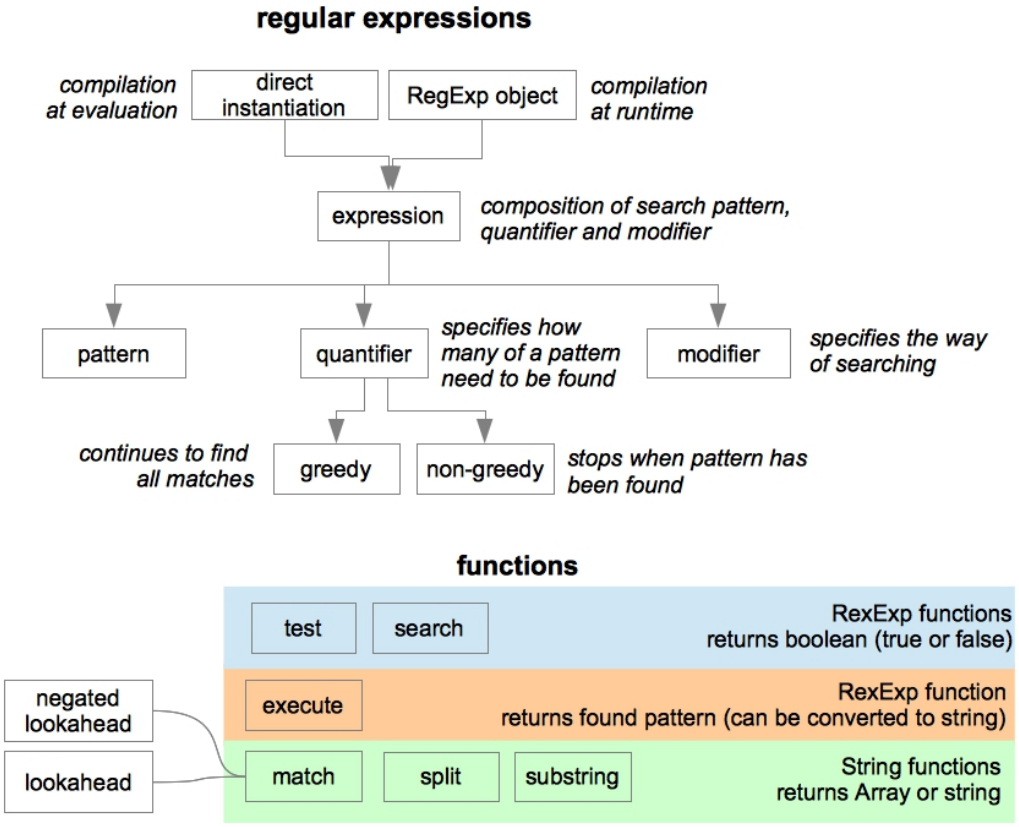


Figure 6.2: Regular expression concepts

Table 6.9: RegExp modifiers

Modifier	*Description*
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Table 6.11: RegExp metacharacters

Metacharacter	*Description*
`.`	Find a single character, except newline or line terminator
`\w`	Find a word character
`\W`	Find a non-word character
`\d`	Find a digit
`\D`	Find a non-digit character
`\s`	Find a whitespace character
`\S`	Find a non-whitespace character
`\b`	Find a match at the beginning/end of a word
`\B`	Find a match not at the beginning/end of a word
`\0`	Find a NUL character
`\n`	Find a new line character
`\f`	Find a form feed character
`\r`	Find a carriage return character
`\t`	Find a tab character
`\v`	Find a vertical tab character
`\xxx`	Find the character specified by an octal number xxx
`\xdd`	Find the character specified by a hexadecimal number dd
`\uxxxx`	Find the Unicode character specified by a hexadecimal number xxxx

Table 6.12: RegExp quantifiers

Quantifier	*Description*
`n+`	Matches any string that contains at least one n
`n*`	Matches any string that contains zero or more occurrences of n
`n?`	Matches any string that contains zero or one occurrences of n
`n{X}`	Matches any string that contains a sequence of X n's
`n{X,Y}`	Matches any string that contains a sequence of X or Y n's
`n{X,}`	Matches any string that contains a sequence of at least X n's
`n$`	Matches any string with n at the end of it
`^n`	Matches any string with n at the beginning of it
`?=n`	Matches any string that is followed by a specific string n
`?!n`	Matches any string that is not followed by a specific string n

