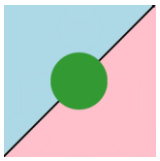
SVG Family-Tree Generator (v6.0.0) — Program Notes



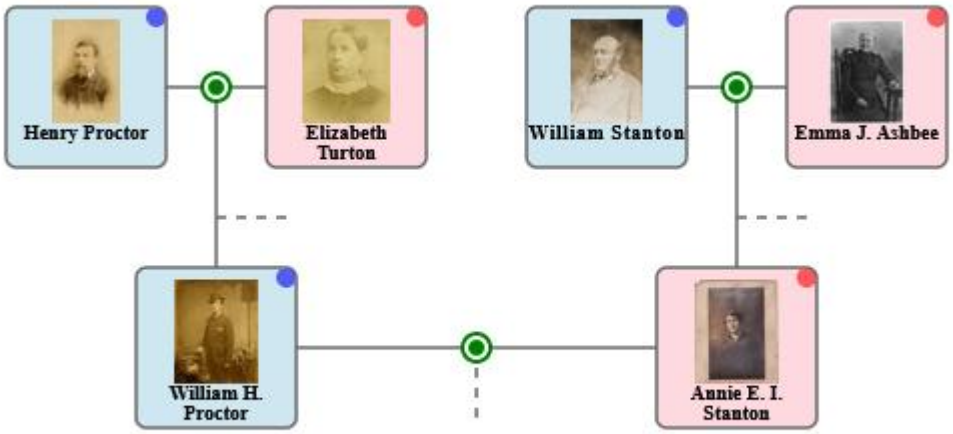
Tony Proctor, 21 Apr 2021

Contents

1 Introduction

This document contains low-level program documentation for the SVG utility (hereinafter called SVG-FTG) that would be useful to application developers and power users. There is a separate User Guide document that explains how to use SVG-FTG.

SVG-FTG generates interactive family trees that may be viewed in a browser such as Firefox, Chrome, or Microsoft Edge. The trees are built using a combination of HTML, SVG, CSS, and JavaScript technologies, and may include clickable actions on the person-boxes and family-circles.



Each tree is defined by a simple text file (usually named Tree*.txt) rather than taking a definition directly from a database or a GEDCOM file. This is because it offers control over which persons and relationships are to be shown, and how they are to be laid out. It may even be used to depict a fictitious or deliberately inaccurate tree, as in Boots Made For Walking.

The initial goal was clean and crisp visualisation to accompany narrative reports (even at large magnification) rather than lots of swirls and scrolls, or gratuitous colours. Scalable Vector Graphics (SVG) is ideal for this as it allows trees to be created that would scale indefinitely rather than going all fuzzy at high magnification. This goal was later supplemented by the addition of these interactive applications and services to family trees, such as 'Timeline Reports' or even custom ones (see Application Development).

As of v5.0.0, SVG-FTG hosts a comprehensive GEDCOM Browser that supports copy-and-paste between the GEDCOM and the Tree Designer. The browser allows navigation by menu of the associated individuals and families, and the ability to copy a person, person with spouse, or a couple and their direct children, onto the clipboard for pasting into the Tree Designer.

2 Configurations

If SVG-FTG generates a single *.html file containing the necessary combination of HTML/SVG/CSS/JavaScript then it may be necessary to copy some or all of this into another file, such as a Web page or blog post. The way this is done depends on the ultimate goal, and there are a number of configuration alternatives.

Here's a small, but typical, output from SVG-FTG when the setting InfoPanels=True is active:

```
<!DOCTYPE html>
<html>
<head>
<title>SVG Family Tree TreeDoc</title>
<meta charset="utf-8"/>
<meta name="generator" content="Generated by Parallax View's SVG Family-Tree Generator V6.0.0.
See https://parallax-viewpoint.blogspot.com/2018/09/svg-family-tree-generator-v50.html"/>
```

```
<link rel='stylesheet' type='text/css'
href='https://parallaxviewpoint.com/SVGcode/infopanels6.css'>
<style>
div.tp-svg {
    display: inline-block;
    background: #f0f0f0;
    border-style: solid;
    border-width: thin;
    margin: 0px;
    padding: 0px;
    width: 100%;
    max-width: 570px;
    height: 77vh;
/*    padding-bottom: 77vh; */
    max-height: 440px;
    position:relative;
}

svg#TreeDoc {
    position:absolute;
    left:0; top:0; width:100%; height:100%;
    margin: 0px;
    padding: 0px;
}
</style>
```

CSS-Text — CSS for styling the main SVG frame.

```
<script src='https://parallaxviewpoint.com/SVGcode/infopanels6.js'></script>
<script>
function clickHandler(ev,type,inst,key) {
// ev=event object, type=P/F, inst=optional instance ID, key=person or family key name
    if (type == 'P') {
        if (ev==null || (!ev.ctrlKey && !ev.shiftKey && !ev.altKey && !ev.metaKey)) {
            ip_showDiv(ev,type,inst,key);
            ev.preventDefault(); return;
        }
        if ((ev.ctrlKey || ev.metaKey) && !ev.shiftKey && !ev.altKey) {
            ip_resetDivs(ev,type,inst,key);
            ev.preventDefault(); return;
        }
    }
    if (type == 'F') {
        if (ev==null || (!ev.ctrlKey && !ev.shiftKey && !ev.altKey && !ev.metaKey)) {
            ip_showDiv(ev,type,inst,key);
            ev.preventDefault(); return;
        }
        if ((ev.ctrlKey || ev.metaKey) && !ev.shiftKey && !ev.altKey) {
            ip_resetDivs(ev,type,inst,key);
            ev.preventDefault(); return;
        }
    }
}

function clickHandlerTR(ev,type,inst,key) {
    alert('TR button clicked for ' + type + inst + '-' + key);
}

function clickHandlerTL(ev,type,inst,key) {
    alert('TL button clicked for ' + type + inst + '-' + key);
}

function clickHandlerBR(ev,type,inst,key) {
    alert('BR button clicked for ' + type + inst + '-' + key);
}

function clickHandlerBL(ev,type,inst,key) {
```

Javascript — Code to handle click actions on boxes, circles, and buttons.

```
    alert('BL button clicked for ' + type + inst + '-' + key);
}
</script>

</head>

<body>
<div class="tp-svg" id="tp-TreeDoc">
<svg viewbox="0 0 570 440" id="TreeDoc" xmlns="http://www.w3.org/2000/svg" version="1.1"
xmlns:xlink="http://www.w3.org/1999/xlink">
<desc xmlns:dc="http://purl.org/dc/elements/1.1/">
<dc:publisher>Generated by Parallax View's SVG Family-Tree Generator V6.0.0. See
https://parallax-viewpoint.blogspot.com/2018/09/svg-family-tree-generator-
v50.html</dc:publisher>
</desc>

<style>
text {
    text-anchor: middle;
    font-size: 13px;
    font-weight: bold;
    font-family: "Times New Roman";
    pointer-events: none;
}

.f, .m, .u, .x {
    stroke: dimgray;
    stroke-opacity: 1.0;
    stroke-width: 1.5;
    fill-opacity: 0.6;
}

.f {
    fill: pink;
}

.m {
    fill: lightblue;
}

.u {
    fill: lightgray;
}

.x {
    fill: lightgray;
}

.f.tr, .f.tl, .f.br, .f.bl {
    fill: red;
    stroke: none;
}

.m.tr, .m.tl, .m.br, .m.bl {
    fill: blue;
    stroke: none;
}

.u.tr, .u.tl, .u.br, .u.bl {
    fill: dimgray;
    stroke: none;
}

.x.tr, .x.tl, .x.br, .x.bl {
    fill: dimgray;
```

CSS-SVG — CSS for styling SVG elements

```
    stroke: none;
}

line {
    stroke: dimgray;
    opacity: 1.0;
    stroke-width: 1.5;
}

line.direct {
    stroke: #7070ff;
    opacity: 1.0;
    stroke-width: 4;
}

line.tentative, line.more {
    stroke-dasharray: 5px;
}
</style>
```

```
<defs>
    <g id="G-FamC">
        <circle fill="green" cx="0" cy="0" r="8"/>
    </g>

    <g id="G-FamCH">
        <circle fill="green" cx="0" cy="0" r="8"/>
        <circle fill="white" cx="0" cy="0" r="6"/>
        <circle fill="green" cx="0" cy="0" r="4"/>
    </g>

    <g id="G-FamC-L">
        <circle fill="gray" cx="0" cy="0" r="8"/>
    </g>

    <g id="G-FamCH-L">
        <circle fill="gray" cx="0" cy="0" r="8"/>
        <circle fill="white" cx="0" cy="0" r="6"/>
        <circle fill="gray" cx="0" cy="0" r="4"/>
    </g>

    <g id="G-Box">
        <rect width="80" height="80" rx="0" ry="0"/>
    </g>

    <g id="G-ButtonTR">
        <rect width="10" height="10" rx="0" ry="0" clip-path="url(#G-ClipTR)"/>
    </g>

    <g id="G-ButtonTL">
        <rect width="10" height="10" rx="0" ry="0" clip-path="url(#G-ClipTL)"/>
    </g>

    <g id="G-ButtonBR">
        <rect width="10" height="10" rx="0" ry="0" clip-path="url(#G-ClipBR)"/>
    </g>

    <g id="G-ButtonBL">
        <rect width="10" height="10" rx="0" ry="0" clip-path="url(#G-ClipBL)"/>
    </g>

    <g id="G-IconTR" style="opacity:0.6;">
        <image width="10" height="10" xlink:href="" clip-path="url(#G-ClipTR)"/>
    </g>
```

