Using and Managing GDM

Jiri (George) Lebl

Abstract

GDM does many things, however it contains almost no documentation. This talk (and paper) is an attempt to correct this situation. GDM was written as a simpler alternative to XDM and is in fact a complete rewrite and does not use any XDM code. It is simpler in places and more advanced in others, so it may not perhaps replace all the applications of XDM, but it aspires to be good enough to replace most, while having some other neat features that XDM doesn't have and overall being more user friendly. This paper will guide users and sysadmins through the design of GDM and the more advanced setups, such as running multiple servers, customizing the login, setting up XDMCP (with a complete explanation of how XDMCP works), running X terminal labs with GDM, communicating with GDM using the socket/pipe protocol. Also tips and solutions for problems in setups that users have encountered are interspersed throughout the discussion. Future planned enhancements are also given some attention.

1 Introduction

GDM, the GNOME Display Manager, is the login manager that is by now familiar to most GNOME users. It does a lot more than logging you in, it starts the X server, manages remote connections using XDMCP, allows users to shut down, reboot or suspend the machine and basically has to do all the behind the scenes stuff so that GNOME (or any other desktop) has an X server to start on. This is basically what the XDM software does, however GDM is a complete rewrite and shares no code with XDM. The KDE Display Manager on the other hand is basically XDM retrofitted with a KDE style login box and setup program.

GDM was originally written by Martin K. Petersen, but he has since stopped maintaining it. He has for a while worked on GDM3, which was to be a cleaner rewrite of GDM2, but this rewrite never happened. GDM2 is really the second version of GDM and this is the GDM currently in use and the one I am describing. Around the spring of 2001, GNOME 2.0 was around the corner (or it seemed more around the corner than it actually was), and we didn't have a working display manager. The problem with GDM2 was that it was not finished and was basically not working at all. Various distributions had their own very large patches which made it work in various ways, but mostly broke it in various different ways. So I started applying the distribution patches and later took over maintaining GDM2 and it became my pet project to add features and easter eggs to (of course the panel still has the best easter eggs, and if you haven't played "gegls from outer space", you have not lived). Lately I have however concentrated a lot more on school and actually finishing a degree and perhaps one day becoming a productive member of society. If you can be productive with a pure mathematics degree is another question.

2 GDM Design

The basic design of GDM stresses security, modularity, robustness and user friendliness much more than XDM. GDM runs as a series of different processes, with different privileges. The main GDM daemon process is the behind the scenes puppet master. It doesn't by itself manage any sessions, however it has information about all the other processes. This master process is run as asynchronously and should never run operations which could cause it to hang even for a short while. It also handles communicating using the XDMCP protocol and using the local socket protocol, but it delegates all the the work of starting and managing an X server and a session to a slave process. A separate slave process is started for each X server. This slave process can run synchronously, thus can hang when for example waiting on the X server, this makes the design a lot simpler, but is also the source of many design headaches and thus this may be changed later. After a session ends, the slave has two options. It either resets the X server and puts up another login box, or it kills the X server and lets the master process run another slave for this display. The latter is now the default setup and should be preferred because of security issues even though it

is a little *dirtier* (there will be a short flicker between sessions). The former is still left as a possibility since some X servers are very buggy tend to lock the machine on such a complete restart. The slave sends all pids about its subprocesses to the master and so if the slave crashes, the master can clean up and try again. Of course crashing is considered very bad form, but GDM can handle crashes somewhat cleanly, and perhaps let the user log in. For a picture of how this all looks see Figure 1.

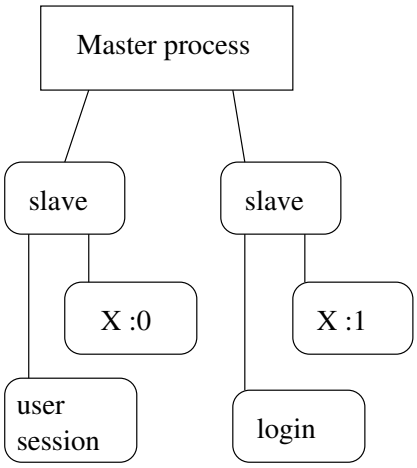

Figure 1: Structure of GDM Processes.

