

AcroTeX.Net

AcroTeX

The AcroTeX and FLEX/Flash Connection

Applications to Graphing

D. P. Story

## Table of Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Background . . . . .	3
1.2	What is AcroTeX? . . . . .	3
<b>2</b>	<b>Requirements</b>	<b>4</b>
2.1	TeX Package Requirements . . . . .	4
2.2	PDF Creator Requirements . . . . .	5
2.3	Installation . . . . .	5
<b>3</b>	<b>The AcroTeXGraphing System</b>	<b>6</b>
3.1	Setting up the Graphing Screen . . . . .	7
3.2	Graphing Screen Controls . . . . .	9
	• Required Controls . . . . .	9
	• Optional Controls . . . . .	10
3.3	Populate and Silent Linking . . . . .	12
3.4	Graphing with <code>\sgraphLink</code> . . . . .	12
3.5	Graphing with <code>\defineGraphJS</code> . . . . .	15
<b>4</b>	<b>Customizations</b>	<b>15</b>

## 1. Introduction

The **AcroF<sub>e</sub>X** Graphing Bundle is used to create a *graphing screen* that can be incorporated into a PDF document and viewed within Adobe Reader, version 9.0 or later. The graphing screen can be interactive or non-interactive.

For the interactive graphing screen, the user can enter an expression representing a function of a single variable  $x$ , a polar function of  $t$ , or a set of parametric equations that are functions of  $t$ . Various controls are provided to change the viewing window, for shifting horizontally and vertically, and for zooming in or out.

For the non-interactive graphing screen, the screen is populated when the user clicks a link created by `\sgraphLink`. `\sgraphLink` passes such information as the function, domain, and range to the graphing routines of AcroF<sub>e</sub>X.

In this version of AcroF<sub>e</sub>X, up to four functions can be graphed and four sets of plotted points can be displayed, on one graphing screen.

The graph screen itself is actually a SWF file, named `acroflex.swf`. This SWF file is part of the AcroF<sub>e</sub>X distribution. This package uses the `rannot` package, also written by this author, to create rich media annotations, to embed `acroflex.swf` in the PDF document, and to display the SWF through the rich media annotation.

### 1.1. Background

Version 9 of Acrobat/Adobe Reader introduces the *rich media annotation* which plays FLV movies, and SWF animations, and MP3 files.<sup>1</sup> Acrobat/Adobe Reader also provides a scripting bridge between JavaScript for Acrobat, and ActionScript, the scripting language of Flash player. This bridge enables the PDF and the Flash widget, embedded in the rich media annotation, to communicate. The scripting bridge opens up wonderful opportunities for applications to the education sector. The AcroF<sub>e</sub>X Graphing Bundle is one such application of the new PDF-Flash connection to education.

**AcroF<sub>e</sub>X** uses the commercial product Adobe FLEX Builder 3 and FLEX 3 SDK to produce Flash widgets, and the AeB to create PDF documents with appropriate JavaScript to communicate with the Flash widget. FLEX Builder 3 is currently free for students and educators, the FLEX 3 SDK is free to all.

### 1.2. What is AcroF<sub>e</sub>X?

The word **AcroF<sub>e</sub>X** is meant to convey a merging of two computer technologies:

- **Acro**: connotes both **Adobe Acrobat (Adobe Reader)** and **AcroT<sub>e</sub>X** (as in the **AcroT<sub>e</sub>X eDucation Bundle** or, just **AeB**).

<sup>1</sup>The rich media annotation is introduced in *Adobe Supplement to the ISO 32000*, which documents BaseLevel 1.7, ExtensionLevel 3, the Adobe's extensions to PDF 1.7.

- **F<sub>e</sub>X**: connotes **Adobe FLEX 3**. FLEX 3 is used to create SWF files to interact with the user. In the case of graphing, plotting information is passed from Acrobat, via JavaScript, to the Flash widget. ActionScript takes the data, and plots the points provided, and connects them with a smooth curve.

## 2. Requirements

In this section we list the requirements for this package.

