Double Buffer Extension Protocol

X Consortium Standard

Ian Elliott, Hewlett-Packard Company
David P. Wiggins
X Consortium

Double Buffer Extension Protocol: X Consortium Standard

by Ian Elliott
David P. Wiggins
X Consortium

X Version 11, Release 7.7

Version 1.0

Table of Contents

Chapter 1. Introduction

The Double Buffer Extension (DBE) provides a standard way to utilize double-buffering within the framework of the X Window System. Double-buffering uses two buffers, called front and back, which hold images. The front buffer is visible to the user; the back buffer is not. Successive frames of an animation are rendered into the back buffer while the previously rendered frame is displayed in the front buffer. When a new frame is ready, the back and front buffers swap roles, making the new frame visible. Ideally, this exchange appears to happen instantaneously to the user and with no visual artifacts. Thus, only completely rendered images are presented to the user, and they remain visible during the entire time it takes to render a new frame. The result is a flicker-free animation.

Chapter 2. Goals

This extension should enable clients to:

- Allocate and deallocate double-buffering for a window.
- Draw to and read from the front and back buffers associated with a window.
- Swap the front and back buffers associated with a window.
- Specify a wide range of actions to be taken when a window is swapped. This includes explicit, simple swap actions (defined below), and more complex actions (for example, clearing ancillary buffers) that can be put together within explicit "begin" and "end" requests (defined below).
- Request that the front and back buffers associated with multiple double-buffered windows be swapped simultaneously.

In addition, the extension should:

- Allow multiple clients to use double-buffering on the same window.
- Support a range of implementation methods that can capitalize on existing hardware features.
- Add no new event types.
- Be reasonably easy to integrate with a variety of direct graphics hardware access (DGHA) architectures.

Chapter 3. Concepts

Normal windows are created using the core `CreateWindow` request, which allocates a set of window attributes and, for InputOutput windows, a front buffer, into which an image can be drawn. The contents of this buffer will be displayed when the window is visible.

This extension enables applications to use double-buffering with a window. This involves creating a second buffer, called a back buffer, and associating one or more back buffer names (XIDs) with the window for use when referring to (that is, drawing to or reading from) the window's back buffer. The back buffer name is a DRAWABLE of type BACKBUFFER.

DBE provides a relative double-buffering model. One XID, the window, always refers to the front buffer. One or more other XIDs, the back buffer names, always refer to the back buffer. After a buffer swap, the window continues to refer to the (new) front buffer, and the back buffer name continues to refer to the (new) back buffer. Thus, applications and toolkits that want to just render to the back buffer always use the back buffer name for all drawing requests to the window. Portions of an application that want to render to the front buffer always use the window XID for all drawing requests to the window.

Multiple clients and toolkits can all use double-buffering on the same window. DBE does not provide a request for querying whether a window has double-buffering support, and if so, what the back buffer name is. Given the asynchronous nature of the X Window System, this would cause race conditions. Instead, DBE allows multiple back buffer names to exist for the same window; they all refer to the same physical back buffer. The first time a back buffer name is allocated for a window, the window becomes double-buffered and the back buffer name is associated with the window. Subsequently, the window already is a double-buffered window, and nothing about the window changes when a new back buffer name is allocated, except that the new back buffer name is associated with the window. The window remains double-buffered until either the window is destroyed or until all of the back buffer names for the window are deallocated.

In general, both the front and back buffers are treated the same. In particular, here are some important characteristics:

- Only one buffer per window can be visible at a time (the front buffer).
- Both buffers associated with a window have the same visual type, depth, width, height, and shape as the window.
- Both buffers associated with a window are "visible" (or "obscured") in the same way. When an Expose event is generated for a window, both buffers should be considered to be damaged in the exposed area. Damage that occurs to either buffer will result in an Expose event on the window. When a double-buffered window is exposed, both buffers are tiled with the window background, exactly as stated by the core protocol. Even though the back buffer is not visible, terms such as obscure apply to the back buffer as well as to the front buffer.
- It is acceptable at any time to pass a BACKBUFFER in any request, notably any core or extension drawing request, that expects a DRAWABLE. This enables an application to draw directly into BACKBUFFERs in the same fashion as it would draw into any other DRAWABLE.

