The Network Access Plugin

The network interface opens custom access to data that is internally used by Turbo-BrainVoyager (TBV) during and after real-time processing including information of regions of interest (ROI), design matrix, (pre-processed) raw data and statistical information inclusive the content of the design matrix, beta maps and contrast t-maps. This plugin allows to perform additional operations and calculations outside the TBV.

Well defined access queries are used to get access to the data via a TCP connection. Thereby, a server implementation allows to get access to the whole data that is also provided by the plugin interface of TBV during and after the real-time processing.

A server client approach is used to get access to the data at any time. The implemented server provides comprehensive features to get access to the data in real-time while running the analysis. To receive the requested information from the server one just need to send a data specific query including the requested information types.

Connect to the server

A connection to the server could be established through a TCP socket implementation. At first the specific server port has to be defined in the plugin. In TBV 4.4.8 the network plugin launches automatically in the background using port 55555. The ports range from 1 to 65535 but the first 1024 well-known ports or system ports are used by system processes and need super user privileges on Unix-like systems to bind a network socket to it. After defining the port the IP of the server will be displayed in the plugin window. Be sure to use a TCP socket to connect the server.

The plugin uses two different sockets. One to receive and answer the queries from the client and one to send execution information from TBV. To specify the type of the socket to use send a ("Request Socket") char array for the query socket or a ("Execute Socket") char array for the execution information socket (prepend the size of the array as a 4 byte unsigned integer for all char arrays and end the array with a null terminator [\0]). Include the specific char array into a byte array and insert as the first element the size of the byte array as an 8 byte signed 64bit integer. An example byte-based representation is shown below:

0	0	0	0	0	0	0	\19	0	0	0	\15	R	e	q	u	e	s	t		S	o	c	k	e	t	\0
0	0	0	0	0	0	0	\19	0	0	0	\15	E	x	e	c	u	t	e		S	o	c	k	e	t	\0

If the request is wrong the connection will be closed. Insert a short delay from around 0.5 seconds between the different socket connection requests to be ensure that the server can assign your connection to the right type and started listening again for a new connection.

After the connection is completely established the data can be directly accessed from TBV.

All data that is send and received is in **big-endian** *(network byte)* **byte order** and all floating-point values are in **single-precision** floating-point format.

Access the data

A query with the included request of the specific data must be send to the server to access the data from the TBV. Every query is defined on the next pages of this guide including a description of the data. A query has to be constructed as a byte array where the first element is a definition of the size of the byte array as an 8-byte signed 64bit integer. The second entry defines the type of query as a char array. After that follows the specific query definitions, for example the time point from which to get the data. Only ask for data that are already available in the TBV otherwise an error string ("Wrong request!") will be returned as shown below.

If the query consists of more than one parameter, attach the other parameters to the query and store it in the byte array.

You will receive a byte array with the respective content as answer. (The first 8 byte of the byte array represents the size of the received byte array as unsigned 64bit integer, here marked in green.) As an example, how a definition of the byte array should look like, see the “tGetCurrentTimePoint” example below (send and receive).

In this example the received information includes the query itself plus the output, in this case 1 formatted as a 64bit integer (4 bytes). The blue highlighted 4-bytes define the length of the text request, in our example 21 symbols (20 letters and the null terminator).

Prestep, Poststep, Postrun calls

Whenever new data is available a specific information will be send to the client. The calls are distinguished into a Prestep, Poststep and a Postrun call. The Prestep call will be send whenever new raw data is available. The Poststep call will be send whenever new pre-processed data is available. The Postrun call will be send when the experiment is completely finished or interrupted by the user. This invocation consists of an at least 14 byte large char array followed by an integer representing the current point in time of the current TBV processing. Both data will be send as a byte array (The first 8 byte of the byte array represents the size of the received byte array as signed 64bit integer)

Wrong requests

Whenever the user sends a wrong query (for example asked for a time point that has not been processed yet) a wrong request information will be send to the client. The first part of the information is the definition that a wrong request was asked. After that follows the description which specific part of the query is wrong. The client receives a 14 byte char array (“Wrong request!”) followed by an char array including the respective error. Keep in mind that this data will also be send in a byte array. (The first 8 byte of the byte array represents the size of the received byte array as unsigned 64bit integer and before each char array the specific length as a 4byte unsigned integer.)

