

# TransDisplay

## A TransSkel Display Window Module

Version 1.0  
11 November 1986

### Introduction

This document describes *TransDisplay*, a plug-in module that runs on top of the *TransSkel* Macintosh application skeleton, and that may be added to any *TransSkel* project to provide an arbitrary number of text display windows. It may be used, for instance, to provide a debug output window without disturbing the normal operation of the application under development, or to display on-line documentation. *TransDisplay* provides no editing capabilities; applications requiring text editing windows may use *TransEdit* instead.

*TransDisplay* provides standard document windows that may be dragged and resized in the usual manner. Output written to the window is displayed and remembered (it's stored in a standard TextEdit record), so anything that goes out of view is not lost, but may be scrolled back for review. To prevent overflow, the text is autoflushed every so often. Autoflush behavior is configurable. Display windows may also be flushed manually by the host.

The host application may exert quite a bit of control over display windows if it wishes, but the minimum investment required to use them is small. For instance, to maintain a help window, the window is created with a single call and the help text is written with another call. To maintain a debug output window, the procedure is similar, except that text is written to the window intermittently with output calls placed at arbitrary places in the host, rather than with a single call at the time the window is created.

At little cost, the host may provide a mechanism allowing the window to be made visible and invisible under user control: a single menu item and a few lines of code suffices.

The window and its data structures may also be destroyed anytime during host execution. Display windows not destroyed explicitly by the host are disposed of automatically by the usual *TransSkel* mechanism (*i.e.*, when the host calls `SkelClobber`). If the destroyed window happens to be the current display output window, output is automatically turned off.

Output to a display window may be turned on or off at any time. For debugging purposes especially, this provides an alternative to the insertion / deletion / reinsertion or commenting / uncommenting or `#ifdef`'ing of debug statements. For instance, a menu item may be established during development to allow run-time toggling of debug output, or a dialog might be provided for selecting one of several levels of debug output. Also, if the window creation call is deleted, all output calls are implicitly disabled and do not need to be deleted. This provides a convenient (*i.e.*, trivial-effort) compile-time mechanism for controlling debug output.

The set of built in output-writing procedures is very simple-minded, consisting of calls for writing one object each of the following types: text, string, char, int, long, boolean (no floating point). However, *TransDisplay* is not tied to any particular output convention. If the built-in calls are inconvenient or insufficient for particular applications, one may use `sprintf` to format a text object to be passed to *TransDisplay*. The `sprintf` routine is found in the LightspeedC stdio library.

Display windows may be told to report activate/deactivate events to the host. This is useful for applications that enable or disable menu items according to which window is frontmost.

# TransDisplay 1.0 Manual

## 2

Those applications not requiring multiple windows may `#define` a symbol in the source to cause *TransDisplay* to compile in single-window mode. This results in less code. The interface is identical in both cases. (The term “single-window” is slightly ambiguous. It means a single window at any one time. One may create and destroy any number of display windows in sequence.)

## Distribution Information

*TransSkel* and *TransDisplay* are public domain, so distribution is unrestricted. I am interested in hearing about any additions or corrections, for possible inclusion in future releases. I may be reached via U.S. mail at:

Paul DuBois  
Wisconsin Regional Primate Research Center  
1220 Capitol Court  
Madison, WI 53706 USA

or via electronic mail at

UUCP: {allegro, ihnp4, seismo}!uwvax!uwmacc!dubois  
ARPA: dubois@unix.macc.wisc.edu  
dubois@rhesus.primate.wisc.edu

The version of *TransDisplay* described in this document is written for LightspeedC. LightspeedC is a trademark of:

THINK Technologies, Inc.  
420 Bedford Street Suite 350  
Lexington, MA 02173 USA

This distribution of *TransDisplay* consists of:

The document “TransDisplay—A TransSkel Display Window Module”

The document “TransDisplay 1.0 Quick Reference”

TransDisplay.h - *TransDisplay* header file.

TransDisplay.c - *TransDisplay* source.

Demonstration programs:

MiniDisplay.c - minimal demonstration.

EventLog.c - event logging demonstration.

EventLog.proj.rsrc - resource file for EventLog demonstration.

The remainder of this document describes the demonstration programs included in the distribution and provides a detailed specification of the *TransDisplay* interface. Familiarity with *TransSkel* is assumed.

## Demonstration Programs

The demonstrations are an introduction to the ways in which *TransDisplay* can be used. MiniDisplay shows the minimum amount of work necessary by the host to use a display window. EventLog demonstrates how a display window may be fully integrated into an application, including appropriate menu item enabling/disabling.

### MiniDisplay

# TransDisplay 1.0 Manual

## 3

This demonstration puts up an Apple menu with desk accessories in it, a File menu with a Quit item and a single display window. The window displays a minimal amount of text, demonstrating the available output calls. Desk accessories may be run as usual. Terminate the application by selecting Quit from the File menu, or by typing command-Q.

