Scripting Reference Manual

for

BrainVoyager QX 2.3

Contents

Chapter 1

Introduction

Welcome to the BrainVoyager Scripting Reference Manual![1] This reference manual can be used for scripting in BrainVoyager QX 2.3, but most commands can also be used in BrainVoyager QX 2.2, and some in BrainVoyager QX 2.1.
Latest update: 13-04-2011

1.1 Location of script files

Scripts (*.js) for BrainVoyager QX 2.1 and higher can be stored in the folder /(My) Documents/BVQXExtensions/Scripts/.

1.2 Switching from scripting in BrainVoyager QX 2.0 to 2.1 or higher

How to use Qt Script and what are the differences with respect to QSA?
- To start, call the scripting dialog clicking on "Scripts" → "Scripting..." menu.
- Files are stored as ".js" (JavaScript) text files allowing to use also external editors (by simple double-clicking).
- Instead of "Application.BrainVoyagerQX", you only need "BrainVoyagerQX" global symbol to get access to the BVQX API, please have a look at the example scripts.
- There (at present) no support for "projects", i.e. only one file is evaluated (this might change).
- The script editor has syntax highlighting. Furthermore the QScript debugger is integrated, which allows for easy hunting of bugs.
- On Windows, COM support is available from version 2.2.
- QScript is much faster than QSA, so some "plugin" processing will be possible without requiring C++ code. - Scripts with a graphical user interface (GUI) have to be started from within the script editor. - Script GUIs are saved in user interface (*.ui) files that can be created with the free Qt Designer software.

See also chapter 5.

[1] This used to be called the 'Scripting Guide' but the new names 'Getting Started Guide for scripting' and 'Scripting Reference Manual' are hopefully more clear.

1.3 History

1.3.1 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.

1.3.2 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.

1.3.3 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 5.3). The parameter `dataType` has been added for creating VTC and VDW files. Two properties for changing the confound in SDM files have been added.

1.3.4 BrainVoyager QX 2.0

No changes.

1.3.5 BrainVoyager QX 1.10

1.10.4

In this BrainVoyager QX version, it is also possible to create VDW files via scripting.

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.

1.3.6 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(), CreateProject-DMR() 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 QX 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.

Chapter 2

Scripting in BrainVoyager QX 2.1-2.3

2.1 The BrainVoyager Script Editor

2.1.1 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.

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.

2.1.2 Creating scripts

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

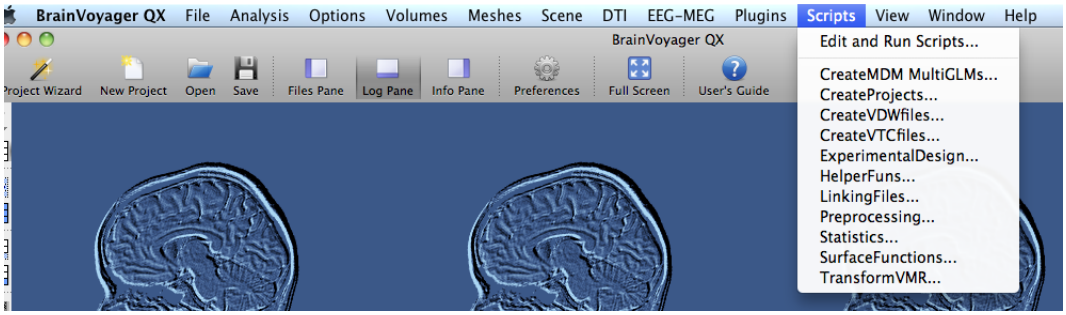

Figure 2.1: Open the Scripting Editor

2.1.3 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/BVQXExtensions/Scripts/`. It is also possible to run the script by clicking the "Run" button in the BrainVoyager QX 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 QX 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).

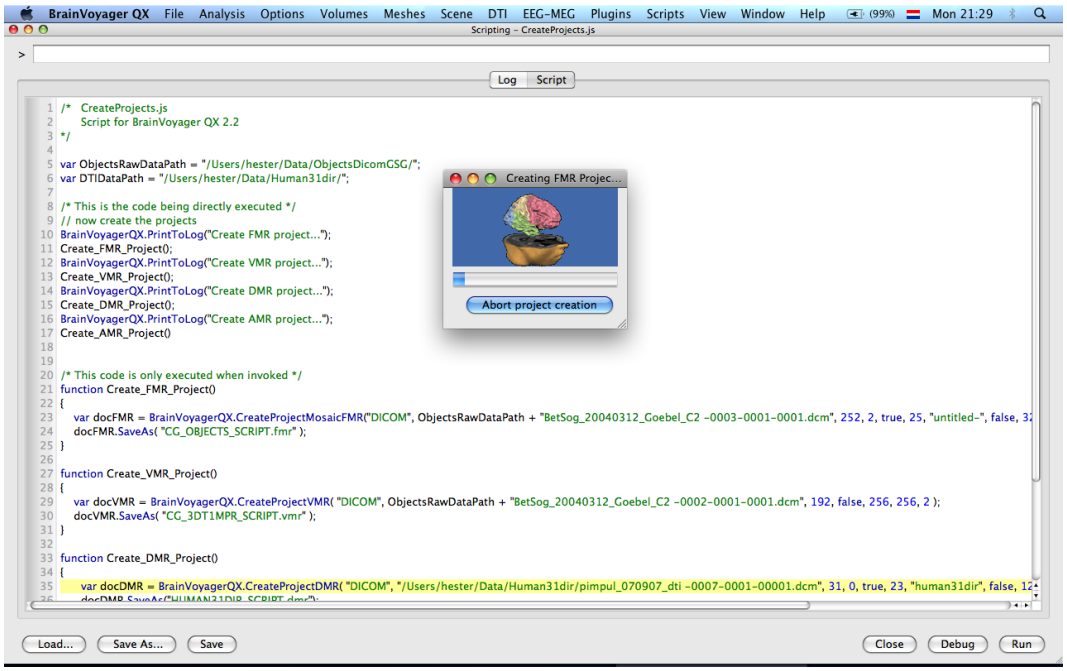

Figure 2.2: Running a script

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

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

2.1.4 Using the interpreter

In the BrainVoyager QX 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.

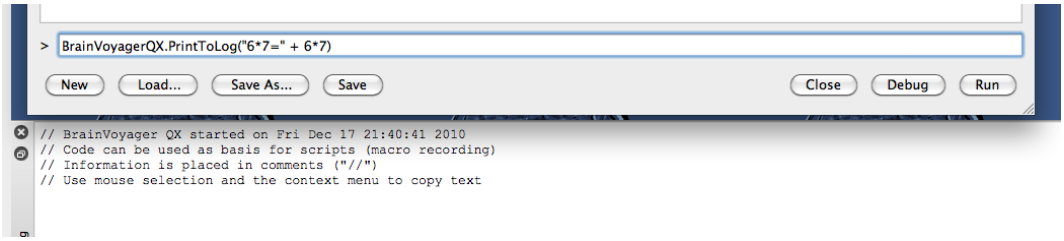

Figure 2.4: Commands typed in this field will be evaluated directly

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.

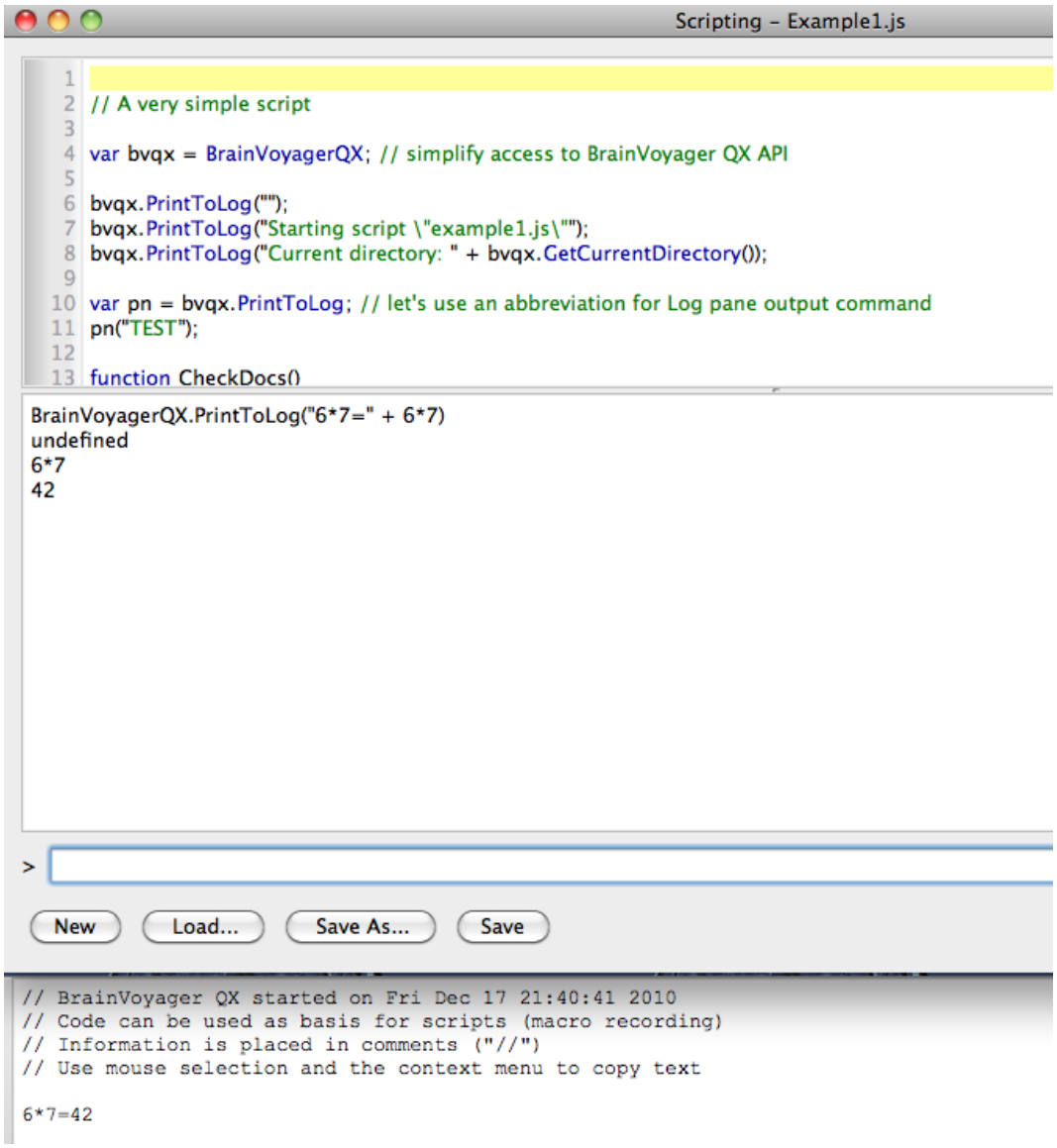

Figure 2.5: Commands typed in the command line are printed in the command history window

2.1.5 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.

