MoBBED Views: A basic viewing infrastructure for mobile brain-body-environment decision-making

By

Kay Robbins

University of Texas at San Antonio

December 2011

UTSA CS Technical Report TR-2011-018

Table of Contents

1. Introduction ........................................................................................................... 4
2. Architectural design ............................................................................................... 4
   2.1 Component features ......................................................................................... 6
      2.1.1 Configurable classes ................................................................................ 6
      2.1.2 Clickable interface for linking .................................................................. 6
      2.1.3 Cursor exploration .................................................................................... 7
      2.1.4 Resizable classes with gaps and margins .................................................. 7
   2.2 Data sources and data slices ............................................................................. 9
   2.3 Event handling to support drill-down exploration ............................................. 9
3. Viewing panels ...................................................................................................... 11
   3.1 The axes panel ................................................................................................ 12
   3.2 The horizontal panel ....................................................................................... 13
   3.3 The vertical panel ............................................................................................. 14
   3.4 The tab panel .................................................................................................. 15
4. Basic views ............................................................................................................ 16
   4.1 Block box plot ................................................................................................ 16
      4.1.1 Configurable properties ........................................................................... 16
   4.2 Block histogram plot ....................................................................................... 17
      4.2.1 Configurable properties ........................................................................... 17
   4.3 Block image plot .............................................................................................. 18
      4.3.1 Configurable properties ........................................................................... 18
   4.4 Element box plot ............................................................................................. 19
      4.4.1 Configurable properties ........................................................................... 19
   4.5 Signal shadow plot .......................................................................................... 20
      4.5.1 Configurable properties ........................................................................... 20
      4.5.2 Handling of slices .................................................................................... 21
      4.5.3 Extensions ................................................................................................ 21
   4.6 Stacked signal plot .......................................................................................... 22
      4.6.1 Configurable properties ........................................................................... 23
      4.6.2 Handling of slices .................................................................................... 23
5. Configuration of managed objects ....................................................................... 25
   5.1 Configuring functions ...................................................................................... 25
   5.2 Configuring plots ............................................................................................. 27
   5.3 Configuring properties ..................................................................................... 29
6. Configuration architecture ..................................................................................... 31
1. Introduction

MoBBED (Mobile Brain-Body-Environment Decision-making) is a data-driven approach for developing assistive and monitoring technologies for soldiers in real world environments. Data acquired from a variety of sources including EEG, eye-tracking, motion capture, muscle activity, heart rate, and environmental sensors must be merged and analyzed to provide the input features for integrated decision and monitor algorithms. This data will be, for the most part, annotated streaming data characterized by diverse formats, high data rates, and complex interrelationships. Analysis and discovery in such a data corpus requires a viewing infrastructure that is robust, extensible, and capable of extracting complex combinations of features for analysis and visualization.

This white paper introduces the MoBBED Views infrastructure – a MATLAB-based toolkit for developing modular data viewers. We discuss a particular application, EEGVIS, built using this toolkit to illustrate the features and capabilities of the toolkit.

2. Architectural design

MoBBED data sets tend to be very large and complex, often exhibiting widely varying scales. Both size and wide swings in scale present difficulties for viewers, which must display information in a limited screen area at a resolution that is comprehensible. A typical EEG apparatus might record 128 channels at 512 Hz resulting in about 4 million data points per minute. While normal EEG signals tend to vary on a scale of approximately 100 $\mu V$, a loose connector can result in voltages in the tens of thousands of microvolts.

MoBBED viewers use flexible drill-down strategies to deal with these issues. Summary views reveal potentially interesting sections of data, which users can explore further by clicking to examine using detailed viewing components. MoBBED has a library of modular viewing components that can be mixed and matched to best reveal structure. This white paper concentrates on two-level viewing architectures, but implementers can organize the modular components in other ways.

Fig. 1 shows an example of a basic two level viewer (visviews.dualView) for an EEG data set with 32 channels and over 30K frames or time samples. The top portion contains multiple summary views organized by tabs. Each of the top tab panels shows a summary (kurtosis and standard deviation, respectively) computed on windows of 1000 frames giving 31 windows of blocks. The kurtosis (K) tab is visible and the three summary views show the distribution of kurtosis for each frame (boxplot), the distribution of the kurtosis by frame and channel (image), and the distribution of kurtosis by channel. When the user clicks on one of these summary views, the detail panels display the clicked region.

The bottom portion contains various detail panels, which display relatively small portions of the data. A user selects detail views by clicking a summary view. The user can configure the arrangement of viewing panels and how summary and detail panels link.

The remainder of this section describes the basic architecture and supporting infrastructure in more detail. Section 2.1 describes the component features and the base classes that typical viewing components extend. Section 2.2 describes the data model. Section 2.3 describes the event model and illustrates how to configure the linkage between viewing components. Subsequent chapters cover the individual view components and configuration in more detail.
Figure 1: A screenshot of the application window for an application based on visviews.dualView. The summary view (top row of plots) is a tab panel with two tabs representing kurtosis and standard deviation for an EEG signal with 32 channels. The data has 31 blocks, each holding approximately 8 seconds worth of data. The lower two panels provide detailed views of channel 1 over all windows.
2.1 Component features

MoBBED components generally extend a series of base classes to add functionality. Table 1 summarizes the base classes and the functionality corresponding to these base classes.

<table>
<thead>
<tr>
<th>Base class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visprops.configurable</td>
<td>Allows a user to configure specified public properties of this object through a property manager GUI</td>
</tr>
<tr>
<td>visviews.clickable</td>
<td>Allows a user to link this component within its group to upstream sources and to use this component as a source for downstream components</td>
</tr>
<tr>
<td>visviews.cursorExplorable</td>
<td>Allows this component to report its values when the user moves the cursor over the component in exploration mode</td>
</tr>
<tr>
<td>visviews.resizable</td>
<td>Allows a supervising visualization to resize the component so that the overall margins of all components align</td>
</tr>
</tbody>
</table>

2.1.1 Configurable classes

A class extending visprops.configurable exposes specified public properties of the objects for configuration using a GUI in much the same way as the MATLAB property manager handles user configuration for figures. The class must provide a static getDefaultProperties() method that specifies which public properties are configurable and how to configure them. Section 5.1 discusses the configuration process in more detail.

2.1.2 Clickable interface for linking

MoBBED's hierarchical viewing structure includes summary plots of user-selectable scale to provide an overview of the data. The user can click on the summaries to drill down on the details for the clicked region. Users can configure how to link particular plots as described in Section 5.2. The visviews.clickable class encapsulates the linkage infrastructure, and panels that participate in this linking infrastructure should extend visviews.clickable. Generally, these panels override the following methods:

```
[dSlice, bFunction] = getClicked(obj)

[cbHandles, hitHandles] = getHitObjects(obj)
```

When the user clicks on a clickable panel, the ButtonDownFcn callback calls its getClicked method. This method returns the data slice corresponding to the clicked region and a reference to the block function represented by this panel, if applicable. The getHitObjects method controls setting the ButtonDownFcn callback. The cbHandles cell array specifies the handles whose ButtonDownFcn callback should be set. The hitHandles cell array specifies the handles whose HitTest property should be set 'on'.

