XVS 2.0 Documentation
Port to Qt4

Magnus Haw

August 20, 2010

Contents

Part I
General Overview

1 What is xvs?

Xvs is a graphing utility for 1D time dependent data. It was made to visualize and examine PDE output from RNPL output files (sdf format).

Xvs has a simple graphical user interface which is composed of a main window, plot widgets, and a control dialog. The gui is entirely written in Qt (to increase portability) and makes use of OpenGL hardware acceleration. Additionally, Qt uses the native window styles so users should not be surprised if xvs looks different on different machines.

The main window is the central window that is executed from main.cpp and contains up to 25 plot widgets. It has a menu bar, several buttons, and update/info bars. The main window offers the most general functions such as opening data, exporting data, exiting, setting global variables, and opening the control dialog.

The plot widgets that open in the main window currently display one or more functions f(x,@t). Each plot has a number of options that are accesible in the form of mouse manipulations, hot keys (switch between modes, translate plot view, advance to next time step etc), and a context menu (menu that appears on right-click).

The control dialog is a popup window that gives greater control over the properties of the plot widgets: labels, modes, view window, etc.

2 How to use xvs

2.1 Open File

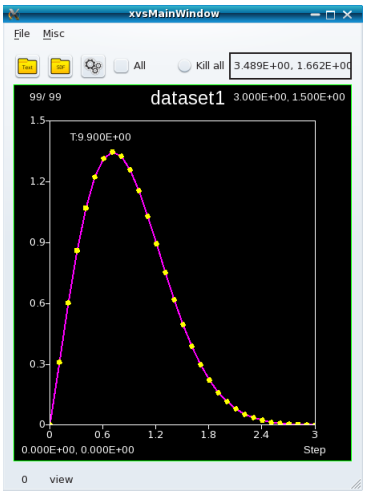
The first step in using xvs is to open a datafile for viewing. To open a file go to the file menu and click on the 'open sdf' or 'open text file' option or click on one of the two toolbar buttons with labeled folder icons. After clicking on an open file option, a dialog will popup where you can select the file you wish to open. You must have a data file in one of these two formats: xvs-compatible text file or a .sdf file (See section 4).

2.2 Basic Features

After opening the file, a plot widget will open and fill the display portion of the main window. The plot widget will start in 'Zoom mode' (hotkey: 'z') you may examine the data by zooming in/out by selecting portions of the screen with the mouse, using the zoom buttons at the top right corner of the screen, toggling autoscale mode by pressing 'w', or by opening the control dialog and setting the view screen manually. The scroll wheel and arrow keys will translate the viewing window in the associated directions (the scroll wheel will switch to the x direction if the 'Alt' or 'Option' key is depressed).

To move through the various time steps of the data, switch to auto mode (hotkey: 'a') or to step mode (hotkey: 's'). The time step can also be set using a number of index modifiers in the control dialog. Auto mode will automatically animate the time steps at approximately 13 fps. A right click will stop the animation and pop up a context menu; left click on the plot window again to restart the animation. Step mode requires manual input to move through the frames: right arrow key and left click move the index forward by one, left arrow and right click move the index back by one. Note: all modifications to the viewing window in one mode will remain the other modes. There is also a freeze mode (hotkey: 'f') that freezes the current view.

Figure 1: Example of single plot window in step mode



Further viewing options are available in the control dialog (line color, width, marker type etc.).

2.3 File Output

After browsing the data, the user may want some form of output. Currently, the only supported export formats are a xvs-compatible text format, sdf, mpeg, and jpeg. These options are available from the file→export menu and will allow the user to direct the export location via a 'save as' type dialog.

For a full description of these options see section 4.

3 Basic Dependencies

The basic dependencies are:

Qt4 Some features will not compile without some version of Qt4. This is because the build depends on qmake 2.x and earlier versions of qt have qmake 1.x

libbbhutil.a The bbhutil library is needed for sdf i/o operations. It can be created by installing RNPL and then putting the location of libbbhutil.a in $PATH.

mjpegtools The mpeg export features depend on these libraries but the rest of xvs will continue to work without mjpegtools.

Part II
XVS Features

4 Supported I/O Formats

The currently supported i/o formats are a simple text format, the binary SDF format accesible via the bbh utility library (libbbhutil.a), and export-only mpeg and jpeg formats.

4.1 Text Format

The text file format consists of a basic 2D list with header information. The first line of the file lists the number of distinct datasets present in the file (i.e the number of plot windows that will be opened). The next line is blank. The next two lines compose the header for the first dataset: name of the data and number of time steps. The following line is a tab separated list of the x-coordinates preceeded by the text "xcoord". Then the main data follows: first column time, remaining columns tab separated y-values...

NumDatasets:2

name_of_data
100

xcoord→	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9...
0.00000	yval	yval	yval	yval	yval	yval	yval	yval	yval	yval...
0.00010	yval	yval	yval	yval	yval	yval	yval	yval	yval	yval...
...										
0.00990	yval	yval	yval	yval	yval	yval	yval	yval	yval	yval...

name_of_data2
50

xcoord→	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9...
0.00000	yval	yval	yval	yval	yval	yval	yval	yval	yval	yval...
...										
0.00490	yval	yval	yval	yval	yval	yval	yval	yval	yval	yval...

This format can be both read and exported for 12 or fewer datasets. Note: changes to the plot options (color, view etc.) are not saved.