- It is an error (Window) to pass a BACKBUFFER in a core request that expects a Window.
- A BACKBUFFER will never be sent by core X in a reply, event, or error where a Window is specified.
- If core X11 backing-store and save-under applies to a double-buffered window, it applies to both buffers equally.
- If the core `ClearArea` request is executed on a double-buffered window, the same area in both the front and back buffers is cleared.

The effect of passing a window to a request that accepts a DRAWABLE is unchanged by this extension. The window and front buffer are synonomous with each other. This includes obeying the `GetImage` semantics and the subwindow-mode semantics if a core graphics context is involved. Regardless of whether the window was explicitly passed in a `GetImage` request, or implicitly referenced (that is, one of the windo's ancestors was passed in the request), the front (that is, visible) buffer is always referenced. Thus, DBE-naive screen dump clients will always get the front buffer. `GetImage` on a back buffer returns undefined image contents for any obscured regions of the back buffer that fall within the image.

Drawing to a back buffer always uses the clip region that would be used to draw to the front buffer with a GC subwindow-mode of `ClipByChildren`. If an ancestor of a double-buffered window is drawn to with a core GC having a subwindow-mode of `IncludeInferiors`, the effect on the double-buffered window's back buffer depends on the depth of the double-buffered window and the ancestor. If the depths are the same, the contents of the back buffer of the double-buffered window are not changed. If the depths are different, the contents of the back buffer of the double-buffered window are undefined for the pixels that the `IncludeInferiors` drawing touched.

DBE adds no new events. DBE does not extend the semantics of any existing events with the exception of adding a new DRAWABLE type called BACKBUFFER. If events, replies, or errors that contain a DRAWABLE (for example, `GraphicsExpose`) are generated in response to a request, the DRAWABLE returned will be the one specified in the request.

DBE advertises which visuals support double-buffering.

DBE does not include any timing or synchronization facilities. Applications that need such facilities (for example, to maintain a constant frame rate) should investigate the Synchronization Extension, an X Consortium standard.

Window Management Operations

The basic philosophy of DBE is that both buffers are treated the same by core X window management operations.

When the core `DestroyWindow` is executed on a double-buffered window, both buffers associated with the window are destroyed, and all back buffer names associated with the window are freed.

If the core `ConfigureWindow` request changes the size of a window, both buffers assume the new size. If the windo's size increases, the effect on the buffers depends on whether the implementation honors bit gravity for buffers. If bit gravity is implemented, then the contents of both buffers are moved in accordance with the windo's bit gravity (see the core `ConfigureWindow` request), and the remaining areas are tiled with the window background. If bit gravity is not implemented, then the entire unobscured region of both buffers is tiled with the window background. In either case, `Expose` events are generated for the region that is tiled with the window background.

If the core `GetGeometry` request is executed on a BACKBUFFER, the returned x, y, and border-width will be zero.

If the Shape extension `ShapeRectangles`, `ShapeMask`, `ShapeCombine`, or `ShapeOffset` request is executed on a double-buffered window, both buffers are reshaped to match the new window shape. The region difference is the following:

D = newshape - oldshape

It is tiled with the window background in both buffers, and `Expose` events are generated for D.

Complex Swap Actions

DBE has no explicit knowledge of ancillary buffers (for example, depth buffers or alpha buffers), and only has a limited set of defined swap actions. Some applications may need a richer set of swap actions than DBE provides. Some DBE implementations have knowledge of ancillary buffers, and/or can provide a rich set of swap actions. Instead of continually extending DBE to increase its set of swap actions, DBE provides a flexible "idiom" mechanism. If an application's needs are served by the defined swap actions, it should use them; otherwise, it should use the following method of expressing a complex swap action as an idiom. Following this policy will ensure the best possible performance across a wide variety of implementations.

As suggested by the term "idiom," a complex swap action should be expressed as a group/series of requests. Taken together, this group of requests may be combined into an atomic operation by the implementation, in order to maximize performance. The set of idioms actually recognized for optimization is implementation dependent. To help with idiom expression and interpretation, an idiom must be surrounded by two protocol requests: `DBEBeginIdiom` and `DBEEndIdiom`. Unless this begin-end pair surrounds the idiom, it may not be recognized by a given implementation, and performance will suffer.