Basic Project Queries

Send: **tGetCurrentTimePoint**
Receive: int CurrentTimePoint
Provides the number of the currently processed step during real-time processing as an integer. Note that this function is 1-based, i.e. when the first step is processed the function returns "1" not "0"; this is important when the return value is used to access time-related information; in this case subtract "1" from the returned value.

Send: **tGetExpectedNrOfTimePoints**
Receive: int NrOfTimePoints
Provides the number of time points as an integer. The name contains the term "expected" since a real-time run might be interrupted by the user, i.e. this is the intended number of volumes as specified in the TBV settings file.

Send: **tGetDimsOfFunctionalData**
Receive: int dim_x, int dim_y, int dim_z
Provides the dimensions of the functional volumes; "dim_x" and "dim_y" are the dimensions of the slices constituting the volume and "dim_z" corresponds to the number of slices.
Example of a Bytearray answer:

Send: **tGetProjectName**
Receive: char[100] cProjectName
Provides the name of the project as specified in the TBV file as a C string; note that the received data must point to a pre-allocated array that is large enough (a buffer of 100 bytes should be sufficient). The returned name can, for example, be used as part of names identifying exported data (see example "Export Volume Data" client).

Send: **tGetWatchFolder**
Receive: char[513] cWatchFolder
Provides the path of the "watch folder" as specified in the TBV file as a C string; Note that the received data must point to a pre-allocated array that is large enough for the returned path (a buffer of 513 bytes is recommended).

Send: **tGetTargetFolder**
Receive: char[513] cTargetFolder
Provides the path of the "target folder" as specified in the TBV file as a C string; note that the received data must point to a pre-allocated array that is large enough for the returned path (a buffer of 513 bytes is recommended). The target folder can be used to export data for custom processing (see example "Export Volume Data" client).

Send: **tGetFeedbackFolder**
Receive: char[513] cFeedbackFolder
Provides the path of the "feedback folder" as specified in the TBV file as a C string; note that the provided data must point to a pre-allocated array that is large enough for the received path (a buffer of 513 bytes is recommended). The feedback folder can be used to store the result of custom calculations, e.g. providing custom input for the "Presenter" software tool.

Protocol, DM, GLM Queries

Send: **tGetCurrentProtocolCondition**
Receive: int CurrentProtocolCondition
Provides the index of the currently "active" condition of the protocol (0-based), i.e. the condition that has a defined interval enclosing the current time point.

Send: **tGetFullNrOfPredictors**
Receive: int FullNrOfPredictors
Provides the number of predictors of the design matrix. Note that this query returns the "full" number of intended predictors while the "tGetCurrentNrOfPredictors" returns the number of predictors currently in use.

Send: **tGetCurrentNrOfPredictors**
Receive: int CurrentNrOfPredictors
Provides the currently effective number of predictors. Note that this query may return a smaller number than the "tGetFullNrOfPredictors" query since the internal GLM calculations use a restricted set of predictors in case that for one or more predictors not enough non-zero data points are available. Roughly speaking, the number of current predictors increases each time when a new condition is encountered during real-time processing.

Send: **tGetNrOfConfoundPredictors**
Receive: int NrOfConfoundPredictors
Provides the full number of confound predictors. To get the full/effective number of predictors-of-interest, subtract the returned value from the "tGetFullNrOfPredictors" or "tGetCurrentNrOfPredictors" function, respectively.

Send: **tGetValueOfDesignMatrix**, int pred, int timepoint
Receive: int pred, int timepoint, float ValueOfDesignMatrix
Provides the value of a predictor at a given time point of the current design matrix. Note that the design matrix always contains the "full" set of predictors, a reduced set of predictors is only used internally (predictors that are not used internally are those containing only "0.0" entries up to the current time point). Note that the "timepoint" parameter must be smaller than the value returned by the "tGetCurrentTimePoint" query. For details, see the provided example clients.