4.2 Scientific Data Format (SDF)

The SDF format is exclusively written and read by the bbhutil library functions. The xvs functions that read sdf files require a locally compiled version of this library to operate (libbbhutil.a). This library was incorporated as is, has its own documentation, and is henforth treated as a black box. Previously, SDF files were sent to the xvs server via various client programs (e.g sdftoxvs).

This version of xvs does not yet have a server and can therefore only access sdf files via the menu option and the toolbar button.

Certain aspects of SDF files may still be unaccounted for such as the multiple-dataset options and the >1D datasets.

Export of data to sdf format only targets the active widget: the apply to all option will not save all the plots in a single sdf file. Additionally, plots with more than one dataset will only save the first dataset.

4.3 Mpeg format

The data in a single plot window can be exported as an mpeg movie via the file→export menu. The export requires: mjpegtools and mjpegtools2-shlibs.

4.4 Jpeg format

The data in a single plot window can be exported to jpeg(s) via the file→export menu. If the apply-all option is set, all timesteps will be exported.

5 Main window

The main window has several featues of interest to the user: the menus, the toolbar, the statusbar, and the hotkeys.

5.1 Menu bar

File Menu This menu contains the most basic functions of the program.

- **Open→Text File** This option will open a file dialog box and allow the user to navigate the directory tree and select a single file that ends with *.txt* or *.dat*. If the file is formatted correctly, it will open with one plot window (in zoom mode) per dataset contained. If formatted incorrectly, the program might crash.
- **Open→Sdf File** This option will open a file dialog box and allow the user to navigate the directory tree and select a single file that ends with *.sdf*. If the file is formatted correctly, it will open with one plot window (in zoom mode).
- **Export→Sdf** Will export the active plot to an sdf file. Not affected by apply-all option.
- **Export→Text** Exports active plot to text format. If apply all is active, all plots are exported to single file.
- **Export→Mpeg** Exports active plot data to mpeg movie. If libraries not available, exports to jpegs.
- **Export→Jpeg** Exports current plot to jpeg format (also can export to png if specified in filename; pdf not supported).
- **Help** This will pop up a dialog box that will direct users to the documentation.
- **XVS Exit** This option will exit xvs with an 'Are you sure you want to exit?' dialog.

Misc Menu This menu contains miscellaneous functions.

- **Show Control Dialog** This option will show the control dialog box.
- **Set Insert Fuzz** This option sets the fuzzy tolerance of comparison functions (i.e the difference is considered zero if less than abs(fuzz)).
- **Reset Insert Fuzz** Sets fuzz to 1e-10.
- **Toggle Ctrl-space kills windows** toggles the shortcut Ctrl-space on/off.

5.2 Toolbar and Statusbar

Toolbar This bar is located at the top of the main window below the menu bar. It contains the most used functions of the program.

Open Text File This is the leftmost button on the toolbar. It calls the open text action which is identical to the one in the filemenu (see previous section)

Open SDF File Second from left on toolbar, calls open sdf action.

Show Control Dialog Shows the control dialog.

Select all checkbox Toggles the select all property. This property generally lets functions act on all the widgets as opposed to just the active one.

Kill all plot windows Kills all plot windows.

Cursor tracking label This label shows the location of the cursor in the active plot window.

Statusbar This bar is located at the bottom of the main window. It displays updates, error messages, and notices.

Index label This label is the small box on the lower left corner. It displays the index of the active plot.

Message label Longer label that displays text messages. Usually shows the current update sent to the control dialog.

Temporary messages These messages will be shown for $\approx$3 seconds and will cover the other labels. Generally these messages will be notices of completion or will display the results of an action.

5.3 Hot Keys

The hotkeys are taken directly from the original xvs

Space: Toggle Single Window View

Ctrl-Space: Kill current window

A: Animate

B: Back (moves one plot backwards)

C: Clone (Duplicate)

D: Bring up Data Subpanel

E: Enter Easy Zoom mode

F: Enter Freeze mode

H: Help (this file)

M: Bring up Markers Subpanel

N: Next (moves to next widget)

S: Enter Single Step Mode

V: Bringup View Subpanel

W: Toggle Full Scale Mode

X: Enter X zoom Mode

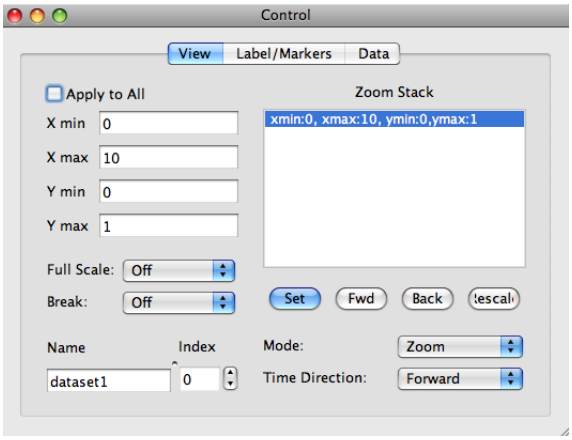
Y: Enter Y zoom Mode

Z: Enter XY zoom Mode

6 Control dialog

Several of the features of the previous xvs control dialog have not been implemented.

Figure 2: Control Dialog View Tab



View tab This tab contains all the functions associated with view screen manipulation (plot boundaries)

Plot boundary inputs (xy min/max) These lines serve as place holders for the current plot boundaries. To change the plot boundaries, the user must edit these lines and then click the *Set* button.

Full scale combo box Toggles autoscale feature on/off.

Zoom stack view Shows the previous window settings. Note: selection via the mouse will NOT change settings. User must use the back, forward, rescale, and set buttons to change window settings.