Table 6.13: RegExp object properties

Property	*Description*
`global`	Specifies if the `g` modifier is set
`ignoreCase`	Specifies if the `i` modifier is set
`lastIndex`	The index at which to start the next match
`multiline`	Specifies if the `m` modifier is set
`source`	The text of the RegExp pattern

Table 6.14: RegExp methods

Method	*Description*
`compile()`	Compiles a regular expression
`exec()`	Tests for a match in a string. Returns a result array
`test()`	Tests for a match in a string. Returns true or false

Examples

Example 1: regular expression in the interpreter

Imagine we would want to find the matrix size of an image from the file name (this is a feature in the batchprocessingwizard 1.8.2). In the following example (figure 6.3), we have used the regular expression `/\d+_\d+/` in the direct interpreter of BrainVoyager's Script Editor to find the two pairs of digits separated by an underscore (_) in the string `brupc96_96.uff`; the regular expression returns `96_96`.

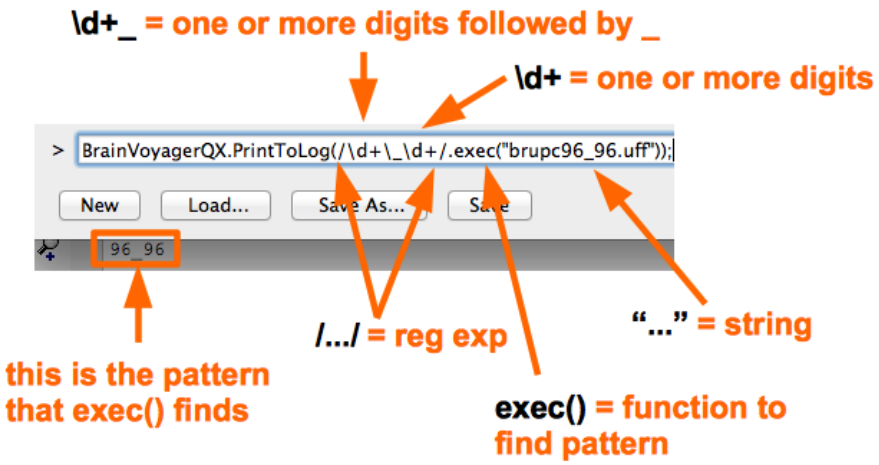


Figure 6.3: Using a regular expression in the interpreter

Example 2: regular expression with exec() function

Likewise, we can use the group parentheses (in regular expression `/(\d+)_(\d+)/`) and use the `RegExp` function `exec()` the two matrix sizes in the string `brupc96_96.uff`. For the `exec()` function, we get the resolution numbers nicely separated in the result (see figure 6.4); we receive an array with `96_96`, `96` and `96`.

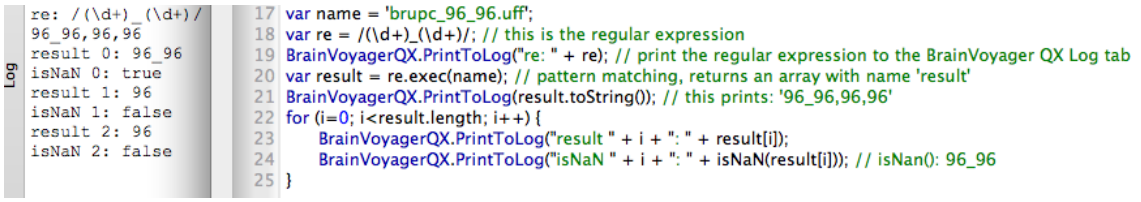


Figure 6.4: Using a regular expression with parentheses

Example 3: using the split() function

When we use the `String` function `split()` to extract the pair of numbers, most of the string is returned as substrings, separated by the numbers: `brupc`, `96`, `96`, `.uff` (see figure 6.5).

```
var name = 'brupc96_96.uff';
var re = /(\d+)_(\d+)/;
var res=name.split(re);
BrainVoyager.PrintToLog("split: " + res); // prints: split: brupc,96,96,.uff
```

Just like in the `exec()` example, we would still need to loop through the array to obtain the x-size 96 and y-size 96. In figure 6.4 a loop is used; in the loop, each entry of the array is tested whether it is a number or not using the function `isNaN()`.