### 2.1. $\LaTeX$ Package Requirements

The preamble of the demo file `afgraph.tex` lists:

```
\usepackage[%
  driver=dvipsones,
  web={nodirectory,pro,tight,usesf},
  eforms,exerquiz,dljslib={ImplMulti},
  graphicxsp={showembeds}
]{aeb_pro}
\usepackage{acroflex}
```

Let me comment on each of these lines.

- `\usepackage{acroflex}`: Of course, we use the `acroflex` package. The `acroflex` package **requires** the `rmannot` package. The latter package is the one that creates the rich media annotation, embeds the graphing Flash widget, `acroflex.swf`, and displays it. The `rmannot` package is the only one listed in `acroflex.dtx` as required, however, more packages are really required, as discussed in the next items.

`graphicxsp`: A **required** package. The `rmannot` package requires `graphicxsp`. The `graphicxsp` package, part of the AeB Pro Bundle, provides embedding of poster graphics for a rich media annotation. A poster graphic is the appearance you see when the annotation is not activated.

- `\usepackage{aeb_pro}`: A **required** package. The AcroF<sub>e</sub>X Graphing Bundle uses the `willClose` environment to assure the document will behave properly when the user closes the document.

Its options, the so-called **AeB Control Central**, represent a convenient way to input the other required packages (in optimal order) needed by the `acroflex` package.

If `aeb_pro` is not used, then the individual required packages must be input using the `\usepackage` mechanism.

We input the `aeb_pro` package before the `acroflex` package.

We now comment on each of the options used in the `aeb_pro` package:

- `driver`: This system uses Acrobat Distiller, which distills a PostScript file. The driver values this package uses are `dvips` and `dvipson`. Setting the driver is important because the `dvi-to-ps` application (`dvips` and `dvipson`) consumes the `dvi` file produced by the  $\TeX$  compiler and writes a PostScript file that Acrobat Distiller consumes.
- `web`: The `web` package is not really required. It is used to create a PDF page size convenient to view on a computer monitor. This package has many options and features for the document author to design a document for screen—or for paper—viewing.  
The `web` package brings in the `hyperref` package, which is a **required package**. If `web` is not used, `hyperref` needs to be input.
- `eforms`: A **required** package. This package provides form field and link support for the Acro $\TeX$  Graphing Bundle. The `eforms`, in turn, inputs the `insdljs` package, which provides support for document-level JavaScript. It is the document-level JavaScript where much of the work is done: parsing input, calculating graphing data, and sending this data off to the `acroflex.swf` widget for display.
- `exerquiz`: A **required** package. The `exerquiz` package has several function parsing methods defined in its document-level JavaScript. Acro $\TeX$  uses these parsing routines. One of these days, I'll separate out the parsing routines from `exerquiz`, but not now.
- `dljslib`: An optional, but recommended package. We use this package for its `ImplMulti` option. This option simplifies the problem of entering functional expressions. Without the `ImplMulti`, to enter  $2x \sin^2(2x)$ , the user would have to type explicit multiplications, `2*x*(sin(2*x))^2`, with the `ImplMulti` option, the user needs only enter `2xsin^2(2x)`.
- `graphicxsp`: The `graphicxsp` is a **required** package of `rmannot`, but we input it earlier so we can set its options through the **AeB Control Central** (part of AeB Pro).

## 2.2. PDF Creator Requirements

This package requires Acrobat Distiller 9.0 (or later) to convert PostScript files to PDF. Because this package uses `rmannot` to create rich media annotations, there is also a requirement that the Distiller must be opened using the `-F` command line flag. See the documentation of the `rmannot` package for more details.

## 2.3. Installation

The installation of the `acroflex` package is straightforward. Place `acroflex.zip` in the search path of your  $\TeX$  system and unzip. Unzipping creates a folder named `acroflex`. Refresh your filename database, if your system requires it.

Accompanying the distribution is a file named `acroflex.cfg`. Open this file in your favorite text editor and you see the following lines.

```
%
% AcroFlex Graphing Bundle Configuration File
% D. P. Story, dpstory@acrotex.net
%
\pathToAcroFlex{C:/acrotex/aepro/acroflex/swf}
```

Edit the argument of `\pathToAcroFlex` (defined in the `acroflex` package) to the path of the folder that contains the `acroflex.swf` Flash file. Save and close `acroflex.cfg`.