SVG-Elements — SVG boxes, circles, and lines representing persons and families

```
    <g id="G-IconTL" style="opacity:0.6;">
       <image width="10" height="10" xlink:href="" clip-path="url(#G-ClipTL)"/>
    </g>

    <g id="G-IconBR" style="opacity:0.6;">
       <image width="10" height="10" xlink:href="" clip-path="url(#G-ClipBR)"/>
    </g>

    <g id="G-IconBL" style="opacity:0.6;">
       <image width="10" height="10" xlink:href="" clip-path="url(#G-ClipBL)"/>
    </g>

    <clipPath id="G-ClipTR"><path d="M0,0.75 L0,15 L14.25,15 L14.25,0.75 L0,0.75
Z"/></clipPath>
    <clipPath id="G-ClipTL"><path d="M0.75,0.75 L0.75,15 L15,15 L15,0.75 L0.75,0.75
Z"/></clipPath>
    <clipPath id="G-ClipBR"><path d="M0,0 L0,14.25 L14.25,14.25 L14.25,0 L0,0 Z"/></clipPath>
    <clipPath id="G-ClipBL"><path d="M0.75,0 L0.75,14.25 L15,14.25 L15,0 L0.75,0
Z"/></clipPath>
</defs>
<line class="bridge" x1="260" y1="220" x2="440" y2="220"/>
<line class="" x1="350" y1="220" x2="350" y2="310"/>
<use id="F-JamesElizabeth2" tabindex="0" class="" xlink:href="#G-FamC" x="350" y="220"
onclick="clickHandler(evt,'F','','JamesElizabeth2');">
</use>
<line class="bridge" x1="390" y1="350" x2="440" y2="350"/>
<use id="F-ThomasMargaret" tabindex="0" class="" xlink:href="#G-FamC" x="415" y="350"
onclick="clickHandler(evt,'F','','ThomasMargaret');">
</use>
<g id="P-JamesA3">
<use tabindex="0" xlink:href="#G-Box" class="m" x="180" y="180"
onclick="clickHandler(evt,'P','','JamesA3');">
</use>
<text x="220" y="180" dy="1.84em">James</text>
<text x="220" y="180" dy="3.07em">Astling</text>
<text x="220" y="180" dy="4.3em">(1727&#x2013;1789)</text>
</g>
<g id="P-ElizabethD">
<use tabindex="0" xlink:href="#G-Box" class="f" x="440" y="180"
onclick="clickHandler(evt,'P','','ElizabethD');">
</use>
<text x="480" y="180" dy="1.84em">Elizabeth</text>
<text x="480" y="180" dy="3.07em">Dickinson</text>
<text x="480" y="180" dy="4.3em">(1743&#x2013;1824)</text>
</g>
<g id="P-MargaretA">
<use tabindex="0" xlink:href="#G-Box" class="f" x="310" y="310"
onclick="clickHandler(evt,'P','','MargaretA');">
</use>
<text x="350" y="310" dy="1.84em">Margaret</text>
<text x="350" y="310" dy="3.07em">Astling</text>
<text x="350" y="310" dy="4.3em">(1784&#x2013;1869)</text>
</g>
<g id="P-ThomasH">
<use tabindex="0" xlink:href="#G-Box" class="m" x="440" y="310"
onclick="clickHandler(evt,'P','','ThomasH');">
</use>
<text x="480" y="310" dy="1.84em">Thomas</text>
<text x="480" y="310" dy="3.07em">Hallam</text>
<text x="480" y="310" dy="4.3em">(1782&#x2013;1850)</text>
</g>

</svg>
</div>
```

```
<br/>
<div class="tp-male" id="P_JamesA3" style="display:none;">
Baptised 17 Sep 1727 at Coddington All Saints.
Buried 10 Oct 1789 at Coddington All Saints.
</div>
<div class="tp-female" id="P_ElizabethD" style="display:none;">
Baptised 4 Jul 1743 in Long Bennington, Lincs.
Buried 11 Nov 1824 at Coddington All Saints.
</div>
<div class="tp-female" id="P_MargaretA" style="display:none;">
Born 28 Sep 1784 in Coddington.
Baptised on 10 Oct 1784 at Coddington All Saints.
Died 1869, aged 92, and was buried at Woodborough St. Swithun on 13 Oct 1869.
</div>
<div class="tp-male" id="P_ThomasH" style="display:none;">
Baptised 14 Feb 1782 at Lowdham St. Mary The Virgin.
Buried 4 Jan 1850 at Bingham St. Mary and All Saints.
</div>
<br/>
<div class="tp-family" id="F_JamesElizabeth2" style="display:none;">
Married on 22 Jul 1784 at Coddington All Saints.
</div>
<div class="tp-family" id="F_ThomasMargaret" style="display:none;">
Married 9 Feb 1803 at Screveton St. Wilfrid.
Thomas was a POW in Napoleonic France until 1814, while Margaret took up with a Thomas Meads
in Epperstone.
</div>
</body>
</html>
```

Biographical notes for the persons and families.

When modifying an existing web page, or a blog-post, using a visual design tool then a useful approach is to leave a place-holder in the normal that marks where you would like to insert your tree(s). When finished, go the HTML view (which is hopefully available) and replace your place-holder with the code for the corresponding tree. Also, it's good practice to put the <head> parts in any existing <head> section in that HTML.

There are two main options for adding trees anywhere: to insert the contributions directly (inline), or to host them elsewhere and simply point your page to them (linked). The original blog-post about this project (Interactive Trees in Blogs Using SVG) put all the possible contributions inline (CSS-Text, JavaScript, CSS-SVG, SVG-Elements , and notes divs). This case uses the pop-up information panels for person and family notes.

Subsequent examples, such as Impediment to Marriage, used the linking method. Because they used InfoPanels=False, and possibly Tooltips=True, then the CSS-Text only styled the SVG frame, and the JavaScript and notes divs sections were omitted. The CSS-Text contribution was placed near the head of these blog posts, and the CSS-SVG and SVG-Elements contributions placed in a *.svg file in a Web host such as https://neocities.org. This could then be linked in two main ways:

Case 1:

```
<embed src='https://parallaxview.neocities.org/Blogger/TreeAstling.svg'/>
```

This case is the easiest, and it ensures that event handling (for tooltips or interactive applications) still works.

Case 2:

```
<img src='https://parallaxview.neocities.org/Blogger/TreeAstling.svg'
onerror="this.src='https://parallaxview.neocities.org/Blogger/TreeAstling.p
ng'"
onclick="window.open(this.src,'_blank');"
```

This case is a little more static as the event handling doesn't work. However, the onerror clause allows some fall-back to display a static image file (*.png in this example) if someone's browser cannot display SVG. The onclick clause further allows a fully working event version to be displayed in a separate page by clicking on the SVG shown in your current page.

A case of "swings and roundabouts" (as they say here) in that you can't get both features (events and fall-back) to work together.

The example at Boots Made For Walking illustrates another case: including multiple SVG images in the same page. It is necessary for the CSS-Text contributions to apply to specific SVG frames, and for any event handling to deal only with the corresponding persons, families, notes, or program data. This is achieved by specifying a digit or letter for 'Instance ID' in the advanced settings, or in an Inst=n setting of a header record in the tree definition file. For instance, Inst=9.

For the frame styling (sizing, in particular), this might appear as follows:

```
<style>
div.tp-svg9 {
    …etc…
}
</style>
```

```
<div class="tp-svg9" id="tp-TreeTest">
<svg viewbox="0 0 1155 440" id="TreeTest"
xmlns="http://www.w3.org/2000/svg" version="1.1"
xmlns:xlink="http://www.w3.org/1999/xlink">
    …etc…
</svg>
</div>
```

When clicking on a person-box or a family-circle then the JavaScript event handler is passed this instance ID, e.g.

```
onclick="clickhandler(evt,'F','9','HenryEmmaE');"
```

This can then act on the corresponding information panel because its 'id' value will contain the instance ID:

```
<div class="tp-family" id="F9_HenryEmmaE" style="display:none;">
After Henry&amp;apos;s death, Emma remarried to Luke Hallam in 1892
</div>
```

This 'instance ID' is omitted if the Inst=n setting is not specified.

If you have set SVGFile=True then the situation is somewhat simpler. The contributions CSS-SVG and SVG-Elements are written to a *.svg file., and this may be linked to your Web page or blog post as shown in the example cases 1 and 2, above. NB: it may be necessary to tell your system that such *.svg files should be opened with the same browser as *.html files as it's not always set by default.

Because CSS-Text , JavaScript, or notes divs contributions can still be generated then some interactive applications are still supported, but the lack of HTML will restrict the list. Setting Tooltips=True still works.

The CSS-Text contribution might contain something like:

```
div.tp-svg {
    display: inline-block;
    background: #f0f0f0;
    border-style: inset;
    border-width: thin;
    margin: 0px;
    padding: 0px;
    width: 100%;
    max-width: 1155px;
    height: 39vw;
    max-height: 440px;
    position:relative;
}

svg {
    position:absolute;
    left:0; top:0; width:100%; height:100%;
    margin: 0px;
    padding: 0px;
}
```

and this would have controlled the size and style of the bounding , which might have looked as follows:

```
<div class="tp-svg" id="tp-TreeHammond1">
… inline SVG …
</div>
```

2.1 Use of the id Attribute

The generated HTML/SVG code uses the following conventions for its 'id=' attributes.

Notes divs: {P/F}[inst]_key (e.g. P_key1, or F2_key2)

Program Data divs: {P/F}[inst].key (e.g. P.key2, or F4.key9)

SVG elements: {P/F}[inst]-key (e.g. P-key3, or F9-key4) [i.e. boxes and circles]

SVG group entities: G[inst]-key (e.g. G-FamC, or G0-FamCH)

2.2 Symbolic Names

Various symbolic names all follow the same character-set rules, being composed of alphanumerics, underscores, and hyphens, starting with an alphabetic character. These include:

- Person, family, place, and viewpoint keys.
- Tree titles.
- Application, service, and data signature ids.
- Tag names for parts of a registered data signature.
- Header settings, and local per-person or per-family settings.
- Event types, used in the event mark-up of HTML notes

2.3 User-defined CSS Classes

A number of global and local header settings are provided to allow the addition of user-defined CSS class names to certain visible entities. These allow the presentation of those entities to be customised for your particular application.

Each of the following settings accepts a "|"-separated list of class names, each of which has certain substitution markers defined to make the actual class name dependent upon other factors. These settings are:

- *ClassPerson*: Adds a class to person-boxes.
- *ClassPersonLines*: Adds a class to the lines onto person-boxes.
- *ClassPersonNotes*: Adds a class to the elements holding biographical notes for persons.
- *ClassFamily*: Adds a class to family-circles.
- *ClassFamilyLines*: Adds a class to the lines associated with families. This is described in more detail below because the tree structure is not straightforward .
- *ClassFamilyNotes*: Adds a class to the elements holding biographical notes for families.

The substitution markers are:

- *?*: Substitutes the value of any instance ID defined in the advanced settings (see Inst header setting and the Advanced-Settings form).

- *??*: Substitutes the associated key value of the respective person or family.

A number of class names are already generated by SVG-FTG, and these should be considered reserved:

- *For persons*: sex classes (f, m, u, x).
- *For lines*: tentative parentage and unshown children (broken), direct lineage from root person (direct), and for differentiating parts of tree structure (bridge, descender, rail).
- *For buttons*: sex classes (f, m, u, x) and corner classes (tr, tl, br, bl).
- *For notes*: sex classes (tp-female, tp-male, tp-unk, tp-other).

So, for instance, with header settings of:

```
Inst=9, ClassPerson=C?-??
```

then a person with a key of JoeBloggs would be given CSS class names as follows:

```
class="m  C9-JoeBloggs"
```

When a class is associated with family lines (via ClassFamilyLines) then it must be considered in combination with other reserved classes in order to differentiate parts of the tree structure. Developers can modify the formatting associated with these reserved classes in order to better visualise how they're used. All of the following are in addition to any ClassFamilyLines that the user (or application) has added:

- The line between two parents has the reserved class 'bridge'.
- If there are several children then their person lines descend from a horizontal line with the reserved class 'rail'. When this rail is present then a vertical line connects it to the family-circle and this has a reserved class of 'descender'.
- With just one child then there may be no descender or rail — just the person line directly to the child — of there may be a "dog leg" consisting of a descender, short rail, and one person line.
- The broken line representing tentative parentage has the reserved class 'tentative'. The broken line representing unshown children, usually displayed at the end of the rail, has the reserved class 'more'.

Hence, the classes will be combined as follows:

- Bridge = (ClassFamilyLines) + 'bridge'
- Descender = (ClassFamilyLines) + 'descender'
- Rail = (ClassFamilyLines) + 'rail'

- Unshown children = (ClassFamilyLines) + 'more'
- Person lines = (ClassFamilyLines) + (ClassPersonLines)
- Tentative parentage = (ClassFamilyLines) + (ClassPersonLines) + 'tentative'

If the tree orientation is vertical, rather than horizontal, then the same classes are applied in the same way. When Fanned=True then the person lines come straight from the family-circle (i.e. no rail or descender), and the 'more children' line (class 'more') is not present as the striped family-circle indicates the same status. The 'Ancestral Links' application provides a good example of how to use these.