In the XDM design, the login dialog runs with the same privileges as XDM itself, that is `root`. So a potential security hole in the GUI could compromise the system. GDM on the other hand sets up an unprivileged user and the login dialog is run as this user. It can only communicate with the slave that started it using it's standard input and output. The basic design is that the slave just gives it a series of questions and the login dialogs asks these questions of the user. When the user selects some action which is not an answer to a question, such as when the user selects the configurator, the login dialog can interrupt the question. However it is important to understand that the login dialog has very little information about the user and has almost no privileges. This can sometimes cause problems in the design, for example with the face browser. The problem here is that the pictures are only accessible to root since they are in users directories. So the slave has to read in the .png file and send it over the standard input to the greeter. But despite these problems this is a much safer and much more secure design than XDM. The login dialog is really in the position of the user, it can only answer questions, so it would be very tough to exploit. Even if the user would somehow, perhaps through some bug in GTK+, run perhaps a shell as the GDM user, he could not do much damage beyond perhaps shutting down GDM or making GDM unusable, but he could do no worse to the system itself. Another example of why this is safer are for example the Shutdown, Reboot Suspend and Configure menus. Usually you would not want these on remote connections, however perhaps there might be a bug (or an exploit) and the menu item would show up and a remote user might think he could shut down the server. But all the login dialog can do is relay this information to the master process and then the master process sees if this display is even allowed to do a shutdown.

GDM manages three different types of displays (X servers). The first type is the static X server which comes from the configuration file. Usually this is the X server set up for the :0 display slot, and it will always run. You can run several different X servers this way if you want. The second type of display is the flexible (or flexi for short) X server. This X server is run on request by running the program `gdmflexiserver` or by clicking on the appropriate icon that is installed with GDM. The flexi server lets a user log in once and then quits. This is for example useful if you are logged in as user A, and user B wants to log in quickly but you don't wish to log out. The X server takes care of the virtual terminal switching so it works quite transparently in fact. There is also a flexi server as an Xnest, that is an X server in a window. This is requested by running `gdmflexiserver --xnest` or again clicking on the appropriate icon. The last display type is an XDMCP connection. In this case there really isn't any local X server run. The X server runs on some remote machine and just requests that it be managed for one session by the local GDM.

All the configuration for GDM is stored in a file called `gdm.conf` and usually this is in the directory `/etc/X11/gdm`. The directory depends on the local installation of course. This is a plain text file with an ini style syntax. One of the many reasons that this is text only is that it is even possible to read from shell scripts. The standard distributed configuration file has many comments which document how to work with this file. The file isn't reread as soon as it's modified. You must tell GDM to restart for changes to take affect (see also Section 6). Alternatively GDM can reread certain keys on the fly without being restarted, but you must use the socket protocol to tell GDM about this. This is what is used by the gdmsetup program to alert GDM to

changes. You will probably want to use the command `gdm-safe-restart` which is usually installed in `/usr/sbin/`. This will restart GDM after all the users have logged out.

You will also notice that the GUI setup program does not contain all the options of the configuration file. This is partly on purpose as the GUI program is designed for the things most normal desktop users would wish to change, while many configuration options are left just in the configuration file for advanced setups only. Some more options are likely to appear in the GUI setup program, but definitely not all. Design goal here is that the graphical setup program should not allow you to completely hose your setup, to do that you should edit the text configuration file by hand.

There is also the possibility of having several different login dialogs with this setup, and in fact two are distributed with GDM. The first is the standard greeter, which uses only standard GTK+ widgets, is low-weight on network performance, and will be the one that will have the accessibility framework plugged in. This is your low fat, conservative and not very glitzy login box that does just that, log you in. On the other hand we also have the graphical greeter which is this über themable, very graphical, non-accessible, but very cool looking login "dialog" (you can't really call it a dialog since it takes over the entire screen). While GDM is by default shipped with the standard greeter turned on, people like Red Hat have made the graphical greeter default for local connection and made very cool looking login screen theme. It is theoretically possible to write third party login dialogs, but the protocol is still subject to change, and it hasn't really been designed this way.