For example, if an application wants to swap buffers for two windows, and use core X to clear only certain planes of the back buffers, the application would issue the following protocol requests as a group, and in the following order:

- DBEBeginIdiom request.
- `DBESwapBuffers` request with XIDs for two windows, each of which uses a swap action of Untouched.
- Core X PolyFillRectangle request to the back buffer of one window.
- Core X PolyFillRectangle request to the back buffer of the other window.
- DBEEndIdiom request.

The `DBEBeginIdiom` and `DBEEndIdiom` requests do not perform any actions themselves. They are treated as markers by implementations that can combine certain groups/series of requests as idioms, and are ignored by other implementations or for nonrecognized groups/series of requests. If these requests are sent out of order, or are mismatched, no errors are sent, and the requests are executed as usual, though performance may suffer.

An idiom need not include a `DBESwapBuffers` request. For example, if a swap action of Copied is desired, but only some of the planes should be copied, a core X `CopyArea` request may be used instead of `DBESwapBuffers`. If `DBESwapBuffers` is included in an idiom, it should immediately follow the `DBEBeginIdiom` request. Also, when the `DBESwapBuffers` is included in an idiom, that request's swap action will still be valid, and if the swap action might overlap with another request, then the final result of the idiom must be as if the separate requests were executed serially. For example, if the specified swap action is Untouched, and if a `PolyFillRectangle` using a client clip rectangle is done to the windo's back buffer after the `DBESwapBuffers` request, then the contents of the new back buffer (after the idiom) will be the same as if the idiom was not recognized by the implementation.

It is highly recommended that Application Programming Interface (API) providers define, and application developers use, "convenience" functions that allow client applications to call one procedure that encapsulates common idioms. These functions will generate the `DBEBeginIdiom` request, the idiom requests, and `DBEEndIdiom` request. Usage of these functions will ensure best possible performance across a wide variety of implementations.

Chapter 4. Requests

The DBE defines the following requests.

DBEGetVersion

This request returns the major and minor version numbers of this extension.

DBEGetVersion

client-major-version	CARD8
client-minor-version	CARD8
=>	
server-major-version	CARD8
server-minor-version	CARD8

The client-major-version and client-minor-version numbers indicate what version of the protocol the client wants the server to implement. The server-major-version and the server-minor-version numbers returned indicate the protocol this extension actually supports. This might not equal the version sent by the client. An implementation can (but need not) support more than one version simultaneously. The server-major-version and server-minor-version allow the creation of future revisions of the DBE protocol that may be necessary. In general, the major version would increment for incompatible changes, and the minor version would increment for small, upward-compatible changes. Servers that support the protocol defined in this document will return a server-major-version of one (1), and a server-minor-version of zero (0).

The DBE client must issue a DBEGetVersion request before any other double buffering request in order to negotiate a compatible protocol version; otherwise, the client will get undefined behavior (DBE may or may not work).

DBEGetVisualInfo

This request returns information about which visuals support double buffering.

DBEGetVisualInfo

screen-specifiers	LISTofDRAWABLE
=>	
visinfo	LISTofSCREENVISINFO

where:

SCREENVISINFO	LISTofVISINFO
VISINFO	[visual: VISUALID depth: CARD8 perflevel: CARD8]

Errors: Drawable

All of the values passed in screen-specifiers must be valid DRAWABLEs (or a `Drawable` error results). For each drawable in screen-specifiers, the reply will contain a list of VISINFO structures for visuals that support double-buffering on the screen on which the drawable resides. The visual member specifies the VISUALID. The depth member specifies the depth in bits for the visual. The perflevel is a performance hint. The only operation defined on a perflevel is comparison to a perflevel of another visual on the same screen. The visual having the higher perflevel is likely to have better double-buffer graphics performance than the visual having the lower perflevel. Nothing can be deduced from any of the following: the magnitude of the difference of two perflevels, a perflevel value in isolation, or comparing perflevels from different servers.

If the list of screen-specifiers is empty, information for all screens is returned, starting with screen zero.

DBEAllocateBackBufferName

This request allocates a drawable ID used to refer to the back buffer of a window.

DBEAllocateBackBufferName

window	WINDOW
back-buffer-name	BACKBUFFER
swap-action-hint	SWAPACTION

Errors: Alloc, Value, IDChoice, Match, Window

If the window is not already a double-buffered window, the window becomes double-buffered, and the back-buffer-name is associated with the window. The swap-action-hint tells the server which swap action is most likely to be used with the window in subsequent `DBESwapBuffers` requests. The swap-action-hint must have one of the values specified for type SWAPACTION (or a Value error results). See the description of the `DBESwapBuffers` request for a complete discussion of swap actions and the SWAPACTION type.