Set button Sets the active plot (or all plots if apply all property is on) to the values in the adjacent plot input lines.

Back button Sets the active plot to the previous plot settings.

Forward button Sets the active plot to the next plot settings in the zoom stack.

Rescale button Fits plot window to current data set.

Select all checkbox Toggles the select all property. This property generally lets functions act on all the widgets as opposed to just the active one.

Mode combo box Selects one of three viewing modes: Zoom, Step, Auto.

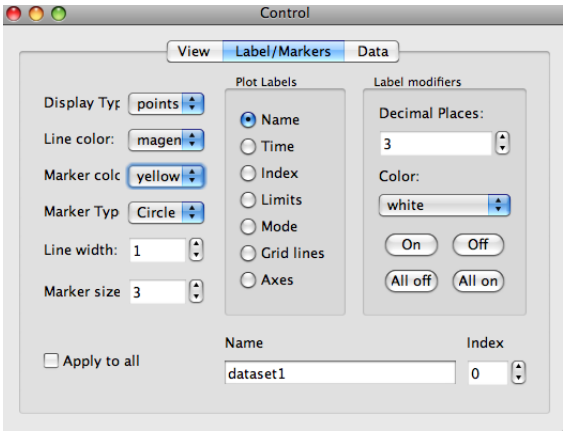
Time direction combo box Selects direction of time animation.

Break combo box Toggles the break status (animation doesn't loop if break is on).

Name input line Displays/changes the name on the active plot.

Time index spinbox Displays/changes time index of active plot.

Figure 3: Control Dialog Label/Markers Tab



Labels/Markers tab Contains modifier functions for label/markers.

Display type combo box selects the visible markers/lines.

Marker color combo box selects color of markers.

Line color combo box selects line color.

Marker type combo box selects marker type: circle, square, or triangle.

Point size spinbox selects pixel width of markers [1-6].

Line width spinbox selects line width [1-4].

Select all checkbox Toggles the select all property. This property generally lets functions act on all the widgets as opposed to just the active one.

Plot label radio buttons These buttons determine which label the other functions on the tab apply to.

Decimal places spinbox determines number of decimal places for given label.

Color combo box determines the color of a given label.

On/Off buttons These will turn the selected label on/off.

All on/all off These will turn all labels on/off.

Browse tab Displays data for the current time step.

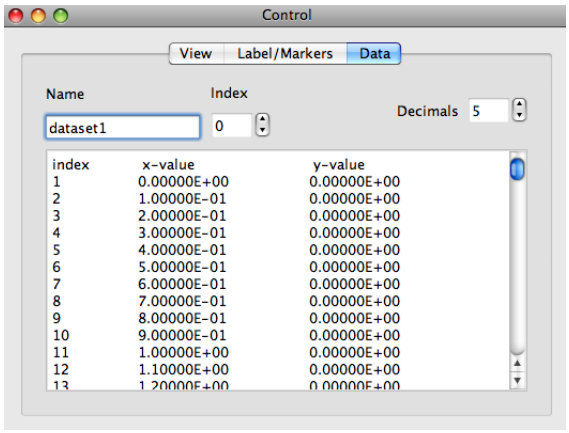
Name input line Displays/changes the name on the active plot.

Time index spinbox Displays/changes time index of active plot.

Decimals input line Displays/changes number of decimals shown in display window.

Display window Shows the index, x-val, and y-val in columns for the current timestep. Will only show data of first function in plot.

Figure 4: Control Dialog Browse Tab



7 Plot widget

The plot widget is the primary display medium of xvs. It has several modes for viewing and selecting data as well as a number of transformation options. In the main window, a plot widget becomes active when moused over and displays an active status via a green border (inactive: red border). To activate keyboard focus when switching between the control dialog and the main window a widget must be clicked on. A number of the hotkeys (section 5.3) relate directly to plot behavior such as the mode selectors, autoscale option (w), clone function (c), and maximization/minimization (space).

7.1 Easy Zoom Mode: e

This mode allows the user to zoom in and out by selecting portions of the window with the mouse. The upper right corner of the plot will also display zoom-in, zoom-out, forward, and back buttons to provide more control. The window can be translated via the arrow keys and the scroll wheel on the mouse (press 'Alt' or 'Option' to scroll right/left with the wheel).

7.2 Step Mode: s

In step mode, the zoom features are disabled. A left-click will advance the timestep by one and a right-click will decrease the timestep by one (consequently, the context menu is not available in step mode). The arrow keys change the time index by one in the appropriate direction. Holding the arrow keys down may allow faster animation would otherwise be shown in automode.

7.3 Auto Mode: a

Auto mode animates the time steps of the respective dataset in a looped manner. The direction of animation is set in the control dialog. If the break option is toggled in the control dialog, the animation will stop after a single period. The animation will stop if right-clicked and will allow context menu functions to be accessed. To restart the animation, left-click on the appropriate plot window.

7.4 Freeze Mode: f

Freeze mode freezes the current screen and disables all interactive mouse actions.

7.5 X Zoom Mode: x

This mode allows the user to zoom in and out by clicking the mouse. A right click will zoom in x by a factor of two around the cursor (zoom will correct to remain inside data domain). Left click will go back to previous zoom.

7.6 Y Zoom Mode: y

This mode allows the user to zoom in and out by clicking the mouse. A right click will zoom in y by a factor of two around the cursor (zoom will correct to remain inside data domain). Left click will go back to previous zoom.