Of course, you need to install the latest versions of AeB (the AcroTeX eEducation Bundle), AeB Pro, `graphicxsp`, and `rmannot`. Follow the package documentation closely for installation, some of the packages require that certain JavaScript file be installed.

### 3. The AcroTeX Graphing System

This package defines several document-level scripts, the two primary ones are `Graph_xy()` and `Graph_xyt()`, the others support these two. `Graph_xy()` and `Graph_xyt()` take the data passed to it, parse it, create plot data, and send it off to the AcroTeX graphing widget to graph the data by way of the infamous scripting bridge. Details of these functions can be found in the documentation in the `acroflex.dtx` file.

In the AcroTeX graphing system there are three modes of operation: interactive, populate, and silent.

- **Interactive:** This occurs when the user enters a function through the UI.

The following controls are *required*:

```
\funcInputField, \graphBtn, \numPoints
\domMin, \domMax, \rngMin, \rngMax,
```

If parametric or polar graphs are to be used, then `\domMinP` and `\domMaxP` are also required. The other controls are *optional*:

```
\graphClrBtn
\amtShift (\hShiftL, \hShiftR, \vShiftD, \vShiftU)
\zoomInOut, \savedelSelBtn, \functionSelect
```

The `\graphClrBtn` button is recommended, though not required. All these commands will be discussed in detail in the pages that follow.

- **Populate:** This mode occurs when the graphing parameters are passed to `Graph_xy` (or `Graph_xyt`) by `\sgraphLink` (or some other command). All the essential information is passed as arguments. The target graphing screen has all the required controls, as listed above. The command initiating the graphing must set the `graph_props.populate` property to `true`. In this case the graphing data populate the required fields and the graph will be drawn. It is the document author's responsibility to only use populate on graphing screens that have all the required control fields.

Populate behaves exactly like interactive, but the graphing data is passed to the graphing routines in pre-packaged form, prepared by the document author; the user, however, can manipulate the curve once it appears.

The required controls are the same as the interactive mode.

- **Silent:** In the non-interactive mode, there must be no controls other than `\graphClrBtn`. Basically, the document author prepares some pre-packaged graphs to be displayed to the user, without interaction. These may go along with a tutorial discussion symmetry, periodicity, tangent lines, etc.

If the document author wants the user to interact with the graph, the required controls need to be supplied and the `graph_props.populate` property needs to be set to `true`. That is, use the populate mode.

### 3.1. Setting up the Graphing Screen

It should be a hard and fast rule that all content concerning a graphing screen should occur on the same page as the rich media annotation that displays the graphing screen. Should discussion cross page boundaries, create another graphing screen for that page. Never fear, the AcroTeX graphing widget is only embedded once, so adding more graphing screens does not bloat the size of the file.

There are three commands to set up an AcroTeX graph screen `\dimScreenGraph`, `\graphName` and `\graphScreen`. The use of the command `\dimScreenGraph` is not required, but recommended.

```
\dimScreenGraph{<width>}{<height>}
\graphName{<unique_name>}
\graphScreen[<rmannot_options>]{<width>}{<height>}
```

**Command Description:** We describe each of these three, and their parameters.

- `\dimScreenGraph`: This command is a convenient way of setting the dimensions of the graphing screen. You specify the width of the screen using the `<width>` parameter and the height of the screen using the `<height>` parameter. These values are passed through a `\setlength`, so simple calculations on the dimension can be performed on the parameters. (The `calc` package is used by the web package.) This command then defines macros `\hScreenGraph` and `\vScreenGraph` to hold these two dimensions, respectively. `\hScreenGraph` and `\vScreenGraph` can be used in `\graphScreen`, or in setting up `minipages` based on these lengths, for example.

If the aspect ratio of all your graphing screens is going to be the same, then it suffices to use `\dimScreenGraph` only once in the document.

- `\graphName`: Use this command to define a unique name for this graphing screen. Each screen must have a different name. This command defines the text macro `\afgraphName`, which expands to the given name.