2.4 Changing the Lines

There are a number of configuration options for changing the style of the lines used between spouses and from family-circles to associated children.

Tentative parentage

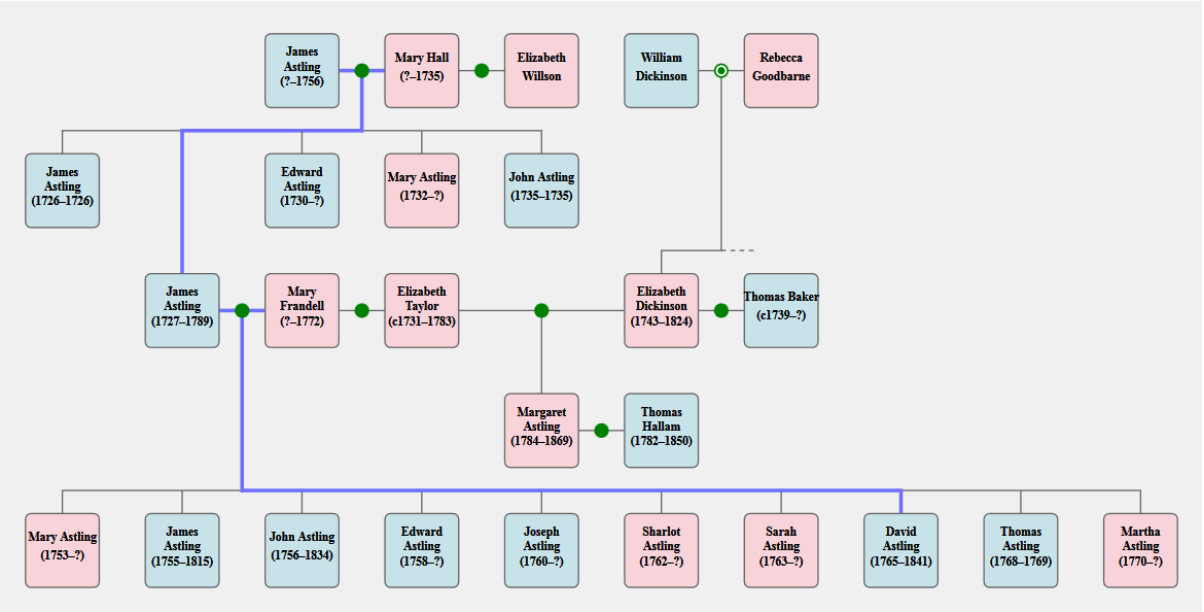
This option is presented to you as a checkbox when you edit the details of a person in the Tree Designer. When selected then any associated child link, from a parent family-circle, will be depicted with a dashed line rather than a solid line.

Fanned=boolean

This tree setting duplicates the Tree Designer's representation of the tree by using direct, angled, lines rather than the horizontal/vertical combination normally associated with family trees. If you find that you have crossing lines, say because you have tried to include ancestors of both spouses when there is *pedigree collapse*, then this representation may be less ambiguous. It is recommended that Opaque=True is always set for fanned lines in order to avoid them showing through the person-boxes.

RootKey=key

This tree setting (which is currently available as a checkbox in the Edit-Person form) identifies a specific person for whom you wish to emphasise all their ancestral lines. What it does is trace the ancestors — both paternal and maternal — for each previous generation, and highlights the paths with a different line style. This is coloured differently and is wider than default lines, but this can be customised by modifying the CSS for the line class "direct".



Using CSS

Prior to V6, SVG-FTG automatically associated the lines of each family with a given distinct class name of C[inst]-familykey. For instance, "C-JohnSarah1880" or "C9-JSmithCJones" (where 'inst' is the optional instance ID that you may have set in the Advanced-Settings form).

This allowed you to change the presentation of these lines on a per-family basis. For instance, you could add extra CSS control in the <style>…</style> section for each family key that you wish to change.

```
line.C-JamesMary2 {
   stroke: red;
}
```

This feature is no longer provided by default, but it can easily be re-added using a header setting of:

```
ClassFamilyLines=C?-??.
```

As of V6.0.0, though, User-defined CSS Classes can be associated with different visible entities, and on a global or local level. See User-defined CSS Classes for more details.

2.5 Changing the Buttons

The Advanced-Settings form has a checkbox for each corner of a person-box that will place a simple button in those corners. This corresponds to the Header record settings of ButtonTL=True, etc. Buttons can also be selectively enabled or disabled per person-box using local settings (see "User Guide:Local Settings").

By default, these buttons are coloured according to the sex of the person, but this can be changed by modifying the corresponding CSS in the <style> section of the header template files (Header*.*).

For instance, the following CSS formats buttons appearing in all four potential corners of a female person-box so that they are coloured red and have no border:

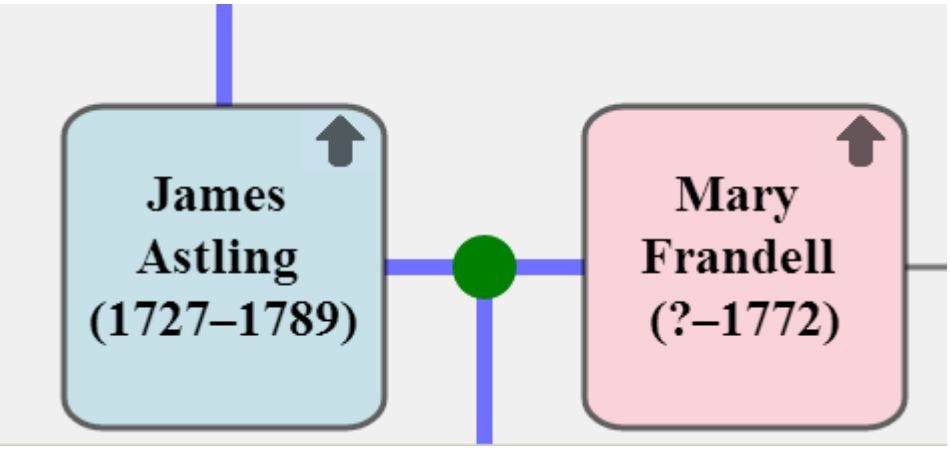
```
.f.tr, .f.tl, .f.br, .f.bl {
    fill: red;
    stroke: none;
}
```

If rounded button corners are requested in the Edit-Settings form (corresponding to the setting Rounded=True) then the buttons will change from a rectangle to a circle (or ellipse if the BW and BH settings are different) so that they fit neatly into the respective corners.

The Advanced-Settings form also offers control over icon replacements for the simple buttons (corresponding to the IconTL, etc., Header record settings) and "hover text" tooltips (corresponding to the TooltipTL, etc., Header record settings). The icon paths are treated as a URL references to icon images to use for the corresponding button corner. For instance, header settings of

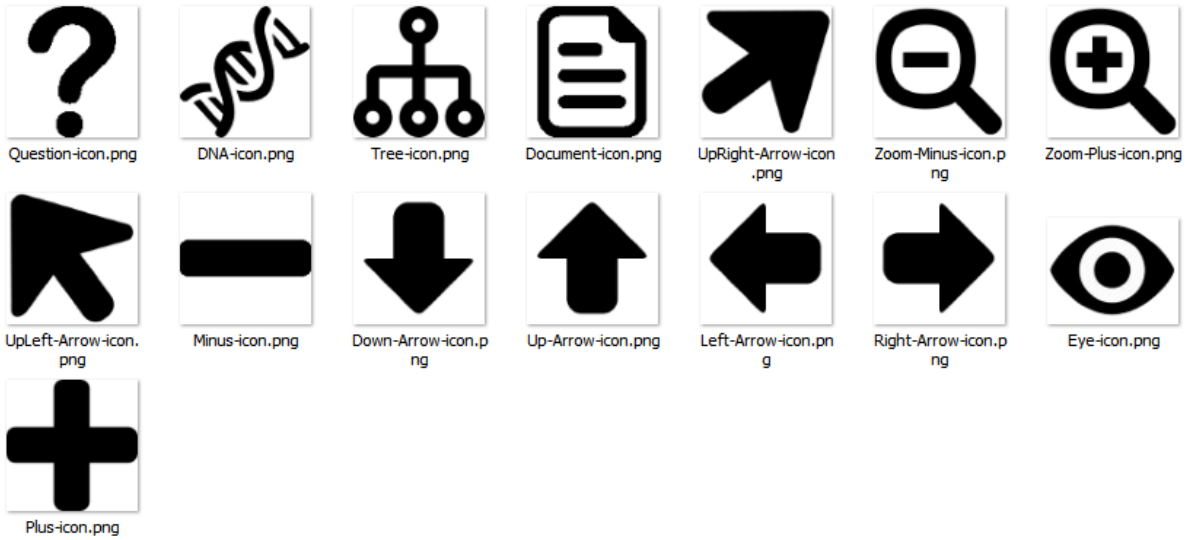
```
IconTR= https://parallaxviewpoint.com/SVGcode/Up-Arrow-icon.png
Rounded=True
```

would result in something like



If the person-box corners are rounded (as in this example) then the image will be clipped according to the normal button size and shape, if necessary, to stay within the person-box border.

That same https://parallaxviewpoint.com/SVGcode/ folder currently contains the following useful icons:



Depending upon the selected image, you may need to keep the coloured shape behind it. This can be done by modifying the associated G-Icon group element (in the Header*.* file) to include both a button rectangle and a button image, in that order. For instance:

```
<g id="G??inst??-IconTR">
  <rect width="??bw??" height="??bh??" rx="??rx??" ry="??ry??"/>
  <image width="??bw??" height="??bh??" xlink:href="??image-tr??"
        clip-path="url(#G??inst??-ClipTR)"/>
</g>
```

The tooltip settings allow a line of text to be specified for a button tooltip, when the mouse hovers over them.

```
TooltipTR=This is my tooltip
```

Suppose, though, that you need a more visual indication that these buttons are selectable. The easiest way is to adjust their formatting when the end-user hovers over them. For instance, the following extra bit of CSS code would temporarily put a border on the button's associated rectangle:

```
#G??inst??-Button:hover, #G??inst??-IconTR:hover {
    stroke: gray;
   s  troke-width: 1.0;
}
```

NB: there are some documented issues with the Google Chrome browser not honouring this 'hover' pseudo-class for SVG elements.