Send: **tGetNrOfContrasts**
Receive: int NrOfContrasts
Provides the number of (automatically or manually) specified contrasts in the TBV settings file. This value is important for accessing t maps, see the "tGetMapValueOfVoxel" and "tGetContrastMaps" queries.

ROI Queries

Send: **tGetNrOfROIs**
Receive: int NrOfROIs
Provides the number of currently available ROIs. Note that the number of ROIs may change during real-time processing since the user may open additional ROI windows or close ROI windows at any time. It is thus important to use this function prior to other functions accessing ROI information.

Send: **tGetMeanOfROI**, int roi
Receive: int roi, float MeanOfROI
Returns the mean signal value of the ROI referenced with the "roi" parameter (0-based index). A valid number must be smaller than the value returned by the "tGetNrOfROIs" query. Note that the voxels defining a ROI might change in case that the user selects another region for the same ROI index (replaces the content of the same plot in the GUI). The query should be used in situations when ROIs are not changed, i.e. when a set of ROIs is pre-loaded for a neurofeedback study. For details, see the "ROI Processing" example client.

Send: **tGetExistingMeansOfROI**, int roi, int toTimePoint
Receive: int roi, int toTimePoint, float [toTimePoint] ExistingMeansOfROI
Returns all mean signal values of the ROI referenced with the "roi" parameter (0-based index) to the specified point in time. A valid ROI number must be smaller than the value returned by the "tGetNrOfROIs" query. Note that the voxels defining a ROI might change in case that the user selects another region for the same ROI index (replaces the content of the same plot in the GUI). The query should be used in situations when ROIs are not changed, i.e. when a set of ROIs is pre-loaded for a neurofeedback study. For details, see the "ROI Processing" example client.

Send: **tGetMeanOfROIAtTimePoint**, int roi, int toTimePoint
Receive: int roi, int toTimePoint, float MeanOfROIAtTimePoint
Returns the mean signal value of the ROI referenced with the "roi" parameter (0-based index) of a defined point in time. A valid ROI number must be smaller than the value returned by the "tGetNrOfROIs" query, a valid toTimePoint number must be smaller than the value returned by the "tGetCurrentTimePoint" query. Note that the voxels defining a ROI might change in case that the user selects another region for the same ROI index (replaces the content of the same plot in the GUI). The query should be used in situations when ROIs are not changed, i.e. when a set of ROIs is pre-loaded for a neurofeedback study. For details, see the "ROI Processing" example client.

Send: **tGetDetrendedValueOfROI**, int roi
Receive: int roi, float DetrendedValueOfROI
Returns the detrended mean signal value of the ROI referenced with the "roi" parameter (0-based index). A valid number must be smaller than the value returned by the "tGetNrOfROIs" query. Note that the voxels defining a ROI might change in case that the user selects another region for the same ROI index (replaces the content of the same plot in the GUI). The query should be used in situations when ROIs are not changed, i.e. when a set of ROIs is pre-loaded for a neurofeedback study. For details, see the "ROI Processing" example client.
NOTE: The data is only detrended if the detrending is enabled in TBV. A switch to display non detrended values will cause that this function will return non detrended data.

Send: **tGetExistingDetrendedMeansOfROI**, int roi, int toTimePoint
Receive: int roi, int toTimePoint, float [toTimePoint] ExistingDetrendedMeansOfROI
Returns all detrended mean signal values of the ROI referenced with the "roi" parameter (0-based index) to the specified point in time. A valid ROI number must be smaller than the value returned by the "tGetNrOfROIs" query. Note that the voxels defining a ROI might change in case that the user selects another region for the same ROI index (replaces the content of the same plot in the GUI). The query should be used in situations when ROIs are not changed, i.e. when a set of ROIs is pre-loaded for a neurofeedback study. For details, see the "ROI Processing" example client.
NOTE: The data is only detrended if the detrending is enabled in TBV. A switch to display non detrended values will cause that this function will return non detrended data.