- `\graphScreen`: This is the main command of this package, it's the one that creates a rich media annotation and associates it with the AcroTeX Graphing widget. It has three parameters:
  1. [`<rmannot_options>`] is optional and just passes to the underlying command `\rmAnnot` (defined in the `rmannot` package) that actually creates the rich media annotation. The most "important" key-value pair, for this package, is the `poster` key, through this key, a poster can be associated with the annotation.
  2. `<width>` is the width of the graph screen, if `\dimScreenGraph` was used, just use `\hScreenGraph` as this value.
  3. `<height>` is height the screen of the graph screen, if `\dimScreenGraph` was used, just use `\vScreenGraph` as this value.

The `\graphScreen` can be resized using `\resizebox` or `\scalebox` (from the `graphicx` package) to obtain a larger or smaller graph screen with the same aspect ratio.

The following is an example of the usage of each of these three commands. Note that the height is three-fourths that of the width.

```
\dimScreenGraph{186bp}{186bp*3/4}
\graphName{graph1}
\graphScreen[poster=aflogo]{\hScreenGraph}{\vScreenGraph}
```

**Graphing Screen in a Floating Window.** The graphing window can appear in a floating window as well. The `\iconFloatGraphScreen` command is used to create such a screen.

```
\iconFloatGraphScreen[<key_values>]{<width>}{<height>}
```

**Parameter Description:** The command has three parameters. The first optional one is passed as the first optional parameter of the underlying `\graphScreen` command. The `\graphScreen` command uses the two parameters `\hScreenGraph` and `\vScreenGraph`, defined through the `\dimScreenGraph` command, to set the dimensions of the graph screen. The graph screen is then resized using `\resizebox` from the `graphicx` package. The other two parameters, `<width>` and `<height>`, are simply passed to `\resizebox`. See the documentation on `\resizebox` for details on these parameters.

For example,

```
\iconFloatGraphScreen[poster=aflogo]{40bp}{!}
```

The first parameter is used to define a poster of the icon, the second parameter is 40bp which means to resize the graphic to a width of 40bp, the third parameter of exclamation point (!) signals `\resizebox` to maintain the aspect ratio of the graphic.

The `\iconFloatGraphScreen` command is implemented by creating a rich media annotation for the AcroTeX Graphing widget, with a form field button on top of it that is transparent. Pressing on the icon is actually pressing on the button. The button action activates the graphing screen



if it is not activated, and deactivates it if it is activated. The graphing screen might be the target of graphing data sent to it by the `\sgraphLink` command, see [Section 3.4](#), page 12, or through the graphing screen controls, these are explained next.

### 3.2. Graphing Screen Controls

The controls described in this section are used for interactive and populate modes.

#### • Required Controls

For interactive or populate mode, in addition to `\graphScreen`, several controls are required so the user can manipulate the graph.

```
\funcInputField[<key_values>]{<width>}{<height>}
```

**Command Description:** The field created by `\funcInputField`<sup>2</sup> is used to enter a function or a set of points to be graphed. The function is parsed by the `exerquiz` routines, so the same syntax that is used for `exerquiz` quizzes and short quizzes is used. The `<key_values>` are passed to the underlining text field and can be used to change the appearance of the field, see the `eformman.pdf` for more information. The `<width>` and `<height>` are the width and height, respectively, of the text field.

```
\graphBtn[<key_values>]{<width>}{<height>}
```

**Command Description:** The graph button. Once the user has entered a required data into the required fields, the user press this button and the graph appears in the graph screen. The parameters are the same as for `\funcInputField`, the descriptions are the same.

```
\numPoints[<key_values>]{<width>}{<height>}
```

**Command Description:** This text field displays the number of points to be plotted. It is editable, the user can change this value. The parameters are the same as for `\funcInputField`, the descriptions are the same.

```
\domMin[<key_values>]{<width>}{<height>}
\domMax[<key_values>]{<width>}{<height>}
\rngMin[<key_values>]{<width>}{<height>}
\rngMax[<key_values>]{<width>}{<height>}
```

**Command Description:** The graphing window is set by these four text fields. When the curve is graphed, only the rectangular window  $[\backslash\text{domMin}, \backslash\text{domMax}] \times [\backslash\text{rngMin}, \backslash\text{rngMax}]$  is displayed.<sup>3</sup> The parameters are the same as for `\funcInputField`, the descriptions are the same.