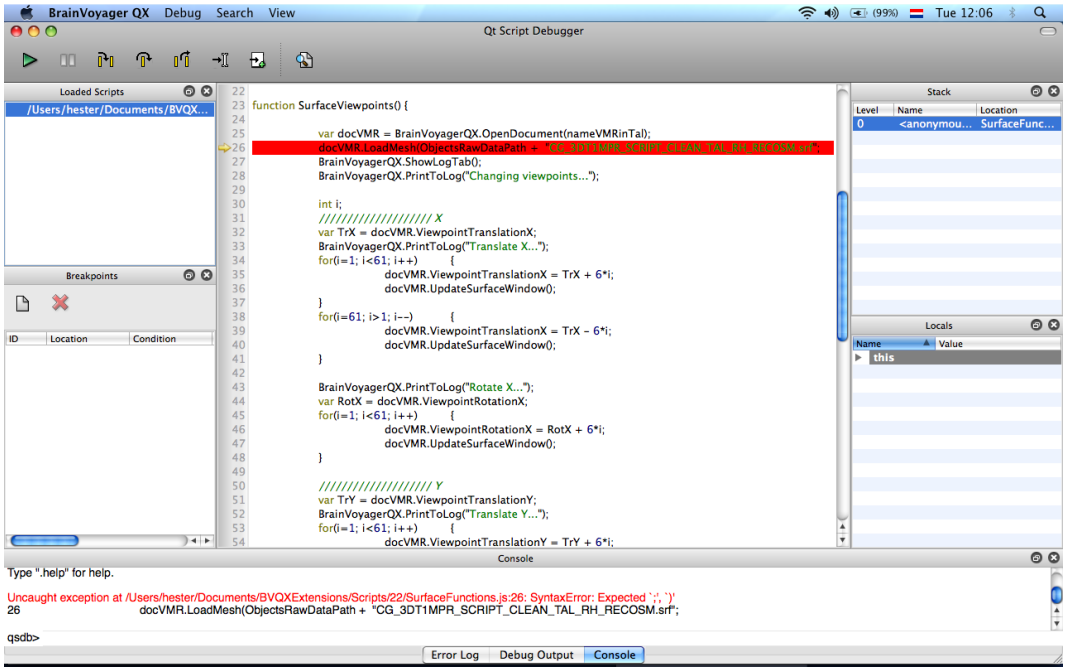

Figure 2.6: Debugging in the Scripting Editor

Chapter 3

BrainVoyager Scripting API

3.1 Introduction

In this chapter is documented which BrainVoyager scripting methods are available.

3.1.1 Invoking BrainVoyager scripting functions from other programs

On Windows, the BrainVoyager scripting functions can also be invoked from COM-enabled programs like Matlab. To use this, open the 'command prompt', go to the BrainVoyager QX directory via the DOS command `cd ..` (change directory). When in the BrainVoyager QX directory, type `BrainVoyagerQX.exe -regserver`. This declares the COM-facilities of BrainVoyager to the rest of Windows.
Then, for example in Matlab, start BrainVoyager by typing:
`bvqx = actxserver('BrainVoyagerQX.BrainVoyagerQXScriptAccess.1').`
Now the BrainVoyager script functions can be used just like they are described below. Details can be found in the "Scripting BrainVoyager QX from Matlab" guide.
The Component Object Model (COM) has been developed by Microsoft. Most applications from Microsoft are COM-enabled. Therefore, another possibility is to use the Windows Scripting Host to execute the BrainVoyager script functions. The script should be saved as a Visual Basic Script (*.vbs) in that case. An example is shown below. To run this script on Windows, one just needs to double-click the *.vbs file.

On Mac, AppleScript can be used. For a nice AppleScript guide (by Bert Altenburg), see
http://files.macscripter.net/sourcebook/AS4ASb2.pdf.

3.1.2 Example script

In this example is shown how to invoke BrainVoyager script functions from other COM-enabled programs.

```
Dim BrainVoyagerQX
Set BrainVoyagerQX = CreateObject("BrainVoyagerQX.BrainVoyagerQXScriptAccess")

'BrainVoyagerQX.TimeOutMessageBox "Welcome to COM scripting", 3

BrainVoyagerQX.PrintToLog "Hello from VBS script!"

'MeshFile = BrainVoyagerQX.BrowseFile("Select Mesh File", "*.srf")
'BrainVoyagerQX.PrintToLog MeshFile

Set docVMR = BrainVoyagerQX.OpenDocument("C:\Users\rainer\Data\CG2_3DT1FL_SINC4_TAL.vmr")
BrainVoyagerQX.TimeOutMessageBox "VMR document loaded!", 3

'Set docVMR = BrainVoyagerQX.CreateProjectVMR "DICOM",
"C:\Users\rainer\Data\BetSog_20040312_Goebel_C2 -0002-0001-0001.dcm", 192, false, 256, 256, 2
'docVMR.SaveAs "CG_3DT1MPR_SCRIPT.vmr"
docVMR.Close
```

3.2 List of all methods

RenameDicomFilesInDirectory()
PrintToLog()
ShowLogTab()
MoveWindow()
OpenDocument()
CreateProjectFMR()
LinkAMR()
CreateProjectMosaicFMR()
CreateProjectFMRSlicesTimeLooping()
CreateProjectDMR()
CreateProjectMosaicDMR()
CreateProjectVMR()
LinkStimulationProtocol()
LinkVTC()
CreateProjectAMR()
Close()
CorrectSliceTiming()
CorrectSliceTimingWithSliceOrder()
CorrectMotion()
CorrectMotionEx()
CorrectMotionTargetVolumeInOtherRun()
CorrectMotionTargetVolumeInOtherRunEx()
TemporalHighPassFilter()
LinearTrendRemoval()
TemporalGaussianSmoothing()
SpatialGaussianSmoothing()
SpatialGaussianSmoothing()
LinearTrendRemoval()
TemporalHighPassFilter()
TemporalGaussianSmoothing()
AutoTransformToIsoVoxel()
AutoTransformToSAG()
SetVoxelIntensity(x, y, z, intensity)
GetVoxelIntensity(x, y, z)
CreateVTCInVMRSpace()
CreateVTCInACPCSpace()
CreateVTCInTALSpace()
CreateVDWInVMRSpace()
CreateVDWinACPCSpace()
CreateVDWInTALSpace()
ClearStimulationProtocol()
LinkStimulationProtocol()
AddCondition()
SetConditionColor()
AddInterval()
SaveStimulationProtocol()
ClearDesignMatrix()
SetPredictorValues()
SetPredictorValuesFromCondition()
ApplyHemodynamicResponseFunctionToPredictor()
ScalePredictorValues()
SaveSingleStudyGLMDesignMatrix()

ClearMultiStudyGLMDefinition()
AddStudyAndDesignMatrix()
SaveMultiStudyGLMDefinitionFile()
LoadSingleStudyGLMDesignMatrix()
LoadMultiStudyGLMDefinitionFile()
ComputeSingleStudyGLM()
ComputeMultiStudyGLM()
ComputeRFXGLM()
LoadGLM()
ShowGLM()
SaveGLM()
ClearContrasts()
SetCurrentContrast()
SetCurrentContrastAtIndex()
AddContrast()
SetContrastValue()
SetContrastString()
SetContrastValueAtIndex()
LoadMesh()
AddMesh()
SaveMesh()
UpdateSurfaceWindow()
SaveSnapshotOfSurfaceWindow()
LinkMTC()
CreateMTCFromVTC()

3.3 BrainVoyager QX-specific classes

A BrainVoyager document in the scripting module is an FMR, VMR, DMR or AMR project. To retrieve a pointer to a BrainVoyager document object, use the property `ActiveDocument`:

```
var doc = BrainVoyagerQX.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 = BrainVoyagerQX.OpenDocument(docname);.
```

3.4 Functions of the BrainVoyagerQX object

3.4.1 List of methods

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

BrainVoyagerQX 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).
PathToData: This property points as default to the "BVQXData" folder in the user's "(My)Documents" folder (added in QX 2.3).
PathToSampleData: This property points as default to the "BVQXSampleData" folder (added in QX 2.3).

3.4.2 Detailed description of methods

RenameDicomFilesInDirectory()

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

PrintToLog()

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

ShowLogTab()

ShowLogTab()
Description: Show the BrainVoyager QX Log tab.