Send: **tGetDetrendedMeanOfROIAtTimePoint**, int roi, int toTimePoint
Receive: int roi, int toTimePoint, float DetrendedMeanOfROIAtTimePoint
Returns the mean signal value of the ROI referenced with the "roi" parameter (0-based index) of a defined point in time. A valid ROI number must be smaller than the value returned by the "tGetNrOfROIs" query, a valid toTimePoint number must be smaller than the value returned by the "tGetCurrentTimePoint" query. Note that the voxels defining a ROI might change in case that the user selects another region for the same ROI index (replaces the content of the same plot in the GUI). The query should be used in situations when ROIs are not changed, i.e. when a set of ROIs is pre-loaded for a neurofeedback study. For details, see the "ROI Processing" example client.
NOTE: The data is only detrended if the detrending is enabled in TBV. A switch to display non detrended values will cause that this function will return non detrended data.

Send: **tGetNrOfVoxelsOfROI**, int roi
Receive: int roi, int NrOfVoxelsOfROI
Provides the number of voxels of the specified ROI. Note that the returned number might change during real-time processing in case that the user replaces a ROI with another set of voxels. The value of this query is important for accessing information of individual ROI voxels (see below).

Send: **tGetBetaOfROI**, int roi, int beta
Receive: int roi, int beta, float BetaOfROI
Retrieves the value of a specified beta (0-based index) of the specified ROI (0-based index). For each ROI a GLM is calculated incrementally using the mean signal time course; the betas of the calculated GLM are accessible with this query; note that the beta indices range from 0 to the full number of predictors; to retrieve only the betas of the predictors of interest, the beta index must be smaller than "tGetFullNrOfPredictors" minus "tGetNrOfConfoundPredictors". For details, see the "ROI Processing" example client.

Send: **tGetCoordsOfVoxelOfROI**, int roi, int voxel
Receive: int roi, int voxel, int x, int y, int z
Provides the coordinates of a voxel (0-based enumeration index) of the ROI specified with the "roi" parameter (0-based index); the value for the "voxel" index ranges from "0" to one less than the value returned by the "tGetNrOfVoxelsOfROI" query; since ROIs content may change, use the latter function for a specific ROI index always before using the current function. For details, see the "ROI Processing" example client.

Send: **tGetAllCoordsOfVoxelsOfROI**, int roi
Receive: int roi, int [tGetNrOfVoxelsOfROI(roi)*3] CoordsOfVoxelsOfROI
Provides the coordinates of all voxels of the ROI specified with the "roi" parameter (0-based index); since ROIs content may change, use the latter function for a specific ROI index always before using the current function. For details, see the "ROI Processing" example plugin. If a coordinate of a specific voxel of a roi needs to be accessed, use the term "x_coord = voxel_roi+0; y_coord = voxel_roi+1; z_coord = voxel_roi+2". For details, see the "ROI Processing" example client.

Send: **tGetNrOfBestVoxelsOfROI**, int roi
Receive: int roi, int NrOfBestVoxelsOfROI
Provides the number of best voxels of the specified ROI. Note that the returned number might change during real-time processing in case that the user replaces a ROI with another set of voxels. The value of this query is important for accessing information of individual ROI voxels (see below). The best voxel selection procedure must be enabled in the options of the Neurofeedback dialog to receive the coordinates of the best voxel sub-ROI.

Send: **tGetCoordsOfBestVoxelOfROI**, int roi, int voxel
Receive: int roi, int voxel, int x, int y, int z
Provides the coordinates of a voxel (0-based enumeration index) within the best voxel ROI (white marked voxels) of the ROI specified with the "roi" parameter (0-based index); the value for the "voxel" index ranges from "0" to one less than the value returned by the "tGetNrOfVoxelsOfROI" query; since ROIs content may change, use the latter function for a specific ROI index always before using the current function. For details, see the "ROI Processing" example client. The best voxel selection procedure must be enabled in the options of the Neurofeedback dialog to receive the coordinates of the best voxel sub-ROI.