If parametric and polar graphing is required of the user, then `\domMinP` and `\domMaxP` are required as well.

<sup>2</sup>The command was originally misnamed `\fileInputField` and is still recognized by the AcroTeX package; however, document authors should use the command `\funcInputField`.

<sup>3</sup>By this notation, I mean the intervals determined by the values of these intervals.

```
\domMinP [<key_values>] {<width>} {<height>}
\domMaxP [<key_values>] {<width>} {<height>}
```

**Command Description:** The interval  $[\text{\domMinP}, \text{\domMaxP}]$  is the interval over which a set of parametric equations is traced; in the case of polar functions, this interval is used for the domain of the polar function. The parameters are the same as for `\funcInputField`, the descriptions are the same.

**Setting the default values.** Whereas it is possible to set the default values of the fields just described, a more convenient method is used.

```
\defaultFunction{<function|points>}
\defaultNumPoints{<postive_integer>}
\defaultDomRng{<x_min>}{<x_max>}{<y_min>}{<y_max>}
\defaultDomP{<t_min>}{<t_max>}
```

These can be executed, along with `\graphName`, just before the `\graphScreen` command. The values of their parameters will then populate the corresponding fields as default values.

The following are the default values of all the required fields, as defined by the `acroflex` package. Note that all of these are parsed (with the exception of the number of points) using `exerquiz`'s parsing routines; consequently, a value such as  $2\pi$  is perfectly legal.

```
\defaultFunction{x^2}
\defaultNumPoints{40}
\defaultDomRng{-2}{2}{0}{4}
\defaultDomP{0}{2*PI}
```

#### • Optional Controls

There are several other optional controls that may be useful in manipulating a graph.

```
\graphClrBtn [<key_values>] {<width>} {<height>}
```

On clicking this button, the current graphing screen is cleared of all graphs and plotted points. Shift-clicking this button deactivates the graphing screen, and the annotation's poster appears.

**Multiple Plots.** By using the `\functionSelect` combo box, the user can graph multiple curves.

```
\savedelSelBtn [<key_values>] {<width>} {<height>}
\functionSelect [<key_values>] {<width>} {<height>}
```

The `\functionSelect` combo box serves several purposes. It consists of eight items that appear as *Curve 1*, *Curve 2*, *Curve 3*, *Curve 4*, *Plot 1*, *Plot 2*, *Plot 3*, *Plot 4*. When this combo box is present, the user is able to graph multiple curves and plots. Changing the

combo box to Curve 2, for example, and pressing the `\graphBtn` button, the function will be graphed on Curve 2. There are four curves possible, and four sets of plotted points. The different curves and plots are color coded.

When the `\savedSelBtn` is also present, the user can click on it and save the function definition under that curve or plot. These expressions will only be saved during the current viewing session in Adobe Reader, but if the user is on Acrobat, the PDF can be saved and the values added to the combo list will be saved as well.

The parameters are the same as for `\funcInputField`, the descriptions are the same.

**Horizontal and Vertical Shifting.** There are several controls that shift the graphing window vertically or horizontally.

```
\amtShift [<key_values>] {<width>} {<height>}
\hShiftL {<text>}
\hShiftR {<text>}
\vShiftU {<text>}
\vShiftD {<text>}
```

The `\amtShift` is a text field, its value is a positive number that will be used to shift the graphing window horizontally or vertically. The user can change this value. The parameters are the same as for `\funcInputField`, the descriptions are the same.

The other four commands are implemented as links, then clicked, the graphing window moves the amount specified in `\amtShift` field left (`\hShiftL`), right (`\hShiftR`), up (`\vShiftU`) or down (`\vShiftD`). The argument `<text>` is the text to be used to identify the link.

**Zoom, zoom, zoom.** The user can be allowed to optionally zoom the graph out or in by providing the control `\zoomInOut`.

```
\zoomInOut [<key_values>] {<width>} {<height>}
```

Click the `\zoomInOut` button zooms out by an amount shown in the `\amtShift` field; shift-clicking will zoom in by the amount shown in the `\amtShift` field. The parameters are the same as for `\funcInputField`, the descriptions are the same.