7.7 XY Zoom Mode: z

This mode allows the user to zoom in and out by clicking the mouse. A right click will zoom in x and y by a factor of two around the cursor (zoom will correct to remain inside data domain). Left click will go back to previous zoom.

7.8 Selection Mode

Selection mode is a mode which is only activated by certain functions (currently only applicator functions). It changes the active/inactive border of a selected widget to blue. In selection mode, click on a widget to select it and a blue border should appear. Widgets that are selected via left clicks can be deselected by right clicks. For certain functions such as the convergence tests, it is important to select widgets in a certain order (e.g least to greatest level).

7.9 Applicator Functions

There are several implemented functions that can be applied to the datasets in xvs and are accessed via the context menu. These include transformations of the (x,y, and/or t) data, comparisons between datasets, and convergence tests. Generally these functions will automatically create a new plot window with the transformed data but the function can be applied in place if the control key is depressed. Also, for those functions that operate on more than one dataset, selection mode will be activated: widgets are then selected via left clicks and deselected by right clicks. A brief overview of the available functions is shown below.

Compare [xyt]
: These functions compare two sets of data. Each comparison uses either a fuzzy comparison (i.e there is a given tolerance for the difference) or an exact comparison (double precision).

Apply [xy]
: These functions apply a transformation to a single data set.

dy/dx	Takes the derivative with repect to x.
x->log(x+c)	Transforms x to log(x+c).
x y	y →x*y.
y/x	y→y/x.

x^p	$x \rightarrow x^p$.
x^p y	$x \rightarrow x^p$ * y.
y^p	$y \rightarrow y^p$

Convergence [O(h^p)]

These functions take multiple plots that are at different levels ($dx_l = dx_0 * \frac{1}{2}^l$, $dt_l = dt_0 * \frac{1}{2}^l$) and difference them such that $\text{diff}_i = (l_{i+1} - l_i) * 2^{pl}$. If these differences appear to converge, then the data can be said to have pth order convergence. The data must be selected in order from lowest level to highest level for the program to recognize the applicability of a convergence test.

Merge/split [plots]

These functions merge and split datasets that have equal time data. Merge activates selection mode and selects 2 datasets: will normally add a new window with both datasets (if the 'ctrl' key is depressed xvs will delete the original plots). Split takes a single widget and splits it into its individual datasets (if the 'ctrl' key is depressed xvs will delete the original plot).

Part III
Description of Classes

This version of xvs was constructed using Qt, a C++ based application framework. With xvs, the basic form of link between the ui and the base code is the 'connect' command. The main form of connection is in the form of signals and slots. Signals are emitted to all objects when certain conditions are fulfilled and slots are member functions of objects that can be connected to (and therefore called) by signals. After a connection has been made between a signal and slot, the slot will be called every time the signal is emitted.

The format of the control dialog and main windows was contructed using the Qt Designer gui builder. The *.ui files are a product of the gui builder and serve mainly as header files where most of the widgets are declared and organized into layouts.

The gui also takes advantage of a number of static Qt library functions that create specific dialogues such as getting the user to input a integer or string.

The following description of the classes is by no means exhaustive but was written to give a basic understanding of the structure and workings of xvs.

8 xvsMainWindow

The xvsMainWindow class is the central entity in xvs. It constructs and contains all of the other classes during runtime. It keeps track of the application-wide variables, the plots, and the control dialog. Furthermore, it implements the context menu and therefore contains all the associated menus and actions.

8.1 State variables

There are currently 9 state variables that the main window keeps track of. They are as follows (default values are in brackets):

double ifuzz [1e-10] This variable contains the fuzz value or the allowed tolerance for error in comparisons between doubles.

int numPlotWidgets [0] Tracks number of plot widgets present. Changed only by addPlotWidget(), removePlotWidget(), and killAllPlotWindows.

int activeWidgetIndex[0] Tracks active plot window. Affected by changeActivePlot(int i).

int gridSize [2] Tracks the size of grid that contains plot widgets. Affected by organizePlotWidgets() and addPlotWidget().

bool allCheckBoxState [false] This contains the state of apply all option. Affected by context menu action, main window and control dialog check boxes

bool ctrl_spaceKills[true] State of shortcut. Only affected by checkable menu option.

bool isPlotMaximized [false] Tracks whether the current plot is maximized or minimized.

bool controlButtonDepressed [false] Tracks state of control button. See re-implemented protected mouse click events for details.

xvsMainWindow::*pt2Member [NULL] Pointer to xvsMainWindow function. Used principally by selection mode to call the necessary function after selection process is complete.

8.2 Initialization functions

These functions are called exclusively in the constructor to set up the features of the main window.

createMenuActions() This function initializes all the main menu (file& misc menus) actions and connects most of them to the appropriate slot.

createPlotActions() This function initializes the killActiveWindow and maximizeActiveWindowAction actions and their shortcuts.

createMenus() This creates the main menus and then adds all the associated actions.

createContextActions() This creates all the context menu options (mostly applicator functions) and connects them to the appropriate slot.

createContextMenus() Creates the context menus and adds associated actions. Context menu creation during runtime handled by contextMenuEvent function.

createStatusBar() Sets up widgets associated with statusbar (statusLabel,formulaLabel) and determines alignment.

8.3 Menus and actions

The menus and actions in the main window form 2 groups: the main menus and the context menus. The main menus are those available from the menu bar (file & misc menus) and the context menus are available on a right-click somewhere on the main window. The basic functionality of the menus is provided by QActions. These actions are objects that can be inserted into menus (becoming menu entries) as well as being individually connected to a slot. See examples in createMenuActions() and createContextActions() for creation of actions and createMenus() for examples of action addition to menus.