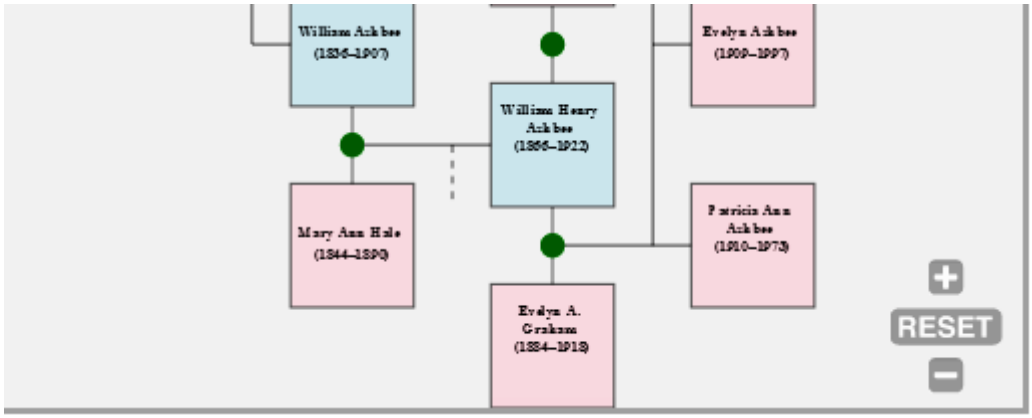
Global settings (via the Advanced-Settings form, or the Header record of the *.txt file) and local settings (via the Edit-Person form, or the "/" record of the *.txt file) can be mixed. For instance, enabling buttons globally but setting tooltips locally, or only enabling buttons locally. Because of the way button icons are implemented, it is not possible to define button icons on a per-box basis.

2.6 Pan-Zoom

In 2009, Andrea Leofreddi developed a JavaScript library for panning and zooming SVG images, called SVGPan. This became an open-source project, called svg-pan-zoom, and is currently available for use under a BSD 2-clause "Simplified" License (a permissive license that comes in two variants: the BSD 2-Clause and BSD 3-Clause. Both have very minute differences from the MIT license). [*Thanks to Bob Fieg for bringing this project to my notice*]

This library can easily be included in a Web page to enable panning around and zooming in/out of a specific SVG image within that page. It is very flexible and includes several customisation options.

If this option is selected then the SVG image will have a couple of controls visible in the bottom-right of the frame.



These allow zoom-in (+) or zoom-out (-) of the current image. Unlike the Ctrl/+ and Ctrl/- operations supported by your browser, these controls apply **only** to the current SVG image, and so are more convenient for examining a particular tree. If you click-and-drag on the image then it allows you to move around the tree in a way superior to clunky scrollbars. The 'Reset' control returns both the zoom scale and the pan position back to their initial values.

Application developers need to be aware of whether Pan-Zoom is configured since the size and location of elements may have been transformed, and no longer match the values directly associated with the elements. The Pan-Zoom code wraps all the SVG content inside an extra <g> element such that panning and zooming is achieved by modifying a transform attribute associated with this element. When the code is initialised, the SVG structure is changed from this:

```
<svg>
  <line ...>
  <line ...>
  ...
</svg>
```

to this:

```
<svg>
  <g transform="matrix(1 0 0 1 0 0)">
```

```
    <line ...>
    <line ...>
    ...
  </g>
</svg>
```

The Pan-Zoom help declares that all content has to be in the DOM before the Pan-Zoom code is initialised. However, SVG-FTG easily overcomes this by wrapping the SVG elements inside an empty <g> element of its own before that initialisation, and this not only avoids Pan-Zoom adding another but it gives an anchor onto which dynamic elements can be added. That is, adding them to the <g> rather than the parent <svg>. Those elements will then scale and move in synch with the statically declared ones. See ancestorlinks6.js for an example.

The id attribute of this extra <g> has the form tp-pz-svg, possible with an instance ID suffix, and its presence may be used as an indication that Pan-Zoom has been configured since it is not used in other contexts. NB: The programdata.js code has some support functions specifically for Pan-Zoom (see programdata.js).

The Pan-Zoom instance is always stored in a variable called tp-panZoomTree, possibly with an instance ID suffix. For instance, if your tree title was TreeAshbee then this is also the id attribute of the <svg> element. If the tree instance ID was x then the Pan-Zoom instance is created by code of the form:

```
var tp_panZoomTreex = svgPanZoom('#TreeAshbee',{
    zoomTree: true,
    panEnabled: true,
    dblClickZoomEnabled: false,
    controlIconsEnabled: true,
    fit: true,
    center: true,
    minZoom: 0.5,
    maxZoom: 10
    });
```

2.7 Foreign Characters

Although the majority of users will be content with ASCII characters, or at least the extended set of characters associated with West Europe (e.g. Windows code page 1252, or ISO 8859/1), some may want to try more exotic (to western usage) characters.

The internals of SVG-FTG are Unicode, which is a good start, and the tree definition files are written in UTF-8 (with an associated BOM, or Byte Order Mark). Also, modern Web browsers generally have good Unicode support. The only problem is the low-level software that SVG-FTG uses for the monitor screen as this is tied to a specific Microsoft system code page, also known as the ANSI character set.

As well as tree definition files, HTML, SVG, and GEDCOM output files are also written in UTF-8 with an accompanying BOM. When reading tree definition files, SVG-FTG supports BOM indications of UTF-8, Unicode little-endian, and Unicode big-endian. The default in the absence of a BOM is the current Windows ANSI character set. This also applies to the reading of GEDCOM files, but if its CHAR record conflicts with a BOM then an error is reported (see GEDCOM Data).

Regarding record terminators, SVG-FTG uses the Windows default of CRLF on output. During input, it additionally supports CR, LF, LFCR (unlikely to be seen outside of Acorn BBC and RISC OS systems), and CRCRLF (which occurs frequently when transferring files between differing systems).

2.8 Event Mark-Up

This section documents the format of the mark-up used in the biographical notes to define events, as described in "User Guide:HTML Editing".

This mark-up occurs on and elements within the notes. Such elements must include a data-start attribute giving the date; attributes for the event description (data-event) and event type (data-evtype) are optional, but note that selected event types are examined during an 'Auto Layout' operation. If you occasionally see attributes of data-gtag, data-gtype, or data-gpayl on this mark-up then they are simply there in order to preserve event details in the GEDCOM representation; it has no other use within SVG-FTG.

The dates have a standard form, and an extended GEDCOM-like form (enclosed in braces).

```
Standard  -  [c]yyyy[-mm[-dd]]

GEDCOM    -  { [kw1] yyyy[-mm[-dd]][J|D] [kw2 yyyy[-mm[-dd]][J|D]] }
```

where 'D' implies dual years showing the possible date alternatives for pre-1752 dates brought about by the changing the beginning of the year from March to January in the English calendar change of 1752, e.g. 1699/00. 'J' implies Julian dates (must apply to both dates if two present).

```
kw1 = ABT, CAL, EST, BEF, AFT, BET, FROM, TO

kw2 = AND (always if kw1=BET), TO (possibly if kw1=FROM)
```

Restrictions:

1) GEDCOM-like keywords must be uppercase
2) Only one space separating keywords from dates, and none separating the braces
3) Only Gregorian and Julian calendars supported
4) No "B.C." suffix supported
5) No DATE_PHRASE syntax supported.

NB: any unsupported or unrecognised date forms imported from GEDCOM are held in the tree definition file as "{@calendar-esc@date-string}" but are not processed.

2.9 GEDCOM Data

The GEDCOM browser accepts versions 5.5 and 5.5.1 of the GEDCOM specification. Extra code was added in order to cope with the variations in common use, including many illegal GEDCOM date forms, and custom tags used in the identification of media files (_FILE, _PRIM, _TYPE, and _KEYS). The BLOB tag, deprecated in 5.5.1, is ignored.

Some of the deviations from the GEDCOM date specification that are accepted by SVG-FTG include:

- Additional whitespace.
- Long (non-abbreviated) month names.
- Variant month abbreviations: 'SEPT'.
- Long (non-standard) keyword names, e.g. BETWEEN instead of BET.
- Trailing periods or commas on elements of the date, e.g. "Abt."
- ISO 8601 format dates: 'yyyy-mm-dd'.
- Day and month interchanged, e.g. 'mmm dd' instead of 'dd mmm'.
- Spurious combinations: 'FROM ABT', 'yyyy?', and 'yyy?'

Some valid GEDCOM dates are currently not supported by SVG-FTG because of the rarity of their usage.

- Only Gregorian and Julian calendars supported.
- .No “B.C.” (or "BC") suffix supported.
- No DATE_PHRASE syntax supported.

Dates below a certain threshold are deemed invalid. The default is 1000 but a field on the GEDCOM Browser form allows that to be changed.

SVG-FTG assumes a valid correspondence between its person/family keys and GECOM XREF identifiers. Those keys can only contain alphanumeric, underscore, and hyphen characters (and starting with an alphabetic character), but GEDCOM's XREF identifiers may contain many more types of character. If your GEDCOM file employs such characters in its XREF identifiers then it will need changes before loading.

SVG-FTG supports BOM (Byte-Order Marks) when reading GEDCOM files, and a variety of record terminators (see Foreign Characters), but it does not support the ANSEL character set. If there is no BOM then the data's CHAR record is honoured. If the CHAR record conflicts with a BOM then an error is reported and the BOM honoured.

2.10 Other Uses

Although the HTML/SVG output was primarily designed for blog posts and other Web pages, it has found some unexpected uses.

Maybe the simplest is that of being able to generate a static image to incorporate into a printed work, but it has also been used as an aid to producing presentations rather than genealogical writings.

One of my initial goals was to use it as a navigational aid. Since the biographical notes can include HTML then it means that they can include links to research articles that I'd already posted on the associated person, or family. In effect, I would be able to post a visual tree that linked to fully researched articles elsewhere — a million miles away from the typical online trees where you would be lucky to find a citation, let alone some justification and reasoning behind it!

With the advent of extra buttons, programmable event handling, application-defined data, and navigational data — culminating in packaged interactive applications (see Application Development) — the output from SVG-FTG can become the user interface to custom interactive applications.

The HTML in the biographical notes would use normal <a> elements to reference such articles with hyperlinks. However, the generated SVG also puts 'id=' attributes on person-boxes and family-circles, meaning that they can be referenced from elsewhere to (i.e. linked trees). The SVG 1.1 statement at Links into SVG has this to say on the subject:

> If the SVG fragment identifier addresses any element other than a 'view' element, then the document defined by the closest ancestor 'svg' element is displayed in the viewport using the view specification attributes on that 'svg' element.

However, during a forum question on this subject (see Best Way to Link into SVG Content), it was found that as long as tabindex=0 or -1 (but not +ve values) then a fragment on the enclosing HTML document pulls the specific SVG image into view, and the corresponding element scrolled across appropriately.

In other words, if an online research article used a hyperlink pointing to https://parallaxviewpoint.com/Blogger/TreeHammond2.svg#P2-HarrietHammond then it would display that tree and scroll the relevant person into view. Note that if the same URL fragment (i.e. the part after the hash character) was specified on a full HTML document address (as with http://parallax-viewpoint.blogspot.com/2017/05/boots-made-for-walking.html#P2-HarrietHammond) then it would also bring the appropriate tree into view (that article contains three of them) before scrolling to the referenced person. NB: this particular example won't work because it links to the SVG contribution via an <img> element in order to implement some fall-back. If it had linked to it via <embed>, or included it inline, then it would have worked. This limitation regarding fall-back is mentioned in the parent section, above.

In real-life, though, there are complications: the scrolling may be under the control of the Pan-Zoom library, and the highlighting of elements may be reliant on classes and CSS rather than any browser default. For this reason, our Linked Trees application performs these aspects manually.

One of the most interesting potential uses is that of a research tool. Since it has a graphical interface, employing mouse operations, it has been tested to see how easy it would be to visualise tentative relationships, and to record research notes for persons and relationships, as you go. It appears to work because the Tree Designer can display disjoint tree segments, and persons who are not yet connected to any tree.