**Setting the default values.** As with the required controls, the optional ones can be given default values through convenience macros.

```
\defaultShiftAmt {<positive_number>}
\defaultShiftAmt {1}
```

The `\defaultShiftAmt` is used to set the default value of the `\amtShift` field; the default value is `\defaultShiftAmt{1}`.

The `\functionSelect` lists four curves and four plots. The text can be changed by through the following text macros. Each command is followed by its default definition.

```

\afCurve{<name_for_curve>}
\afCurve{Curve}
\afPoint{<name_for_point>}
\afPoint{Point}
\afUnused{<unused>}
\afUnused{--unused}

```

The definitions values of `\afCurve` and `\afPoint` are the target of several search using regular expressions. If the values of `\afCurve` and `\afPoint` are too complex, the regular expression search may fail. Try to keep these definitions to ASCII characters.

### 3.3. Populate and Silent Linking

The previous section details the interactive mode, where the `\graphScreen` is present with all its required controls, and possibly some optional controls. Curves are generated purely through the user interface, that is, the user enters data into the various form fields, clicks the `\graphBtn`, and *voilà*, the graph is drawn!

In this section, the populate mode is discussed as well as silent mode.

### 3.4. Graphing with `\sgraphLink`

The document author can prepare function/points to be graphed, along with all the essential data needed to view the graph. For populate, the graphing data populate the required text field, and is available for the user then to manipulate. The population of an interactive graphing screen is done though a special link, the `\sgraphLink`. (The “s” in `\sgraphLink` stands for “silent,” but that was before I made the design decision to have a populate mode.)

The syntax for `\sgraphLink` is

```

\sgraphLink[<appearance>]{<graph_key_vals>}{<func|points>}{<text>}

```

**Parameter Description:** The command takes four parameters, the first is the usual optional parameter that can be used to change the appearance of the link. The others we present in detail.

1. [`<appearance>`]: Key-value pairs that are used to change the appearance of the link.
2. `<graph_key_vals>`: Key-value pairs, some of which are used on the  $\TeX$  side, some on the PDF side, while others on SWF side.
  - `graph`: The value of this key determines which *chart series* (FLEX terminology) the data will appear on. The values of this key are
    - `c1`, `c2`, `c3`, and `c4`: Use one of these values to graph a function, a polar function, or a set of parametric functions. Up to four curves can be displayed on the graphing screen at once. These values are displayed using the `LinearSeries` (FLEX terminology).

- $a_1$ ,  $a_2$ ,  $a_3$ , and  $a_4$ : Same as above, but the region between the horizontal axis, and the graph is shaded in. These values are displayed using the `AreaSeries` (FLEX terminology).
- $p_1$ ,  $p_2$ ,  $p_3$ , and  $p_4$ : Use one of these values to plot points. These values are displayed using the `PlotSeries` (FLEX terminology).

Thus, `graph=c2` tells the graphing routines of AcroTeX and the AcroTeX Graphing widget to display this data on series  $c_2$ .

If two curves or plots have the same value for `graph`, then the the one graphed last will overwrite the earlier one. If you want both curves or plots to appear on the graph together, give them different `graph` values.

When this key is not given a value, the default is  $c_1$ .

- `type`: This key declares the type of curve, possible values are `cart`, `para`, and `polar`. This key is used mostly internally, and is normally not used. There is one situation that it is used. When defining a polar function, use `type=polar`. Thus, to define a polar function, type something like this:

```
\sgraphLink{type=polar,xInterval={[-1.5,1.5]},yInterval={[-1,2]},
tInterval={ [0,2*PI]},points=40,populate}
{1+sin(t)}{\$r = 1 + \sin(\theta)$ }
```

Note the explicit use of `type=polar`; the parsing can identify a function of  $x$  and a set of parametric equations that are function of  $t$ , but help is needed for polar.

- `populate`: Possible values are `true` or `false`, typing `populate` is the same as `populate=true`. This switch signals the graphing routines on the PDF side to populate the required fields with the graphing data. The default is `populate=false`, do not populate, use silent mode.