8.4 Plot Window Variables

The plot widgets for the main window are tracked using several variables.

PlotWidget *arrayPlotWidgets[25] Array of plots; stores pointers to all plotWindows.

int activeWidgetIndex Tracks active widget index with respect to cursor.

int numPlotWidgets Tracks number of plot widgets.

int gridSize Gridsize for windows (i.e gridsize of 2 holds 4 windows)

bool isPlotMaximized Property that tracks maximization of active plot.

8.5 Plot Window Management

The plot widget management is accomplised using the following functions. These functions update and utilize the various plot window variables. Generally, each function is designed to take advantage of the signal-slot framework and has input arguments tailored to match relevant signals.

void addPlotWidget() Adds new plot. Takes care of plot reorganization. Updates all plot variables. Should probably not be modified.

void removePlotWidget() Removes active plot and reorganizes if necessary. Updates all plot variables.

void organizePlotWidgets() Reshuffles plots into best fit grid.

void maximizeActivePlot() Hides all other plots. Updates isPlotMaximized.

void changeActivePlot(int i) Changes active plot to plot i. Updates activeWidgetIndex.

void killAllPlotWindows() Kills all plot widgets. Updates plot variables.

8.6 Selection Mode Management

The selection mode is implemented in xvsMainWindow because it involves the selection of multiple plots. General use of selection mode is as follows (NOTE: user should have own names for myFunction, isDataOkToUse and executeActionHere):

```
void xvsMainWindow::myFunction()
{
    pt2Member = &xvsMainWindow::myFunction;
    if (selectedWidgets.count() == numNeeded) {
        if (isDataOkToUse() == false) return;
        else executeActionHere();
        deactivateSelectionMode();
    }//end if
    else activateSelectionMode(numNeeded);
}
```

After selection mode is activated, the getSelectedWidgets thread will run until the specified number of widgets have been selected. The activation also connects the finished() signal of the getSelectedWidgets thread to the pt2Member pointer of the main window. This is why the pt2Member function needs to be set if selection mode is to be used. The general scheme is to call the slot, have it skip to activateSelectionMode, and then have the same slot called again when the selection mode returns with the desired number of widgets.

There is also an additional option to select an arbitrary number of widgets: activateSelectionMode(-1), where a center-mouse click will end the selection mode.

bool selectionMode Tracks activity of selection mode. true=on.

QList<PlotWidget * > selectedWidgets Keeps track of widgets selected in selection mode

void activateSelectionMode(int num) Activates selection mode by using a getSelected-Widgets thread (see types.h). Returns when num plots have been selected.

void deactivateSelectionMode() Deactivates selection mode and clears the selectedWidgets array.

void (xvsMainWindow::*pt2Member)() Provides pointer to member function for use when getSelectedWidgets returns.

void selectFunction() Calls the function specified by pt2Member. Connects to finished() signal of getSelectedWidgets thread.

void addSelectedWidget(int i) Adds widget to selectedWidgets.

void removeSelectedWidget(int i) Removes widget from selectedWidgets.

8.7 Control Dialog

The control dialog is initialized as a member of the main window. This is done for convenience: gets a pointer to the dialog as well as insuring that it closes along with xvs. The following functions are the primary interactions of the main window with the control dialog.

void setupControlDialog() Initializes control dialog connections to apply-all variable.

void setupPlotWindow(PlotWidget *w) Creates plot connections to control window upon creation of plot widget. This is a rather long function since it connects each plot widget to every operation of the control dialog. These connections will only work when the plot is active.

8.8 Event Reimplementations

This section covers all the protected event reimplementations.

void closeEvent(QCloseEvent *event) Reimplemented to give "are you sure you want to exit" dialog.

void contextMenuEvent(QContextMenuEvent *event) Creates the custom context menu. Uses menus created in createContextMenus().

void keyPressEvent(QKeyEvent *event) Implements keyboard shortcuts.

void keyReleaseEvent(QKeyEvent *event) Deals with control key toggling.

void mousePressEvent(QMouseEvent *event) Kills selection mode if the middle mouse button is clicked.

8.9 Signals and Update Functions

These signals and update functions serve to provide realtime feedback for certain aspects of xvs, such as the cursor position.

void closeWindow() Quit/exit xvs signal

void allButtonToggled(bool) Toggles apply to all property

void widgetChanged(QString) Signals a change in active window, used to update controlDialog

void selectionModeActivated(int) Changes all plot modes to selection if int=3 and to zoom if int=-1. uses this special feature of changeDisplayMode(int).

void selectedWidgetsChanged(int) Tracks number of selected widgets. Sends data to getSelectedWidgets thread.

void endSelectionMode(int) Activated by middle mouse button click. Ends arbitrary numWidgets selection mode. Changes number of selected widgets in getSelectedWidgets thread to -1. This will match the original initialization parameter (-1: for arbitrary number of selections) and kill selection mode.

void updateControlDialogValues(QString s) Resets values when active plot changes; string arg options: updateAll, view, name, index, mode; connected to needControlDialogUpdate signal in plotWidget class.

void cursorPosUpdate(QPointF pos, int i) Updates cursor position label

void setAllCheckBoxState(bool check) Keeps the "apply to all" property modifier buttons in sync

8.10 File i/o

Most of the file i/o operations are implemented here in the main window. However, the mpeg and jpeg export routines are in the plotWidget class due to ease of implementation (private variables accesible).

void openTextFile() Opens a text file in new window

FofX readSdfFile(const char *filename, int tLevel) Uses exterior lib here (bbhutil)

void openSdfFile() Opens sdf file in new window. Uses readSdfFile to retrieve data.

void exportToTextFile() Exports active plot (or all plots if apply-all property is active) to a single text file.

void exportToSdfFile() Exports active plot to sdf. Uses exterior lib here (bbhutil).