If the window already is a double-buffered window, nothing about the window changes, except that an additional back-buffer-name is associated with the window. The window remains double-buffered until either the window is destroyed, or until all of the back buffer names for the window are deallocated.

The window passed into the request must be a valid WINDOW (or a Window error results). The window passed into the request must be an InputOutput window (or a Match error results). The visual of the window must be in the list returned by `DBEGetVisualInfo` (or a Match error results). The back-buffer-name must be in the range assigned to the client, and must not already be in use (or an IDChoice error results). If the server cannot allocate all resources associated with turning on double-buffering for the window, an Alloc error results, the windo's double-buffer status (whether it is already double-buffered or not) remains unchanged, and the back-buffer-name is freed.

DBEDeallocateBackBufferName

This request frees a drawable ID that was obtained by `DBEAllocateBackBufferName`.

DBEDeallocateBackBufferName

back-buffer-name	BACKBUFFER

Errors: Buffer

The back-buffer-name passed in the request is freed and no longer associated with the window. If this is the last back-buffer-name associated with the window, then the back buffer is no longer accessible to clients, and all double-buffering resources associated with the window may be freed. The window's current front buffer remains the front buffer.

The back-buffer-name must be a valid BACKBUFFER associated with a window (or a Buffer error results).

DBESwapBuffers

This request swaps the buffers for all windows listed, applying the appropriate swap action for each window.

`DBESwapBuffers`

windows	LISTofSWAPINFO

where:

SWAPINFO	[window: WINDOW swap-action: SWAPACTION]
SWAPACTION	{ Undefined, Background, Untouched, Copied }

Errors: Match, Window, Value

Each window passed into the request must be a valid WINDOW (or a `Window` error results). Each window passed into the request must be a double-buffered window (or a `Match` error results). Each window passed into the request must only be listed once (or a `Match` error results). Each swap-action in the list must have one of the values specified for type SWAPACTION (or a `Value` error results). If an error results, none of the valid double-buffered windows will have their buffers swapped.

The swap-action determines what will happen to the new back buffer of the window it is paired with in the list in addition to making the old back buffer become visible. The defined actions are as follows:

Undefined	The contents of the new back buffer become undefined. This may be the most efficient action since it allows the implementation to discard the contents of the buffer if it needs to.
Background	The unobscured region of the new back buffer will be tiled with the window background. The background action allows devices to use a fast clear capability during a swap.
Untouched	The unobscured region of the new back buffer will be unmodified by the swap.

Copied	The unobscured region of the new back buffer will be the contents of the old back buffer.

If `DBESwapBuffers` is included in a "swap and clear" type of idiom, it must immediately follow the `DBEBeginIdiom` request.

DBEBeginIdiom

This request informs the server that a complex swap will immediately follow this request.

```
DBEBeginIdiom
```

As previously discussed, a complex swap action is a group/series of requests that, taken together, may be combined into an atomic operation by the implementation. The sole function of this request is to serve as a "marker" that the server can use to aid in idiom processing. The server is free to implement this request as a no-op.

DBEEndIdiom

This request informs the server that a complex swap has concluded.

```
DBEEndIdiom
```

The sole function of this request is to serve as a "marker" that the server can use to aid in idiom processing. The server is free to implement this request as a no-op.

DBEGetBackBufferAttributes

This request returns information about a back buffer.

```
DBEGetBackBufferAttributes
```

back-buffer-name	BACKBUFFER
=>	
attributes	BUFFER_ATTRIBUTES

where:

BUFFER_ATTRIBUTES: [window: WINDOW]

If back-buffer-name is a valid BACKBUFFER, the window field of the attributes in the reply will be the window which has the back buffer that back-buffer-name refers to. If back-buffer-name is not a valid BACKBUFFER, the window field of the attributes in the reply will be None.

Chapter 5. Encoding

Please refer to the X11 Protocol Encoding document as this section uses syntactic conventions and data types established there.

The name of this extension is "DOUBLE-BUFFER".

Type

The following new types are used by the extension.

