Xserver Provider for DTrace

Alan Coopersmith, Oracle Corporation

X Server Version 1.12.1

Table of Contents

Introduction

This page provides details on a statically defined user application tracing provider [http://wikis.sun.com/display/DTrace/Statically+Defined+Tracing+for+User+Applications] for the DTrace [http://hub.opensolaris.org/bin/view/Community+Group+dtrace/] facility in Solaris™ 10, MacOS X™ 10.5, and later releases. This provider instruments various points in the X server, to allow tracing what client applications are up to.

The provider was integrated into the X.Org git master repository with Solaris 10 & OpenSolaris support for the Xserver 1.4 release, released in 2007 with X11R7.3. Support for DTrace on MacOS X was added in Xserver 1.7.

These probes expose the request and reply structure of the X protocol between clients and the X server, so an understanding of that basic nature will aid in learning how to use these probes.

Available probes

Due to the way User-Defined DTrace probes work, arguments to these probes all bear undistinguished names of *arg0*, *arg1*, *arg2*, etc. These tables should help you determine what the real data is for each of the probe arguments.

Table 1. Probes and their arguments

Probe name	**Description**	**arg0**	**arg1**	**arg2**	**arg3**	**arg4**
Request Probes						
request-start	Called just before processing each client request.	*requestName*	*requestCode*	*requestLength*	*clientId*	*requestBuffer*
request-done	Called just after processing each client request.	*requestName*	*requestCode*	*sequenceNumber*	*clientId*	*resultCode*
Event Probes						
send-event	Called just before send each event to a client.	*clientId*	*eventCode*	*eventBuffer*		
Client Connection Probes						
client-connect	Called when a new connection is opened from a client	*clientId*	*clientFD*			
client-auth	Called when client authenticates (normally just after connection opened)	*clientId*	*clientAddr*	*clientPid*	*clientZoneId*	
client-disconnect	Called when a client connection is closed	*clientId*				
Resource Allocation Probes						

Probe name	**Description**	**arg0**	**arg1**	**arg2**	**arg3**	**arg4**
resource-alloc	Called when a new resource (pixmap, gc, colormap, etc.) is allocated	*`resourceId`*	*`resourceTypeId`*	*`resourceValue`*	*`resourceTypeName`*	
resource-free	Called when a resource is freed	*`resourceId`*	*`resourceTypeId`*	*`resourceValue`*	*`resourceTypeName`*	

Data Available in Probe Arguments

To access data in arguments of type string, you will need to use `copyinstr()` [http://wikis.sun.com/display/DTrace/Actions+and+Subroutines#ActionsandSubroutines-{{copyinstr}}]. To access data buffers referenced via uintptr_t's, you will need to use `copyin()` [http://wikis.sun.com/display/DTrace/Actions+and+Subroutines#ActionsandSubroutines-{{copyin}}].

Table 2. Probe Arguments

Argument name	**Type**	**Description**
`clientAddr`	string	String representing address client connected from
`clientFD`	int	X server's file descriptor for server side of each connection
`clientId`	int	Unique integer identifier for each connection to the X server
`clientPid`	pid_t	Process id of client, if connection is local (from `getpeerucred()`)
`clientZoneId`	zoneid_t	Solaris: Zone id of client, if connection is local (from `getpeerucred()`)
`eventBuffer`	uintptr_t	Pointer to buffer containing X event - decode using structures in <`X11/Xproto.h` [http://cgit.freedesktop.org/xorg/proto/xproto/tree/Xproto.h]> and similar headers for each extension
`eventCode`	uint8_t	Event number of X event
`resourceId`	uint32_t	X resource id (XID)
`resourceTypeId`	uint32_t	Resource type id
`resourceTypeName`	string	String representing X resource type (`"PIXMAP"`, etc.)
`resourceValue`	uintptr_t	Pointer to data for X resource
`resultCode`	int	Integer code representing result status of request
`requestBuffer`	uintptr_t	Pointer to buffer containing X request - decode using structures in <`X11/Xproto.h` [http://cgit.freedesktop.org/xorg/proto/xproto/tree/Xproto.h]> and similar headers for each extension

Argument name	**Type**	**Description**
`requestCode`	uint8_t	Request number of X request or Extension
`requestName`	string	Name of X request or Extension
`requestLength`	uint16_t	Length of X request
`sequenceNumber`	uint32_t	Number of X request in in this connection

Examples

Example 1. Counting requests by request name

This script simply increments a counter for each different request made, and when you exit the script (such as by hitting **Control+C**) prints the counts.

```
#!/usr/sbin/dtrace -s

Xserver*:::request-start
{
	@counts[copyinstr(arg0)] = count();
}
```

The output from a short run may appear as:

```
  QueryPointer                                                      1
  CreatePixmap                                                      2
  FreePixmap                                                        2
  PutImage                                                          2
  ChangeGC                                                         10
  CopyArea                                                         10
  CreateGC                                                         14
  FreeGC                                                           14
  RENDER                                                           28
  SetClipRectangles                                                40
```

This can be rewritten slightly to cache the string containing the name of the request since it will be reused many times, instead of copying it over and over from the kernel:

```
#!/usr/sbin/dtrace -s

string Xrequest[uintptr_t];

Xserver*:::request-start
/Xrequest[arg0] == ""/
{
```

```
    Xrequest[arg0] = copyinstr(arg0);
}

Xserver*:::request-start
{
    @counts[Xrequest[arg0]] = count();
}
```