Most visualization panels extend visviews.axesPanel, which is a clickable class. This class implements the getHitObjects() method by setting the cbHandles and hitHandles cell arrays to contain only the panel axes. However, panels containing box plots or individual selectable lines require a larger list of callbacks and hittable objects and must override the getHitObjects method of the visviews.clickable base class.

The visviews.clickable base class defines a buttonDownCallback function that performs the linkage and a registerCallbacks function that sets the ButtonDownFcn property of the handles returned by getHitObjects. Extending classes should not override these functions to perform additional actions in the callback. Instead, individual classes should override the following two functions:
The buttonDownCallback function calls the buttonDownPreCallback at the beginning of buttonDownCallback and the buttonDownPostCallback after processing the linkage.

2.1.3 Cursor exploration
MoBBED supports the continuous display of data coordinates or other information as the user moves the cursor over the viewing area. The visviews.cursorExplorer class manages the interrogation and continuous updating of information in response to window motion events. Application developers add a visviews.cursorExplorer object to their viewing supervisor class to enable cursor exploration.

MATLAB only supports a single type of window motion event and uses this event for resizing, pan, and zoom. As a result, cursor exploration must disable these other uses to work without interference. The viewing supervisor application should provide a mechanism for the user to enter and exit cursor exploration mode. The supervisor should call cursorOn and cursorOff methods of visviews.cursorExplorer to enter and exit cursor exploration mode. These methods disable or enable zoom, pan, and some resizing in exploration mode as well as saving and restoring state information.

Panels or other components that are available for cursor exploration must extend the visviews.cursorExplorable class, which has two methods:

\[
[x, y, xInside, yInside] = \text{getDataCoordinates}(\text{obj, point})
\]

\[s = \text{updateString}(\text{obj, point})\]

Both methods take a point in pixel coordinates relative to the enclosing figure. The getDataCoordinates method returns the x and y data coordinates of the point and indicators of whether these coordinates are inside the axes in the x and y directions, respectively. The updateString method returns the string displayed when the user moves the mouse over the designated point. This string should be empty if the point is not within this object's panel area.

Most visualization panels extend visviews.axesPanel, which is an explorable class. These panels usually override the updateString method to provide information that is more detailed than basic coordinates. The container panels: visviews.verticalPanel, visviews.horiztonalPanel, and visviews.tabPanel, each extend visviews.cursorExplorable and interrogate their child panels to find the appropriate string to return to the visviews.cursorExplorer.

2.1.4 Resizable classes with gaps and margins
When multiple panels appear in a single figure, the figure can appear misaligned when it uses the default MATLAB resizing. The supervising visualization should adjust the panel margins to eliminate the incongruities. Generally, viewing panels use the box on option to display the axes bounding boxes. The box edges forming the outside borders should align. Furthermore, panels displayed in the same horizontal row should have top and bottom box edges that align.

The visviews.resizable is an interface with two methods for extending classes to implement:

\[\text{gaps} = \text{getGaps}(\text{obj})\]

\[\text{reposition}(\text{obj, margins})\]
The `getGaps()` method returns a vector containing the number of pixels for [Left, Bottom, Right, Top] margins formed between the axes box and the edge of the panel. Composite panels such as the `visviews.tabPanel` and `visviews.horizontalPanel` report the maximum for the outer margins over all of their child panels. Supervising applications find the maximum required gaps and call the `reposition()` methods of their child resizable panels to perform realignments. The individual resizable panels should always preserve these margins on resizing, resulting in a more unified view. The individual views described in this white paper all use the resizable `visviews.axesPanel` as a base class, eliminating the need for individual visualizations to handle resizing explicitly.
2.2 Data sources and data slices

MoBBED's data summary paradigm computes summarizing values on blocks of data, where a block of data is a set of consecutive data values associated with a set of elements (such as channels for EEG). These data values may be time samples of the original signal, a group of blocked values, or some more complicated representation. When the user clicks on a point in the summary, the application changes the view in the detail panels to reflect the information associated with this point.

MoBBED encapsulates the data for the visualization in the viscore.blockedData class. The current implementation of this class expects a 2D or 3D array of data that may be epoched or unepoched. This class supports reblocking of data along a particular dimension if the original data is not epoched. The class also pads along the blocked dimension to accommodate the specified block size. This block size represents the window of data over which to compute a summary value.

A viscore.dataSlice is a specification of a subarray. A data slice does not itself contain data, but contains methods for extracting data from an array based on the slice specification. For example, a viscore.dataSlice with specification `{':', '4', ':'}` extracts an unsqueezed subarray from a two-dimensional or three-dimensional array by setting the index in dimension two to 4, provided that 4 is a valid index for the array. Otherwise, the slice extracts an empty array. When presented with a one-dimensional array, this slice extracts a copy of the original array. When presented with a data array of dimension higher than three, this slice replaces the dimensions above three with ':' when evaluating.

2.3 Event handling to support drill-down exploration

The individual MoBBED viewing components can link together in a variety of configurations. The two-level architecture supports linkage at two levels: summary and detail (hence the designation two-level viewer) organized in a two-level forest. Users configure their displays by specifying the summary and detail components as well as whether components have upstream sources.

Table 2 shows a typical specification of plot linkage. The plot name uniquely identifies the plot, allowing users to specify multiple copies of the same plot class with different configurations. Summary plots appear in the upper tabbed summary area, while detail plots appear in the detail area below. The summary tab panel has a tab for each type of summary function, and the summary plots appear on each tab.

<table>
<thead>
<tr>
<th>Category</th>
<th>Plot name</th>
<th>Plot class</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>summary</td>
<td>Block box</td>
<td>visviews.blockBoxPlot</td>
<td>None</td>
</tr>
<tr>
<td>summary</td>
<td>Element box</td>
<td>visviews.elementBoxPlot</td>
<td>None</td>
</tr>
<tr>
<td>summary</td>
<td>Block image</td>
<td>visviews.blockImagePlot</td>
<td>None</td>
</tr>
<tr>
<td>summary</td>
<td>Block image linked</td>
<td>visviews.blockImagePlot</td>
<td>Block box</td>
</tr>
<tr>
<td>detail</td>
<td>Stacked signal</td>
<td>visviews.signalStackedPlot</td>
<td>Master</td>
</tr>
<tr>
<td>detail</td>
<td>Shadow signal</td>
<td>visviews.signalShadowPlot</td>
<td>Master</td>
</tr>
</tbody>
</table>

The sources control how the plot receives data. Top-level summary plots do not have a source and present a single static view of the entire data set initialized by the master application. Summary plots with a designated source receive their data from the slice selected when the user clicks on the source. Initially, the master application initializes these downstream plots with the first slice of their respective sources. When the user clicks on any summary plot that has link details set to true, the master application delivers its slice information and data to detail plots that have master as a source.
Initially, detail plots receive their information from the first slice in the first plot of the summary unless they have a specific detail-level source.

![Diagram of linkage represented by Table 2]

**Figure 2**: A diagram of the linkage represented by Table 2. The unshaded boxes represent summary plots and the shaded boxes represent detail plots. The ‘Block box’ plot has its link details set to false.