3 Initialization Scripts

The setup of the X servers and of the user sessions is handled by scripts, so that local installations can customize their setup appropriately. The base setup directory is `/etc/X11/gdm`. Before I discuss the different sessions let's discuss the setup scripts. These are stored in the `Init/`, `PostLogin/`, `PreSession/` and `PostSession/` directories. Note that the `PostLogin/` is a very new addition which at the time of writing is only in CVS. It will be available in versions 2.4.2.x and higher. There can be several different files in each directory and the first one found according to the following rule is executed. First the display name (e.g `:0` or `remotehost:0`) is tried, then the hostname of the connecting host (useful for remote connections), then `XDMCP` is tried for all XDMCP remote connections, then `Flexi` is tried for all flexible servers and finally `Default` is tried if no other file was found.

The `Init` script is run as root when GDM sets up the login display. It is run when the greeter is started and can be used to initialize the display further, perhaps run some other programs to run on the login screen. The `PostLogin` script is run just after the user has successfully entered the login information, but before any session setup is done. This script is useful for perhaps doing some setup on the home directory, something that must be done before the user is actually logged in. The `PreSession` script is run when the user is actually being logged in and some setup has already been done, this script is for example used to register the session with `utmp/wtmp`. The `PostSession` script is run after the session ends and is used for example to unregister the session with `utmp/wtmp`.

Fairly standard environment is set for these scripts. The `PostLogin`, `PreSession` and `PostSession` scripts have the USER variable set to the user who is logging in. If this is a remote XDMCP session then REMOTE_HOST is set to the host from which the user is connecting from. HOME will be set to the user's home directory as set in the password file.

Once the user has logged in, then one of the session scripts is run. These are in `/etc/X11/gdm/Sessions/`. Either the default one (named `Default` or having a symbolic link named `default` pointing to it) or the one the user picked is run. This is normally run in a login shell so login stuff need not be read in again. The contents of this file is different depending on which session is selected, and different distributions have their own session scripts usually. This is one part of the design likely to change at some point in the future. Both GDM and KDM lack a very good session setup and it would be good if such a setup were in fact common to both. It would seem better to have one script to do the most common setup and then just run the desktop or window manager selected. However the existing setup makes it very simple to add new sessions. You just drop in a session script in the `/etc/X11/gdm/Sessions/` directory.

4 Multiple X Servers

As I mentioned before, GDM can manage a lot more than just your one console display. In can manage several local and remote connections as well. We will focus on the remote connections in Section 5, so for now let's concentrate on the local case, that is on the static and the flexible servers.

As I said before not all the configuration options are in the graphical setup program, and this is one area where you have to get your hands dirty and edit the text configuration file. First GDM has definitions of X server types. Normally you would really have only one server type, the `Standard`. This server type is defined by default as follows:

```
# Definition of the standard X server.
[server-Standard]
name=Standard server
command=/usr/X11R6/bin/X
flexible=true
```

The `name` is what you want GDM to tell the user if there is a choice of servers somewhere, and `flexible` means that this server is available as a flexible server, that is a local user may start a new login at any time by running `gdmflexiserver` (or selecting the appropriate icon from the GNOME menu). If there is more than one server defined with `flexible=true`, then the user is given a dialog with those choices upon running `gdmflexiserver`.

Now that we have some servers defined, we need to define some static servers, that is servers that run all the time, and if they ever are killed, they are respawned again. This is done in the `[servers]` section in the configuration file. This section just lists all the servers with keys being the display numbers. For example to run two `Standard` servers, one on display `:0` and one on display `:1`, you would have:

```
[servers]
0=Standard
1=Standard
```

You can also pass an extra command line argument here just by putting it after the `Standard`. You could also create a new server type for this, but that's not necessary. So for example if we want to run the standard server on `:0` but with bell volume 0, then we could do

```
[servers]
0=Standard -f 0
```

Some people have a tendency to want to put the virtual terminal argument on the server command line. This should *not* be done for Linux as GDM already takes care of that if you have `VTAllocation=true` in the `[daemon]` section of the configuration file (which is the default as shipped). You could also tell GDM which virtual terminal to start with by using `FirstVT=7` in the same section. The default is 7, which will start X servers on the first available virtual terminal from vt7 up. The X server can really do this by itself, but there are cases where it gets it wrong and you could end up with an unusable console. Again this only works on Linux currently, but of course patches are welcome to make it work on other systems as well.