Send: **tGetAllCoordsOfBestVoxelsOfROI**, int roi
Receive: int roi, int [tGetNrOfBestVoxelsOfROI(roi)*3] CoordsOfVoxelsOfROI
Provides the coordinates of all voxels of the ROI specified with the "roi" parameter (0-based index); since ROIs content may change, use the latter function for a specific ROI index always before using the current function. For details, see the "ROI Processing" example plugin. If a coordinate of a specific voxel of a roi needs to be accessed, use the term "x_coord = voxel_roi+0; y_coord = voxel_roi+1; z_coord = voxel_roi+2". For details, see the "ROI Processing" example client. The best voxel selection procedure must be enabled in the options of the Neurofeedback dialog to receive the coordinates of the best voxel sub-ROI.

Neurofeedback Data Access Queries

Send: **tGetNFBaseline**
Receive: float NFBaseline
The NFBaseline defines the baseline level calculated from the (detrended) time course values of the first ROI.

Send: **tGetNFValue**
Receive: float NFValue
Provides the current basis value for the NF calculation. By just subtracting the NFBaseline value from the received NFValue you can calculate the exact Neurofeedback value used in TBV. For example the returned tGETNFValue 1.1595937 and the tGetNFBaseline is -0.0710152909 then the actual feedback value used in TBV is

fb = (1.1595937 - -0.0710152909) = 1.23

Send: **tGetNFMaxPSC**
Receive: float NFMaxPSC
Provides the maximum percent signal change (MaxPSC) value used for the NF scaling. Relating the calculated absolute value to the MaxPSC of for example value 2, the rescaled feedback value results in 1.23 / 2.0 = 0.615. This value is converted to the number of filled entries in the thermometer by multiplying with the number of available levels (usually 10 if no negative values are shown) followed by a simple rounding operation which results in value 6 (rounding value 6.15).

Send: **tGetNFAvgLastN**
Receive: int NFAvgLastN
Provides the number of averaging samples used for the moving average temporal smoothing of the NF signal. The smoothing has not yet been applied to the tGetNFValue data and must be applied on the receiver side.

Send: **tGetNFFbLevel**
Receive: int NFFbLevel
Provides the achieved feedback level of the thermometer as shown in the NF dialog in TBV. The received NFFbLevel values have no limit and don’t apply the MaxPSC value as a cutoff but only as a scaling.

Send: **tGetNFTargetLevel**
Receive: int NFTargetLevel
Provides the target level that the participant should reach in the current trail if set in the protocol.

Send: **tGetNFBLWndShiftStart**
Receive: int NFBLWndShiftStart
Provides the shift at begin parameter defined in the Feedback baseline data points options within the TBV NF dialog.

Send: **tGetNFBLWndShiftEnd**
Receive: int NFBLWndShiftEnd
Provides the shift at end parameter defined in the Feedback baseline data points options within the TBV NF dialog.

Volume Data Access Queries

Send: **tGetValueOfVoxelAtTime**, int x, int y, int z, int timePoint
Receive: int x, int y, int z, int timepoint, float ValueOfVoxelAtTime
Provides the signal value as a 4-byte float value of the voxel specified by the coordinate parameters "x", "y" and "z" for the given time point (0-based indices). The given "timepoint" parameter must be smaller than the value obtained by the "tGetCurrentTimePoint" query.

Send: **tGetValueOfAllVoxelsAtTime**, int timePoint
Receive: short int [dim_x*dim_y*dim_z] TimeCourseData
Provides the signal value of all voxels to a given time point that is also used internally in TBV. Individual values are 2-byte short integers. Note that the "timepoint" parameter must be smaller than the value returned by the "tGetCurrentTimePoint()" function. If a voxel with specific coordinates needs to be accessed, use the term "z_coord*dim_x*dim_y + y_coord*dim_x + x_coord". For details, see the provided example clients.

