Graphviz plugins

Plugins provide a means of flexibly extending the graphviz tools: **dot**, **neato**, etc.

The objectives for graphviz plugins are to:
- Support independent compilation so that features can be provided by third-parties.
- Support independent packaging to that features can be installed at the user's discretion, at a later time than the base package.
- Support optional platform-specific functionality without adding complexity or dependencies to the base package (e.g. renderers for a specific windowing system).
- Unbundle some lesser-used functionality from dot into plugins **s**o that the core of dot can be smaller and faster.
- Unbundle some older functionality from dot into plugins so that the facilities are still available, but optionally and not in the core. For example the use of libgd.

Some constraints considered in there design were:
- To minimize any performance penalty from the use of plugins.
- To make the use of plugins transparent to the user and sys admin.
- To permit an option to link plugins statically at build time.
- To permit an option to always load plugins at run-time, not waiting for demand.
- To continue to support the old-style codegens, at least until completely replaced by plugins.

Terminology

A **plugin** provides a facility like a PNG renderer or an Xlib window driver, Each **plugin** implements a **plugin-api**.

A **plugin-api** is the exposed interface between the graphviz core and the **plugin**.
The set of apis is described in *lib/gvc/gvplugin.h* and each api is individually described in, e.g. *lib/gvc/gvplugin_render.h*. The set of apis currently consists of:

render:	output formats such as: png, svg, ps, ...
layout:	layout engines such as: dot, neato, circo, ...
textlayout:	code that estimates the size of a block of text. Uses pango if available.
device:	output devices, independent of format, e.g. Xlib, gtk.
loadimage:	code that loads an image in one format and makes it available in another format.

A **plugin-library** is an arbitrary collection of **plugins**, for one or more **plugin-apis**, that are loaded all together when any one of the contained **plugins** are demanded. The contents of a **plugin-library** are typically determined by other resources that are common to the set of **plugins**. For example, all the code that uses *libgd.so* is in the “gd” plugin.

A **demand-loaded plugin** is one that is not loaded until its need for use on the has been determined from the current usage.

A **builtin-plugin** is one that is available at program start, whether or not it will be needed for the current usage. **Builtin-plugins** can still be in the form of shared libraries, or they can be statically linked

A **statically-linked builtin-plugin** is one that is linked to the binary image at build time, rather than from a shared library at run-time.

Mechanism

The contents of a **plugin-library** are self-describing. The **plugin-libraries** are named like: *libgvplugin_core.so* where the "*core*" of the library name matches the name of the single exposed symbol: *gvplugin_core_LTX_library*. This symbol is a pointer to a table of **plugin-apis** supported in this **plugin-library**, and for each **plugin-api** the list of **plugins** for that api. Each **plugin** provides a "quality" value which is used to control the default selection when there are multiple **plugins** of the same type loaded.

At installation time (with the file write privileges of the installer) "**dot -c**" is run to search for and verify loadability of all available **plugin-libraries**, and then to generate */usr/lib/graphviz/config* which records their capabilities. This is a human-readable text file. When **dot** is run by a regular user to process a graph, only this */usr/lib/graphviz/config* file is loaded at first so that the set of available **plugins** and their capabilities is known and the **plugin** can be loaded if needed.

When a **plugin** is demanded, the **plugin-library** containing that **plugin** is loaded in its entirety and all of the **plugins** it provides are recorded as loaded.

Running dot with plugins

Dot attempts to be transparent to the user about its use of **plugins**. By default it will always choose the highest "quality" **plugin** listed in */usr/lib/graphviz/config*.

Sometimes it may be necessary to override the default. This can be done for renderers by indicating the plugin-library containing the plugin: e.g. -Tpng:gd

To allow the user to determine what renderers are available, the user can ask for any invalid plugin and dot will return the set available **plugins**:

```
$ dot -Txxx
Renderer type: "xxx" not recognized. Use one of: canon cmap cmapx cmapx_np dia dot fig gd gd2 gif gtk
hpgl imap imap_np ismap jpeg jpg mif mp pcl pdf pic plain plain-ext png ps ps2 svg svgz vml vmlz vrml vtx
wbmp xdot xlib
```

Or, if the **plugin-library** is not recognized then dot will return the set of alternatives:

```
$ dot -Tpng:
Renderer type: "png:" not recognized. Use one of: png:cairo png:gd
```

The order of the alternatives corresponds to their quality ranking, so -Tpng:cairo is the default.

Also, **dot -v** will identify all available **plugins** for all **plugin-apis**:

```
$ dot -v
The plugin configuration file:
    /usr/lib/graphviz/config
        was successfully loaded.
  render      :  canon cmap cmapx cmapx_np dia dot fig gd gd2 gif gtk hpgl imap imap_np
                 ismap jpeg jpg mif mp pcl pdf pic plain plain-ext png ps ps2 svg svgz vml
                 vmlz vrml vtx wbmp xdot xlib
  layout      :  circo dot fdp neato nop nop1 nop2 twopi
  textlayout  :  textlayout
  device      :  gtk xlib
  loadimage   :  2ps gif2fig gif2gd gif2ps gif2svg gif2vrml gif2xdot jpeg2fig jpeg2gd jpeg2ps
                 jpeg2svg jpeg2vrml jpeg2xdot png2cairo png2fig png2gd png2ps png2svg
                 png2vrml png2xdot ps2ps
```

Installing plugins

On package managed systems using .rpm or .deb or similar, the details of the installation process are handled automatically.

The key point about plugin-library installation or removal is that the */usr/lib/graphviz/config* file must be kept up to date by using "**dot -c**".

Builtins and static linking

A **plugin** is **builtin** if it is linked to the executable at startup.

Dot can be configured and built with an arbitrary combination of **builtin** and **demand loaded plugins,** however p**lugins** that are to be **builtin** must be explicitly listed and linked at build time.
This is done by providing a source file like *lib/gvc/dot_builtins.c* that is linked to the executable, and adding the matching -lgvplugin_xxx to satisfy the symbol reference.

To control whether demand loading is available at all, link with *lib/gvc/demand_loading.c* or *lib/gvc/no_demand_loading.c*

For the common case of maximal use of **demand-loaded plugins** the *libgvc.so* library is already linked with *lib/gvc/demand_loading.c* and *lib/gvc/dot_builtins.c*. To provide you own choice of **builtins**, and **demand-loading** use *libgvc_builtins.so* and your own versions of *builtins.c* and *demand_loading.c*.

Static linking will link all graphviz libraries, and any **plugins-libraries** that are **builtin**, into a single binary. Note that system libraries are not statically linked to the image.

See *cmd/dot/Makefile.am* for the technique used to build **dot_static**

ToDo

- "file" and "compressed file" output should be implemented as device plugins.
- Implement a mechanism for overriding **plugins** for other **plugin-apis** from the command line.
- Provide a new **plugin-api** and **plugins** for input-parsing – add support for e.g. GXL input.

Author

John Ellson - August 10, 2007