5 XDMCP

As you know, X11 was designed with network in mind, and so it makes sense to be able to run a graphical session remotely on another computer. For example you may have one large server with lots of power, and thin clients which just manage the graphics and input hardware, but all the programs run on the server. Most of the time clients don't really do anything intensive, reading email and browsing the web really doesn't use up too much processing power. So as long as you have fast network, since all the graphics display must be passed through, it makes sense to have only one powerful computer to do all this.

XDMCP is the X Display Manager Control Protocol and was designed to allow client to run a login remotely on another machine. Since the terminology *client* and *server* gets all weird at this point, we'll use the following terminology. We'll use *X server* for the actual X server, we'll use *machine A* for the machine with the user at the keyboard (the machine running the X server) and we'll use *machine B* for the machine with GDM that the user is trying to connect to. For all this to work of course GDM must have XDMCP enabled by setting `Enable=true` in the `[xdmcp]` section, and this can in fact be done through the GUI setup program. The protocol uses UDP packets on port 177, but you should refrain from allowing this over the internet, because of possible DoS attacks. GDM takes great care to try to prevent these, but there is no need for this to be open on the internet anyway. GDM normally compiles with tcp wrappers support and so you could just have tcp wrappers not allow connections on port 177 from untrusted servers.

So the basically the way this works is that the user runs an X server on machine A and gives it an argument of `-query hostname`, which results in this server sending a QUERY opcode of the XDMCP protocol to GDM running on machine B. Then GDM responds and tells the X server if it is OK or not to try to connect. Then the X server works out the details with the GDM running on machine B to set up a session. After the user logs in and quits (or just quits the login dialog), the corresponding slave is killed. Usually the X server then resets and tries to connect again, so it may seem to work in the same sort of

manner as static X servers.

You may notice that I said the user on machine A runs an X server and this may not seem like something you'd want your users to have to do. You could write a script or some such to start this automatically, but the best way would be to also run GDM on machine A (here, XDMCP can be disabled) and have it manage the X server for you. However normally when GDM runs a server it will want to take it over and run a login dialog on it, so you must stop it from doing that by putting `handled=false` in the server definition. So let's define a server type `Terminal` for this

```
[server-Terminal]
name=Terminal server
command=/usr/X11R6/bin/X -terminate
flexible=false
handled=false
```

And in your `[servers]` section you would have

```
[servers]
0=Terminal -query machineB
```

The `-terminate` option is not strictly necessary here, but without it GDM will always assume a logged in state (since it can't tell what machine B's GDM is doing) and then soft restarts and other things like that won't work at all without the user zapping the server with `Ctrl-Alt-Backspace`. With `-terminate`, the X server will kill itself on every reset and give GDM a chance to catch up.

You could even have this available for flexible servers. There however you would need to add the `-query` argument in the server definition. So let's say we normally don't connect to machine B, but sometimes we wish to run a flexible login there. You could define a server called `MachineB` as follows, note the `flexible=true`. Then when you wish to connect to machine B, you run `gdmflexiserver`.

```
[server-MachineB]
name=Connection to Machine B
command=/usr/X11R6/bin/X -terminate -query machineB
flexible=true
handled=false
```