MoveWindow()

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

OpenDocument()

OpenDocument()
Description: Open a BrainVoyager project.
Parameter 1: Name of the FMR/AMR/VMR/DMR project to open.
Returns: The project.

3.5 Create projects

3.5.1 List of Methods

CreateProjectFMR()
LinkAMR()
CreateProjectMosaicFMR()
CreateProjectFMRSlicesTimeLooping()
CreateProjectDMR()
CreateProjectMosaicDMR()
CreateProjectVMR()
LinkStimulationProtocol()
LinkVTC()
CreateProjectAMR()
Close()

3.5.2 Create FMR project

CreateProjectFMR()

CreateProjectFMR() *Function:* CreateProjectFMR(FileType, FirstFileName, NrOfVolumes, nrOfVolumesToSkip, createAMR, nrOfSlices, prefixSTCs, swapBytes, resX, resY, nrBytes, savingDir).
Description: FMR projects consist of a set of functional data in the original "slice space".
Member of class: BrainVoyagerQX.
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 project.
Parameter 4: nrOfVolumesToSkip: number of volumes that should be skipped at the beginning, so number of volumes in FMR project 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 project.
Returns: Document

CreateProjectMosaicFMR()

CreateProjectMosaicFMR() *Description:* FMR projects 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: BrainVoyagerQX.
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 = BrainVoyagerQX.CreateProjectMosaicFMR("DICOM",
ObjectsRawDataPath + "BetSog_20040312_Goebel_C2 -0003-0001-0001.dcm", 252, 2, true,
 25, "untitled-", false, 320, 320, 2, ObjectsRawDataPath, 1, 64, 64 );
```

CreateProjectFMRSlicesTimeLooping()

CreateProjectFMRSlicesTimeLooping() *Function:* CreateProjectFMRSlicesTimeLooping(FileType, FirstFileName, NrOfVolumes, nrOfVolumesToSkip, createAMR, nrOfSlices, prefixSTCs, swapBytes, resX, resY, nrBytes, savingDir)
Description: See CreateProjectFMR(). The current function is similar to using the "Slices x time" checkbox via the BrainVoyager Create Project dialog. *Member of class:* BrainVoyagerQX.
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 project.
Parameter 4: nrOfVolumesToSkip: number of volumes that should be skipped at the beginning, so number of volumes in FMR project 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 project.
Returns: Document

LinkAMR()

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

FMR project 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 project has been set properly. Value is either true or false (boolean).

3.5.3 Create DMR projects

CreateProjectDMR()

CreateProjectDMR()
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 project 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 project should be saved.
Returns: Document

CreateProjectMosaicDMR()

Function: CreateProjectMosaicDMR()
Member of class: BrainVoyagerQX.
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 projects

CreateProjectVMR()

Function: CreateProjectVMR()
Creates an anatomical project 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 project 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 Talairach 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.

3.5.5 Create AMR projects

CreateProjectAMR()

CreateProjectAMR() *Function:* CreateProjectAMR()
Description: Creates an AMR project file. AMR projects 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 Project Wizard or Create Project 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 projects

Close()

Description: Close the current FMR/VMR/AMR/DMR project.

3.5.7 Example scripts

```
/*  CreateProjects.js
Script for BrainVoyager QX 2.1
*/

var ObjectsRawDataPath = "/Users/hester/Data/ObjectsDicomGSG/";
var DTIDataPath = "/Users/hester/Data/Human31dir/";

/* This is the code being directly executed */
// now create the projects
BrainVoyagerQX.PrintToLog("Create FMR project...");
Create_FMR_Project();
BrainVoyagerQX.PrintToLog("Create VMR project...");
Create_VMR_Project();
BrainVoyagerQX.PrintToLog("Create DMR project...");
Create_DMR_Project();
BrainVoyagerQX.PrintToLog("Create AMR project...");
Create_AMR_Project()


/* This code is only executed when invoked */
function Create_FMR_Project()
{
    var docFMR = BrainVoyagerQX.CreateProjectMosaicFMR("DICOM", ObjectsRawDataPath +
     "BetSog_20040312_Goebel_C2 -0003-0001-0001.dcm",
     252, 2, true, 25, "untitled-", false, 320, 320, 2, ObjectsRawDataPath, 1, 64, 64 );
    docFMR.SaveAs( "CG_OBJECTS_SCRIPT.fmr" );
}

function Create_VMR_Project()
{
    var docVMR = BrainVoyagerQX.CreateProjectVMR( "DICOM", ObjectsRawDataPath +
     "BetSog_20040312_Goebel_C2 -0002-0001-0001.dcm",
     192, false, 256, 256, 2 );
    docVMR.SaveAs( "CG_3DT1MPR_SCRIPT.vmr" );
}

function Create_DMR_Project()
{
var docDMR = BrainVoyagerQX.CreateProjectDMR( "DICOM",
"/Users/hester/Data/Human31dir/pimpul_070907_dti -0007-0001-00001.dcm",
31, 0, true, 23, "human31dir", false, 128, 128, 2, "/Users/hester/Data/Human31dir" );
docDMR.SaveAs("HUMAN31DIR_SCRIPT.dmr");
}

function Create_AMR_Project()
{
    var docAMR = BrainVoyagerQX.CreateProjectAMR( "DICOM",
     ObjectsRawDataPath + "BetSog_20040312_Goebel_C2 -0001-0001-0001.dcm",
     3, false, 256, 256, 2 );
    docAMR.SaveAs("CG_SLICELOCALIZER_SCRIPT.amr");
}
```

3.6 BVQX Project functions: Preprocessing of functional data (FMR)

3.6.1 List of methods

CorrectSliceTiming()
CorrectSliceTimingWithSliceOrder()
CorrectMotion()
CorrectMotionEx()
CorrectMotionTargetVolumeInOtherRun()
CorrectMotionTargetVolumeInOtherRunEx()
TemporalHighPassFilter()
LinearTrendRemoval()
TemporalGaussianSmoothing()
SpatialGaussianSmoothing()

3.6.2 Detailed description of methods

CorrectSliceTiming()

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 projects because these projects 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 project and the new STC prefix is determined as in the GUI version, i.e., if the FMR project "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`.

CorrectSliceTimingWithSliceOrder()

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`.
Example: The following string specifies the same order as the "ascending interleaved" option using cubic spline interpolation:

```
docFMR.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);
```

CorrectMotion()

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. 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 log file. 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_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.
Returns: True or false (boolean).
Discussion: The applied method for this function is trilinear detection and sinc interpolation.

CorrectMotionEx()

CorrectMotionEx()
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. 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 (default in GUI).
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()

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 project to which the current project should be aligned.
Parameter 2: Target volume number: The number of the volume within that run to which the project should be aligned.

CorrectMotionTargetVolumeInOtherRunEx()

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 project 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).

TemporalHighPassFilter()

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

LinearTrendRemoval()

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

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()

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

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

/* This code will be executed when clicking 'Run' */
Preprocess_FMR();
//MotionCorrectionISA();
//var fmrname = MocoISAPath  + "FFA_localizer_2/series0003_SCLAI2.fmr";
//MotionCorrection(fmrname, 1);

/* These functions can be invoked */
function Preprocess_FMR()
{
    var ret = BrainVoyagerQX.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;

    // Create a new FMR or open a previously created one.  Here we open the "CG_OBJECTS_SCRIPT.fmr" file
    var docFMR = BrainVoyagerQX.OpenDocument(ObjectsRawDataPath  + "CG_OBJECTS_SCRIPT.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;
docFMR.InterSliceTime = 80;
docFMR.TimeResolutionVerified = true;
    }
    if( !docFMR.VoxelResolutionVerified )
    {
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 dcoument)
    docFMR.LinkStimulationProtocol( "CG_OBJECTS.prt" );

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

    //
    // Preprocessing step 1: Slice time correction
    ret = BrainVoyagerQX.TimeOutMessageBox("Preprocessing step 1: Slice time correction.\n\n" +
    " To skip this step, press the 'ESCAPE' key.", 5);
    if(ret)
    {
docFMR.CorrectSliceTiming( 1, 0 ); // 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 = BrainVoyagerQX.OpenDocument( ResultFileName );
 }

 // Preprocessing step 2: 3D motion correction
 ret = BrainVoyagerQX.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;  // the current doc (input FMR) knows
  // the name of the automatically saved output FMR
  docFMR.Close();            // close input FMR
  docFMR = BrainVoyagerQX.OpenDocument( ResultFileName ); // Open motion corrected file (output FMR)
  // and assign to our doc variable
 }

 // Preprocessing step 3: Spatial Gaussian Smoothing
 // (not recommended for individual analysis with a 64x64 matrix)
 ret = BrainVoyagerQX.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 = BrainVoyagerQX.OpenDocument( ResultFileName );
 }

 // Preprocessing step 4: Temporal High Pass Filter, includes Linear Trend Removal
 ret = BrainVoyagerQX.TimeOutMessageBox("Preprocessing step 4: Temporal high-pass filter.\n\n" +
 " To skip this step, press the 'ESCAPE' key.", 5);
 if(ret) {
   docFMR.TemporalHighPassFilter( 3, "cycles" );
   ResultFileName = docFMR.FileNameOfPreprocessdFMR;
   docFMR.Close();           // close input FMR
   docFMR = BrainVoyagerQX.OpenDocument( ResultFileName );
 }

 // Preprocessing step 5: Temporal Gaussian Smoothing  (not recommended for event-related data)
 ret = BrainVoyagerQX.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 = BrainVoyagerQX.OpenDocument( ResultFileName );
 }

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

function MotionCorrectionISA()
{

var docFMR = BrainVoyagerQX.OpenDocument(MocoISAPath  + "FFA_localizer_2/series0003_SCLAI2.fmr");
// docFMR.CorrectMotionTargetVolumeInOtherRun(MocoISAPath + "FFA_localizer_1/series0002_SCCAI2_3DMCT.fmr", 1);
docFMR.CorrectMotionTargetVolumeInOtherRunEx(MocoISAPath + "FFA_localizer_1/series0002_SCCAI2_3DMCT.fmr",
1, 1, 1, 100, 0, 1 );
}

function MotionCorrection(fmrname, targetvolume)
{
var docFMR = BrainVoyagerQX.OpenDocument(fmrname);
  docFMR.CorrectMotion(targetvolume);

  // for intra-session motion correction use this command (with approprate file name):
     // docFMR.CorrectMotionTargetVolumeInOtherRun("run1.fmr", 1);

 var ResultFileName = docFMR.FileNameOfPreprocessdFMR;
  docFMR.Close();
docFMR = BrainVoyagerQX.OpenDocument( ResultFileName );
}
```