**Populate versus Silent Modes:** The `populate` key is how populate mode is distinguished from silent mode: `populate=true` is populate mode, `populate=false` (or the `populate` parameter not listed) is silent mode. In populate mode, the target graphing screen must have all required control fields; in silent mode, the only control should be the `\graphClrBtn` button.

- `connectwith`: The method used to connect consecutive points on the graph, possible values are `curve` and `segment`. This value is passed to the AcroTeX graphing widget. For function of  $x$ , the default is `curve`; otherwise, the default is `segment`. This value is ignored when the `graph` property signals plotting ( $p_1$ - $p_4$ ).
- `points`: The number of points to generate for plotting the current function. When the `graph` property signals plotting ( $p_1$ - $p_4$ ), the `points` property is ignored. If the `graph` property signals graphing ( $c_1$ - $c_4$ ;  $a_1$ - $a_4$ ), and argument #3 is a set of rectangular points, the `points` property must either not be present, or set to zero (`points=0`).
- `xInterval`: (Required) An interval on the  $x$ -axis, the interval must be in the form  $\{[a, b]\}$ , for example, `xInterval={ [0, 1] }`. For functions of  $x$ , this interval represents the domain over which the function is graphed. It also represents the left and right boundaries of the graphing window.

**Important:** The `xkeyval` package parses these parameters. Because the interval notation contains a comma (`,`), the whole interval must be enclosed in braces so the parsing will be correct, as illustrated above.

- `yInterval`: (Required) An interval on the  $y$ -axis, the interval must be in the form `{[a, b]}`, for example, `yInterval={ [0, 1]}`. It represents the upper and lower boundaries of the graphing window.

As with `xInterval`, the interval needs to be enclosed in braces.

- `tInterval`: When plotting a set of parametric equations, or a polar function, this interval is required as a parameter. The interval is of the form `{[a, b]}`, including the braces, and represents the domain of the parameter. The `tInterval` must not be included otherwise, that is, for graphing a function of  $x$ . Some early  $\TeX$  parsing tests whether the value of `tInterval` is empty (the default) or not. If nonempty, we assume the graphing is parametric or polar. For point plotting, `tInterval` must not be included in the parameter list.
- `noquotes`: When argument #3 is parsed, it is, by default, placed in double quotes, for example, `"x^2"`; however, there are some situations where the double quotes should not be used. (See the `afgraph.tex` file for one such example.) Possible values for `noquotes` are `true` and `false`. Including `noquotes` in the option list is equivalent to `noquotes=true`. The default is `noquotes=false`.
- `wait`: Possible values for `wait` are `true` and `false`. Including `wait` in the option list is equivalent to `wait=true`. The default is `wait=false`. When using `\defineGraphJS` to create a JavaScript action that will execute multiple calls to `Graph_xy` or `Graph_xyt`, list `wait` in the option list. This will cause a slight delay that allows the graphing screen to become activated, (if not already activated) before the graphing data is created and sent to the AcroTeX graphing widget. See the example below in [Section 3.5](#), page 15.

3. `<func|points>`: This argument can be a function or a set of points.

- A function can be three types: (1) a function of  $x$ ; (2) a function of  $t$ ; and a pair of function of  $t$ . If there is a single function of  $t$ , case (2), that is interpreted as a polar function, and graphed accordingly. The pair of functions must be functions of  $t$  and separated by a semi-colon (`;`); these are then interpreted as a set of parametric equations. For example, `x^2` would be graphed as a parabola; `1+sin(t)` would be graphed as a Cardioid in the polar coordinate system; and `cos(t); sin(t)` would be graphed as a circle.
- Points can be input as a semi-colon-delimited list of rectangular coordinates. For example, `(1,2); (2,3); (5,6)`. Points can be plotted discretely, or plotted and connected with either a smooth curve, or line segments.

4. `<text>`: The text that the link is attached to, when this text is clicked, the defined action of populating the graph occurs.

### 3.5. Graphing with `\defineGraphJS`

The `\defineGraphJS` is a command that expands to either `Graph_xy()` or `Graph_xyt()`, and is essentially the code used by `\sgraphLink`. Use `\defineGraphJS` to create a custom link action or form field action to graph pre-packaged functions.

```
\defineGraphJS{<graph_key_vals>}{<func|points>}{<command>}
```

**Command Description:** `\defineGraphJS` defines a new command `\<command>` what will expand to `Graph_xy()` or `Graph_xyt()` fully populated by its arguments. This command can be used to create new actions that involve multiple calls to the AcroTeX graphing routines.