8.11 Assorted Applicator Slots

Most of these applicator slots have a set format:

1. Get relevent input: could be an input dialog and/or selection mode.
2. Validate Input: check lengths of relevant data to determine if they are compatible.
3. Perform operation: usually create a new window unless ctrl key is depressed (transforms in-place).

A number of the applicator slots simply call functions that are implemented in the xytVector class. Others that involve more than one plot will utilize selection mode. For the interested user, see source code for further examples.

9 PlotWidget

9.1 State variables

There are 10 primary state variables for each plotWidget object. They are as follows:

int timeIndex Current time index.

int timeDirection Determines iterative direction for animation (0:forward 1:backward).

int breakStatus (0:off 1:on) when activated this property will restrict the animation mode to 1 cycle.

QString name title or name of window.

int index grid index for main window (where it is located wrt other plots).

int displayMode Display mode (0:zoom 1:step 2:animate 3:selection).

bool isActive is considered active by main window.

bool isSelected is selected by mouse in selection mode.

bool isApplyAllActive apply to all property. Synced with main window status.

bool fullScale When true, plots autoscale.

9.2 Data structure

The data structure of the plotWidget class has two data repositories. The primary data structure is the xytVector specified by the *data* pointer. This contains all the information needed for the plot to display a single function. However, since it is beneficial for multiple plots to be shown in the same window, there is another member variable called $curveMap$ which contains a list of xytVectors. For plots with more than one dataset, both of these variables are occupied, namely the first xytVector in $curveMap$ will be copied to *data*. Some of the main window functions (control dialog dataview, applicator functions, etc.) will only notice the *data* xytVector and act accordingly. The plotting routines will always show all the contained functions.

xytVector data data storage object- holds 'n' arrays of xy points (see types.h)

QList<xytVector> curveMap Array of xytVectors. Only used if > function in plot.

void setDataVec(xytVector vec) sets data to vec.

void addCurveMap(xytVector vec) adds xytVector to curveMap. Used for > 1 function.

9.3 Time Management

The time index mangament was significantly more complicated than expected. Most of these functions handle looping past a single period and can therefore be used at any index. The first timePlusPlus and timeMinusMinus form the core of the time management and provide the basis for the animate routines.

void setTimeIndex(int t) sets time index to 't' provided it is in the correct range

void changeTimeDirection(int d) Sets time direction to d (0:forward, 1:backward).

void changeBreakStatus(int b) Sets breakStatus to b (0:off 1:on).

void timePlusPlus() Increases time index by one. Handles looping over multiple periods.

void timeMinusMinus() Decreases time index by one. Handles looping over multiple periods.

void animate() Calls time++, time– when timer goes off (every 60 millisec) depending on timeDirection. Connected to QTimer in constructor. Animation controled by start(stop)Animate().

void startAnimate() Starts animation (starts timer).

void stopAnimate() Stops animation (stops timer).

9.4 View Management

The view management is primarily dependent on the plotSettings class declared in types.h, types.cpp. The current view is determined by the plotSettings object pointed to by zoomStack[curZoom]. zoomStack is a list of plotSettings objects that is appended every time the view changes; this allows the view history to be browsed through using the forward() and back() slots. Other related functions are either accessors or autoscale helpers.

QVector<PlotSettings> zoomStack Keeps record of zoom information to allow back &forward actions.

int curZoom Current zoomstack index.

plotLimits getBoundingLims(QList<double> x, QList<double> y) Returns the minimum box showing all the data.

void forward() moves one index forward in zoomstack to those PlotSettings.

void back() moves one index back in zoomstack to those PlotSettings.

void zoomIn() zooms to half span in xy.

void zoomOut() zooms out to 2x span in xy.

QMap<int,QVector<double> > getZoomStack() Returns zoomStack.

plotLimits getPlotLimits() Returns plot limits. Used by clone and updateControlDialog-Values in mainWindow.

void setPlotSettings(const PlotSettings &settings) imports plot settings in contructor.

void initializePlotBoundaries() fits plot boundaries to initial data (max&mins).

void setPlotBoundaries(plotLimits p, PlotSettings *settings = 0) changes view without adding to zoomstack.

void optimizePlotBoundaries() Fits plot boundaries to current data (max&mins). Appends zoomStack

void resetPlotLimits(plotLimits p) Changes current plotSettings to have boundaries from plotLimits *p*.

9.5 Display Properties

There are a wide range of display properties. Most of these are color, size, or shape properties associated with lines and markers. The colors are declared in the various drawing functions and the color properties are stores as integer indexes of this list. There are also a large number of accessor and modifier functions associated with these variables which will not be described here.

int lineColor 1:red, 2:green, 3:blue, 4:cyan, 5:magenta, 6:yellow

int markerColor same refs as lineColor

bool linesShown hide/show curve lines

bool markersShown hide/show points

int markerType 1:circle 2:square, 3:triangle

int markerSize 1-6 pixels (radius)

int lineWidth 1-4 pixels

QString titleLabel plot title label.