3 Application Development

SVG-FTG is designed to support custom application development, and the 'Timeline Reports' application was produced partly to demonstrate this. When event handlers are employed then it furnishes the means by which family trees can be used as the user interface to interactive applications, i.e. they can provide functionality in response to mouse operations by the end-user, and so add value to what would otherwise be a static image.

There three sets of data applicable to this goal: 'notes data' (which is the biographical or historical information added by the end-user), 'program data' (which is totally application-defined) output by setting ProgData=True, and 'navigation data' (which describes the persons and families in the tree) output by setting NavData=True.

3.1 Application Registration

Interactive applications are registered in the file ApplicationsServices.xml, located in the same directory as the svg.exe file. It defines the standard interactive applications and services available to SVG-FTG. It also provides default configurations for their possible mouse and button operations, and the format of any data they require.

A service differs from an application in that it provides interactive functionality to applications rather than directly to the end-user.

These registrations may be overridden or supplemented by using custom versions of this file, called ApplicationsServices*.xml (in the same directory). Since SVG-FTG will automatically load all such files then the id attribute values are expected to be unique across all of files matching this wildcard.

Each application or service is defined by an XML <Application> element having the following attributes:

- *id*: Symbolic name of the application or service. As with person/family keys, these ids must be composed of alphanumerics, underscores, and hyphens and must start with an alphabetic character. In order to avoid a clash with any predefined ones, it is recommended that any custom ones should employ underscores and/or hyphens.
- *name*: Short name of the application or service. Used on menus.
- *descr*: Longer description of the application or service. Used in tooltips.
- *valid-svg*: Boolean value indicating whether the application or service is valid in an SVG-only file (see SVGFile header setting). The default is False.

- *srv* : Boolean value indicating that the current entry is a service rather than an application, and so it may be selected using <RequireSrv> but will not be presented to the end-user for manual selection. The default is False.

Each application or service may include a number of sub-elements to define and configure them. Multiple instances of each are valid, unless stated otherwise:

- *<IncludeCSS>*: Payload is the URL of any CSS file required by the application or service.
- *<IncludeJS>*: Payload is the URL of any JS file required by the application of service.
- *<RequireSetting name='header-setting' [weak='Boolean']>*: Declares that the application requires the specified header setting, using the payload as its value. For instance, <RequireSetting name='NotesData'>True</RequireSetting>. The 'weak' attribute is discussed below.
- *<RequireLocal name='setting' if='data-id'>*: Declares that the application requires the specified local setting (i.e. per-person or per-family), using the payload as its value. The 'if' attribute declares that the presence of the setting depends on the presence of any data of the specified type (see Data Registration). For instance <RequireLocal name='MyApp' name='ButtonBL' if='References'>True</RequireLocal> would result in a corresponding bottom-left button if, and only if, the person or family had program data of type 'References'. See below.
- *<RequireLib name='lib'>*: Declares that a documented library of support code is required (or not required) by the application or service. The payload must be a Boolean. The only permissible names are currently "support.js", "navigation.js" or "programdata.js". In the case of "programdata.js", this also causes the associated programdata.css file to be included, and so no explicit <IncludeCSS> is needed. NB: In order to retain backwards compatibility, the actual support files have a major-version suffix as of V6 (e.g. navigation6.js), but the name specified in this element must always be the generic un-versioned one.
- *<RequireSrv id='id'>*: Declares that the application requires (or not requires) the service entry specified by the 'id' attribute. The payload must be a Boolean. Such services are included is a manner identical to applications selected by the end-user, but just before code generation. That is, applications are selected by the end-user but services are selected by applications. They differs from the capabilities of <IncludeJS> elements in that sections of HTML and CSS may also be part of a service definition.
- *<Op id='id' name='name' key='mouse-click' on='place'>*: Specifies a piece of JavaScript to handle the specified mouse-click operation. The attribute 'name' is a short name for the functionality associated with the operation. 'Key' must be one of 'Click', 'Ctrl+Click', 'Shift+Click', or 'Alt+Click' (see "User Guide:Modifier-key Usage in Browsers"). 'On' must be one of 'box', 'image', 'circle', or a button-corner acronym (TR, TL, BR, or BL). For instance, <Op id='SelectFamily' name='Select Family' key='Shift+Click' on='circle'>. All such JavaScript will have access to the event parameters defined in the various Header Files. This is, ev

(event object), type (P or F indicating person or family), inst (instance ID), and key (symbolic name of person or family).

- *<Preamble>*: Declares an optional single block of elements to be inserted just after the <body> element (or the <svg> element if SVGFile=True). This block may utilise the substitution markers ('individual values' and 'line control', but no 'line substitution' except "help-point") defined in Header Files.
- *<Postamble>*: Declares an optional single block of elements to be inserted just before the </body> element (or the </svg> element if SVGFile=True). This block may utilise the substitution markers ('individual values' and 'line control', but no 'line substitution' except "help-point") defined in Header Files.

In selecting the appropriate <RequireSetting> and <RequireLib> elements to include, it is worth noting the following dependencies

- The support.js library is low-level and has no dependencies.
- NavData generates the navigation data: the data describing the person/family elements and their relationships.
- The navigation.js library provides helpful functions for accessing the navigation data and so it requires NavData.
- ProgData generates the application-defined program data (see tabs in Edit-Person and Edit-Family forms).
- The programdata.js library provides higher-level helpful functions for general application development, including access to any application-defined program data, although ProgData is not necessarily a requirement. It does require support.js, and both navigation.js and NavData=True. Its own programdata*.css file is included automatically by the <RequireLib> element.

Note that header settings may be requested by both the user (either through the corresponding forms, or by editing the tree definition file directly) or by the selected applications and services. Ones requested by applications and services would implicitly include header settings corresponding to any required buttons, just in case the user forgets to configure such buttons. For instance, if an operation was associated with a top-right button then a header setting of ButtonTR=True will be requested, just as though it was in an explicit <RequireSetting> element.

Any settings (local or global) requested by the applications are only set just prior to generating code (i.e. when the main 'Process' button has been pressed), which means that you may not see them selected in the corresponding settings forms or persisted in the tree definition file. If you would like to see them explicitly recorded in the tree definition file (even though it is not necessary) then you can force a Save operation after generating the output code, using the 'File->Save' option either in the main form or the Tree Designer.

The 'weak' attribute on <RequireSetting> controls whether application or end-user requirements take precedence. Normally, application ones take precedence, but weak='True' changes this. An example would be where an application requires a button: it may weakly require a tooltip or icon but a specification by the end-user in the Edit-Settings form would override that.

3.1.1 Modifiers

Entries in the standard list may be modified by including a corresponding <Modify> element with a matching 'id' in another file — no need to edit the standard file. Such modifiers only need to include the elements (or <Application> attributes) that are different. For instance, placing the following element in (say) ApplicationsServicesMods.xml, would reconfigure the 'Ancestor Links' application to use a TL button rather than a TR one, to give this button a tooltip and icon, and to give the application a modified name.

```
<Applications>
      <Modify id='AncestorLinks' name='Ancestor Linkage (mod)'>
            <RequireSetting name='ButtonTL'>True</RequireSetting>
            <RequireSetting
name='IconTL'>https://parallaxviewpoint.com/SVGcode/UpLeft-Arrow-
icon.png</RequireSetting>
            <RequireSetting name='TooltipTL'>Show ancestors for this
person</RequireSetting>

            <Op id='ShowPaths' name='Show Paths' key='Click'
on='TL'>al_showPaths(inst,key);</Op>
            <Op id='ShowBoxes' name='Show Boxes' key='Shift+Click'
on='TL'>al_showBoxes(inst,key);</Op>
            <Op id='ClearSel' name='Clear Highlights' key='Ctrl+Click'
on='TL'>al_clear(inst);</Op>
      </Modify>
</Applications>
```

Alternatively, the following modifier will add a TR button to the 'Expand Image' application, and configure it with an eye icon.

```
<Modify id='ExpandImage' name='Expand Image (mod)'>
      <RequireSetting name='ButtonTR'>True</RequireSetting>
      <RequireSetting
name='IconTR'>https://parallaxviewpoint.com/SVGcode/Eye-
icon.png</RequireSetting>
      <RequireSetting name='TooltipTR'>Expand any thumbnail
image</RequireSetting>

      <Op id='Expand B' name='Expand Image B' key='Click'
on='TR'>ei_showImage(ev,type,inst,key);</Op>
</Modify>
```

Both the normal click on an image and a click on this new TR will result in the same operation. However, in this example, the button supplements the standard click operation rather than replacing it since the 'id' is different on its <Op> element.

In general, modifiers can:

- Include additional files or header settings.
- Add additional <Op> operations, or reconfigure existing ones if 'id' is the same.
- Disable an <Op> operation by removing the payload of the element.
- Change attribute values on the original <Application> element (other than 'id').
- Define a new, or override a prior, Preamble or Postamble.

If an application requires that buttons are per-person, based on the presence of program data of a given type, then this is achieved using modifiers by first disabling the above automatic button configuration by specifying a corresponding <RequireSetting> element with a payload of False. It must then use a <RequireLocal> to request the local buttons in connection with a particular data format. This request is re-evaluated whenever the main 'Process' button is pressed so that it corresponds to the latest program data entered by the user. For instance, the following <Modify> entry would change the 'Compendia' application to only place buttons on person-boxes that actually have 'Reference' data in their associated program data:

```
<Modify id='Compendia' >
      <RequireSetting name='ButtonTL'>False</RequireSetting>
      <RequireLocal name='ButtonTL'
            if='References'>True</RequireLocal>
</Modify>
```

3.1.2 Data Registration

As well as <Application> and <Modify> elements, these files can contain <ProgramData> elements. These define a type of data that will be used by one or more applications. These elements have the following attributes:

- *id*: Symbolic name of the data. As with SVG-FTG keys and other ids in these registration files, these ids must be composed of alphanumerics, underscores, and hyphens, starting with an alphabetic character. In order to avoid a clash with any predefined ones, it is recommended that any custom ones should employ underscores and/or hyphens.
- *descr*: Longer description of the data that can be presented to a user in a menu.
- *place-keys*: Boolean indicating whether this type of data employs place-keys. If true then they are expanded to their Internet equivalent during output. The default is False.
- *alias-of*: Id of another registered data format that this one is identical to. This feature is currently used to distinguish user-defined TreeLink data from the auto-generated TreeLinkV contributions related to viewpoints.

- *hidden*: Boolean indicating that the format should be hidden from the end-user Edit menu. Default is False.

Each element may include the following sub-elements to define the data. Multiple instances of each are valid, unless stated otherwise:

- *<UsedBy id='app id'>*: Indicates which applications utilise this type of data.
- *<Item tag='tag name'>*: Defines one item on the records associated with this data. The 'tag' attribute gives it a symbolic name (to be used in code) while the payload is a meaningful title that could be displayed to an end-user.

The following example shows the 'References' data, as used by the 'Compendia' application. Each record consists of two items: a title for the reference, and a URL data link.

```
<ProgramData id='References' descr='Article or image references'>
      <UsedBy id='Compendia'/>
      <Item tag='title'>Reference Title</Item>
      <Item tag='link'>Data Link</Item>
</ProgramData>
```

3.2 support.js

There is a library of low-level utility functions at https://parallaxviewpoint.com/SVGcode/support*.js to help with common operations required by applications and services. See <RequireLib> element in Application Registration.