3.7 BVQX Project functions: Preprocessing of functional data (VTC)

3.7.1 List of methods

SpatialGaussianSmoothing()
LinearTrendRemoval()
TemporalHighPassFilter()
TemporalGaussianSmoothing()

3.7.2 Detailed description of methods

SpatialGaussianSmoothing()

Function: SpatialGaussianSmoothing()
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()
Description: Removes temporal linear trends in VTC file.

TemporalHighPassFilter()

Function: TemporalHighPassFilter()
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()
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 to 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 = BrainVoyagerQX.ActiveDocument;

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

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

BrainVoyagerQX.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")
//BrainVoyagerQX.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))
BrainVoyagerQX.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 BVQX Project functions: Transformations and Normalization

3.8.1 List of methods

AutoTransformToIsoVoxel()
AutoTransformToSAG()
SetVoxelIntensity(x, y, z, intensity)
GetVoxelIntensity(x, y, z)
CreateVTCInVMRSpace()
CreateVTCInACPCSpace()
CreateVTCInTALSpace()
CreateVDWInVMRSpace()
CreateVDWinACPCSpace()
CreateVDWInTALSpace()

3.8.2 Detailed description of methods

Transforming VMR files

AutoTransformToIsoVoxel()

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.

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 3.1).
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.

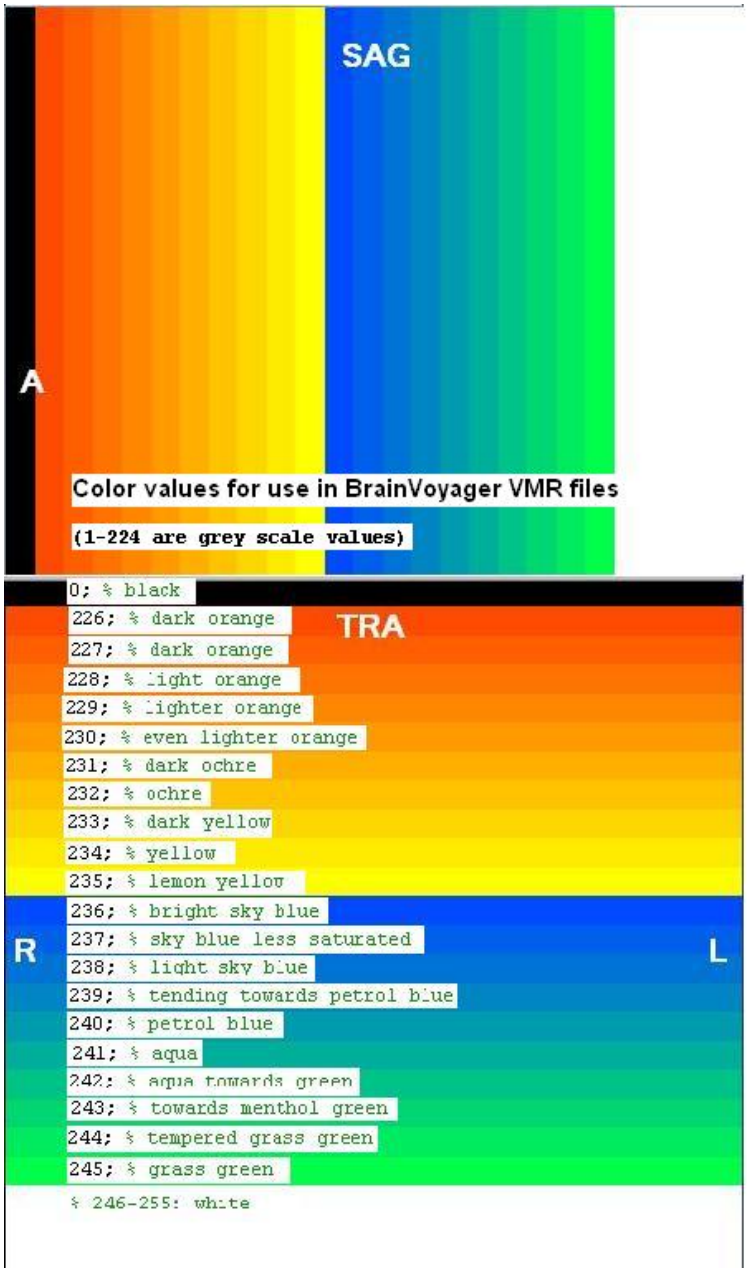

Figure 3.1: Intensity values for an anatomical file (*.vmr)

CreateVTCInVMRSpace()

CreateVTCInVMRSpace() *Function:* CreateVTCInVMRSpace()
Description: Transforms the time course data of an FMR project into a defined 3D space, in this case native space. The result of this transformation is a VTC file.
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()

CreateVTCInACPCSpace() *Function:* CreateVTCInACPCSpace()
Description: Create a volume-time-course (VTC) file in AC-PC space.
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()

CreateVTCInTALSpace() *Function:* CreateVTCInTALSpace()
Description: Valid only if the opened document is of type VMR. FMR projects 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 project whose functional data should be transformed in 3D space (the functional data actually resides in STC files which are referenced by the FMR project). 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 QX 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.
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.

3.8.3 Example script to create VTC files

```
/* This information is used in the functions */
var ObjectsRawDataPath = "/Users/hester/Data/ObjectsDicomGSG/";
var DTIDataPath = "/Users/hester/Data/Human31dir/";
var today = new Date();
var nameFMR = ObjectsRawDataPath + "CG_OBJECTS_SCRIPT_SCLAI_3DMCTS_SD3DSS4.00mm_LTR_THP3c_TDTS10.0s.fmr";
var nameVMRinNative = ObjectsRawDataPath +"CG_3DT1MPR_SCRIPT.vmr";
var nameVMRinAcpc = ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_ACPC.vmr";
var nameVMRinTal = ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_TAL.vmr";
var nameIAfile = ObjectsRawDataPath +
"CG_OBJECTS_SCRIPT_SCLAI_3DMCTS_SD3DSS4.00mm_LTR_THP3c_TDTS10.0s-TO-CG_3DT1MPR_SCRIPT_IA.trf";
var nameFAfile = ObjectsRawDataPath +
"CG_OBJECTS_SCRIPT_SCLAI_3DMCTS_SD3DSS4.00mm_LTR_THP3c_TDTS10.0s-TO-CG_3DT1MPR_SCRIPT_FA.trf";
var nameACPCfile = ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_ACPC.trf";
var nameTALfile = ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_ACPC.tal";
var nameVTCinNative= ObjectsRawDataPath + "CG_OBJECTS_SCRIPT_VMR.vtc";
var nameVTCinACPC = ObjectsRawDataPath + "CG_OBJECTS_SCRIPT_ACPC.vtc";
var nameVTCinTAL= ObjectsRawDataPath + "CG_OBJECTS_SCRIPT_TAL.vtc";
var dataType = 2; // 1: int16, 2: float32
var resolution = 2; // one of 1, 2 or 3 mm^2
var interpolation = 1;
var threshold = 100;
var extendedBoundingBox = false;

/*  This code is executed when clicking 'Run'
If an error occurs, it is printed to the BrainVoyager QX Log tab
*/
BrainVoyagerQX.ShowLogTab();
BrainVoyagerQX.PrintToLog("Start creating VTCs...");
try {
CreateVMRinNativeSpace();
CreateVMRinAcpcSpace();
// make a VTC in Talairach space with extended bounding box = false
CreateVMRinTalairachSpace(extendedBoundingBox);
extendedBoundingBox = true;
// rename, so that the other file won't be overwritten
var nameVTCinTAL= ObjectsRawDataPath + "CG_OBJECTS_SCRIPT_TAL_ext.vtc";
// make a VTC in Talairach space with extended bounding box including cerebellum (z-dir)
CreateVMRinTalairachSpace(extendedBoundingBox);
} catch (e) {
BrainVoyagerQX.PrintToLog("Error: " + e);
}

/* These functions are invoked from the section above */
function CreateVMRinNativeSpace()
{
var docVMR = BrainVoyagerQX.OpenDocument(nameVMRinNative);
docVMR.ExtendedTALSpaceForVTCCreation = false; // this is true or false
var success = docVMR.CreateVTCInVMRSpace(nameFMR, nameIAfile, nameFAfile,
nameVTCinNative, dataType, resolution, interpolation, threshold);
docVMR.Close();
}

function CreateVMRinAcpcSpace()
{
var docVMR = BrainVoyagerQX.OpenDocument(nameVMRinAcpc);
docVMR.ExtendedTALSpaceForVTCCreation = false; // this is true or false
var success = docVMR.CreateVTCInACPCSpace(nameFMR, nameIAfile, nameFAfile, nameACPCfile, nameVTCinACPC,
dataType, resolution, interpolation, threshold);
docVMR.Close();
}

function CreateVMRinTalairachSpace(useExtendedBoundingBox)
{
var docVMR = BrainVoyagerQX.OpenDocument(nameVMRinTal);
docVMR.ExtendedTALSpaceForVTCCreation = useExtendedBoundingBox; // this is true or false
var success = docVMR.CreateVTCInTALSpace(nameFMR, nameIAfile, nameFAfile, nameACPCfile, nameTALfile,
nameVTCinTAL, dataType, resolution, interpolation, threshold);
docVMR.Close();
}
```

3.8.4 Creating diffusion weighted (VDW) files

CreateVDWInVMRSpace()

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()

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()

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.