BACKBUFFER: XID

SWAPACTION

```
#x00  Undefined
#x01  Background
#x02  Untouched
#x03  Copied
```

SWAPINFO

```
4     WINDOW                 window
1     SWAPACTION             swap action
3                            unused
```

```
VISINFO
4     VISUALID                visual
1     CARD8                   depth
1     CARD8                   perflevel
2                             unused

SCREENVISINFO
4     CARD32                  n, number in list
8n    LISTofVISINFO           n VISINFOs

BUFFER_ATTRIBUTES
4     WINDOW                 window
```

Error

```
Buffer
```

```
1     0                       error
1     error base + 0          code
2     CARD16                  sequence number
4     CARD32                  bad buffer
2     CARD16                  minor-opcode
```

```
1      CARD8                  major-opcode
21                            unused
```

Request

```
DBEGetVersion
1      CARD8                  major-opcode
1      0                      minor-opcode
2      2                      request length
1      CARD8                  client-major-version
1      CARD8                  client-minor-version
2                             unused
=>
1                             unused
2      CARD16                 sequence number
4      0                      reply length
1      CARD8                  server-major-version
1      CARD8                  server-minor-version
22                            unused
```

```
DBEAllocateBackBufferName


1      CARD8                  major-opcode
1      1                      minor-opcode
2      4                      request length
4      WINDOW                 window
4      BACKBUFFER             back buffer name
1      SWAPACTION             swap action hint
3                             unused
```

```
DBEDeallocateBackBufferName


1      CARD8                  major-opcode
1      2                      minor-opcode
2      2                      request length
4      BACKBUFFER             back buffer name
```

```
DBESwapBuffers


1      CARD8                  major-opcode
1      3                      minor-opcode
2      2+2n                   request length
4      CARD32                 n, number of window/swap action pairs in list
8n     LISTofSWAPINFO         window/swap action pairs
```

```
DBEBeginIdiom


1      CARD8                  major-opcode
1      4                      minor-opcode
2      1                      request length
```

```
DBEEndIdiom


1     CARD8                   major-opcode
1     5                       minor-opcode
2     1                       request length

DBEGetVisualInfo


1     CARD8                   major-opcode
1     6                       minor-opcode
2     2+n                     request length
4     CARD32                  n, number of screen specifiers in list
4n    LISTofDRAWABLE          n screen specifiers
=>
1     1                       Reply
1                             unused
2     CARD16                  sequence number
4     CARD32                  reply length
4     CARD32                  m, number of SCREENVISINFOs in list
20                            unused
4j    LISTofSCREENVISINFO     m SCREENVISINFOs

DBEGetBackBufferAttributes


1     CARD8                   major-opcode
1     7                       minor-opcode
2     2                       request length
4     BACKBUFFER              back-buffer-name
=>
1                             unused
2     CARD16                  sequence number
4     0                       reply length
4     BUFFER_ATTRIBUTES       attributes
20                            unused
```

Chapter 6. Acknowledgements

We wish to thank the following individuals who have contributed their time and talent toward shaping the DBE specification:

T. Alex Chen, IBM; Peter Daifuku, Silicon Graphics, Inc.; Ian Elliott, Hewlett-Packard Company; Stephen Gildea, X Consortium, Inc.; Jim Graham, Sun; Larry Hare, AGE Logic; Jay Hersh, X Consortium, Inc.; Daryl Huff, Sun; Deron Dann Johnson, Sun; Louis Khouw, Sun; Mark Kilgard, Silicon Graphics, Inc.; Rob Lembree, Digital Equipment Corporation; Alan Ricker, Metheus; Michael Rosenblum, Digital Equipment Corporation; Bob Scheifler, X Consortium, Inc.; Larry Seiler, Digital Equipment Corporation; Jeanne Sparlin Smith, IBM; Jeff Stevenson, Hewlett-Packard Company; Walter Strand, Metheus; Ken Tidwell, Hewlett-Packard Company; and David P. Wiggins, X Consortium, Inc.

Mark provided the impetus to start the DBE project. Ian wrote the first draft of the specification. David served as architect.

Chapter 7. References

Jeffrey Friedberg, Larry Seiler, and Jeff Vroom, "Multi-buffering Extension Specification Version 3.3."

Tim Glauert, Dave Carver, Jim Gettys, and David P. Wiggins, "X Synchronization Extension Version 3.0."