This file also provides support for IE 11 by adding missing and differently named methods to its prototypes.

Data or function	Notes
function su_getClassList (elem) {}	Returns a reference to the classList associated with the given person-box or family-cirlce element. If not null then this will be a DOMTokenList object, to which you can apply properties and methods such as: • .length — count of tokens • .value — normalised list as string • .item(index) • .contains(token) — test if token present • .add(token[,...]) • .remove(token[,...]) • .replace(oldToken,newToken) • .toggle(token) — add or remove token
function su_SVGFile (inst) {}	Tests whether the specified tree instance is SVG-only (i.e. the SVGFile header setting was selected), or a mix of HTML/SVG. Returns a Boolean.

function su_pz (inst) {}	Tests whether the Pan-Zoom feature was selected for the specified tree instance. Returns a Boolean.
Function su_ie () {}	If running within Internet Explorer (IE) then it returns the major version number (e.g. 11), otherwise it returns 0.
function su_showFile (file,title) {}	Show the contents of the given file in a new browser tab with the specified title. The file may be a data file (e.g. image, text, pdf) or a document file (e.g. html or svg). Note that html/svg may have a title of its own.
function su_dragDialog (idCont, idHdr,enable) {}	Support for dragging dialog boxes around the browser window. idCont is the id of the main container div, and idHdr (if specified) is the id of the header div (i.e. title bar). Specifying the latter means that you have to grab the header to move the dialog. If 'enable' is false then the dragging is disabled and the event handler removed.

3.3 NavData

A set of data may be deposited in the code that indicates how the various person-boxes and family-circles are related, such as enumerating the children of a family pair. This is deposited in JavaScript format if the option NavData=True is set (or 'Generate Navigation Data' checkbox in the Advanced-Settings form).

The individual parts of the data are as follows:

Data name	Data key	Notes
--	title	Title given to the tree. See 'Title' header setting.
nd_persons {}	persons	Maps person keys to person indexes (used in other data).
nd_families {}	families	Maps family keys to family indexes (used in other data).
nd_sex []	sex	Sex of each person, by person index. Female=0, male=1, unknown/unspecified=2, other/indeterminate=3.
nd_personalNames []	personalNames	"|"-separated list of alternative personal names (not captions) for the given person. Surnames may be enclosed in slashes, as per GEDCOM. The list is empty if none are defined for the person.
nd_parentFam []	parentFam	Index of family in which each person is a child, by person

		index. ‘No family’ is represented by -1.
nd_spouseFam []	spouseFam	Array of indexes for all families in which each person is a spouse, by person index. Each array may be empty.
nd_famParents []	famParents	Two-element array of person indexes for the parents in each family, by family index.
nd_famChildren []	famChildren	Array of person indexes for all children in each family, by family index. Each array may be empty.
nd_colours []	colours	Array of colour names or values used in the tree. Entries 0–3 are indexed by the integers from nd_sex. Entry 4 is the border colour used for all person boxes. Entries 5–8 are used in family circles: 5 = main circle, 6- separator for banded main circle (i.e. indicating the presence of 'more children'), 7 = liaison circle, 8 = separator for banded liaison circle. Entry 9 is the single component used in the background for R, G, and B (e.g. 255 if white). NB: Use ND_C_* constants defined in navigation*.js.
nd_dims []	dims	Array of numeric data. Entry 0 is the opacity (1.0 if Opaque=True). Entries 1–6 correspond to the header settings: W, H, MW, MH, BW, BH. Entries 7-9 are line widths. NB: Use ND_D_* constants defined in navigation*.js.

If your page contains multiple tree instances then these data names will be supplemented by the respective instance IDs in order to keep them distinct: nd<instance>_<name>. For instance, nd3_persons. As a convenience to hide this, a single nd_index object is provided to group each set of data names under corresponding instance ID, and allow them to be accessed by a data key (see table above). For example, nd_index[“3”].persons would yield nd3_persons — note that the actual instance ID to use is given as a parameter to each event handler.

3.4 navigation.js

There is a library of utility functions at https://parallaxviewpoint.com/SVGcode/navigation*.js to help with access to navigation data. See NavData, and also the <RequireLib> element in Application Registration.

Data or function	Notes
Conversion between access keys (pkey=person key, fkey=family key, key=either key, index=person or family index, elem=SVG element, inst/type/key as per handler calls).	

function nd_pkeyToIndex (inst,key) { }	Maps a person key to a person index.
function nd_indexToPkey (inst,index) { }	Maps a person index to a person key.
function nd_fkeyToIndex (inst,key) { }	Maps a family key to a family index.
function nd_indexToFkey (inst,index) { }	Maps a family index to a family key.
function nd_keyToElem (inst,key,type) {}	Maps a person/family key to a corresponding SVG element with the corresponding id.
function nd_indexToElem (inst,index,type) {}	A s per nd_keyToElem but by person/family index.
function nd_elemToInst (elem) {}	Returns the instance ID for the given person or family element.
function nd_elemToKey (elem) {}	Returns the key for the given person or family element.
function nd_elemToType (elem) {}	Returns the type character for the given person or family element.
function nd_elemToIndex (elem) {}	Returns the index corresponding to the given person or family element.
Data access functions (note=notes div, data=program data div, sexIndex=0-3, sexClass=f/m/u/x, caption=person caption, perCol=person-box colour).	
function nd_getNoteByKey (inst,key,type) {}	Maps a person/family key to a corresponding div element holding the “information panel” notes. This may be null if none defined.
function nd_getNoteByElem (elem) {}	As per nd_getNoteByKey but working from a person/family element.
function nd_getDataByKey (inst,key,type) {}	Maps a person/family key to a corresponding div element holding the “program data". This may be null of none defined.
function nd_getDataByElem (elem) {}	As per nd_getDataByKey but working from a person/family element.
function nd_isFamilyKey (inst,key) {}	Tests whether the given key represents a family rather than a person.
function nd_getSexClassByPkey (inst,key) {}	Returns a class name, or a root thereof, for the sex of the specified person key, Values are ‘f', 'm', 'u', or 'x'.

function nd_getSexClassByIndex (inst,index) {}	As per nd_getSexClassByPkey but by person index.
function nd_getSexIndexByIndex (inst,index) {}	As per nd_getSexClassByIndex but returning the sex index.
function nd_getSexIndexByElem (elem) {}	As per nd_getSexIndexByIndex but by element.
function nd_getCaptionByPkey (inst,key) { }	Returns the displayed caption for the specified person key. The text is plain text with no character entities or HTML numeric character references.
function nd_getCaptionByIndex (inst,index) { }	As per nd_getCaptionByPkey but by person index.
function nd_getCaptionByElem (elem) {}	As per nd_getCaptionByPkey but by person element.
function nd_description (inst,key,type) {}	Returns a description for a person (caption) or a family ("family of" plus both spouse captions)
function nd_getPersonalNamesByIndex (inst,index) { }	Returns any "|"-separated list of personal names for the given person index, or empty string.
function nd_getPersonalNamesByElem (elem) {}	As per nd_gePersonalNamesByIndex but by person element.
Function nd_getPersonalNamesByPkey (inst,key) {}	As per nd_gePersonalNamesByIndex but by person key.
function nd_getPerColByIndex (iinst,ndex) { }	Returns the colour name, or value string, used for the given person index.
function nd_getPerColByElem (elem) {}	As per nd_ getPerColByIndex but by person element.

3.5 ProgData

An application's own data may be created in the 'Program Data' tab within the Edit-Person or Edit-Family forms, and this is stored in separate invisible elements if ProgData=True (or 'Generate Program Data' checkbox in the Advanced-Settings form).

The Application Registration file(s) also define the data signature for such applications, and this facilitates both editing by the user (at design-time) and access by the application (at run-time). Although XML would often be used to represent such program data, the current version of SVG-FTG opts, instead, for a lightweight implementation that still accommodates mixing of data for different applications (i.e. using the same storage mechanism) and symbolic editing my the user.

Each line of the program data consists of a "|"-separated list of items, the first of which is the relevant data id, and the remainder being a set of related data. For instance, the program data for a given person might contain the following lines in it:

```
References | A Life Revealed | http://parallax-viewpoint.blogspot.com/2014/08/a-life-revealed.html

References | More of a Life Revealed | http://parallax-viewpoint.blogspot.com/2014/10/more-of-life-revealed.html

References | Mary Phyllis Ashbee | https://parallaxviewpoint.com/Images/Mary_Ashbee.JPG
```

The Application Registration file(s) define the names of these data items to be "title" and "link". When pd_getAppData () is called to return a subset of this data, it is provided with the data id (e.g. "References") so that it only returns relevant data; other types of data are ignored.

NB: empty ProgData elements are not generated. This means that a test must be performed to see if specific program data exists. Also, if SVGFile=True then the associated elements are enclosed in unnamed <foreignObject> elements in order to make them valid. See Notes and Program Data.

3.6 programdata.js

There is a library of utility functions at https://parallaxviewpoint.com/SVGcode/programdata*.js to help with accessing program data, and with general higher-level requirements of applications and services.

Data or function	Notes
pd_dataSignatures = {}	Signatures of the data for each application in each instance. pd_dataSignatures[inst] returns a collection of data-item names for each data id. The data-item names are in a "|"-separated list, e.g. {References: 'title|link'}.
function pd_clearPer (class) {}, function pd_clearFam (class) {}	Un-highlights all person-boxes or family-circles, as set by the corresponding pd_mark* variation (i.e. puts them back to normal display mode). The default class (if unspecified) is pd-person and pd-family, respectively.
function pd_markPer (elem,class) {}, function pd_markFam (elem,class) {}	Highlights the given person-box or family-circle element. The default class (if unspecified) is pd-person and pd-family, respectively. See below.
function pd_unmarkPer (elem,class) {}, function pd_unmarkFam (elem,class) {}	Un-highlights the given person-box or family-circle, as set by the corresponding pd_mark* variation (i.e. puts it back to normal display mode). The default class (if unspecified) is pd-person and pd-family,

	respectively.
function pd_mark (elem) {}, function pd_unmark (elem) {}, function pd_clear () {}	Convenience interfaces to the above functions that are independent of type (P/F) , but only apply to the default highlighting using classes pd-person and pd-family.
function pd_compareProgData (type,inst,key,dataId,match,options) {}	Compares the program data of this object (if any defined) with every other object of both types (P/F). DataId is relevant data id, match is a comparison function(dict1,dict2) to implement the comparison semantics between any two program-data records. options.Mark is an optional function to highlight matching elements, default is pd_mark. options.Clear is an optional function to remove current highlights, default is pd_clear.
function pd_scrollIntoView (pElem) {}	If the specified person-box is currently out of view (see below) then it is scrolled back into view, just inside the relevant border(s).
function pd_getAppData (data,dataId) {}	Extracts the records from the given program-data element that relate to the given data id. It builds a dictionary of data-items for each such record, and returns an array of all such dictionaries. The returned array may be zero-length.

By default, pd_markPer() highlights a person-box by thickening the border and changing its colour, dependent on the associated sex, and pd_markFam thickens the border of a family-circle and changes it to black. For person-boxes, an explicit class of 'pd-personBox' may be specified to change the fill colour. The default classes are defined in programdata*.css, but in all cases, an application-defined or programmer-defined class can be specified and defined in an appropriate CSS file..