3.8.5 Example script to create VDW files

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

/* This information is used in the functions */
var DTIDataPath  = "/Users/hester/Data/Human31dir/";
var today  = new Date();
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";
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
*/
BrainVoyagerQX.ShowLogTab();
BrainVoyagerQX.PrintToLog("Start creating VDWs...");
try {
//CreateVDWinNativeSpace();
//CreateVDWinAcpcSpace();
CreateVDWinTalSpace();
} catch (e) {
BrainVoyagerQX.PrintToLog("Error: " + e);
}

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

BrainVoyagerQX.TimeOutMessageBox("This function creates a VDW file in native space...", 5);
var vmr = BrainVoyagerQX.OpenDocument(nameVMRinNative);
BrainVoyagerQX.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) {
BrainVoyagerQX.PrintToLog("Created VDW: " + nameVDWinNative);
} else {
BrainVoyagerQX.PrintToLog("VDW creation did not succeed.");
}
  vmr.Close();
}

function CreateVDWinAcpcSpace() {

BrainVoyagerQX.TimeOutMessageBox("This function creates a VDW file in AC-PC space...", 5);
var vmr = BrainVoyagerQX.OpenDocument(nameVMRinAcpc);
BrainVoyagerQX.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) {
BrainVoyagerQX.PrintToLog("Created VDW: " + nameVDWinAcpc);
} else {
BrainVoyagerQX.PrintToLog("VDW creation did not succeed.");
}
  vmr.Close();
}

function CreateVDWinTalSpace() {

BrainVoyagerQX.TimeOutMessageBox("This function creates a VDW file in Talairach space...", 5);
var vmr = BrainVoyagerQX.OpenDocument(nameVMRinTal);
BrainVoyagerQX.PrintToLog("Start creating VDW in Talairach space...");
// vmr.ExtendedTALSpaceForVTCCreation = false; // this is true or false
var success = vmr.CreateVDWInTALSpace(nameDMR, nameIAfile, nameFAfile, nameACPCfile,
nameTALfile, nameVDWinTal, dataType, resolution, interpolation, threshold);
```

```
if (success) {
BrainVoyagerQX.PrintToLog("Created VDW: " + nameVDWinTal);
} else {
BrainVoyagerQX.PrintToLog("VDW creation did not succeed.");
}
  vmr.Close();
}
```

3.9 BVQX Project functions: Experimental design

3.9.1 List of Methods for stimulation protocols

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

3.9.2 List of Properties for stimulation protocols

StimulationProtocolFile: String
StimulationProtocolExperimentName: String
StimulationProtocolResolution: Number
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()

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

LinkStimulationProtocol()

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()

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

SetConditionColor()

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 3.2.
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)

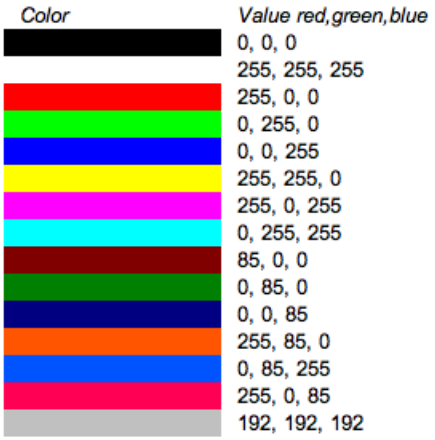

Figure 3.2: Example colors for protocol conditions

AddInterval()

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()

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

3.9.4 Example script

Creating a stimulation protocol

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

```
MakeProtocol(); // invoked function below

function MakeProtocol() {

var interval, intervalstart, intervalend;
BrainVoyagerQX.PrintToLog("Create stimulation protocol...");
var doc = BrainVoyagerQX.ActiveDocument;
// = BrainVoyagerQX.OpenDocument( ObjectsRawDataPath + "CG2_3DT1FL_SINC4_TAL.vmr" );
doc.ClearStimulationProtocol();
doc.StimulationProtocolExperimentName = "Objects Experiment";
doc.StimulationProtocolResolution = 1;
doc.AddCondition("Fixation");
doc.AddCondition("Objects in LVF");
doc.AddCondition("Objects in RVF");
doc.AddCondition("Objects in BVF");
for (interval = 0; interval < 10; interval++) {
intervalstart = interval*27 -1;
intervalend = intervalstart + 8;
if (interval == 0) {
intervalstart = 1; intervalend = 7;
}
doc.AddInterval("Fixation", intervalstart, intervalend);
}
for (interval = 0; interval < 3; interval++) {
intervalstart = 35+interval*81;
intervalend = intervalstart + 17;
doc.AddInterval("Objects in LVF", intervalstart, intervalend);
}
for (interval = 0; interval < 3; interval++) {
intervalstart = 8+interval*81;
intervalend = intervalstart + 17;
doc.AddInterval("Objects in RVF", intervalstart, intervalend);
}
for (interval = 0; interval < 3; interval++) {
intervalstart = 62+interval*81;
intervalend = intervalstart + 17;
doc.AddInterval("Objects in BVF", intervalstart, intervalend);
}
doc.SetConditionColor("Fixation", 100, 100, 100);
doc.SetConditionColor("Objects in LVF", 255, 0, 0);
doc.SetConditionColor("Objects in RVF", 0, 255, 0);
doc.SetConditionColor("Objects in BVF", 0, 0, 255);
doc.StimulationProtocolBackgroundColorR = 0;
doc.StimulationProtocolBackgroundColorG = 0;
doc.StimulationProtocolBackgroundColorB = 0;
doc.StimulationProtocolTimeCourseColorR = 255;
doc.StimulationProtocolTimeCourseColorG = 255;
doc.StimulationProtocolTimeCourseColorB = 255;
doc.StimulationProtocolTimeCourseThickness = 4;
doc.SaveStimulationProtocol("CG_OBJECTS_FROMSCRIPT.prt"); //Users/hester/Data/bvqxdata/
doc.Save(); // to save link to protocol permanently
}
```

3.9.5 List of Methods for design matrices

ClearDesignMatrix()
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()

ClearDesignMatrix()
Description: Removes any current design matrices.

SetPredictorValues()

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()

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()

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 QX User's Guide).
Parameter 1: Name of the condition that should be convolved with the HRF.

ScalePredictorValues()

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()

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

ClearMultiStudyGLMDefinition()

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

AddStudyAndDesignMatrix()

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()

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

3.9.8 Example scripts

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

```
/* Declarations */
var ObjectsRawDataPath = "/Users/hester/Data/ObjectsDicomGSG/";
var RFXDataPath = "/Users/hester/Data/Exercise_RFX_ANOVA/";

/* This code is executed when clicking 'Run' */
CreateDesignMatrix();
CreateMultiStudyDesignMatrix();

function CreateDesignMatrix()
{
BrainVoyagerQX.PrintToLog("Create single subject design matrix...");
var doc = BrainVoyagerQX.OpenDocument(ObjectsRawDataPath + "CG2_3DT1FL_SINC4_TAL.vmr" );
doc.LinkVTC(ObjectsRawDataPath + "CG_OBJECTS_SCRIPT_TAL.vtc");
doc.LinkStimulationProtocol(ObjectsRawDataPath + "CG_OBJECTS_FROMSCRIPT.prt");
doc.ClearDesignMatrix();

doc.AddPredictor("LVF");
doc.SetPredictorValuesFromCondition("LVF", "Objects in LVF", 1.0);
doc.ApplyHemodynamicResponseFunctionToPredictor("LVF");

doc.AddPredictor("RVF");
doc.SetPredictorValuesFromCondition("RVF", "Objects in RVF", 1.0);
doc.ApplyHemodynamicResponseFunctionToPredictor("RVF");

doc.AddPredictor("BVF");
doc.SetPredictorValuesFromCondition("BVF", "Objects in BVF", 1.0);
doc.ApplyHemodynamicResponseFunctionToPredictor("BVF");

// 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.SaveSingleStudyGLMDesignMatrix("CG_OBJECTS_FROMSCRIPT.rtc");
doc.SaveSingleStudyGLMDesignMatrix("CG_OBJECTS_FROMSCRIPT.sdm");
}

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

}
```

3.10 BVQX Project functions: Statistics

3.10.1 List of Methods

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

3.10.2 List of Properties

CorrectForSerialCorrelations: Boolean (true or false). If true, the AR(1) correction method will be applied.
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()

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

LoadMultiStudyGLMDefinitionFile()

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

ComputeSingleStudyGLM()

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

ComputeMultiStudyGLM()

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

ComputeRFXGLM()

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

LoadGLM()

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

ShowGLM()

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

SaveGLM()

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

Setting contrasts

ClearContrasts()

ClearContrasts()
Description: Remove any current contrasts.

SetCurrentContrast()

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.

SetCurrentContrastAtIndex()

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.

AddContrast()

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

SetContrastValue()

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

SetContrastString()

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"

SetContrastValueAtIndex()

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.

3.10.4 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.