Sometimes however you have many machines to connect to and you don't want to change the setup of every thin client to update this list of machines. So what you want is run a chooser. This is handled by the INDIRECT_QUERY opcode, which corresponds to running the X server with an argument of `-indirect machineB`. Machine B must have indirect connections turned on by setting `HonorIndirect=true`. What happens here is that GDM on machine B will run a chooser application instead of a login box. This chooser application in turn sends a QUERY to the network, usually by broadcasting, though this can be changed to a hardwired list of hosts. The machines that send a WILLING opcode, meaning they are willing to accept a connection, will be listed in the chooser dialog. When the user chooses a machine to connect to, say machine C, the chooser dies and the connection closes, but machine B remembers where machine A wanted, and will remember it for about 30 seconds. Next the X server on machine A restarts and sends another INDIRECT_QUERY to machine B. By this time machine B knows where machine A wants to go and forwards the query to machine C. This design is a little bit weird, and that's because originally it was supposed to work differently and the X server itself was supposed to provide a chooser. But that's not the case (at least not with XFree86). GDM actually extends the protocol here by adding another two opcodes by which the machine C will tell machine B to forget about the choice once the connection succeeds. This way if the user picked the wrong machine, he doesn't keep getting machine C login box for the next 30 seconds.

Again you'd probably want GDM on machine A to again manage the X servers. But there is a small snag as it is not easy to use the indirect setup with flexible servers (it works just fine with static servers). The problem is that if you have `-terminate` in the command line, the flexi server will end after the choice has been made but will never run the actual login connection to machine C. You could remove `-terminate` from the command line, but in that case you would have to kill the X server with `Ctrl-Alt-Backspace`, since the local GDM has no clue when to terminate the server.

Sometimes it would be nice to run an XDMCP session in Xnest. And for this you don't really need GDM on machine A, you could just run Xnest yourself. But running X nest is a little involved and so GDM provides the `gdmXnestchooser` command. This runs Xnest with `-indirect localhost` by default. If you supply a hostname on the command line it will connect to that server instead. If you want to run a direct session then supply a `-d` argument. So to connect to machine B directly you would run `gdmXnestchooser -d machineB`. Of course this all assumes you have Xnest installed.

One thing about XDMCP that comes up quite often is that the GDM on machine B has no way of knowing that for example the user on machine A hit the reset switch. It will just think that the user is not doing anything. This is why GDM on machine

B periodically pings the X server on machine A. The interval is controlled by the `PingInterval` setting on current GDM versions and `PingIntervalSeconds` on future versions of GDM (version 2.4.2.x and later). The problem was that `PingInterval` was in minutes with a default of 5, and this is way too long. If your connectivity goes out for 5 minutes at a time, you really don't want to use X over such a connection anyway. Unfortunately this was copied from XDM which was designed in the 80's when such occurrences were common and people had more patience. A reasonable time is in fact a couple of seconds. In CVS and in new releases of GDM version 2.4.2.x and up, you will have only the `PingIntervalSeconds` with a default of 15 seconds. GDM will then try to ping every 15 seconds and if it doesn't get a response before the next time it tries to ping the X server it assumes the connection is down and shuts down the user's session. The default may still be fine tuned in the future as perhaps a ping every 15 seconds may be too taxing on the network. If you set this ping interval to 0 no pings will be done.

You can also run a machine with no local static servers just by having an empty `[servers]` section in the configuration file. In this case XDMCP must be turned on since it wouldn't really make sense to run GDM otherwise. If you don't have any static servers defined and XDMCP is off, GDM will run a single local server and report an error as it considers this a misconfiguration. Running only XDMCP and no local X servers is again useful for those big machines that serve entire labs of thin clients, but you don't want a local X server taking up memory since perhaps it's never used.

6 Communicating with GDM

GDM provides a unix socket protocol in `/tmp/.gdm_socket` for communication. Basically you give commands to GDM and it responds to you. Each command is given on a separate line and each command (except CLOSE) should have a one line response from GDM. The arguments just come space separated after the command. All of this is case sensitive so make sure you use upper case. The client should first give the VERSION command to check that the connection is working and get the version of GDM to see if the command you desire is supported. Some commands may require that they be run from a user that is logged in on the console, only FLEXI_XSERVER requires that right now. The way to do that is to first run the AUTH_LOCAL command with the X cookie that the user got for his console session. From then on the connection is authenticated as local. Of particular interest here is the CONSOLE_SERVERS command which lists all the servers running on the machine. This should be useful to anyone wishing to write some virtual console switcher or manager.