Figure 6.5: Using a regular expression with parentheses and splitting

Example 4: replacing a subject name

In this example, we try to replace the subject number in a filename, so that it is more convenient to preprocess FMR files in a loop. We use the regular expression `/[^_]+/`; the slashes (`/ /`) indicate that this is a regular expression. The quantifier hat (`^`) means that it looks for a pattern at the beginning of the string; the underscore (_) indicates the character until where the pattern needs to be found. So in case the string is `AA_run1.fmr` and the regular expression is executed, it will find `AA`.
Now we now that it will match any characters until the underscore, we can skip using the `exec()` function and straight away replace the `AA` by the next subject's initials `BB`.
We use the function `<string>.replace(<regular expression>, <new substring>)`:

```
var subjname = 'AA_run1.fmr';
var re = /[^_]+/; // matches everything at the beginning until the '_'
var newstr = subjname.replace(re, 'BB');
BrainVoyager.PrintToLog(newstr); // prints 'BB_run1.fmr'
```

Figure 6.6 depicts the use of this regular expression in BrainVoyager.

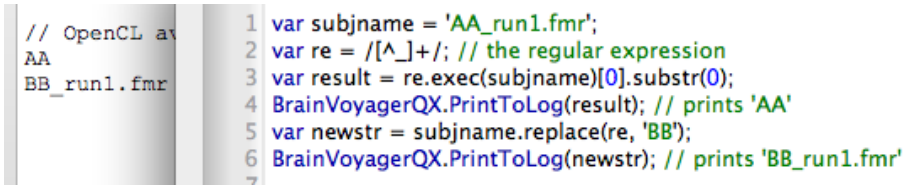


Figure 6.6: Using a regular expression with the replace function

Example 5: replacing subject names in a loop

If we extend example 4 with a loop, we can use our regular expression in a preprocessing procedure:

```
var subjname =  'XX_run1.fmr';
var re = /[^_]+/; // matches until '_'
var namesArray = ['AA', 'BB', 'CC', 'DD', 'EE'];
var subj; // counter
for (subj=0; subj<namesArray.length; subj++) {
var newstr = subjname.replace(re, namesArray[subj]);
BrainVoyager.PrintToLog(newstr);
// ... do some preprocessing for this subject
}
```

See also figure 6.7.

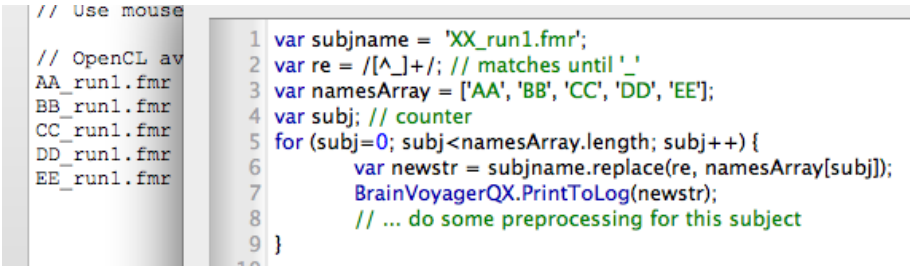


Figure 6.7: Using a regular expression for creation of subject names

6.3.5 Date and time

For the object `Date` we have the following `get()` functions:

```
getDay(), getFullYear(), getHours(), getMilliseconds(),
getMinutes(), getMonth(), getSeconds(), getTime(),
getTimeZoneOffset()
Universal Coordinated Time (=Greenwich Mean Time, GMT)
getUTCDate(), getUTCDay(), getUTCFullYear(),
getUTCHours(), getUTCMilliseconds(), getUTCMinutes(),
getUTCMonth(), getUTCSeconds(), getUTC()
```

The following `set()` functions are available:

```
setDate(), setFullYear(), setHours(), setMilliseconds(), setMinutes(),
setMonth(), setSeconds(), setTime(), setUTCDate(), setUTCFullYear(),
setUTCHours(), setUTCMilliseconds(), setUTCMinutes(),
setUTCMonth(), setUTCSeconds(), setUTCTime()
parse(), toString(), toLocaleString(), toUTCString
```

6.4 Objects

6.4.1 Creating objects

There are two ways for creating objects: via *direct instantiation* and via a *constructor*.