Fig. 2 shows the linkage represented in Table 2. The summary plots named 'Block box', 'Block image', and 'Element box' are static in the sense that they show the same data independent of button down events. These plots only change when the data or function changes. However, the three plots have different behavior when clicked. 'Block image' and 'Element box' both deliver a slice to the top-level detail plots but 'Block box' delivers its slice to 'Image linked'. This later link allows users to construct linked visualizations on the fly to drill down multiple levels.
3. Viewing panels

Viewing panels are general-purpose containers that form the building blocks for more complicated viewing mosaics. MoBBED provides several basic containers: visviews.axesPanel, visviews.horizontalPanel, visviews.verticalPanel, and visviews.tabPanel. These panels provide capabilities for resizing with fixed borders. They support cursor interrogation within their areas and manage integration of mouse click events.

Fig. 3 shows the container structure of the two level viewer of Fig. 1. The visviews.dualView class provides top-level management and acts as the event master for the application. The application has two top-level containers a visviews.tabPanel for the summary and a visviews.verticalPanel for the details. Each tab panel and each individual vertical panel contains a visviews.horizontalPanel. The individual visualizations in the panel extend the visviews.axesPanel base class.

![Diagram of panel organization for the two-level viewer of Fig. 1. The classes are part of the visviews package.](image)

The application delivers resizing, mouse clicks, cursor exploration, and configuration information down through the hierarchy of Fig. 3. For example, when user resizes a figure, the top-class (dualView) asks its child tabPanel and verticalPanel for the size of the outer margins (getGaps()) required to accommodate the graphics. Each of these classes in turn interrogates its children to find their required margins and assembles the results to give the over-all outer margins required by the application. The top-level class then computes the overall margins for the figure and calls the setMargins() method on its respective children to reposition the panels.
3.1 The axes panel

The visviews.axesPanel is a resizable panel containing a single main MATLAB axes. Application developers generally extend this base class to implement a resizable viewing component. Application writers are free to add other axes, but should generally use these axes as overlays and keep them within the borders of the main axes to avoid alignment problems. Fig. 4 shows an example of an axes panel whose margins have been set to minimize the gap between the axis box and the outer borders of the panel.

![Figure 4: An example of an axes panel with minimal margins.](image)

The visviews.axesPanel's primary purpose is to allow an application to maintain fixed margins of consistent size in a larger mosaic when the overall figure resizes. The visviews.axesPanel extends uiextras.Panel, which provides the basic resizing infrastructure. The class also extends visviews.resizable and implements the needed infrastructure for the fixed margins. Applications call the getGaps and reposition methods to maintain the margins. The resizing infrastructure is transparent to most visualization classes that extend visviews.axesPanel. The only exception is when a graphics object has special behavior when resized (e.g., the MATLAB boxplot adjusts its own labels after resizing).

The visviews.axesPanel also extends visviews.clickable and supports the infrastructure for determining pixel positions and data coordinates of a point within the axes by providing the following utility functions:

```matlab
function [x, y, xInside, yInside] = getDataCoordinates(obj, point)

function hPosition = getPixelHitPosition(obj)
```

The visviews.cursorExplorable class requires an implementation of the first of these methods, while extending classes that must implement their own getDataCoordinates method use the second utility method. Most visualization classes that extend visviews.axesPanel only have to provide an updateString method specific to the visualization to enjoy cursor explorability as the axesPanel implementation of getDataCoordinates suffices to identify the cursor point for exploration.

Similarly, most extending classes need only implement a getClicked method to participate in linked navigation because visviews.axesPanel provides an implementation of getHitObjects.
3.3 The horizontal panel

The visviews.horizontalPanel class creates a resizable container panel that accommodates an arbitrary number of resizable panels arranged in a horizontal row. The class extends the uiextras.HBoxFlex class. The visviews.horizontalPanel class is configurable, explorable, clickable, and resizable, handling these features by propagating them appropriately to its subpanels. Fig. 5 shows an example container holding three axes panels.

![Figure 5: An example of a horizontal panel (visviews.horizontalPanel) container with three panels.](image)

The visviews.horizontalPanel class is a building block for other containers such as the visview.tabPanel. Individual child panels contained within these views must extend visviews.resizable. The visviews.horizontalPanel class is also clickable and cursor explorable. This class expects its child panels to extend the visviews.cursorExplorable and visviews.clickable classes. The class ignores child panels that do not extend visviews.cursorExplorable during cursor exploration.
3.4 The vertical panel

The visviews.verticalPanel class creates a resizable container panel that accommodates an arbitrary number of resizable panels arranged in a single column. The class extends the uiextras.VBoxFlex class. The visviews.verticalPanel class is configurable, explorable, clickable, and resizable, handling these features by propagating them appropriately to its subpanels. The subpanels are visviews.horisontalPanel objects to allow placement of dependent views in the same row. Fig. 6 shows an example container holding two axes panels.

![Figure 6: An example of a vertical panel object (visviews.verticalPanel) with two panels.](image)

The visviews.verticalPanel class is also clickable and cursor explorable. This class expects its child panels to extend the visviews.cursorExplorable and visviews.clickable classes. The class ignores child panels that do not extend visviews.cursorExplorable during cursor exploration.
3.5 The tab panel

The visviews.tabPanel class is a resizable tab panel that extends uiextras.tabPanel. The class extends the uiextras.tabPanel class. The visviews.tabPanel class is configurable, explorable, clickable, and resizable, handling these features by propagating them appropriately to its subpanels. The subpanels each contain a visviews.horizontalPanel. Fig. 7 shows an example of a tab panel with two tabs each having a horizontal panel with three axes panels.

Figure 7: An example of a tab panel object (visviews.tabPanel) containing two tabs.
4. Basic views

This section describes the basic viewing building blocks currently implemented in the +visviews package. Each of the visualizations described here extends visviews.axesPanel.

4.1 Block box plot

The block box plot (as implemented by the visviews.blockBoxPlot class) displays a series of vertical box plots using a compressed style. The block box plot displays the distribution of values of a summarizing function for a clump of consecutive time windows or epochs for all channels. Each window or epoch produces a single value for each element. The visviews.blockBoxPlot is configurable, resizable, clickable, and cursor explorable. Fig. 8 shows an example that groups clumps of 3 consecutive time windows into one box for display.

![Block box plot example](image)

**Figure 8**: A block box plot (visviews.blockBoxPlot). Each box represents three consecutive windows.

The block box plot can link to other plots. When the user clicks on a box, the plot reports the slice represented by the clicked box, allowing downstream link plots to display a more detailed view. For example, suppose a summary block box plot has clumps of 10 windows and has a downstream link to another summary block box plot with no groups (clumps of size 1). When the user clicks on a box in the first block box plot, the downstream plot displays 10 boxes corresponding to the blocks represented by the clicked box in the initial plot.

4.1.1 Configurable properties

The block box plot has five configurable parameters: the BoxColors, ClumpFactor, CombineMethod, IsClickable, and LinkDetails. BoxColors provides a list of colors used to alternate through in displaying the boxes. For data with lots of clumps, the boxes appear highly compressed due to limited viewing space and alternating colors help users distinguish the individual boxes.