```
/*  Statistics.js
Script for BrainVoyager QX 2.2
*/
/* Declarations */
var ObjectsRawDataPath = "/Users/hester/Data/ObjectsDicomGSG/";
var RFXDataPath = "/Users/hester/Data/Exercise_RFX_ANOVA/";

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

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

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

/* These functions can be invoked */
function RunSingleSubjectGLM()
{
try {
var ObjectsRawDataPath = "/Users/hester/Data/ObjectsDicomGSG/";
var doc = BrainVoyagerQX.OpenDocument(ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_CLEAN_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 = true;
doc.ComputeSingleStudyGLM();
doc.ShowGLM();
doc.SaveGLM(ObjectsRawDataPath + "CG_OBJECTS_FROMSCRIPT.glm");
} catch (error) {
BrainVoyagerQX.PrintToLog(error);
}
}

function RunMultiSubjectGLM()
{
var doc = BrainVoyagerQX.OpenDocument(RFXDataPath + "Average_Tal.vmr");
doc.CorrectForSerialCorrelations = false;
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 = BrainVoyagerQX.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 = BrainVoyagerQX.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();
}
```

3.11 BVQX Project functions: Surface functions

These functions can be invoked using an anatomical volume (VMR).

3.11.1 List of Methods

LoadMesh()
AddMesh()
SaveMesh()
UpdateSurfaceWindow()
SaveSnapshotOfSurfaceWindow()
LinkMTC()
CreateMTCFromVTC()

3.11.2 List of Properties

ViewpointTranslationX
ViewpointRotationX
ViewpointTranslationY
ViewpointRotationY
ViewpointTranslationZ
ViewpointRotationZ

3.11.3 Detailed description of methods

LoadMesh()

LoadMesh() *Description:* Load a mesh into the current scene.
Parameter 1: Name of the polygon mesh (*.srf).

AddMesh()

AddMesh() *Description:* Add a mesh to the current scene.
Parameter 1: Name of the polygon mesh (*.srf).

SaveMesh()

SaveMesh() *Description:* Save the current mesh.
Parameter 1: Name for the polygon mesh (*.srf).

UpdateSurfaceWindow()

UpdateSurfaceWindow() *Description:* After changing the viewpoint settings or loading meshes, invoke this function to see the effect.

SaveSnapshotOfSurfaceWindow()

SaveSnapshotOfSurfaceWindow() *Description:* Save a snapshot of the current OpenGL window.
Parameter 1: Name for the snapshot.
Note: The file type of the snapshot is determined the filetype selected in BrainVoyager → Preferences.

LinkMTC()

Description: Link a mesh time series file to a surface file (*.srf).
Parameter 1: Name of the mesh time course file including a path.

CreateMTCFromVTC()

CreateMTCFromVTC() *Description:* Create a mesh time series to be displayed on 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”.

3.11.4 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.js
Script to visualize surface files
*/

/* This information is used in the functions */
var ObjectsRawDataPath = "/Users/hester/Data/ObjectsDicomGSG/";
var nameVMRinTal = ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_CLEAN_TAL.vmr";

/* This code will be executed when clicking 'Run' */
LoadVMRandSRF();
SurfaceViewpoints();

/* These functions can be invoked */
function LoadVMRandSRF()
{
var docVMR = BrainVoyagerQX.OpenDocument(nameVMRinTal);
docVMR.LoadMesh(ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_CLEAN_TAL_LH_RECOSM.srf");
docVMR.AddMesh(ObjectsRawDataPath + "CG_3DT1MPR_SCRIPT_CLEAN_TAL_RH_RECOSM.srf");
docVMR.Close();
}

function SurfaceViewpoints() {

var docVMR = BrainVoyagerQX.OpenDocument(nameVMRinTal);
docVMR.LoadMesh(ObjectsRawDataPath +  "CG_3DT1MPR_SCRIPT_CLEAN_TAL_RH_RECOSM.srf");
BrainVoyagerQX.ShowLogTab();
BrainVoyagerQX.PrintToLog("Changing viewpoints...");

var i;
//////////////////// X
var TrX = docVMR.ViewpointTranslationX;
BrainVoyagerQX.PrintToLog("Translate X...");
for(i=1; i<61; i++) {
docVMR.ViewpointTranslationX = TrX + 6*i;
docVMR.UpdateSurfaceWindow();
}
for(i=61; i>1; i--) {
docVMR.ViewpointTranslationX = TrX - 6*i;
docVMR.UpdateSurfaceWindow();
}

BrainVoyagerQX.PrintToLog("Rotate X...");
var RotX = docVMR.ViewpointRotationX;
for(i=1; i<61; i++) {
docVMR.ViewpointRotationX = RotX + 6*i;
docVMR.UpdateSurfaceWindow();
}

//////////////////// Y
var TrY = docVMR.ViewpointTranslationY;
BrainVoyagerQX.PrintToLog("Translate Y...");
for(i=1; i<61; i++) {
docVMR.ViewpointTranslationY = TrY + 6*i;
docVMR.UpdateSurfaceWindow();
}
BrainVoyagerQX.PrintToLog("Rotate Y...");
for(i=61; i>1; i--) {
docVMR.ViewpointTranslationY = TrY - 6*i;
docVMR.UpdateSurfaceWindow();
}
var RotY = docVMR.ViewpointRotationY;
for(i=1; i<61; i++) {
docVMR.ViewpointRotationY = RotY + 6*i;
docVMR.UpdateSurfaceWindow();
}

//////////////////// Z
var TrZ = docVMR.ViewpointTranslationZ;
BrainVoyagerQX.PrintToLog("Translate Z...");
for(i=1; i<61; i++) {
docVMR.ViewpointTranslationZ = TrZ + 6*i;
docVMR.UpdateSurfaceWindow();
}
```

```
for(i=61; i>1; i--) {
docVMR.ViewpointTranslationZ = TrZ - 6*i;
docVMR.UpdateSurfaceWindow();
}

BrainVoyagerQX.PrintToLog("Rotate Z...");
var RotZ = docVMR.ViewpointRotationZ;
for(i=1; i<61; i++) {
docVMR.ViewpointRotationZ = RotZ + 6*i;
docVMR.UpdateSurfaceWindow();
}

docVMR.SaveSnapshotOfSurfaceWindow(ObjectsRawDataPath + "Surface.png");
BrainVoyagerQX.PrintToLog("Finished.");
}
```

3.11.5 Example script: create MTC

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

```
var path = "C:/Data/bvqxdata/CBA_RFX_June2009/subj_AA/";
var docVMR = BrainVoyagerQX.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" );
```

Chapter 4

File I/O

4.1 Class QFile

4.1.1 List of methods

QFile()
open()
close()

QFile properties

4.1.2 Detailed description of methods

open()

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 4.4).

close()

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

4.2 Class QIODevice

4.2.1 List of methods

QIODevice properties

WriteOnly: ....

4.2.2 Detailed description of methods

4.3 Class QTextStream

4.3.1 List of methods

writeString()
writeDouble()

4.3.2 Detailed description of methods

QTextStream()

QTextStream() *Description:* Create a textstream object.

writeString()

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

writeInt()

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

writeDouble()

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

4.4 Example script: write a file I

```
// 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();
```

4.5 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";

     BrainVoyagerQX.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();
    BrainVoyagerQX.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;
}
```

4.6 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/hester/Data/testdata/filenames.txt");
var projfls = this.readTextFile(filename);


function readTextFile(name) {

    var projfilenames = new Array();
    var filecounter;
    BrainVoyagerQX.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();
        BrainVoyagerQX.PrintToLog(filename);
        projfilenames.push(filename);
    }
     filenamesfile.close();
     return projfilenames;
}
```

4.7 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).
Thanks to Dirk Heslenfeld

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

function Remove(file)
{
  BrainVoyagerQX.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

Switching from BrainVoyager QX 2.0 to 2.1 scripting

The scripting module in BrainVoyager QX 2.1 uses the new script module from Qt. The previous language was called QSA. The script language in BrainVoyager QX 2.1 in called Qt Script. The differences between the languages - as described by Nokia (see `http://qt.nokia.com`) - can be found in this chapter.

5.1 The Scripting Language

The scripting language used in QSA, from here on referred to as QSA, was derived from ECMAScript 3.0 and 4.0 and is a hybrid of these standards. Most of the run-time logic, such as classes and scoping rules, is based on the ECMAScript 4.0 proposal, while the library implementation is based on the ECMAScript 3.0 standard. Qt Script on the other hand is solely based on the ECMAScript 3.0 standard. Though the languages look identical at first glance, there are a few differences that we'll cover in the sections below.

5.1.1 Classes vs. Objects and Properties

QSA implements classes and inheritance much in a familiar way to users of other object oriented languages, like C++ and Java. However, the ECMAScript 3.0 standard defines that everything is an object, and objects can have named properties. For instance to create an point object with the properties x and y one would write the following Qt Script code:

```
point = new Object();
point.x = 12;
point.y = 35;
```

The object point in this case is constructed as a plain object and we assign two properties, x and y, to it with the values 12 and 35. The point object is assigned to the "Global Object" as the named property point. The global object can be considered the global namespace of the script engine. Similarly, global functions are named properties of the global object; for example:

```
function manhattanLength(point) {
```

```
    return point.x + point.y;
}
```

An equivalent construction that illustrates that the function is a property of the global object is the following assignment:

```
manhattanLength = function(point) {
   return point.x + point.y;
}
```

Since functions are objects, they can be assigned to objects as properties, becoming member functions:

```
point.manhattanLength = function() {
    return this.x + this.y;
}
```

print(point.manhattanLength()); // prints 47 In the code above, we see the first subtle difference between QSA and Qt Script. In QSA one would write the point class like this:

```
class Point() {
    var x;
    var y;
    function manhattanLength() { return x + y; }
}
```

where in the manhattanLength() function we access x and y directly because, when the function is called, the this object is implicitly part of the current scope, as in C++. In Qt Script, however, this is not the case, and we need to explicitly access the x and y values via this.