QString timeLabel displays actual time

QString indexLabel displays time index

QString lowerLimitLabels shows lower x,y values

QString upperLimitLabels shows upper x,y values

QString displayModeLabel shows display mode (zoom, auto, step etc)

bool labelsVisible[7] Determines visibility.

int labelColors[7] Determines label colors.

int labelFormats[7] Determines label format (# decimals).

int Margin Determines size of margin in pixels. Standard size =45. Removed size =1.

9.6 Drawing Routines

The drawing routines consist of three primary functions. The primary procedure consists of a painter object drawing shapes onto a pixmap which is then copied to the widget display (double buffering). A margin will be introduced if either gridlines or the axes are displayed, otherwise there will be no margin.

void drawGrid(QPainter *painter) Draws grid and ticks & tick mark labels

void drawCurves(QPainter *painter) Draws curves

void drawLabels(QPainter*painter) Draws labels

void refreshPixmap() calls all drawing functions

9.7 Export Routines

The export routines in the plotWidget class are makeMpeg() and exportToImage(). It's called exportToImage because it can export to png and jpeg formats. The routines will take the plot window settings as is. The exported frames will not be displayed on the screen during the export.

void makeMpeg() exports plot data to temp jpeg file using Qt export functions and then uses mpeg2enc to create the mpeg file.

void exportToImage() exports plot data to jpeg files using Qt export functions.

9.8 Keyboard and Mouse

The keyboard and mouse implementations are generally designed to switch with the display modes.

void mousePressEvent(QMouseEvent *event) Implements different response actions for different modes (zoom mode: rubber band zoom, step mode: time++, automode: stop/start animation).

void mouseMoveEvent(QMouseEvent *event) reimplemented to add updates of mouse position to info bar in main window.

void mouseReleaseEvent(QMouseEvent *event) reimplemented to add rubberband zoom functionality

void keyPressEvent(QKeyEvent *event) implements all keyboard shortcuts

void wheelEvent(QWheelEvent *event) allows for scrolling with mouse wheel

10 ControlWindow

The ControlWindow class is predominantly made in the *.ui files. It is highly recommended when altering the appearance of the control dialog to use QDesigner. The remainder of the class is toothless and consists of several functions involving status updates and a variety of signals that connect to the active plot.

10.1 Label Controls

The label control members deal with internal consistency of button values and insuring they match the plot display.

QList<QRadioButton *> labelList list of pointers to radiobuttons that select labels.

int findSelectedLabel() uses labelList to determine which button is currently selected.

void labelOn() This function shows the currently selected label (labelToggled signal).

void labelOff() This function hides the currently selected label (labelToggled signal).

void allLabelsOn() Sets all labels visible.

void allLabelsOff() Sets all labels hidden.

void colorChanged(int color) Emits labelColorChanged signal to change selected label's color.

void formatChanged(int format) Emits labelFormatChanged signal to change selected label's decimal places.

void labelToggled(int label, bool toggle) [**signal**] Signal contains label index and activate status.

void labelColorChanged(int label, int color) [**signal**] Contains label index and new color.

void labelFormatChanged(int label, int format) [**signal**] Contains label index and new number of deciamls.

10.2 View Stack

The view stack controls are fairly straightforward. The pushOntoStack slot takes the values in the adjacent input lines and sends them off in the pushOntoStackSignal which gets received by the active plot. The view stack window update is determined in the xvsMainWindow control dialog section.

void pushOntoStack() Collects input data and sends off pushOntoStackSignal.

void pushOntoStackSignal(plotLimits p) [signal] Received by active plot & connected to resetPlotLimits(PlotLimits p).

10.3 Data View Window

Unlike the other aspects of the control dialog, the data view window holds plot data internally. This data is then updated as needed (determined by the updateControlDialogValues(QString arg) slot in xvsMainWindow.

xytVector myData Holds a dataset.

int myTime current timestep.

int myDecimals current format.

void updateData(xytVector data) sets myData to data.

void updateDataIndex(int timeIndex) updates myTime.

void showCurrentData() displays current dataset on QTextEdit object.

void changeDecimals(int d) changes myDecimals.

11 Types

11.1 xytVector

The xytVector class was designed to hold values for a 1D time dependent function. It also contains a set of applicator functions to alter the internal data.

QString name name of dataset.

QList< QList<double> > Y list of list of y-values (Y[time][xpos]).

QList<double> X list of x-values. Constrains data to set values of x.

QList<double> T time at each step.

void x_log_c_plus_x(double c) x→log(x+c)

void x_times_y() y→y*x

void y_divide_x() y→y/x

void x_p(double p) x→x^p

void x_p_times_y(double p) x→y*x^p

void y_p(double p) y→y^p

void dy_dx() y→ $\frac{dy}{dx}$

11.2 getPlotWidgets thread

The getPlotWidgets thread is used to determine when to return the currently selected widgets. When created, it runs a while loop until numSelected=numNeeded. The changeSelected slot is attached to signals from the various plots and keeps the thread numSelected member updated.

int numSelected current number of selected widgets

int numNeeded number of widgets needed→ when numSelected=numNeeded, thread returns.

getSelectedWidgets(int numNeeded) constructor with initialized numNeeded

virtual void run() provides run routine that has event loop.

void changeSelected(int n) updates number of selected widgets

11.3 plotSettings

The plotSettings class is designed to take care of the plot view and axes. It has members determining the plot boundaries, axes label positions, and rounding functions.