The ClumpFactor specifies the number of consecutive windows represented by each box. When the ClumpFactor is one (the default), each box represents a single window or epoch. If ClumpFactor is greater than one, each box represents several consecutive blocks. Users can trade-off clump size versus block size to see different representations of the data.
The CombineMethod specifies how to combine multiple blocks into a single block to determine an overall block value. The value can be be 'max' (default), 'min', 'mean', or 'median'. Detail plots use this block value to determine slice colors.

For example, with 32 channels, a clump size of 3, and a block size of 1000 samples, the blockBlockPlot delivers a slice representing $32 \times 1000 \times 3$ worth of data. A detail plot such as signalStackedPlot combines this data based on its own CombineMethod property, say by taking the mean to plot $32 \times 1000$ data points on 32 line graphs. However, we would like to use line colors for the signals based on the block function values in the box plot. The detail plots use box plot's CombineMethod to combine the blocks to get appropriate colors for the slice. Usually signal plots combine signals using mean or median, while the summary plots such as blockBlockPlot use the max, although users may choose other combinations.

IsClickable, and LinkDetails are boolean properties of clickable objects. If the IsClickable property is true (the default), the blockBlockPlot will respond to user mouse clicks if incorporated into a linked figure by providing a slice of data to be linked. If the LinkDetails property is true (the default), an appropriate user mouse click will cause the slice to be displayed in a detail panel.

### 4.2 Block histogram plot

The block histogram plot (as implemented by visviews.blockHistogramPlot) shows two views of a distribution of summarizing values: a horizontal box plot and a scaled histogram. The horizontal line marks the scale of the fraction of data points represented by the tallest bar.

![A block histogram plot](image)

**Figure 9:** A block histogram plot (visviews.blockHistogramPlot)

### 4.2.1 Configurable properties

The block histogram plot has two settable properties, HistogramColor and NumberBins. These properties control the color of the histogram bars and the number of bins, respectively. The plot doesn’t support clicking to supply downstream targets.
4.3 Block image plot

The block image plot (as implemented by the visviews.bblockImagePlot class) displays the values of a summarizing function as an image (elements × clump), with pixel color representing the value of the function. The y-axis corresponds to elements (e.g., channels) and the x-axis corresponds to time (e.g., window or clump number). The visviews.blockImagePlot is configurable, resizable, clickable, and cursor explorable. Fig. 10 shows an example of a block image plot.

![Figure 10: A block image plot (visviews.blockImagePlot).](image)

The block function associated with this visualization controls the colors of the image and their correspondence to summary function values. The user sets these values when specifying the functions during function configuration. Generally, the block function thresholds its values based on a user-specified criteria such as z score, and the block image visualization maps the thresholded values to specified colors.

4.3.1 Configurable properties

The block image plot has four configurable properties: ClumpFactor, CombineMethod, IsClickable, and LinkDetails similar to the corresponding properties in visviews.boxBlockPlot.
4.4 Element box plot

The element box plot (as implemented by the visviews.elementBoxPlot class) displays a series of horizontal box plots using a compressed style. The element box plot displays the distribution of values of a summarizing function for each element (e.g., channel) as a horizontal box plot. The visviews.elementBoxPlot is configurable, resizable, clickable, and explorable. Fig. 11 shows an example of an element box plot.

![Element box plot](image1.png)

Figure 11: An element box plot (visviews.elementBoxPlot).

The element box plot can link to other summary or to detail plots. When the user clicks on a box, the underlying data represented by this box appears in the downstream linked plots. For example, if the user clicks on the second box from the top in visualization shown in Fig. 11, the details views display all of the windows or epochs associated with channel 2.

4.4.1 Configurable properties

The element box plot has the same configurable properties as the block box plot (visviews.blockBoxPlot). The main difference is that boxes represent the distribution of block values over channels for a fixed interval of blocks in the block box plot. In contrast, element box plot boxes represent the distribution of block values over all blocks for an individual channel or group of channels.
4.5 Signal shadow plot

The shadow signal plot (as implemented by visviews.signalShadowPlot) presents a compact summary of multiple signals over a fixed period. The shadow signal plot shows an envelope of the signals as a gray shadow. All signals fall within this shadow. The plot only displays individual signals designated as outliers for some time points. The shadow signal plot uses the signal z score at each time point to determine outliers. By default, outliers are those signals whose amplitude has a z score of at least three at some point in time. The visviews.signalShadowPlot is configurable, resizable, clickable, and explorable.

Fig. 12: A shadow signal plot (visviews.signalShadowPlot). The plot shows 32 sinusoidal functions that have random amplitude and phase.

Fig. 12 shows an example of a shadow signal plot for 32 sinusoidal functions with random amplitudes and phases. The three functions with large amplitudes appear as individual curves with colors assigned from the jet color map, which ranges from blue through green to red. The dark blue curve corresponds to the first function.

The shadow signal plot changes the labeling of the horizontal axis depending on whether the display is for epoched data or not. For window slices of non-epoched data, the plot uses the sampling rate to calculate the actual time in seconds corresponding to the data. For channel slices of non-epoched data, the plot labels the horizontal axis with the duration of the slice in seconds starting from zero. For window or channel slices of epoched data, the plot labels the horizontal axis using the epoch times in ms of the samples within the epoch. The plot always labels the horizontal axis with the window number (or range of windows numbers) of the corresponding slice.

Clicking one of the signals causes it to become the selected signal. The object displays the selected signal using a wider line and adds an indicator identifying the selected line to the label on the vertical axis. Selecting a signal causes dependent views to update their values. Unselect a signal by clicking in an empty part of the plot area.

4.5.1 Configurable properties

The shadow signal plot has eight configurable properties: CombineMethod, CutoffScore, RangeType, RemoveMean, ShowMean, ShowStd, SignalLabel, and TrimPercent. The CombineMethod specifies how
to combine multiple blocks when displaying a clumped slice. The CutoffScore parameter specifies the size of the z-score cutoff for outliers. A RangeType of 'both' indicates that outliers can occur in either direction from the mean, while 'upper' and 'lower' indicate outliers that occur only above or below the mean respectively. If RemoveMean is true, the visualization removes the mean of each element before trimming or plotting. If ShowMean is true, the plot displays the signal mean as a dark gray line. If ShowStd is true, the plot displays light gray lines to mark a distance of one standard deviation from the mean at each time point. SignalLabel designates the units for the y-axis. TrimPercent is a value in \([0, 100)\) indicating the percentage of extreme points to discard. For example, a TrimPercent value of 10 causes the visualization to remove the largest 5% and smallest 5% of the points before plotting.

4.5.2 Handling of slices

The shadow signal plot takes a two-dimensional slice of data and displays the time course of signals for the specified channels in a particular block or window. For example, in the default configuration, the block box plot (visviews.blockBoxPlot) and block image plot (visviews.blockImagePlot) deliver a slice corresponding to the signals for all elements when the user clicks a block. Similarly, the element box plot (visviews.elementBoxPlot) delivers a slice corresponding to the signals for all blocks when the user clicks an element. When the visualization calls the plot method of the shadow signal plot with this slice, shadow signal slice creates a display similar to that in Fig. 12.

Grouping supports more complicated slicing. A slice may represent any general subarray of elements \(\times\) samples \(\times\) blocks. Strategies appropriate 2D representations of the time courses of these depend on the type of slice and on whether or not the data is epoched. Table 3 summarizes the display strategies.