Send: **tGetRawValueOfAllVoxelsAtTime**, int timePoint
Receive: short int [dim_x*dim_y*dim_z] TimeCourseData
Provides raw (not pre-processed) the signal value of all voxels to a given time point that is also used internally in TBV. Individual values are 2-byte short integers. Note that the "timepoint" parameter must be smaller than the value returned by the "tGetCurrentTimePoint()" function. If a voxel with specific coordinates needs to be accessed, use the term "z_coord*dim_x*dim_y + y_coord*dim_x + x_coord". For details, see the provided example clients.

Send: **tGetBetaOfVoxel**, int beta, int x, int y, int z
Receive: int beta, int x, int y, int z, float BetaOfVoxel
Provides the value of a beta indexed by the "beta" parameter as an 4-byte float value for the voxel specified by the coordinate parameters "x", "y" and "z" (0-based indices). This function allows to access estimated beta values resulting from the incremental GLM performed by TBV. Note that the beta index ranges from 0 to the current number of predictors; to retrieve only the betas of the predictors of interest, the beta index must be smaller than "tGetCurrentNrOfPredictors" minus "tGetNrOfConfoundPredictors". For details, see the "Export Volume Data" example client.

Send: **tGetBetaMaps**
Receive: float [CurrentNrOfPredictors*dim_x*dim_y*dim_z] BetaMaps
Provides the full stack of beta maps that is also used internally in TBV. Individual entries are 4-byte float values. The data is organized as a flat array; in order to obtain the beta value of a specific predictor index for a voxel with specific coordinates, use the term "beta_i*dim_xyz + z_coord*dim_xy + y_coord*dim_x + x_coord". Note that the beta_i index must be in the ranges from 0 to the current number of predictors; to retrieve only the betas of the predictors of interest, the beta index must be smaller than tGetCurrentNrOfPredictors" minus "tGetNrOfConfoundPredictors". For details, see the provided "Export Volume Data" client.

Send: **tGetMapValueOfVoxel**, int map, int x, int y, int z
Receive: int map, int x, int y, int z, float MapValueOfVoxel
Provides the value of a t map indexed by the "map" parameter as a 4-byte float value for the voxel specified by the coordinate parameters "x", "y" and "z" (0-based indices). This function allows to access calculated contrast values that are calculated based on the beta values from

the incremental GLM performed by TBV. The "map" index ranges from 0 to one less than the number of contrasts specified in the TBV settings file (implicitly or via a specified contrast ".ctr" file); the number of contrasts can be retrieved using the "tGetNrOfContrasts" query. For details, see the "Export Volume Data" example client.

Send: **tGetContrastMaps**
Receive: float [tGetNrOfContrasts*dim_x*dim_y*dim_z] ContrastMaps
Provides the full stack of contrast maps that is also used internally in TBV. Individual entries are 4-byte float values. The data is organized as a flat array; in order to obtain the t value of a specific contrast map index for a voxel with specific coordinates, use the term "map_i*dim_xyz + z_coord*dim_xy + y_coord*dim_x + x_coord". The "map_i" index ranges from 0 to one less than the number of contrasts specified in the TBV settings file (implicitly or via a specified contrast ".ctr" file); the number of contrasts can be retrieved using the "tGetNrOfContrasts" query. For details, see the provided "Export Volume Data" client.

SVM Access Functions

TBV provides access to classification output values calculated during real-time SVM classification. The tGetCurrentClassifierOutput() function provides both a single integral value informing which class is predicted at the current time point as well as a detailed vector of float values that can be used for custom classifier-based neurofeedback. The latter information is returned in a vector since the number of values depend on the number of classes used for classification (see below). It is, thus, important to call the tGetNumberOfClasses() function to ensure that the right number of values is used to prepare an array with sufficient size for retrieving the output values.