1. Create a direct instance of an object
The following code creates an new instance of an object, and adds four properties to it:

```
personObj=new Object();
personObj.firstname="John";
personObj.lastname="Doe";
personObj.age=50;
personObj.eyecolor="blue";
```

alternative syntax (using object literals):

```
personObj={firstname:"John",lastname:"Doe",age:50,eyecolor:"blue"};
```

Adding a method to the personObj is also simple. The following code adds a method called `eat()` to the `personObj`:

```
personObj.eat=eat;
```

2. Create an object constructor
Create a function that construct objects:

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;
}
```

Inside the function you need to assign things to this.propertyName. The reason for all the ”this” stuff is that you’re going to have more than one person at a time (which person you’re dealing with must be clear). That’s what ”this” is: the instance of the object at hand.
Once you have the object constructor, you can create new instances of the object, like this:

```
var myFather=new person("John","Doe",50,"blue");
var myMother=new person("Sally","Rally",48,"green");
```

You can also add some methods to the person object. This is also done inside the function:

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;

this.newlastname=newlastname;
}
```

In short, independent functions can be declared function

```
nameOfFunction(<possible arguments>) {
...
}
```

while a function of an object uses the syntax

```
<objectName>.nameOfFunction = function(<possible arguments>) {
...
}
```

The use of member functions in BrainVoyager scripting can be found in the scripts (*.js) that operate with the user interface (*.ui) for scripts: functions of the script object are declared using

```
scriptOjb.<nameOfFunction> = function() { ... }
```

for example in the `scriptObj.initDlg()` function:

```
scriptObj.initDlg = function() {

    this.exampleDlg.windowTitle = "example dialog";

(etc)
}
```

6.4.2 Using Qt Objects

It is also possible to use Qt Objects via scripting. All available objects can be found in the 'Locals' window of the debugger (see figure 6.8). When clicking on the objects, its functions will be revealed (at some point). Via these Qt objects some functions can be retrieved that are not directly available via BrainVoyager's scripting functions or JavaScript.
For example, the script below, which actually just seems to work on Windows, uses the function `grabWindow()` of the Qt object `QPixMap` to make a screenshot of the BrainVoyager volume window.

```
 var test = QPixmap.grabWindow(true,0,0,500, 500);
test.save('test.png');
```

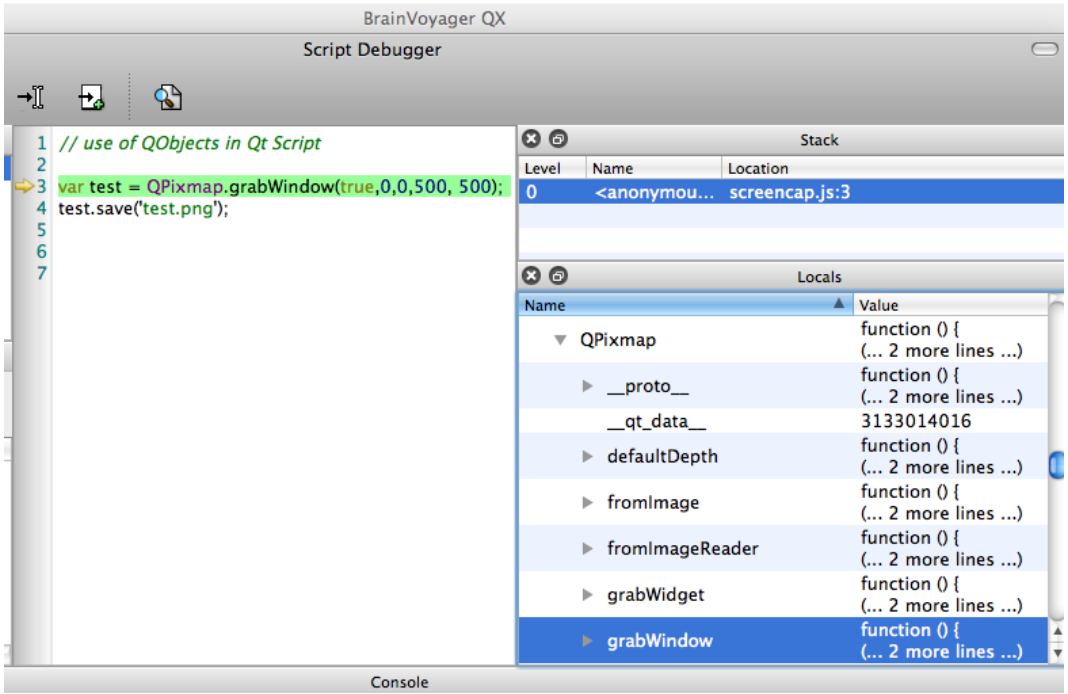


Figure 6.8: List of available Qt Objects in Locals window of BrainVoyager Script Debugger

With thanks to Holger Hecht and Dirk Heslenfeld