All the code above runs with QSA except the assignment of a function to point.manhattanLength, which we repeat here for clarity:

```
point.manhattanLength = function() {
    return this.x + this.y;
}
print(point.manhattanLength()); // prints 47
```

This is because, in QSA, the value of this is decided based on the location of the declaration of the function it is used in. In the code above, the function is assigned to an object, but it is declared in the global scope, hence there will be no valid this value. In Qt Script, the value of this is decided at run-time, hence you could have assigned the manhattanLength() function to any object that had x and y values.

5.1.2 Constructors

In the code above, we use a rather awkward method for constructing the objects, by first instantiating them, then manually assigning properties to them. In QSA, the proper way to solve this is to implement a constructor in the class:

```
class Car {
    var regNumber;
    function Car(regnr) {
        regNumber = regnr;
    }
}
var car = new Car("ABC 123");
```

The equivalent in Qt Script is to create a constructor function:

```
function Car(regnr) {
    this.regNumber = regnr;
}
var car = new Car("ABC 123");
```

As we can see, the constructor is just a normal function. What is special with is how we call it, namely prefixed with the new keyword. This will create a new object and call the Car() function with the newly created object as the this pointer. So, in a sense, it is equivalent to:

```
var car = new Object();
car.constructor = function(regnr) { ... }
car.constructor();
```

This is similar to the `manhattenLength()` example above. Again, the main difference between QSA and Qt Script is that one has to explicitly use the keyword this to access the members and that instead of declaring the variable, regNumber, we just extend the this object with the property.

5.1.3 Member Functions and Prototypes

As we saw above, one way of creating member functions of a Qt Script object is to assign the member function to the object as a property and use the this object inside the functions. So, if we add a toString function to the Car class

```
class Car {
    var regNumber;
    function Car(regnr) {
        regNumber = regnr;
    }
    function toString() {
        return regNumber;
    }
}
```

one could write this in Qt Script as:

```
function Car(regnr) {
    this.regNumber = regnr;
    this.toString = function() { return this.regNumber; }
}
```

In QSA, the member functions were part of the class declaration, and were therefore shared between all instances of a given class. In Qt Script, each instance has a instance member for each function. This means that more memory is used when multiple instances are used. Qt Script uses prototypes to remedy this.

The basic prototype-based inheritance mechanism works as follows. Each Qt Script object has an internal link to another object, its prototype. When a property is looked up in an object, and the object itself does not have the property, the interpreter searches for the property in the prototype object instead; if the prototype has the property then that property is returned. If the prototype object does not have the property, the interpreter searches for the property in the prototype of the prototype object, and so on.

This chain of objects constitutes a prototype chain. The chain of prototype objects is followed until the property is found or the end of the chain is reached.

To make the toString() function part of the prototype, we write code like this:

```
function Car(regnr) {
    this.regNumber = regnr;
}
Car.prototype.toString = function() { return this.regNumber; }
```

Here, we made the toString() function part of the prototype so that, when we call car.toString() it will be resolved via the internal prototype object of the car object. Note, however, that the this object is still the original object that the function was called on, namely car.

5.1.4 Inheritance

Now that we've seen how to use prototypes to create a "class" members in Qt Script, let's see how we can use prototypes to create polymorphism. In QSA you would write

```
class GasolineCar extends Car {
    function GasolineCar(regnr) {
        Car(regnr);
    }
    function toString() {
        return "GasolineCar(" + regNumber + ")";
    }
}
```

With Qt Script, we acheive the same effect by creating a prototype chain. The default prototype of an object is a plain Object without any special members, but it is possible to replace this object with another prototype object.

```
function GasolineCar(regnr) {
    Car(regnr);
}
GasolineCar.prototype = new Car();
GasolineCar.prototype.toString = function() {
    return "GasolineCar(" + this.regNumber + ")";
}
```

In the code above, we have a constructor, GasolineCar, which calls the "base class" implementation of the constructor to initialize the this object with the property regNumber, based on the values passed in the constructor. The interesting line in this case is the line after the constructor where we change the default prototype for GasolineCar to be an instance of type Car. This means that all members available in a Car object are now available in all GasolineCar objects. In the last line, we replace the toString() function in the prototype with our own, thus overriding the toString() for all instances of GasolineCar.

5.1.5 Static Members

QSA allowed users to declare static members in classes, and these could be accessed both through instances of the class and through the class itself. For example, the following variable is accessed through the Car class:

```
class Car {
    static var globalCount = 0;
}
print(Car.globalCount);
```

The equivalent in Qt Script is to assign variables that should appear as static members as properties of the constructor function. For example:

```
Car.globalCount = 0;
print(Car.globalCount);
```

Note that in QSA, static member variables were also accessible in instances of the given class. In Qt Script, with the approach illustrated above, the variable is a member of the constructor object only, and thus only accessible through Car.globalCount.

5.2 The Built-in Functions and Library

The built-in functions in QSA are based on those defined in the ECMAScript 3.0 standard, the same standard used for Qt Script, but QSA adds some extensions to this, specifically for the String and RegExp types. QSA also lacked some functions from the standard, most notably the Date type. Below we list all the differences. All changes made to Qt Script are to increase compliance with ECMAScript 3.0.

QSA Function and notes about Equivalent Qt Script Functions `eval()` The eval function in QSA opened a new scope for code being executed in the eval function, so locally declared variables were not accessible outside. In Qt Script, the `eval()` function shares the current scope, making locally declared variables accessible outside the `eval()` call.
`debug()` This function is not available in Qt Script. Use print() instead.
`connect()` QSA had closures, meaning that a member function reference implicitly contained its this object. Qt Script does not support this. See the Qt Script documentation for details on using the connect function.
`String.arg()` This function is not available in Qt Script. Use replace() or concat() instead.
`String.argDec()` This function is not available in Qt Script. Use replace() or concat() instead.
`String.argInt()` This function is not available in Qt Script. Use replace() or concat() instead.

`String.argStr()` This function is not available in Qt Script. Use replace() or concat() instead.
`String.endsWith()` This function is not available in Qt Script. Use lastIndexOf() instead.
`String.find()` This function is not available in Qt Script. Use indexOf() instead.
`String.findRev()` This function is not available in Qt Script. Use lastIndexOf() and length instead.
`String.isEmpty()` This function is not available in Qt Script. Use length == 0 instead.
`String.left()` This function is not available in Qt Script. Use substring() instead.
`String.lower()` This function is not available in Qt Script. Use toLowerCase() instead.
`String.mid()` This function is not available in Qt Script. Use substring() instead.
`String.right()` This function is not available in Qt Script. Use substring() instead.
`String.searchRev()` This function is not available in Qt Script. Use search() / match() instead.
`String.startsWith()` This function is not available in Qt Script. Use indexOf() == 0 instead.
`String.upper()` This function is not available in Qt Script. Use toUpperCase() instead.
`RegExp.valid` This property is not available in Qt Script because it is not required; a SyntaxError exception is thrown for bad RegExp objects.
`RegExp.empty` This property is not available in Qt Script. Use toString().length == 0 instead.
`RegExp.matchedLength` This property is not available in Qt Script. RegExp.exec() returns an array whose size is the matched length.
`RegExp.capturedTexts` This property is not available in Qt Script. RegExp.exec() returns an array of captured texts.
`RegExp.search()` This function is not available in Qt Script. Use RegExp.exec() instead.
`RegExp.searchRev()` This function is not available in Qt Script. Use RegExp.exec() or String.search()/match() instead.
`RegExp.exactMatch()` This function is not available in Qt Script. Use RegExp.exec() instead.
`RegExp.pos()` This function is not available in Qt Script. Use String.match() instead.
`RegExp.cap()` This function is not available in Qt Script. RegExp.exec() returns an array of captured texts.
QSA also defined some internal Qt API which is not present in Qt Script. The types provided by QSA which are not provided by Qt Script are:

```
  Rect
Point
Size
Color
Palette
ColorGroup
Font
Pixmap
ByteArray
```

5.3 Creating a script with a user interface

It is possible to connect a user interface (*.ui) to the script. This is an XML file that can be created by Qt Designer (is included when downloading the Qt Creator package from
http://qt.nokia.com/.
A nice description how to create a user interface for the script can be found at:
http://www.brainvoyager.com/resources_for_users/bvblog/bvblog.html.
See also the "Plugin" chapter in the BrainVoyager User's Guide:
http://www.brainvoyager.com/bvqx/doc/UsersGuide/WebHelp/BrainVoyagerQXUsersGuide.htm
since this also explains the GUI scripts.
An example script is shown in the section below. For more script examples for BrainVoyager QX 2.1, 2.2 and 2.3, please see
http://support.brainvoyager.com/ → "Available Tools → "Available Scripts" → "Scripts for BrainVoyager QX 2.1 and higher".