<table>
<thead>
<tr>
<th>Plot type by</th>
<th>Epoched?</th>
<th>Multiple blocks</th>
<th>Multiple elements</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Blocks plotted individually - time in seconds</td>
</tr>
<tr>
<td>Block</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Epochs plotted individually - epoch relative times in milliseconds</td>
</tr>
<tr>
<td>Block</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Blocks displayed consecutively - time in seconds</td>
</tr>
<tr>
<td>Block</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Epochs plotted individually - epoch relative times in milliseconds</td>
</tr>
<tr>
<td>Element</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Blocks plotted individually - time in seconds</td>
</tr>
<tr>
<td>Element</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Epochs plotted individually - epoch relative times in milliseconds</td>
</tr>
<tr>
<td>Element</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Blocks displayed consecutively - time in seconds</td>
</tr>
<tr>
<td>Element</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Epochs plotted individually - epoch relative times in milliseconds</td>
</tr>
</tbody>
</table>

4.5.3 Extensions

To implement a shadow plot with different criteria for signal outliers, extend the visviews.signalShadowPlot and override the getOutliers method.
4.6 Stacked signal plot

The stacked signal plot (as implemented by visviews.signalStackedPlot) shows each member of a slice of signals offset vertically, with the lowest numbered member at the top and the highest number member at the bottom. The stacked signal plot can show three possible slices: by channel, by sample, or by window. Plotting by window is the most traditional display. The plot displays the channels vertically, with the times for the window or block along the horizontal axis. Slicing by channel displays the signal for a fixed channel for a group of consecutive windows. Slicing by sample is most relevant for epoched signals. The visviews.signalStackedPlot is configurable, resizable, clickable, and explorable.

![A stacked signal plot](image)

**Figure 13:** A stacked signal plot (visviews.signalStackedPlot). The plot shows 20 blocks of 1000 samples consisting of Gaussian noise.

Fig. 13 shows an example of a stacked signal plot for 20 blocks of a random signal. The stacked signal plot takes its color scheme from the block function passed to the plot method (in this case the kurtosis function). For this example, only block 20 had a kurtosis with a z score greater than 3.

The stacked signal plot changes the labeling of the horizontal axis depending on whether the display is for epoched data or not. For window slices of non-epoched data, the plot uses the sampling rate to calculate the actual time in seconds corresponding to the data. For channel slices of non-epoched data, the plot labels the horizontal axis with the duration of the slice in seconds starting from zero. For window or channel slices of epoched data, the plot labels the horizontal axis using the epoch times in ms of the samples within the epoch. For sample slices, the plot always labels the horizontal axis with the window number of the corresponding point.

Clicking one of the signals causes it to become the selected signal. The object displays the selected signal using a wider line and adds an indicator identifying the selected line to the label on the vertical axis. Selecting a signal causes dependent views to update their values. Unselect a signal by clicking in an empty part of the plot area.
4.6.1 Configurable properties

The stacked signal plot has six configurable properties: ClippingOn, CombineMethod, RemoveMean, SignalLabel, SignalScale, and TrimPercent.

The logical ClippingOn flag indicates whether to clip individual signals so that their amplitudes do not exceed the extent of the graph. The amount of clipping depends on the vertical position of the signal within the display. The CombineMethod specifies how to combine multiple blocks when displaying a clumped slice. The SignalLabel should be a string specifying the signal units. The plot uses this string to display the signal scale as part of the horizontal axis label. The scale specifies the size in real data units of the spacing between tick marks. If RemoveMean is true, the visualization removes the mean of each element before trimming or plotting. SignalLabel designates the units for the y-axis.

The plot uses the SignalScale, which must be a positive numeric value, to determine the spacing between the individual line graphs. The plot calculates the spacing as the signal scale times the 10% trimmed mean of the standard deviations of the signal. That is, after calculating the standard deviation of each graph, the plot removes the lower and upper 5% of the values before calculating mean standard deviation. The resulting trimmed mean standard deviation times the SignalScale corresponds to the spacing between plots or vertical tick marks.

TrimPercent is a value in [0, 100) indicating the percentage of extreme points to discard. For example, a TrimPercent value of 10 causes the visualization to remove the largest 5% and smallest 5% of the points before plotting.

4.6.2 Handling of slices

Like the shadow signal plot, the stacked signal plot takes a two-dimensional slice of data and displays the time course of signals for the specified channels in a particular block or window. For example, in the default configuration, the block box plot (visviews.boxBoxPlot) and block image plot (visviews.blockImagePlot) deliver a slice corresponding to the signals for all elements when the user clicks a block. Similarly, the element box plot (visviews.elementBoxPlot) delivers a slice corresponding to the signals for all blocks when the user clicks an element. When the visualization calls the plot method of the stacked signal plot with this slice, the stacked signal plot creates a display similar to that in Fig. 13.

Grouping supports more complicated slicing. A slice may represent any general subarray of elements × samples × blocks. Strategies appropriate 2D representations of the time courses of these depend on the type of slice and on whether or not the data is epoched. Table 4 summarizes the display strategies.
<table>
<thead>
<tr>
<th>Plot type by</th>
<th>Epoched?</th>
<th>Multiple blocks</th>
<th>Multiple elements</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Blocks plotted individually - time in seconds</td>
</tr>
<tr>
<td>Block</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Epochs plotted individually - epoch relative times in milliseconds</td>
</tr>
<tr>
<td>Block</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Blocks displayed consecutively - time in seconds</td>
</tr>
<tr>
<td>Block</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Epochs for same element combined using CombineMethod - epoch relative times in milliseconds</td>
</tr>
<tr>
<td>Element</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Blocks plotted individually - time in seconds</td>
</tr>
<tr>
<td>Element</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Epochs plotted individually - epoch relative times in milliseconds</td>
</tr>
<tr>
<td>Element</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Blocks for same time combined over elements using CombineMethod - time in seconds</td>
</tr>
<tr>
<td>Element</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Epochs for same time combined over elements using CombineMethod - epoch relative times in milliseconds</td>
</tr>
</tbody>
</table>
5 Configuration of managed objects

MoBBED supports three types of configuration: specification of which summary functions to use (function configuration), specification of which visualizations to use (plot configuration), and setting of public properties of various objects (property configuration). This section describes how to use the configuration. Section 6 describes the underlying design and explains how to create new types of configurable objects.

5.1 Configuring functions

The EEGVIS application of Fig. 1 summarizes a dataset by windowing the data and evaluating specified functions such as the standard deviation or kurtosis on each window. The summary visualizations in the upper tab panel display these values, and the user can click on a plot to view the actual signals corresponding to that window in the detail panels below. Larger windows provide coarser summaries. Summary visualizations such as the block box plot or the block histogram plot display the distribution of the summary values without further processing. Other visualizations such as the block image plot use threshold values to focus visual attention.

The EEGVIS application displays each summary function evaluation in a separate tab panel in the summary section of the view. Users can add, remove, or edit the summary functions used by EEGVIS during visualization or by EEBROWSE during browsing.