The `gdmflexiserver` command actually provides a way to send arbitrary commands to GDM without writing any code and could be used to debug or perhaps in scripts (however gdmflexiserver does require X to be running). Arbitrary command is run with the `--command=COMMAND` option. If you wish to run AUTH_LOCAL first you would also add `--authenticate`, although only FLEXI_XSERVER uses this currently. You can also pass `--debug` so that you get line by line output from the connection. As an example we get a list of the console X servers with the following (see description of CONSOLE_SERVERS command below)

```
$ gdmflexiserver --command=CONSOLE_SERVERS
OK :0,jirka,7
```

Following is a reference for communicating with GDM through the `/tmp/.gdm_socket` socket.

Command: **VERSION**

Description: Query version

Supported since: Supported since: 2.2.4.0

Arguments: None

Answers: GDM *gdm_version*

Command: **AUTH_LOCAL**

Description: Setup this connection as authenticated for running FLEXI_XSERVER, because all full blown (non-Xnest) servers can be started only from users logged in locally, and here GDM assumes only users logged in from GDM. They must pass the xauth MIT-MAGIC-COOKIE-1 that they were passed before the connection is authenticated.

Supported since: 2.2.4.0

Arguments: *xauth_cookie*
xauth_cookie is in hex form without a 0x prefix

Answers: OK
ERROR *err_number description*

0 = Not implemented

100 = Not authenticated

999 = Unknown error

Command: **FLEXI_XSERVER**

Description: Start a new X flexible server. Only supported on connection that passed AUTH_LOCAL.

Supported since: 2.2.4.0

Arguments: *xserver_type*
If no arguments, start the standard X server.

Answers: OK *display*
ERROR *err_number description*

0 = Not implemented
1 = No more flexi servers
2 = Startup errors
3 = X failed
4 = X too busy
6 = No server binary
100 = Not authenticated
999 = Unknown error

Command: **FLEXI_XNEST**

Description: Start a new flexible Xnest server

Supported since: 2.3.90.4
Note: supported on an older version from 2.2.4.0, but since 2.3.90.4 you must supply 4 arguments or ERROR 100 will be returned. This will start Xnest using the XAUTHORITY file supplied and as the uid same as the owner of that file (and same as you supply). You must also supply the cookie as the third argument for this display, to prove that you indeed are this user. Also this file must be readable ONLY by this user, that is have a mode of 0600. If this all is not met, ERROR 100 is returned.

Arguments: *display uid xauth_cookie xauth_file*
The *display* is the display the Xnest should run on, the *uid* is the user id of the requesting user, the *xauth_cookie* should be the MIT-MAGIC-COOKIE-1, the first one gdm can find in the X authority file for this display, and the *xauth_file* is the X authority file for that display. If that's not what you use you should generate a cookie first. The cookie should be in hex form.

Answers: OK *display*
ERROR *err_number description*

0 = Not implemented
1 = No more flexi servers
2 = Startup errors
3 = X failed
4 = X too busy
5 = Xnest can't connect
6 = No server binary
100 = Not authenticated
999 = Unknown error

Command: **CONSOLE_SERVERS**

Description: List all console servers, useful for Linux mostly. Doesn't list XDMCP and Xnest non-console servers.

Supported since: 2.2.4.0

Arguments: None

Answers: OK *server*;*server*;...
Note: The format for *server* is *display,user,vt*. *user* can be empty in case no one logged in yet, and *vt* can be -1 if it's not known or not supported (on non-Linux for example). If the display is an Xnest display and is a console one (that is, it is an Xnest inside another console display) it is listed and instead of vt, it lists the parent display in standard form.