**Parameter Description:** There are three required parameters.

1. `<graph_key_vals>`: The same key-value pairs as described for `\sgraphLink`.
2. `<func|points>`: An expression representing a function of  $x$ , a polar function of  $t$ , a set of parametric equations, or a set of points.
3. `<command>`: A command that this JavaScript code will be saved under.

An example of usage can be found in `afgraph.tex`, we present another example here, also included in `afgraph.tex`, that might suggest the value of this command. We construct a link that graphs a function and plots discrete points.

```
\def\DomX{[0,2PI]}\def\DomY{[-1,1]}
\defineGraphJS{graph=c1,wait,xInterval=\DomX,yInterval=\DomY,
  points=40}{sin(x)}{\mySineCurve}
\defineGraphJS{graph=p1,wait,xInterval=\DomX,yInterval=\DomY}
  {(0,sin(0));(PI/2,sin(PI/2));(PI,sin(PI));(3PI/2,sin(3PI/2));
  (2PI,sin(2PI))}{\mySinePoints}
\setLinkText[\A{JS{%
  \clearGraphJS\r
  \mySineCurve\r
  \mySinePoints
}}]{Consider the sine function and indicated points}%
```

Note the use of the `wait` key in both the definitions to give the graphing screen time to be activated and ready to receive data. Observe also the list of points is given in symbolic form, we let JavaScript calculate the values for us.

The command `\clearGraphJS` is used to clear the graphing screen before new curves are written to the screen. `\clearGraphJS` expands to the document-level JavaScript function that clears the graphing screen.

## 4. Customizations

There are a number of English phrases that appear as tooltips or as messages in alert dialog boxes, as a result, the acroflex package has a language option.

```
\usepackage[lang=english|german]{acroflex}
```

Specifying `english` as the value of `lang` inputs the file `afcustom_us.def`, which normally does nothing; the definitions made in this file are the hard-wired defaults of the package. As an English speaker, you can edit this file, and improve the phrasing, if you wish. Specifying `german` as the value of `lang` inputs the file `afcustom_de.def`; you can, of course, edit this file to get a preferred phrasing. If not `lang` key-value pair is specified, the `acroflex` package inputs the file `afcustom.def` (found in the examples folder). This file is intended for local use. Place it in the folder where the source file resides, modify it as desired to get custom messages. The file `afcustom.def` contains some instructions and guidelines for editing.

- ▶ If the file `afcustom.def` is placed on the latex search path, it will be found and input for each source file; if `afcustom.def` is in the source file folder, it is this version that is found first and input.

Should the phrases entered in `afcustom.def` require special accents, use the `unicode` option of `Web` (which just passes the `unicode` option on to `hyperref`), and enter any special characters using  $\TeX$  notation. For example, to address my formerly favorite friend, Jürgen, we can write,

```
\ttgraphBtn{J\"{u}rgen, press to graph the function}
```

For the alert box messages, use JavaScript unicode notation, for example

```
\defineJSstr{\af@badNumberMsg}{%
  J\u00FCrgen, the value input does not appear to be a number,
  please enter a number, or an expression that evaluates to a
  number. \dps}
```

This latter example does not require the `unicode` option.

Note that `\defineJSstr` is a new command (defined in `eforms`) that enables you to enter unicode, for example, `\u00FC` is the u-umlaut (`"{u}` or `ü`). Also, within the argument string, you can use `\r` (carriage return) and `\tab` to format your lines as needed. Double backslash `\\` is converted into single backslash `\`, so for example, `\\defineJSstr` appears in the dialog box as `\defineJSstr`. The string argument is immediately expanded, so a command like `\dps` (in the above definition) gets expanded at definition time. Use `\protect` to delay the expansion until the tex compiler finally expands the JS command string (useful here, if `\dps` gets redefined).

That's all for now, I simply must get back to my retirement.  $\mathcal{D}$