![Example block function configuration GUI](visfuncs.functionConfig.png)

Fig. 14: Example block function configuration GUI (visfuncs.functionConfig).

Fig. 14 shows an example of the GUI used to configure functions. The GUI specifies two summary functions: kurtosis and standard deviation. Both functions threshold at a single level (using a z score of 3) using a global threshold (over all windows from all channels). The kurtosis function assigns red ([1, 0, 0]) to function values above the threshold and the default background color to other values. Thus, a window whose kurtosis value is an outlier appears highlighted in image plots and in direct plots of the signal. The standard deviation function uses cyan ([0, 1, 1]) to highlight its outliers.

The GUI in Fig. 14 follows the standard MoBBED configuration conventions. The Apply button appears grayed out until the user modifies the functions. Edits to the fields of the GUI table do not trigger a change in the application until the user presses the Apply button. The Reset button causes the GUI to
restore the values in effect at the time of GUI opening, not the values at the time of the previous apply operation. Resetting does not affect the visualization until the user presses the Apply button.

Users can modify entries in the table after pressing the Edit button to enter editing mode. They must press the Edit button again for the changes to take effect locally. MoBBED uses the MATLAB `eval` function to evaluate the function definition on the windowed data. Bad function evaluation strings cause MoBBED to throw an error, but should allow the user to continue.

MoBBED represents functions in the configuration GUI using an array of structures and in the visualization as `visfuncs.functionObj` objects. The MoBBED managed object infrastructure (`viscore.manageObj`, `viscore.dataManager`) handles the conversion between the representations in a transparent way. Table 5 summarizes the function structure fields and their meanings. The Short Name serves as the key for identifying functions in the visualization. These values, which must be unique, label the tabs in the summary visualization. Currently block is the only supported category for functions

<table>
<thead>
<tr>
<th>Structure field name</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackgroundColor</td>
<td>vector</td>
<td>Numeric color vector (1 × 3) giving background color</td>
</tr>
<tr>
<td>Enabled</td>
<td>logical</td>
<td>If false, the object should be disabled in the visualization</td>
</tr>
<tr>
<td>Category</td>
<td>string</td>
<td>Group heading for organizing objects</td>
</tr>
<tr>
<td>DisplayName</td>
<td>string</td>
<td>Name identifying the object in the visualization</td>
</tr>
<tr>
<td>Definition</td>
<td>string</td>
<td>String representation of function for evaluation with <code>eval</code></td>
</tr>
<tr>
<td>Description</td>
<td>string</td>
<td>Description used in tooltips in the visualization</td>
</tr>
<tr>
<td>ShortName</td>
<td>string</td>
<td>Brief name identifying object (key must be unique)</td>
</tr>
<tr>
<td>ThresholdColors</td>
<td>vector</td>
<td>Numeric color vector (n × 3) where n is the number of threshold levels</td>
</tr>
<tr>
<td>ThresholdLevels</td>
<td>numeric</td>
<td>Vector of cutoff levels</td>
</tr>
<tr>
<td>ThresholdScope</td>
<td>string</td>
<td>Indicates whether thresholds are computed globally or by element</td>
</tr>
<tr>
<td>ThresholdType</td>
<td>string</td>
<td>Criteria used for thresholding function values (e.g., z score or value)</td>
</tr>
</tbody>
</table>

Fig. 15 shows a sample function structure corresponding to Fig. 14. Users may configure either EEGBROWSE or EEGVIS with a different set of initial functions either by editing the `getDefaultFunctions` static method or by passing the structure array as the 'Functions' parameter to the class on construction.

```matlab
function fStruct = getDefaultFunctions()
    % Structure specifying the default functions (one per tab)
    fStruct = struct(...
        'Enabled', {true, true}, ...
        'Category', {'block', 'block'}, ...
        'DisplayName', {'Kurtosis', 'Standard Deviation'}, ...
        'ShortName', {'K', 'SD'}, ...
        'Definition', {'@(x) (kurtosis(x, 1, 2))', ...
                        '@(x) (std(x, 0, 2))'}, ...
        'ThresholdType', {'z score', 'z score'}, ...
        'ThresholdScope', {'global', 'global'}, ...
        'ThresholdLevels', {3, 3}, ...
        'ThresholdColors', {{[1, 0, 0], [0, 1, 1]}, ...
                            {[0.7, 0.7, 0.7], [0.7, 0.7, 0.7]}}, ...
        'BackgroundColor', {[0.7, 0.7, 0.7]}, ...
        'Description', {'Kurtosis computed for each (element, block)', ...
                        'Standard deviation for each (element, block)'} ...
    );
end % getDefaultFunctions
```
5.2 Configuring plots

Users can also control which visualizations to display using plot configuration in a similar way to function configuration. Fig. 16 shows a typical plot configuration GUI (visviews.plotConfig). Both EEGVIS and EEBROWSE support this type of configuration. Edits in the GUI table apply locally until the user presses the Apply button. The Apply button triggers redisplay of visualizations in EEGVIS so that they conform to the specification. In EEBROWSE, the Apply button changes the plots for the next visualization to be created. In both cases, changes to the plot configuration trigger updates in property configuration. For example, if the user deletes a plot, EEGVIS removes its property configuration from the property configuration window. Similarly, if the user adds a plot, EEGVIS includes its property configuration in the property configuration window, irrespective of the Enabled setting.

MoBBED represents plot specifications in the configuration GUI using an array of structures and in the visualization as visviews.plotObj objects. As with functions, the MoBBED managed object infrastructure (viscore.manageObj, viscore.dataManager) handles the conversion between the representations in a transparent way. Table 6 summarizes the plot structure fields and their meanings. The Plot name column values are the unique keys that identify plots for configuration purposes.

<table>
<thead>
<tr>
<th>Structure field name</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>logical</td>
<td>If false, the plot should not appear in the visualization</td>
</tr>
<tr>
<td>Category</td>
<td>string</td>
<td>Either summary or detail depending on section in which the panel appears</td>
</tr>
<tr>
<td>DisplayName</td>
<td>string</td>
<td>Name identifying the plot in the visualization</td>
</tr>
<tr>
<td>Definition</td>
<td>string</td>
<td>String representation of the class name of this plot</td>
</tr>
<tr>
<td>Sources</td>
<td>string</td>
<td>None and Master indicate top-level in summary and details, respectively</td>
</tr>
<tr>
<td>Description</td>
<td>string</td>
<td>Description used in tooltips in the visualization</td>
</tr>
</tbody>
</table>
Fig. 17 shows a sample structure for a top-level block image plot. Users may configure either EEGBROWSE or EEGVIS with a different set of initial plots either by editing the getDefaultPlots static method or by passing the structure array as the 'Plots' parameter to the class on construction.

function pStruct = getDefaultPlots()
    % Structure specifying the individual visualizations used
    pStruct = struct(...
        'Enabled', {true}, ...
        'Category', {'summary'}, ...
        'DisplayName', {'Block image'}, ...
        'Definition', {'visviews.blockImagePlot'}, ...
        'Sources', {'None'}, ...
        'Description', {'Image of blocked value array'});
end

Figure 17: An example structure for initializing plots.

If the entry's enabled checkbox is checked, the viewer displays the corresponding visualization. If unchecked, the viewer does not show the panel, but retains the property configuration information. This allows the user to change which items to view without having to reconfigure later.