The pd_compareProgData() function will invoke a caller-defined comparison between each item of program data (having the specified id) for every person/family, For instance, the following code will look at the 'References' data (as used by the 'Compendia' application) and highlight all persons/families having a reference to the same URL link (e.g. appearing in the same article or in the same image):

```
function matchDict (dict1,dict2) {
    return (dict1.link == dict2.link);
}
... etc ...
pd_compareProgData (type, inst, key, 'References', matchDict, {});
```

The function pd_scrollIntoView only scrolls the SVG area rather than the whole page. It works in two very different cases: when the Pan-Zoom code is handling tree-specific panning and rescaling, and

when the tree content is panned using normal browser scrollbars (or even when overflowed parts of a tree have simply been clipped). In both cases, it looks to see which border(s) the specified person-box is beyond and bring it back just inside those borders.

3.7 Notes and Program Data

Notes can employ the place-key feature as a short-hand for URLs, and these are expanded to their Internet equivalent during HTML/SVG output. For program data, it depends on the type of the data, and the 'place-key' attribute on the <ProgramData> element (see Data Registration).

In HTML mode (SVGFile=False), both notes data and program data are stored in areas, each with an 'id' attribute derived from the relevant key (see Use of the id Attribute). In SVG mode, however, because SVG is an XML-based language (just like XHTML) then every tag must be closed (e.g. ...), or be self-closing (e.g.
). In order to prevent this being a burden on the end-user, such content is wrapped in CDATA sections which characterise it as unparsed data, and so avoiding any conflicts. The corresponding elements are also wrapped in SVG <foreignObject> sections in order to make them palatable.

For instance:

```
<foreignObject>
<div class="tp-male" id="P_WilliamProctor"
    xmlns="http://www.w3.org/1999/xhtml"><![CDATA[
Mary Phyllis Ashbee was born 28 May 1905 in Bradford, Yorkshire, and died
13 Jun 1984 while on holiday in Switzerland. She never married.
<br><br><b>Articles and mentions:</b>
<br>
<a href="http://parallax-viewpoint.blogspot.com/2014/08/a-life-
revealed.html" target="_blank">A Life Revealed</a><br>
<a href="http://parallax-viewpoint.blogspot.com/2014/10/more-of-life-
revealed.html" target="_blank">More of a Life Revealed</a>
<br>
<img border="0" height="400"
    src="https://parallaxviewpoint.com/Images/Mary_Ashbee.JPG">
]]></div>
</foreignObject>
```

A consequence of this is the handling of the content. Both cases can be located by passing the id to getElementByID(), but whether you look at the innerHTML or innerText property depends on whether SVGFile=True for that tree instance (see pd_SVGFile() test). An example case exists in expandnotes6.js.

NB: Microsoft Internet Explorer (IE) does not support the <foreignObject> SVG element. What this means is that it may still be used as a container (the syntax is accepted by IE) but it is not a cooperating object as far as the UI is concerned, and you cannot make the content visible without some script code (as in the 'Expand Notes' application).

NB: No program-data element is created for a person or family that has none associated with it. A default notes-data used to be created for a person or family that has none associated with it (incorporating a fixed message), but this is no longer true as of V6.0.

3.8 Header Files

A standard header file will be selected from the code directory of SVG-FTG and included in order to seed the output file. These contain place-markers for SVG-FTG to substitute statements, code, and values dependent upon the options that you selected. The standard files are:

Mode	Filename	Notes
HTML	Header_pz.html	Selected if Pan-Zoom is enabled.
	Header.html	The default in other circumstances HTML.
SVG	Header_js.svg	Selected if compatible applications or services have been selected
	Header.svg	The default in other circumstances SVG.

When developing your own application then you may need to work using a variant of one of these. The name of your custom file can be specified using the Header= file setting in the header records of the tree definition file, or specified in the Advanced-Settings form of the Tree Designer, but a properly registered application would normally declare any required header settings so that they're selected automatically. See Application Registration.

Each place-marker is preceded-by and follow-by a double question mark, e.g. "??marker??. Although most of the place-markers will not be of interest during your development, here is a complete list of them:

Name	**Notes**
Individual value substitutions	
box-col-border	Colour of borders for boxes.
box-col-f, box-col-m, box-col-u, box-col-x	Colours of boxes for each sex setting.
box-opacity	Box opacity (e.g. 0.6), or 1.0 if Opaque=True.
bw, bh	Button width and height (pixels implied).
circle-col-c, circle-col-c2, circle-col-cl, circle-col-cl2	Colours of family-circle variations: C (main circle), C2 (spacer for banded main circle), CL (main liaison circle), CL2 (spacer for banded CL circle).

clip-tr, clip-tl, clip-br, clip-bl	Clip path for button icons.
description	For an output file representing a full tree, this is currently the same as the Title= header setting, preceded by "SVG Family Tree". For a set of output files representing viewpoints, it is the name (not the key) of the viewpoint for the current file, and with no additional text.
font	"10px" if Small=True, else "13px".
image-tr, image-tl, image-br, image-bl	Icon path for the 4 button corners (if any).
inst	The current Inst= header setting.
line-wid, line-wid-sh, line-wid-root,	Pixel width (respectively) of normal lines, shape borders, and the blue "direct" lines from any RootKey.
r1, r2, r3	Radii of the concentric family circles when there are unshown children (pixels implied). r1 is the outer circle.
root-key	The key of any "root key" person (see RootKey header setting), or empty.
rx, ry	Radii of rounded buttons (pixels implied), or 0 if not rounded.
title	The current Title= header setting.
version	Version string, e.g. "V6.0.0".
viewbox	viewBox attribute for SVG element, or empty.
viewpoint	Key name of the current viewpoint, or empty if none.
viewpoint-set	Comma-separated list of all the viewpoint keys for the set of files generated with the current one.
w, h	Box width and height (pixels implied).
width, length	Full width and height of the tree, in units determined by SVG-FTG.
width-div, height-div	Full width and height of the container div (HTML

	mode), in units determined by SVG-FTG.
width-max, height-max	Maximum tree dimensions. May leave room for scrollbars and so would be greater than normal tree width/height.
Line control	
if-blog, ifn-blog	Conditional support to enable or disable a whole depending on the Blog header setting. 'if-' will eliminate current line if Blog=True, and 'ifn-' will eliminate it if Blog=False. See below
If-scaled, ifn-scaled	As per '-blog' markers but dependent upon the Scaled header setting. See below.
if-svg, ifn-svg	As per '-blog' markers but dependent upon the SVGFile header setting. See below.
Line substitutions	
application-preamble	Place to insert any "preamble" code from selected applications and services.
bg-image-point-1, bg-image-point-2	Deal with any background image. First is the object specifying the image file, opacity, etc., and the second is the frame in which it is used.
handler-point	Place to insert JavaScript statements handling the click operations on person-boxes, images, or family-circles required to supported selected applications and services.
handler-point-tr, -tl, -br, -bl	Place to insert JavaScript statements handling the click operations on person-box buttons required to supported selected applications and services.
help-point	Deposits brief HTML-formatted help instructions for which mouse and modifier-key combinations are configured for the currently selected interactive applications. NB: this information is specific to each tree instance, and it may be a consideration if you have several on the same page.
script-point	Place to insert <script> elements that nominate

	other script files.
stock-image-point	Place to include SVG objects representing stock images for each sex.
style-point	Place to include stylesheets.

Note that conditionals, such as ??if-blog??, must be at the start of a line, although several similar conditional can be placed in sequence. For instance, a line beginning "??ifn-svg????if-blog??..." will be deposited only if SVGFile=False and Blog=True.

3.9 Services

Services are similar to applications but offer interactive support to applications.

3.9.1 Message Box

The 'Message Box' service presents a message to the user in conjunction with one button to confirm they've seen it, or two buttons to confirm a response. Being a 'service' then an application must declare a dependency on the service (using <RequireSrv>) before the application code can call on it.

When SVGFile=True (or the 'Basic' option is true) then it uses the standard JavaScript functions alert() and confirm(), otherwise it provides something a bit more pretty and functional.

The main code entry point is:

```
mb_show (title, message, callback, options)
```

It executes asynchronously and so does not return a value directly. The 'title' heads the dialog, and defaults to simply "Message". The 'message' is the message intended for the user. It may include line-breaks using the escape sequence '\n'. The 'callback' is a function to be called when the user has confirmed or dismissed the message-box dialog. It must accept an integer that indicates which button the end-user pressed (i.e. 1 or 2).The 'options' parameter is a collection of specifications that the programmer can use to control the message box.

- Basic: Boolean indicating whether the fancy or the basic edition of the dialog is to be used. Only the basic edition is available if SVGFile=True.
- Button1: Specifies an alternative title for the main button. The default is "OK". Ignored in basic mode.
- Button2: Specifies a title for the optional second button. If not specified then no second button is used. The text is ignored in basic mode, but the presence of this option still controls whether two buttons are available.
- Fixed: Boolean specifying that the message-box dialog is to stay put, in the centre of the screen area, and not be draggable. The default is false. Ignored in basic mode.

- Icon: URL of an optional icon to place in the message-box dialog, possibly indicating the severity of the message. No icon is shown if no URL is specified. Ignored in basic mode. See below.
- Screen: A count in seconds for a backdrop screen to obscure the other elements, and so focus the attention of the end-user on the message. If unspecified then no screen is displayed. Ignored in basic mode.

For example:

```
mb_show ("Error message", "There are no notes for this user",
    function(iBut){}, {
    Button1: "Ok",
    Icon: 'https://parallaxviewpoint.com/SVGcode/Error-Dialog.png',
    Screen: 5
} );


mb_show ("Confirm", "Should the person be deleted?",
    function(iBut) {if (iBut == 1) doDelete();}, {
    Button1: "Ok",
    Button2: "Cancel",
    Basic: false,
    Fixed: false,
    Icon: 'https://parallaxviewpoint.com/SVGcode/Question-Dialog.png'
} );
```

You can use any icon file you like, but a number of convenient ones are available in the http:parallaxviewpoint.com/SVGcode folder:

Information-Dialog.png

Error-Dialog.png

Warning-Dialog.png

Question-Dialog.png

3.9.2 Choose

The 'Choose' service allows an end-user to select an item from a list. When SVGFile=True (or the 'Basic' option is true) then it uses the standard JavaScript function prompt() to achieve something approaching the same functionality, otherwise it provides something a bit more pretty and functional.

The main code entry point is:

```
ch_choose (titles, ids, callback, options)
```

where the parameters are defined as follows:

- 'Titles' is an array of item titles to present to the user. The array must not be null or empty.
- 'Ids' is an array of corresponding identifiers or objects that the code can utilise on return. The array must be the same size as 'titles'.
- 'Callback' is a function that will be notified of the user's selection. It will be given the title and corresponding id of the actual selection, and both will be null if the user cancelled the dialog.
- 'Options' is a collection of specifications that the programmer can use to control the choose dialog. These include:
    - Basic: Boolean indicating whether the fancy or the basic edition of the dialog is to be used. Only the basic edition is available if SVGFile=True.
    - TakeOne: Boolean indicating whether the dialog should automatically take an entry without prompting if there's only one. The default is false (i.e. always prompt).