Example 2. Get average CPU time per request

This script records the CPU time used between the probes at the start and end of each request and aggregates it per request type.

```
#!/usr/sbin/dtrace -s

Xserver*:::request-start
{
    reqstart = vtimestamp;
}

Xserver*:::request-done
{
    @times[copyinstr(arg0)] = avg(vtimestamp - reqstart);
}
```

The output from a sample run might look like:

```
  ChangeGC                                                          889
  MapWindow                                                         907
  SetClipRectangles                                                1319
  PolyPoint                                                        1413
  PolySegment                                                      1434
  PolyRectangle                                                    1828
  FreeCursor                                                       1895
  FreeGC                                                           1950
  CreateGC                                                         2244
  FreePixmap                                                       2246
  GetInputFocus                                                    2249
  TranslateCoords                                                  8508
  QueryTree                                                        8846
  GetGeometry                                                      9948
  CreatePixmap                                                    12111
  AllowEvents                                                     14090
  GrabServer                                                      14791
  MIT-SCREEN-SAVER                                                16747
  ConfigureWindow                                                 22917
  SetInputFocus                                                   28521
  PutImage                                                       240841
```

Example 3. Monitoring clients that connect and disconnect

This script simply prints information about each client that connects or disconnects from the server while it is running. Since the provider is specified as `Xserver$1` instead of `Xserver*` like previous examples, it won't monitor all Xserver processes running on the machine, but instead expects the process id of the X server to monitor to be specified as the argument to the script.

```
#!/usr/sbin/dtrace -s

Xserver$1:::client-connect
{
 printf("** Client Connect: id %d\n", arg0);
}

Xserver$1:::client-auth
{
 printf("** Client auth'ed: id %d => %s pid %d\n",
  arg0, copyinstr(arg1), arg2);
}

Xserver$1:::client-disconnect
{
 printf("** Client Disconnect: id %d\n", arg0);
}
```

A sample run:

```
# ./foo.d 5790
dtrace: script './foo.d' matched 4 probes
CPU     ID                    FUNCTION:NAME
  0  15774 CloseDownClient:client-disconnect ** Client Disconnect: id 65

  2  15774 CloseDownClient:client-disconnect ** Client Disconnect: id 64

  0  15773 EstablishNewConnections:client-connect ** Client Connect: id 64

  0  15772            AuthAudit:client-auth ** Client auth'ed: id 64 => local host

  0  15773 EstablishNewConnections:client-connect ** Client Connect: id 65

  0  15772            AuthAudit:client-auth ** Client auth'ed: id 65 => local host

  0  15774 CloseDownClient:client-disconnect ** Client Disconnect: id 64
```

Example 4. Monitoring clients creating Pixmaps

This script can be used to determine which clients are creating pixmaps in the X server, printing information about each client as it connects to help trace it back to the program on the other end of the X connection.

```
#!/usr/sbin/dtrace -qs

string Xrequest[uintptr_t];
string Xrestype[uintptr_t];

Xserver$1:::request-start
/Xrequest[arg0] == ""/
{
 Xrequest[arg0] = copyinstr(arg0);
}

Xserver$1:::resource-alloc
/arg3 != 0 && Xrestype[arg3] == ""/
{
 Xrestype[arg3] = copyinstr(arg3);
}


Xserver$1:::request-start
/Xrequest[arg0] == "X_CreatePixmap"/
{
 printf("-> %s: client %d\n", Xrequest[arg0], arg3);
}

Xserver$1:::request-done
/Xrequest[arg0] == "X_CreatePixmap"/
{
 printf("<- %s: client %d\n", Xrequest[arg0], arg3);
}

Xserver$1:::resource-alloc
/Xrestype[arg3] == "PIXMAP"/
{
 printf("** Pixmap alloc: %08x\n", arg0);
}


Xserver$1:::resource-free
/Xrestype[arg3] == "PIXMAP"/
{
 printf("** Pixmap free:  %08x\n", arg0);
}

Xserver$1:::client-connect
{
 printf("** Client Connect: id %d\n", arg0);
}

Xserver$1:::client-auth
{
 printf("** Client auth'ed: id %d => %s pid %d\n",
  arg0, copyinstr(arg1), arg2);
}
```

```
Xserver$1:::client-disconnect
{
 printf("** Client Disconnect: id %d\n", arg0);
}
```

Sample output from a run of this script:

```
** Client Connect: id 17
** Client auth'ed: id 17 => local host pid 20273
-> X_CreatePixmap: client 17
** Pixmap alloc: 02200009
<- X_CreatePixmap: client 17
-> X_CreatePixmap: client 15
** Pixmap alloc: 01e00180
<- X_CreatePixmap: client 15
-> X_CreatePixmap: client 15
** Pixmap alloc: 01e00181
<- X_CreatePixmap: client 15
-> X_CreatePixmap: client 14
** Pixmap alloc: 01c004c8
<- X_CreatePixmap: client 14
** Pixmap free:  02200009
** Client Disconnect: id 17
** Pixmap free:  01e00180
** Pixmap free:  01e00181
```