Send: **tGetNumberOfClasses**
Receive: int n_classes
Provides the number of classes for which values are provided. In case that the real-time SVM classifier is not used, this function returns -3; in case that the real-time SVM classifier dialog is open but the classifier is not producing incremental output, this function returns -2; if the classifier is working but no output has been generated yet, this function returns 0. You only should use the tGetCurrentClassifierOutput() function (see below) if this function returns a positive value. Based on the returned (positive) value (assigned to e.g. variable *n_classes*), the size of the array needed for the tGetCurrentClassifierOutput() function can be calculated as the number of pair comparisons *n_pairs*: *n_pairs* = *n_classes* * (*n_classes* - 1) / 2

Send: **tGetCurrentClassifierOutput**
Receive: float [n_pairs] ClassifierOutput
Provides results during real-time SVM classification for the current time point. The function returns an integral value indicating which class is predicted, i.e. which class label has been assigned to the current brain activity pattern. Note that the returned value is 1-based, i.e. if the first class is predicted, value 1 is returned, if the second class is predicted, value 2 is returned and so on. In addition to returning the predicted class, the function also fills a provided float array with detailed classification values. Since the SVM procedure internally finds the predicted class ("winner") by comparing the results obtained for all possible unique pairs of classes, the array needs to be large enough to receive all pairwise classification results (for calculation, see above). In case of a two-class problem, the array will contain only one entry for the pair "1 against 2" or "1-2". For multi-class (> 2) problems, the order of pairs will be starting with all pairs containing class 1 on the left side, then all remaining pairs that have class 2 on the left side and so on. For a 3-class problem, the order would be "1-2", "1-3", "2-3" and for a 4-class problem, the order would be "1-2", "1-3", "1-4", "2-3", "2-4" and "3-4". A positive value for a pair indicates that the class on the left side has "won" whereas a negative value indicates that the class on the right side has "won" the respective pairwise comparison; the size of the value(s) may be used to calculate a continuous value as a feedback signal. As with the tGetNumberOfClasses() function, this function returns value -3 in case that the real-time SVM classifier is not open and -2 in case that the real-time SVM classifier dialog is open but the classifier is not producing incremental output. The function returns value -1 if the SVM dialog is used but no output data is available for the current time point. Only access the provided output_array if the returned value of the function is a positive number. Use the tGetNumberOfClasses() function to retrieve the number of classes from which you can calculate the number of pairs (see above) to determine the necessary size of the array used to receive the pairwise classification values.

Semantic Neurofeedback Data Access Queries

Send: **tGetSemanticNrOfROIs**
Receive: int SemanticNrOfROIs
Provides the number of ROIs used for semantic NF.

Send: **tGetSemanticNrOfConditions**
Receive: int SemanticNrOfConditions
Returns the number of conditions used for the semantic NF paradigm.

Send: **tGetSemanticNrOfVoxels**, int roi
Receive: int SemanticNrOfVoxels
Returns the number of voxels used for the semantic NF paradigm in the specified roi.

Send: **tGetSemanticMultiVoxelPattern**
Receive: float [SemanticNrOfROIs][SemanticNrOfConditions][SemanticNrOfVoxels] SemanticMultiVoxelPattern
Returns the semantic multi voxel base pattern used for the semantic NF paradigm.

Send: **tGetSemanticFeedbackPattern**
Receive: float [SemanticNrOfROIs][SemanticNrOfVoxels] SemanticFeedbackPattern
Returns the semantic multi voxel base pattern for each ROI used for the semantic NF paradigm.

Functional Connectivity Functions

The network interface provides basic functional connectivity measures like the Pearson correlation or partial correlation. These measures are based on the selected ROI's and calculated for the current and previous points in time and a specified window for the correlation. The correlation results stored in the float PCorrelation[ncon] array can be accessed as shown in the example below:

```
int n_rois = tGetNrOfROIs();
int ncon = 0;
for(int i=1; i<n_rois; i++)
                for(int j=i+1; j<=n_rois; j++)
                        {
                                printf("Correlation: ROI %i and ROI %i: %f",i,j,PCorrelation[ncon]);
                                ncon++;
                        }
```

The total number of correlations (ncon) can be calculated using the equation below.

int ncon=(n_rois(n_rois-1))/2; // total number of correlations (edges)*

Send: **tGetPearsonCorrelation**, int windowSize
Receive: float PearsonCorrelation[ncon]
Provides the calculated Pearson Correlation results of the current point in time for all combinations of selected ROI's. At least two ROI's must be selected to calculate a correlation.