The category value determines the location of the panel in the viewer. Summary panels (top display) provide condensed or summary information, such as the kurtosis, computed on different time windows. Detail panels (bottom) display information about the signal. The plot name (the value of the DisplayName field in the structure) uniquely identifies the plot in the visualization, and the class name specifies the class that actually creates the visualization.

A panel can have an optional upstream source (identified by plot name) that determines from where it receives its data. Summary panels can only have summary panels as sources, while detail panels can only have detailed panels as sources. Summary panels that do not have an upstream source, compute their values directly from the data, while summary panels with an upstream source compute their values on a portion of the data selected from their source. A summary panel can be the target of an upstream source of multiple summary panels, but can only have one parent upstream source. See Section 2.3 for more details.

Detail panels with a Master source receive their data from all summary panels whose LinkDetails property is true. When the visualization starts, the detail panels receive their results about the first block. When the user clicks on a leaf summary panel, all top-level detail panels update their data with the data selected from the leaf summary panel. When a user clicks on a detail panel that is acting as an upstream source, the child panels update based on the selection.
5.3 Configuring properties

The MoBBED data infrastructure supports user configuration of public properties through a configuration GUI that is similar to the MATLAB property manager. Configurable objects must extend `visprops.configurable` and provide implementations of the static method `getDefaultProperties()`.  

![Configuration GUI](image)

**Figure 18**: An example of the property configuration GUI for a sample test class.

Fig. 18 shows an example of the configuration GUI for a test class with an example of each type of property. For example, ‘Block name’ is a simple string property (`visprops.stringProperty`), while ‘Block size’ is a double value (`visprops.doubleProperty`), restricted to be in $(0, \infty)$. Users specify such range restrictions in the options, and MoBBED validates the value and restores the previous value if the user tries to enter an invalid item.

Table 7 shows a list of the available properties. Each of these classes extends `visprops.property` and provides methods for converting between the GUI internal representation and MATLAB values along with validation.
Table 7: Property classes in +visprops representing different types of configurable properties

<table>
<thead>
<tr>
<th>Class</th>
<th>Data</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>visprops.colorListProperty</td>
<td>List of a fixed number of colors</td>
<td>Sublist of color choosers</td>
</tr>
<tr>
<td>visprops.colorProperty</td>
<td>Single color</td>
<td>Color chooser</td>
</tr>
<tr>
<td>visprops.enumeratedProperty</td>
<td>Double value in specified interval</td>
<td>Text (entered and checked)</td>
</tr>
<tr>
<td>visprops.integerProperty</td>
<td>Integer value in specified interval</td>
<td>Text (entered and checked)</td>
</tr>
<tr>
<td>visprops.intervalProperty</td>
<td>Real interval</td>
<td>A list of two text boxes</td>
</tr>
<tr>
<td>visprops.logicalProperty</td>
<td>Boolean value (true or false)</td>
<td>Pull-down menu</td>
</tr>
<tr>
<td>visprops.stringListProperty</td>
<td>Editable list of strings (fixed length)</td>
<td>List of text boxes</td>
</tr>
<tr>
<td>visprops.stringProperty</td>
<td>String</td>
<td>Text</td>
</tr>
<tr>
<td>visprops.unsignedIntegerProperty</td>
<td>Unsigned integer in specified interval</td>
<td>Text (entered and checked)</td>
</tr>
<tr>
<td>visprops.vector</td>
<td>Row or column vector</td>
<td>Text (entered and checked)</td>
</tr>
</tbody>
</table>

Users must implement the static getDefaultProperties() method for their configurable objects. This method returns a structure array, which encapsulates how the GUI objects map into public properties of the object. Table 8 describes the structure field names for configuration. In particular, FieldName specifies the name of the public property in the configurable object.

Table 8: The configuration structure returned by visprops.configurable.getDefaultProperties

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>logical</td>
<td>If false, the property entry appears grayed out in the Property Manager</td>
</tr>
<tr>
<td>Category</td>
<td>string</td>
<td>Group heading for organizing properties in the Property Manager</td>
</tr>
<tr>
<td>DisplayName</td>
<td>string</td>
<td>Name identifying the property in the Property Manager</td>
</tr>
<tr>
<td>FieldName</td>
<td>string</td>
<td>Name of public property being configured by this entry</td>
</tr>
<tr>
<td>Value</td>
<td>---</td>
<td>Value of the property being configured by this entry</td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
<td>Class name of the property type (for GUI representation and conversion)</td>
</tr>
<tr>
<td>Editable</td>
<td>logical</td>
<td>If true, the property value can be modified by the user through the GUI</td>
</tr>
<tr>
<td>Options</td>
<td>---</td>
<td>Special parameters depending on type of property</td>
</tr>
<tr>
<td>Description</td>
<td>string</td>
<td>Description of property suitable for display in status window of GUI</td>
</tr>
</tbody>
</table>

Fig. 19 shows an example of a property structure for a list of colors of fixed length.

```matlab
function settings = getDefaultProperties()
    % Structure specifying how to set configurable public properties
    settings = struct(...
        'Enabled', true, ...
        'Category', 'visviews.blockBoxPlot', ...
        'DisplayName', 'Box plot colors', ...
        'FieldName', 'BoxColors', ...
        'Value', [0.7, 0.7, 0.7; 0, 0, 1], ...
        'Type', visprops.colorListProperty, ...
        'Editable', true, ...
        'Options', '', ...
        'Description', 'alternating box plot colors');
end % getDefaultProperties
```

Figure 19: Sample getDefaultProperties() method for a two-color list.
6 Configuration architecture

MoBBED allows users to configure different aspects of the visualization including which visualizations to show, how to arrange them, how to connect them, what their visual properties should be, and which functions should be used for summaries. Section 5 described configuration from the user perspective. This section describes the overall architecture from the application developer perspective.

The MoBBED's architecture follows a high-level modified model-view-controller (MVC) architecture in order to support well-structured interaction among configuration GUIs. User changes of visualization properties or of functions trigger a reset and redrawing of the visualization as one would expect. However, user changes to plot specifications such as adding or deleting a visualization panel must trigger changes to the properties configuration in addition to redrawing the visualization.

Fig. 20 shows the classical model view control (MVC) architecture. The controller handles user events. The model does not contain a direct reference to either the controller or the view, but it may listen for changes in the GUI and respond by modifying its data. The controller handles user events by directly modifying model data or GUI elements in the controller. The view needs access to the model in order to fill its elements with data values. In the classical MVC design pattern, the view does not have direct access to the controller, but may register to receive notification of events.

![Figure 20: Classic model-view-controller (MVC) architecture. The model holds the data, the view displays a GUI, and the controller handles user events. Solid arrows indicate direct access, while dashed lines indicate indirect access, usually implemented by a listener-callback mechanism.](image)

Fig. 21 shows the MoBBED version of MVC. An application that wishes to configure objects using the MoBBED infrastructure creates and holds a reference to a data selector (the controller) (Fig. 21a). The data selector holds references to both the model (Fig. 21c) and the view (Fig. 21d). The application should not keep separate references to these objects but request information through the data selector instead. The application uses the direct path to the selector (Fig. 21a) to access data or to create a view in response to a user request.
The application should also register as a listener for the StateChanged event of its selector (Fig. 21b). The application uses this indirect path to respond to changes in view state, for example by redrawing the visualization if the user adds a new visualization panel.

