Xserver provider for DTrace

Alan Coopersmith, Oracle Corporation

X.Org Xserver version 1.9.99.901

Copyright (c) 2005, 2006, 2007, 2010, Oracle and/or its affiliates. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice (including the next paragraph) shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Table of Contents

Introduction ...................................................... 1
Available probes .................................................. 2
Data Available in Probe Arguments ............................ 2
Examples .......................................................... 4

Introduction

This page provides details on a **statically defined user application tracing provider** [http://wikis.sun.com/display/DTrace/Static+Defined+Tracing+for+User+Applications] for the **DTrace** [http://hub.opensolaris.org/bin/view/Communi-

ty+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 \textit{arg0}, \textit{arg1}, \textit{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

<table>
<thead>
<tr>
<th>Probe name</th>
<th>Description</th>
<th>\textit{arg0}</th>
<th>\textit{arg1}</th>
<th>\textit{arg2}</th>
<th>\textit{arg3}</th>
<th>\textit{arg4}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Probes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>request-start</td>
<td>Called just before processing each client request.</td>
<td>request-name</td>
<td>request-code</td>
<td>request-name</td>
<td>client-request-name</td>
<td>buffer</td>
</tr>
<tr>
<td>request-done</td>
<td>Called just after processing each client request.</td>
<td>request-name</td>
<td>request-code</td>
<td>sequence-number</td>
<td>client-result-code</td>
<td></td>
</tr>
<tr>
<td>Event Probes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send-event</td>
<td>Called just before send each event to a client.</td>
<td>client-id</td>
<td>event-code</td>
<td>event-buffer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client Connection Probes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>client-connect</td>
<td>Called when a new connection is opened from a client</td>
<td>client-id</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>client-auth</td>
<td>Called when client authenticates (normally just after connection opened)</td>
<td>client-id</td>
<td>client-addr</td>
<td>client-pid</td>
<td>client-zone-id</td>
<td></td>
</tr>
<tr>
<td>client-disconnect</td>
<td>Called when a client connection is closed</td>
<td>client-id</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource Allocation Probes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resource-alloc</td>
<td>Called when a new resource ( pixmap, gc, colormap, etc.) is allocated</td>
<td>resource-id</td>
<td>resource-type-id</td>
<td>resource-value</td>
<td>resource-type-name</td>
<td></td>
</tr>
<tr>
<td>resource-free</td>
<td>Called when a resource is freed</td>
<td>resource-id</td>
<td>resource-type-id</td>
<td>resource-value</td>
<td>resource-type-name</td>
<td></td>
</tr>
</tbody>
</table>

Data Available in Probe Arguments

To access data in arguments of type string, you will need to use \texttt{copyinstr()} [http://wikis.sun.com/display/DTrace/Actions+and+Subroutines#ActionsandSubroutines-\{copyinstr\}]. To access data buffers referenced via \texttt{uintptr_t}'s, you will need to use \texttt{copyin()} [http://wikis.sun.com/display/DTrace/Actions+and+Subroutines#ActionsandSubroutines-\{copyin\}].
Table 2. Probe Arguments

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clientAddr</td>
<td>string</td>
<td>String representing address client connected from</td>
</tr>
<tr>
<td>clientFD</td>
<td>int</td>
<td>X server's file descriptor for server side of each connection</td>
</tr>
<tr>
<td>clientId</td>
<td>int</td>
<td>Unique integer identifier for each connection to the X server</td>
</tr>
<tr>
<td>clientPid</td>
<td>pid_t</td>
<td>Process id of client, if connection is local (from getpeeruid())</td>
</tr>
<tr>
<td>clientZoneId</td>
<td>zoneid_t</td>
<td>Solaris: Zone id of client, if connection is local (from getpeeruid())</td>
</tr>
<tr>
<td>eventBuffer</td>
<td>uintptr_t</td>
<td>Pointer to buffer containing X event - decode using structures in &lt;X11/Xproto.h&gt; and similar headers for each extension</td>
</tr>
<tr>
<td>eventCode</td>
<td>uint8_t</td>
<td>Event number of X event</td>
</tr>
<tr>
<td>resourceId</td>
<td>uint32_t</td>
<td>X resource id (XID)</td>
</tr>
<tr>
<td>resourceType-Id</td>
<td>uint32_t</td>
<td>Resource type id</td>
</tr>
<tr>
<td>resourceType-Name</td>
<td>string</td>
<td>String representing X resource type (&quot;PIXMAP&quot;, etc.)</td>
</tr>
<tr>
<td>resourceValue</td>
<td>uintptr_t</td>
<td>Pointer to data for X resource</td>
</tr>
<tr>
<td>resultCode</td>
<td>int</td>
<td>Integer code representing result status of request</td>
</tr>
<tr>
<td>requestBuffer</td>
<td>uintptr_t</td>
<td>Pointer to buffer containing X request - decode using structures in &lt;X11/Xproto.h&gt; and similar headers for each extension</td>
</tr>
<tr>
<td>requestCode</td>
<td>uint8_t</td>
<td>Request number of X request or Extension</td>
</tr>
<tr>
<td>requestName</td>
<td>string</td>
<td>Name of X request or Extension</td>
</tr>
<tr>
<td>requestLength</td>
<td>uint16_t</td>
<td>Length of X request</td>
</tr>
<tr>
<td>sequenceNumber</td>
<td>uint32_t</td>
<td>Number of X request in this connection</td>
</tr>
</tbody>
</table>
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.

```bash
#!/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.

```bash
#!/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:

```bash
# ./foo.d 5790
```
```bash
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

Example 4. Monitoring clients creating Pixmaps