Command: **UPDATE_CONFIG**

Description: Tell the daemon to update configuration of some key. Any user can really request that values are re-read but the daemon caches the last date of the configuration file and a user can't actually change any values unless they can write the configuration file. The keys that are currently supported in version 2.3.90.2 and later are: `security/AllowRoot`, `security/AllowRemoteRoot`, `security/AllowRemoteAutoLogin`, `security/RetryDelay`, `daemon/Greeter`, `daemon/RemoteGreeter`, `xdmcp/Enable`, `xdmcp/Port` and `xdmcp/PARAMETERS` (this updates all the XDMCP parameters). Keys that are supported in 2.3.90.3 and later are: `xdmcp/TimedLogin`, `xdmcp/TimedLoginEnable`, `xdmcp/TimedLoginDelay`, `greeter/SystemMenu` and `greeter/ConfigAvailable`.

Supported since: 2.3.90.2

Arguments: *key*
key is just the base part of the key such as `security/AllowRemoteRoot`.

Answers: OK
ERROR *err_number description*

0 = Not implemented
50 = Unsupported key
999 = Unknown error

Command: **GREETERPIDS**

Description: List all greeter pids so that one can send SIGHUP to them for configuration file rereading. Of course one must be root to do that.

Supported since: 2.3.90.2

Arguments: None

Answers: OK *pid*;*pid*;...

Command: **CLOSE**

Description: Close the connection.

Supported since: 2.2.4.0

Arguments: None

Answers: None

There is also a fifo protocol which is in the server authorization directory (usually `/var/gdm`) and this is called `.gdmfifo`. Only root can access this file and it is mostly used for internal GDM chatter between the slave and the main process. It can however be useful for controlling GDM in various ways as well, and it is possible to do this from scripts with a simple echo command. What follows is a list of useful commands

Command: **SOFT_RESTART**

Description: Restart GDM but only after everyone has logged out. This has been supported as long as there was a fifo protocol.

Command: **DIRTY_SERVERS**

Description: All X servers should be restarted rather than regenerated. Useful if you have updated the X configuration. Note that this happens only when the user logs out or when we otherwise would have restarted a server, nothing immediate is done by this command. This is supported since 2.4.0.6.

Command: **SOFT_RESTART_SERVERS**

Description: Restart all X servers that people aren't logged in on. Maybe you may not want to do this on every change of X server configuration since this may cause flicker on screen and jumping around on the vt. Perhaps useful to do by asking the user if they want to do that. Note that this will not kill any logged in sessions. This is supported since 2.4.0.6.

Command: **SUSPEND_MACHINE**

Description: Have the master run the suspend command if possible. Note that this command is only available in CVS currently and will only be available in version 2.4.2.x and higher.

Command: **HUP_ALL_GREETERS**

Description: Tell all the greeters to reread their configuration files by signaling them with a HUP. This has been available since 2.3.90.2.

An example might be in a script that modifies the X server configuration and wants all the X servers to actually restart next time they reset. So you would run

```
echo DIRTY_SERVERS > /var/gdm/.gdmfifo
```

Some things can be achieved by sending signals or running scripts. For example to tell GDM to stop itself you can run `gdm-stop` or send SIGTERM to the main process. To tell it to immediately restart you can run `gdm-restart` or send SIGHUP to the main process. You can also run `gdm-safe-restart` or send SIGUSR1 which is the same as sending SOFT_RESTART in the fifo protocol and will restart GDM once everyone logs out.

7 Future

There are of course many future plans for GDM. Besides general cleanup and minor feature work, it is planned to improve the accessibility and scalability of GDM. GDM is already good for small desktop installations, but it currently isn't all that good for very large installations. It also lacks any accessibility support. Sun is one company that is investing developer time in improving GDM in both of these directions. Some specific things that are planned that are user visible are things like smart card access support, better face browser (plus face browser for the graphical greeter), better PAM support, better support for multihead environments, RandR support, better language selection support, better session script setup common with KDM, etc...

While GDM development has slowed down recently, it really doesn't need all that much more to gain all, or most, of the above. I think most focus will be in cleaning up existing features and generally stabilizing GDM. Of course one thing that is still on the TODO list is to actually document GDM properly. And I hope that this talk and this paper has been at least helpful in this regard.