5.4 Writing GUI scripts

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 (also 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)
ComboBox: currentIndex, currentText (QX 2.2 and higher)
```

The use of these signals has been illustrated in the extPrintDlg.js/ui script below.

Procedure

1. In Qt Creator, select to Create a Qt Designer Form (*.ui)(figure 5.1)
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

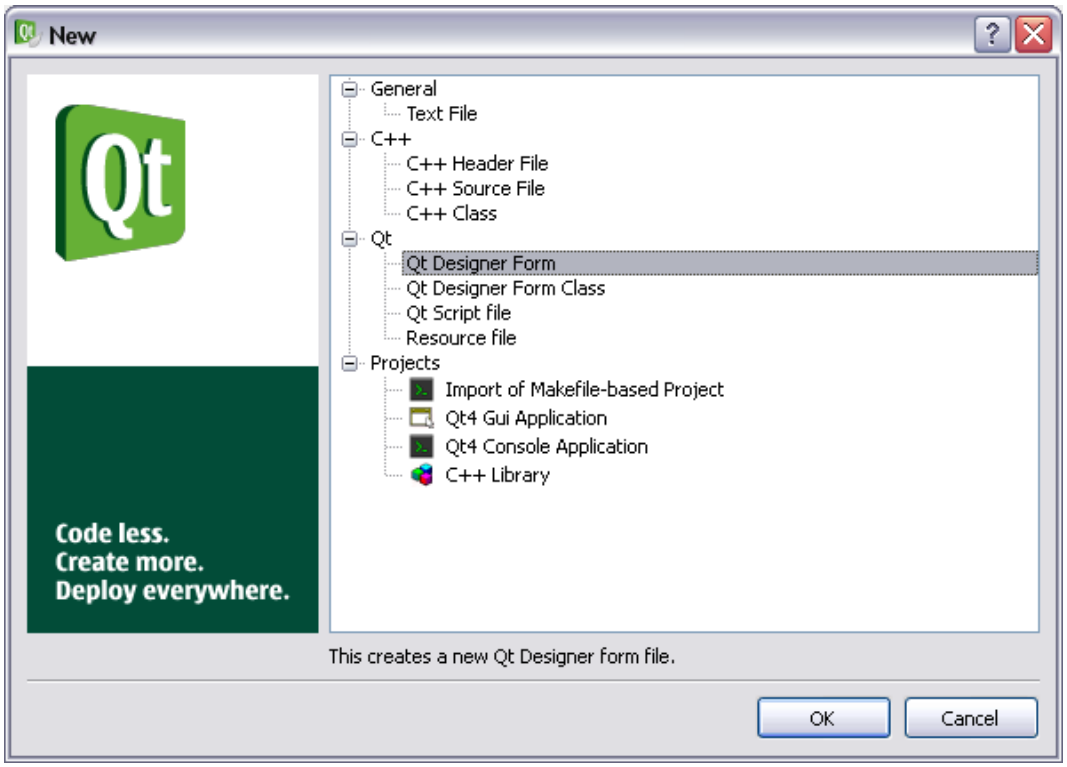

Figure 5.1: Select the Designer Form in Qt Creator

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() {
BrainVoyagerQX.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 5.2).

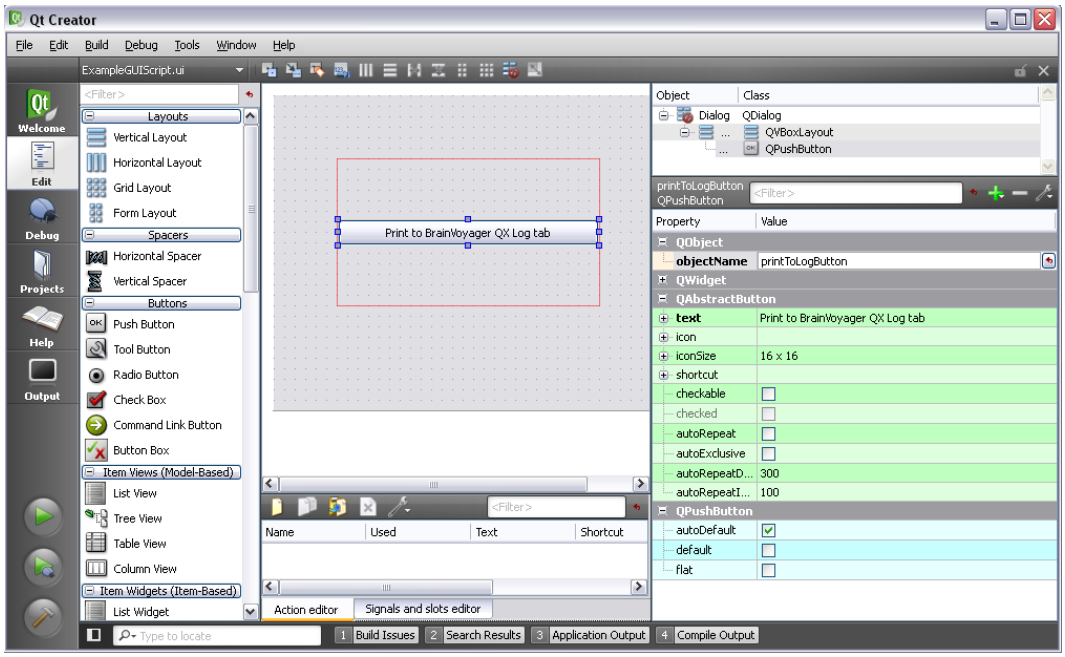

Figure 5.2: Creating the user interface in Qt Creator

When there are many widgets (graphical components), the function `BrainVoyagerQX.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 = BrainVoyagerQX.FindChild("btnGetFiles");`
3. Use the variable `getFileButton` anywhere in the script.

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 5.3). Both script (*.js) and user interface (*.ui) should be placed in the directory `/Documents/BVQXExtensions/Scripts/`.

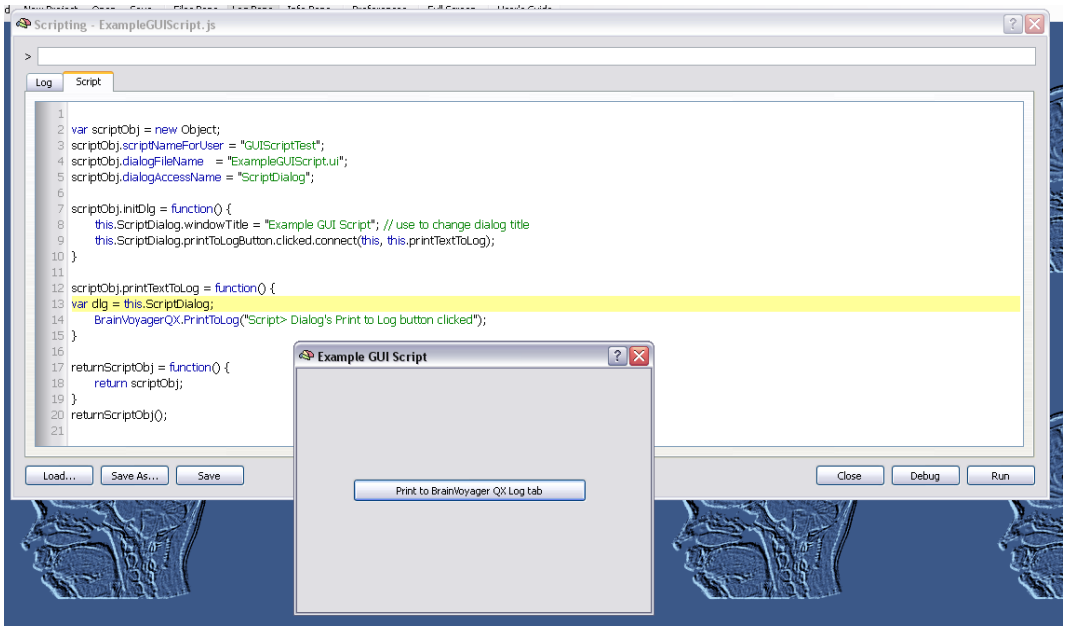

Figure 5.3: Using the user interface in BrainVoyager

5.4.2 Catching errors

If errors occur, it is good to be aware of them. They can be caught using the `try { ... } catch (e) { ... }` keywords. The errors will be text so they can be printed to the BrainVoyager QX Log tab (see figure 5.4).

```
BrainVoyagerQX.OpenDocument( "/Users/hester/Data/ObjectsDicomGSG/CG_3DT1MPR_SCRIPT_CLEAN.vmr" );
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 5.4: Print an error to the BrainVoyager QX 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

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 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 6.2.3).

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().

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)`	
`charCodeAt([i])`	
`concat(string2)`	
`fixed()`	
`fontcolor(#rrggbb)`	
`fontsize(1to7)`	
`fromCharCode(n1...)*`	
`indexOf("str" [,i])`	
`italics()`	
`lastIndexOf("str" [,i])`	
`link(url)`	
`localeCompare()`	
`match(regexp)`	
`replace(regexp,str)`	
`search(regexp)`	
`slice(i,j)`	
`small()`	
`split(char)`	
`strike()`	
`sub()`	
`substring(from, to)`	Return part of the string, starting with position “from”. “to” is optional.
`sup()`	
`toLocaleLowerCase()`	
`toLocaleUpperCase()`	
`toLowerCase()`	
`toString()`	
`toUpperCase()`	
`valueOf()`	

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

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.

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

6.2.3 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)
}
```

6.2.4 Declarations

`var, const, class, this, function`

6.2.5 Control statements

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

6.2.6 Error handling

`try...catch`

6.2.7 Comments

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

6.2.8 Date and time

```
Date, 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()
Set functions
setDate(), setFullYear(), setHours(), setMilliseconds(), setMinutes(),
setMonth(), setSeconds(), setTime(), setUTCDate(), setUTCFullYear(),
setUTCHours(), setUTCMilliseconds(), setUTCMinutes(),
```

```
setUTCMonth(), setUTCSeconds(), setUTCTime()
parse(), toString(), toLocaleString(), toUTCString
```