The model (as implemented by viscore.dataManager) is simply a data container for managed objects (viscore.managedObj) and has no contact with either the view or the controller. The data manager provides fast key lookup of individual managed objects via a map (Fig. 21d).

A managed object has a structure array of properties and an object ID that identifies it with a particular MATLAB entity such as a viewing panel or block function. Managed objects provide methods for converting between the structure and object representations of data. The views work with the object representation, but users configure values in terms of the structure. The default structure fields include ID, Enabled, Category, DisplayName, Definition, and Description. Extending classes often include additional fields and override the conversion methods. Three classes: visfuncs.functionObj,
visprops.configurableObj, and visviews.plotObj extend viscore.managedObj to provide GUI configuration for block functions, plot specifications, and visualization properties, respectively.

The MoBBED design deviates from the classical MVC because top-level applications such as EEGVIS and EEGBROWSE participate in three separate MVC units: configuration of plots (specify the visualizations to show), configuration of functions (specify the functions to use as summaries), and configuration of properties (describe the visual aspects of the graphs themselves). Thus, these top-level applications cannot be their own controllers by simply extending a base case, since this would result in conflicts among the three units. Each of these units has its own configuration GUI, and the GUIs for the three units can be simultaneously active. Furthermore, in addition to triggering redisplay, a change in the plot configuration should trigger a change in the property GUI because different visualizations may require configuration.

All of the configuration units use the same basic strategy for responding to user requests. The dataConfig class keeps two private copies of the dataManager: the original version of its data when the GUI opened (for responses to Reset), and the current version of the data (as edited by the user through the GUI). When the user presses the Apply button on the GUI, the GUI calls the updateState method of its selector in the Apply button callback (Fig. 21g). In other words, the data selector's updateState is essentially a callback for the GUI's Apply button. The updateState method performs bookkeeping and then notifies all listeners of itsStateChanged event. Typically, applications redisplay as needed after providing their own bookkeeping in the individual listener callbacks.

The MVC pattern of Fig. 21 only controls the Apply button. The view keeps its own local copies of the data and responds to user editing events with no outside interaction. Thus, we refer to this as a high-level MVC. Because the view only updates the model through the selector during the Apply callback, the design does not require the usual indirect link between view and model. If the application imposes a change in model, the selector forces an update of the view's local copy.
Figure 22: The overall architecture of EEGVIS and EEGBROWSE. The application manages how changes in one MVC branch affect other MVC branches.
7 Deployment

Users can create visualization shown in Fig. 1 in several ways. Fig. 23 shows an example that creates a 3D array of random exponentially distributed values and creates a visualization using the EEGVIS wrapper function `eegvis`. The `eegvis` function returns a handle to the figure window.

```matlab
data = random('exp', 2, [32, 1000, 20]);
hfig = eegvis(data);
```

Figure 23: MATLAB code to create a view for some random 3D array data.

Alternatively, users can choose to work directly from the class library by calling the `visviews.dualView` constructor. The `bv1` variable contains a handle for the newly created object.

```matlab
data = random('exp', 2, [32, 1000, 20]);
bv1 = visviews.dualView('VisData', data);
```

Figure 24: MATLAB code to create a view for some random 3D array data.

Users can also could create the viewer first and then set and change the data. Fig. 25 shows an example of for this scenario. The example also illustrates using a blocked data object to encapsulate the data, allowing the user to specify some data-specific information.

```matlab
bv2 = visviews.dualView();
data = random('exp', 2, [32, 1000, 20]);
testVD = viscore.blockedData(data, 'Rand1 normal constructor');
bv2.setDataSource(testData);
bv2.reset(true, true, true);
```

Figure 24: Another scenario for creating a viewer from MATLAB

The `visviews.dualView` has four optional parameters described in Table 9.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'VisData'</td>
<td>blockedData object or a 3D array of data</td>
</tr>
<tr>
<td>'Functions'</td>
<td>dataManager, structure, or cell array of initial functions</td>
</tr>
<tr>
<td>'Plots'</td>
<td>dataManager, structure, or cell array of initial plots</td>
</tr>
<tr>
<td>'Properties'</td>
<td>dataManager, structure, or cell array of initial properties</td>
</tr>
</tbody>
</table>
8 Interaction with EEGLAB

As illustrated in Fig 25, the toolkit also comes with a browser for reading in files. Currently this browser, only works for EEGLAB .set files, but support for additional formats should be available. After moving to a directory containing .set files, the user clicks on a file name to choose a file. The actions EEGBROWSE takes depend on which check boxes the user chooses at the bottom of the screen. If the Load workspace on select box is checked, EEGBROWSE loads the selected file into the MATLAB workspace as an EEGLAB EEG structure. If Preview on select is checked, EEGBROWSE displays an EEGVIS figure window with the data. If Preview in new figure is checked, EEGBROWSE displays each preview in a new EEGVIS figure.

![Figure 25: EEGBROWSE allows users to browse a directory of data files.](image)

The Functions, Plot list, and Properties buttons allow users to configure the function tabs, visualization display, and the public properties of the visualizations that EEGBROWSE creates. To start EEGBROWSE from the command line or from within a script, use:

```matlab
eb = eegbrowse();
```

The call to start EEGBROWSE includes a number of optional parameters for setting the default starting directory and the default plots, functions and properties.
The EEGBROWSE application uses some of EEGLAB's functions for reading EEG data and EEGLAB needs to be in MATLAB's path for EEGBROWSE to work. If the entire EEEVIS package is unzipped in the EEGLAB plugins directory, EEGBROWSE and EEEVISI will integrate themselves into the EEGLAB menu system. Fig. 26 shows how EEGBROWSE adds itself as a previewer for EEGLAB. The EEEVIS makes itself available under the EEGLAB Plot menu.

![EEGBROWSE](image)

*Figure 26: The EEGBROWSE application is available as a data previewer for EEGLAB.*

The EEGBROWSE application is also available as a standalone application that users can run from their desktop on machines without an installed MATLAB.

### 9 Availability

The MoBBED viewing infrastructure is freely available under the GNU General Public License and is included as part of the EEEVIS project (http://visual.cs.utsa.edu/eegvis).

**Acknowledgments:** EEGLAB is produced by the UCSD Swartz Center for Computational Neuroscience. The +uiextras package is part of the Gui Layout Toolbox was written by Ben Tordoff of MATHWORKS. This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-10-2-0022. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.
References:


Notes: This report was modified on 2/2/2012. Tables 3 and 4 were added to provide additional details about the handling of slices. A few typographical corrections were also made.

This report was modified on 2/11/2012 to include a revised acknowledgment.

This report was modified on 3/3/2012 to change the capitalization of eegVis to EEGVIS and eegBrowse to EEGBROWSE.

This report was modified on 3/20/2012 to change the name of shadowSignalPlot to signalShadowPlot and stackedSignalPlot to signalStackedPlot. Also, the example of linking was modified to remove the scalp map as an example.