double minX minimum x value shown

double maxX max x value shown

int numXTicks number of tick marks on x-axis

double minY minimum y value shown

double maxY max y value shown

int numYTicks number of tick marks on y-axis

void scroll(int dx,int dy) moves view by specified amount

static void adjustAxis(double &min, double &max, int &numTicks) adjusts single axis tickmarks to more friendly numbers

void adjust() adjusts both axes to rounder, more human friendly numbers (calls adjustAxis twice).

void info() gives info to std output on xy boundaries

double spanX() x span in x units (not in pixel width)

double spanY() y span in y units (not in pixel width)

11.4 FofX

FofX is a simple structure with two lists of doubles and a time variable. Used primarily in sdf reading functions.

```
struct FofX
{
  QList<double> X;
  QList<double> Y;
  double time;
};
```

11.5 plotLimits

Simple data structure with min and max doubles for x and y.

```
struct plotLimits
{
  double xMin;
  double xMax;
  double yMin;
  double yMax;
};
```

Part IV
Advice for Expanding xvs

Since it is highly probable that users will want to extend xvs, this section was written to make any extension as painless as possible. The sections are written in an arbitrary order and it is not neccesary (but still recommended) to read all of them.

12 General Advice

12.1 Stuff You Should Probably Not Touch

If greater functionality is needed there are an array of useful functions already in place. Please don't screw these things up:

Plot Management Please use the changeActivePlot, addPlotWidget, clone, and removePlotWidget functions in future code. They already deal with the best fit grid (main window plot organization) and set up all the necessary connections with the control dialog. A good example would be a new function:

```
xvsMainWindow::removePlotWidgetAt(int i)
{
    changeActivePlot(i);
    removePlotWidget();
}
```

Rubber Band Zoom Anything with the words rubberband will have something to do with this. It provides the mouse selection zooming. Very aggravating to code and extends to a large range of functions.

Slot/Signal pairs These can probably be better organized so beware: make sure that you have taken all the affected connections into account before editing them.

12.2 Plot Widgets

The plot widget is more or less self-sufficient. Not much could be added to this widget without the need for major overhauls. Most new functions will only be interested in the xytVector data and no modifications to the plot widget should be needed. If the user wants to add more graphics to the plot window, it would be easiest to write another void function drawSomething(QPainter *painter) and then put it in the refreshPixmap() function. The drawing functions themselves are not terribly complicated (most of the unclear parts involve converting to pixel coordinates) and could easily be edited to improve functionality (e.g put in an option to not use a border).

12.3 Code Refactoring/Possible Overhauls

applicator class: there are remnants of the attempt to include the applicator functions from the old xforms xvs in the 'src/include' folder. To correctly include these functions, all of the user input mechanisms (GET_DOUBLE, GET_STRING, etc.) and all references to plot windows must be removed from apply.[ch]. A number of other functions used by the applicators might not be defined (dmin, dmax, xmin_wt etc.) and would have to either be found and included or

re-implemented. These changes would ensure that the applicators' functionality is encapsulated and independent of the gui, at the cost of more input parameters. The next step would be to divide the names of the applicators into lists based on their input params, and then feed those lists into a code generating script. This script would follow the 'adding applicator functions' set of directions and generate code for:
1) the header declarations of slots and QActions.
2) the slot implementation: getting the necessary user input, calling the included applicator, applying results of applicator
3) the slot to QAction connections.
4) add the QAction to the appropriate Qmenu

xvs server: a qt server class exists. To use the old sdftoxvs utilities, the server side must know how to translate the data being sent.

Types: the use of types might make the code more readable instead of using integers defined by outdated comments

13 Directions

13.1 Adding Applicator Functions

Currently, creating a new applicator function is not easy. Several steps are required:

1. Create a relevant slot in xvsMainWindow.
2. This slot must implement the applicator function and apply it to the correct plot(s) (See selection mode mangament 8.6 for instructions on selecting multiple plots).
3. It is generally more convenient to locate the data manipulation somewhere else (global functions or in xytVector).
4. Next, a QAction should be declared in xvsMainWindow and connected to the slot. The QAction can then be configured (hotkey, menu, etc.)
5. Then the action should be added to either the context menu or main menu bar. This will allow the applicator to be called via the gui.
6. To add an action, put the constructor and connect statement in the relevant 'create*Actions()' function, and add it to the appropriate menu via the associated 'create*Menus()' function.
7. Update this documentation.

For further instruction see examples in source code.

13.2 Messing with the Control Dialog

The control dialog is the most connection heavy object in xvs. The current design connects the control dialog options to every active plot. This means that all the plotWidget slots that connect to the control dialog must have an if (isActive || isApplyAllActive) statement surrounding the implementation otherwise all options would apply to every plot.
To add an option to the control dialog that affects the plots:

1. First, the relevant functions in the PlotWidget class must be declared in the public slots section.

2. Then, the button/inputLine/etc. should be added to the control dialog via QDesigner (or by hand but that's onerous). If added in QDesigner, make sure to rename the widget so as to access it from the source code.

3. Next, declare the connection between the control dialog option and the plotWidget slot in xvsMainWindow::setupPlotWindow(). This will connect each plotWidget to the control dialog as it is created.

4. Finally, the update mechanism must be appended to ensure that the control dialog is not showing misleading information. User will have to alter xvsMainWindow::updateControlDialogValues() and place the update implementation in one of the categories associated with updates (view, index, name, mode, all).

5. If none of the categories is appropriate, then the user must implement a new category with its own text argument and place 'emit needControlDialogUpdate(QString newArg)' in the relevant functions.

6. Update the documentation.