Send: **tGetPearsonCorrelationAtTimePoint**, int windowSize, int timePoint
Receive: float PearsonCorrelation[ncon]
Provides the calculated Pearson Correlation results of the point in time defined by the timePoint (0-based) parameter for all combinations of selected ROI's. At least two ROI's must be selected to calculate a correlation.

Send: **tGetPartialCorrelation**, int windowSize
Receive: float PartialCorrelation[ncon]
Provides the calculated partial correlation results of the current point in time for all combinations of selected ROI's. At least two ROI's must be selected to calculate a correlation.

Send: **tGetPartialCorrelationAtTimePoint**, int windowSize, int timePoint
Receive: float PartialCorrelation[ncon]
Provides the calculated partial correlation results of the point in time defined by the timePoint (0-based) parameter for all combinations of selected ROI's. At least two ROI's must be selected to calculate a correlation.

Send: **tGetDetrendedPearsonCorrelation**, int windowSize
Receive: float PearsonCorrelation[ncon]
Provides the calculated detrended Pearson Correlation results of the current point in time for all combinations of selected ROI's. At least two ROI's must be selected to calculate a correlation.

Send: **tGetDetrendedPearsonCorrelationAtTimePoint**, int windowSize, int timePoint
Receive: float PearsonCorrelation[ncon]
Provides the calculated detrended Pearson Correlation results of the point in time defined by the timePoint (0-based) parameter for all combinations of selected ROI's. At least two ROI's must be selected to calculate a correlation.

Send: **tGetDetrendedPartialCorrelation**, int windowSize
Receive: float PartialCorrelation[ncon]
Provides the calculated detrended partial correlation results of the current point in time for all combinations of selected ROI's. At least two ROI's must be selected to calculate a correlation.

Send: **tGetDetrendedPartialCorrelationAtTimePoint**, int windowSize, int timePoint
Receive: float PartialCorrelation[ncon]
Provides the calculated detrended partial correlation results of the point in time defined by the timePoint (0-based) parameter for all combinations of selected ROI's. At least two ROI's must be selected to calculate a correlation.

Instantaneous proxy for correlation

The network interface provides access to the method described by Ramot and colleagues 2019. The implementation in the network interface uses the correlation approach described in section 2.7.
The method is adapted to allow to use N Regions + 1 Control region. The correlations are automatically balanced.

$$InstantProxyCorrelation = \left(\frac{1}{ncon}\right) * \ = \ 1^{Nr\ ROIscorr(ROIi, ROIi+1)}$$

Send: **tGetInstantProxyCorrelation**, int windowSize
Receive: float **tGetInstantProxyCorrelation**
Provides the calculated proxy correlation result of the current point in time for all combinations of selected ROI's. At least three ROI's must be selected to calculate a correlation.

Send: **tGetInstantProxyCorrelationAtTimePoint**, int windowSize, int timePoint
Receive: float tGetInstantProxyCorrelationAtTimePoint
Provides the calculated proxy correlation result of the point in time defined by the timePoint (0-based) parameter for all combinations of selected ROI's. At three two ROI's must be selected to calculate a correlation.

Send: **tGetDetrendedInstantProxyCorrelation**, int windowSize
Receive: float tGetDetrendedInstantProxyCorrelation
Provides the calculated detrendedproxy correlation result of the current point in time for all combinations of selected ROI's. At least three ROI's must be selected to calculate a correlation.

Send: **tGetDetrendedInstantProxyCorrelationAtTimePoint**, int windowSize, int timePoint
Receive: float tGetDetrendedInstantProxyCorrelationAtTimePoint
Provides the calculated detrended proxy correlation result of the point in time defined by the timePoint (0-based) parameter for all combinations of selected ROI's. At three two ROI's must be selected to calculate a correlation.