For example:

```
ch_choose ( ['Item 1', 'Item 2', 'Item 3'],
      [id1, id2, id3],
      function (title,id) { alert ( "You selected " + title +
```

```
                " (" + id + ")"); }, {}
    );
```

3.10 z-order

Various applications and services specify an explicit z-order for some of their components, thus ensuring which are above others. The following values are used:

- 50: Pick-list for 'Compendia' application and the 'Choose' service.
- 60: Search box for 'Find Persons' application.
- 70: Optional screen for 'Message Box' service.
- 80: Dialog for 'Message Box' service.

4 Using SVG in the Real World

Although SVG is nearly 20 years old, support for it by many websites can be problematic, even though most Web browsers now render it properly. This section will visit a few of the more popular sites and make some notes that may be helpful.

4.1 Browsers and Windows

SVG-FTG is designed to run under Windows, but has been run on other platforms using virtual machines and with WINE-based compatibility layers: specifically PlayOnMac and CrossOver. It has also been run using multiple monitors, and in different locales. It was originally developed under Windows XP, but is now only tested against Windows 7 and Windows 10.

NB: When dragging a person-box in the Tree Designer, the cursor should change to a combined rectangle and arrow to indicate that a drag-and-drop operation is in progress, known as a 'Drag Icon'. This does not show when running with a WINE-based compatibility layer, which would leave the end-user with no visual indication of what's happening. If the '-I' command line is specified then it will force the Tree Designer to use an alternative private 'Drag Icon'. See "User Guide:Command-line and Shortcuts". Interestingly, the absence of the 'Drag Icon' also affects the recording software used to make the instructional videos that are available in the Facebook group.

The output from SVG-FTG is designed for viewing on a normal computer screen. Although it has been tried on hand-held devices (including phones), the size of the family trees makes them hard to view, and the processing of SVG may be noticeably slower on such devices. The main browsers that have been used during development and testing are Firefox, Chrome, and Edge. No two of these behave exactly the same , and it is not easy to get the output from SVG-FTG to behave consistently in all of them, but effort has been expended in making this happen. Internet Explorer (IE) has a reputation for being very different and non-standard, to the point where some developers just ignore it. IE tends to be used more in corporate scenarios where a company has committed to Microsoft, but (for now at least) effort has been expended to make IE 11 a usable option, although note the restriction documented at Notes and Program Data.

As of SVG-FTG V6.0, changes had to be made to the generation of the SVG output that could, in some circumstances, take more time than it did before. The essence of the problem is a deficiency in SVG that requires the dimensions of images to be known before they can be centred in a person-box. The output already uses the `preserveAspectRatio="xMidYMin meet"` attribute, but the interpretation of this is browser-dependent. When the image is scaled and centred, in accordance with this attribute, it usually entails some padding either side of the image. Firefox doesn't interpret that as part of the image and so a click in the padding is effectively a click on the box. The other browsers interpret it as part of the image, and so a click appears (to the code) to be on an image even when the user didn't click on the visible image. The latter appears to be closer to the SVG specification, but this is fundamentally flawed. It has been necessary, therefore, for SVG-FTG to calculate the aspect ratio of each image beforehand and avoid this padding area. If all images have a local copy then this is fine, but if it has to look at remote copies (on the Internet) then there will be a performance implication when you press the 'Process' button.

4.2 Security

SVG gets bad press for being insecure, but it's not quite as simple as that. SVG files are actually "document files" rather than "image files" as they can contain CSS, script, HTML (inside foreignObject elements), and be represented using a DOM. Unfortunately, that makes "SVG files" (not SVG in general) insecure because you could have malicious script or other content in one, and then have it deployed as a CSS background image, or a logo, or in some image gallery. Treating such files as simple images is where the insecurity really comes from.

SVG-FTG uses HTML and inline SVG together to create responsive graphical UI, and in this guise there is no real difference between HTML and SVG: they can both contain script and CSS., and it's hard these days to imagine a responsive site that does not make use of these technologies.

In view of these acknowledged issues with "SVG files", where they're treated as simple images, then we will not focus much on being able to upload them to some site. Instead, we will look at the more realistic cases where someone needs to upload the combination of HTML, SVG, CSS, and JavaScript to some page.

4.3 neocities.org

This site provides free websites, but the tools are limited because it is aimed more at people with HTML experience who can do it all themselves. However, a nice feature is that is supports clean-and-simple file hosting, meaning that you can put images in different folders and reference each folder by a simple URL. For instance, I have a website at parallaxview.neocities.org that has a subfolder called Images containing the images used in my sample trees. For this, I can use a place-key mapping of:

MySite = https://parallaxview.neocities.org/

MyGallery = $MySite:Images/

A subsequent image reference might be $MyGallery:Mary_Ashbee.JPG, which then expands to https://parallaxview.neocities.org/Mary_Ashbee.JPG.

Neocities.org is very accommodating, and will accept the normal HTML/SVG output from SVG-FTG, or the pure SVG format mentioned above. If you need to create other content, including formatted text and images, then you will need to use a separate tool. Even Microsoft Word can be used in many cases and there's a mention of this in the instructional video on 'Notes'.

The main failing of this site is if you have a custom domain name (i.e. a special URL that you want to use), then you will then need to be a subscriber, with a continuous payment, but the overall cost is very low. You're also limited to the optional sub-domain of "www" (e.g. www.mydomain.com); there is no facility to define others. There are also issues when mixing http/https on the same domain, but this is unlikely to affect all but the most advanced users.

4.4 Dropbox.com

You can also host files in Dropbox, but their shared links are a little messy. I previously wrote about this at Using Documents or Script with Blogger.

There are two types of sharing supported by Dropbox: you can share a file or folder with specific people once you identify them by their email address, or you can create a shared link for a file or folder. The first scheme is by invitation only (adding someone to the list of sharers causes an email invitation to be sent to them), and it allows you to control the type of access they're granted. The second scheme shares the file or folder with anyone who has a copy of the link, and only for read access (they cannot change anything).

Unfortunately, the links it gives you have to be modified if you want your browser to have control over how they're accessed, rather than letting Dropbox "present" them.

Luckily, this is quite easy. A typical Dropbox shared link for a file will look as follows:

https://www.dropbox.com/s/m4r9mnynnvhb46p/calendar.jpg?dl=0

What you need to do is change the "www" to "dl", and then remove the "?dl=0" parameter, resulting in something like:

https://dl.dropbox.com/s/m4r9mnynnvhb46p/calendar.jpg

If you share a folder in this way, and obtain a shared link for the whole folder, then you cannot simply append an image file name, which then makes place-key mappings a little difficult. For instance, you might have been given a folder link of:

https://www.dropbox.com/sh/p05mzbuu4sbd6ty/AABHIAu4mzjCliq2UuhL6EkFa

but if you right-click on any images within the folder and ask for their specific links, you might see something like:

https://www.dropbox.com/sh/p05mzbuu4sbd6ty/AABiU1mJgZVbC9vDGgFrP1X9a/GGFather.JPG?dl=0

https://www.dropbox.com/sh/p05mzbuu4sbd6ty/AABrVhmBAdWKCTA8KHJ7w1hqa/GGMother.JPG?dl=0

You can see that the file links are not the result of concatenating the folder link and file names; there's an extra somewhat-messy term in between. This means that each image requires a specific place-key of its own.

So, you can use it to host files, but they deliberately disabled it being used to host websites some years ago. Hence, it is not possible to upload a family tree generated by SVG-FTG and share it with friends and family.

4.5 Blogger

Blogger doesn't get upset with the HTML contents of your page, and it will accept the inline SVG, CSS, and JavaScript generated by SVG-FTG. If you're not uploading a complete HTML/SVG output from SVG-FTG (e.g. you want to embed a tree is a normal blog-post) then their HTML editor allows this. As well as trees, it is possible to include formatted text, other script, and interactive Google Maps, as described at Summarised Blogger Tips.

When you insert an image into a Blogger blog-post then it won't allow a *.svg file to be selected, but this is to be unexpected because of the security issues. You can generate an SVG file (rather than an HTML one), host it somewhere that will give you an equivalent URL (such as neocities.org), insert a dummy picture into your blog-post, and then edit the HTML to replace that dummy picture with your SVG. The picture will be using an HTML <img> element, but the SVG file is best linked using an <embed> element since it ensures that browser tooltips still work; the <img> would still be accepted but no interaction would be possible with the SVG.

If you want to insert HTML output that includes inline SVG, as recommended here, or if you want to include multiple trees in the same page, as in Boots Made For Walking, then it's advisable to combine all your <script> and <style> elements near the top of the page.

4.6 WordPress

There are two WordPress sites: WordPress.org (self-hosted and unrestricted) and WordPress.com (hosted for you). Most blogs and personal websites will be using the latter.

WordPress.org is the more forgiving one, and will allow you to copy an HTML/SVG output from SVG-FTG into your page. The following instructions apply to any theme.

Ensure that you are running a version of Wordpress later than v5.0 (v5.7 is the current at the time of writing this). Ensure that you are editing the page using the Block Editor, and not the Classic Editor.

- Open your Treexxx.html file in a plain text editor such as Notepad and copy all the code.

- In the page, select the custom HTML Block.
- Paste the content from the html file into the custom block.
- Remove the <html>, <head>, and <body> tags, but not the contents of their elements.
- Give the page a title.
- Save your page.

WordPress.com has a number of different plans upon on which the supported features will depend: Free, Personal, Premium, and Business.

Features that are normally disabled:

- <script> elements containing JavaScript code. Apparently, onclick events are available in the Business plan, but I suspect (unproven) then any JavaScript would have to be inline on the associated attribute.

- <style> elements with CSS presentation control. Interestingly, the same CSS can be used inline, on 'style=' attributes.

- <embed> and <object> elements. The <img> element does render the SVG correctly, but then you lose any interaction with the SVG elements, including browser tooltips.

Note that the <svg> element is not mentioned at all on their page: https://wordpress.com/support/code/, suggesting that its viability could be in some doubt.

However, Wordpress.com demonstrated to me that it is actually possible to host these SVG-FTG applications on their sites under the Business Plan, and that the process is not too different from the Wordpress.org one given above. In their words: "It is also a simple process of copy and pasting. Although, I added all the CSS and JS to the head of the site as on your example URL and only loaded the content in the body via the Custom HTML block".

4.7 Adobe Spark

Adobe Spark is a tool for creating pictorial content for Web pages and blogs. There are various types of project you can create: Video, Page, or Post, and I have previously used a feature known as a Page "glideshow" to good effect at A Box of Peaches: Picture Book.

One problem with these projects is that the content is not entirely representable in HTML, and so they cannot be hosted anywhere else but on their servers.

So is there a case for using SVG images in such content? Glideshows take a number of images and roll them together so that the transitions from image-to-image (and caption-to-caption) are

smoothed. As such, there is no room for interacting with any particular component image, other than the browser tooltips available on SVG elements.

Having tried this (see https://spark.adobe.com/page/FKOqepFrE7OWc/), it seems that images do not appear in the boxes, and very occasionally neither do the captions. Also, the tooltips do not appear. From that perspective, incorporating them is inferior to simply generating a screen snapshot of the SVG, as rendered by your browser, and incorporating a static JPG or PNG edition.

At the time of writing, Adobe Spark support has not responded to my report of this issue.

4.8 Google Sites

Google Sites (classic) was always a little different, if not "clunky", but the newer Google Sites is no good at all for hosting SVG-based applications. They have made a policy decision to not offer an HTML editor, and yet their graphical design tool has many missing features, even for basic text.