### EventLog

This demonstration uses multiple display windows. One is a help window (supplied with text from a resource), while the other is a window that reports events. Another (non-display) window is used to select the types of events that are logged. This demonstration shows how to change text attributes of display windows.

### Note

EventLog requires *TransSkel* version 1.01 or higher.

## The *TransDisplay* Interface—General Information

TransDisplay.c contains the source of the *TransDisplay* module. It can be made into a project for inclusion in the host application project, or the source can be included directly in the host project. *TransDisplay* compiles to different amounts of code, depending on whether it is compiled in single-window or multiple-window mode (2.6K and 3.0K, respectively). You may wish to compile two projects, one for single-window and one for multiple-window mode.

The available calls are:

<b>NewDWindow</b>	Create display window
<b>GetNewDWindow</b>	Create display window from resource template
<b>SetDWindow</b>	Set window used for output
<b>GetDWindow</b>	Get window currently used for output
<b>GetDWindowTE</b>	Get TextEdit record associated with window
<b>SetDWindowNotify</b>	Install activate notification procedure
<b>SetDWindowStyle</b>	Set window text display characteristics
<b>SetDWindowFlush</b>	Set autoflush parameters
<b>SetDWindowPos</b>	Scroll window to given line
<b>FlushDWindow</b>	Flush output from display window
<b>IsDWindow</b>	Test whether window is a display window.
<b>DisplayText</b>	Write text to display window
<b>DisplayString</b>	Write string to display window
<b>DisplayChar</b>	Write character to display window
<b>DisplayInt</b>	Write integer to display window
<b>DisplayLong</b>	Write long integer to display window
<b>DisplayHexChar</b>	Write hex character to display window
<b>DisplayHexInt</b>	Write hex integer to display window
<b>DisplayHexLong</b>	Write hex long integer to display window
<b>DisplayBoolean</b>	Write boolean to display window
<b>DisplayLn</b>	Write carriage return to display window

All other variables and procedures are declared `static`, to preclude name conflicts with the host. The interface to the host application is mostly procedural, but the header file `TransDisplay.h` should be `#include'd` in source files containing *TransDisplay* calls.

# TransDisplay 1.0 Manual

## 4

The general logic of host applications using *TransDisplay* is:

- (i) Initialize *TransSkel*, plus whatever other initialization is desired.
- (ii) For each display window to be used, call *NewDWindow* or *GetNewDWindow* to create it.
- (iii) Set the window to be written to with *SetDWindow* and write to it (unnecessary unless more than one display window is used).
- (iv) To destroy display windows explicitly, call *SkelRmveWind*. Otherwise, display windows are disposed of automatically when the host calls *SkelClobber*.

## The *TransDisplay* Interface—Procedural Specification

Each of the *TransDisplay* interface routines is described in detail below.

### *Note*

The value of *nil* is understood to be equal to 0L (long zero).

Except where noted, routines expecting a *WindowPtr* to a display window do nothing if the pointer is not pointing to a display window.

### *Warning*

Some of the routines below take procedure addresses as parameters. All procedures passed to *TransDisplay* routines should be C procedures. Do not pass addresses of ToolBox routines unless you have a predilection for bomb boxes.

## Standard Interface Routines—Control Routines

**WindowPtr NewDWindow (boundsRect, title, visible, behind,  
goAwayFlag, refCon)**

**Rect**            **\*boundsRect;**  
**StringPtr**    **title;**  
**Boolean**       **visible;**  
**WindowPtr**    **behind;**  
**Boolean**       **goAwayFlag;**  
**long**          **refCon;**

*NewDWindow* creates a new display window and makes it the current window for display output. The *WindowPtr* of the new window is the return value. All the other parameters have the same meanings as the corresponding parameters of the ToolBox routine *NewWindow* (see *Inside Macintosh*). The window is created as a standard document window, with a size box and a scroll bar along the right edge. (The window is subject to whatever the current *TransSkel* window sizing defaults are. They may be changed with *SkelGrowBounds* in the usual manner.)

The initial defaults for display window creation are as follows. The text display characteristics of the window are set to monaco 9-point, word wrap on, and left justification. The autoflush values are: allow 30,000 characters maximum, flush 25,000 when that limit is exceeded. There is no activation/deactivation notification procedure. These may be changed with *SetDWindowStyle*, *SetDWindowFlush* and *SetDWindowNotify*.

To destroy the display window and its data structures, pass the window pointer to *SkelRmveWind*.

When *TransDisplay* is compiled in single-window mode, *NewDWindow* does nothing and returns *nil* if a display window already exists. Once that window has been destroyed, it allows another window to be created.

### *Note*

If the window being destroyed is the current display output window, output is implicitly turned off until the current display window is reset with `SetDWindow` or `NewDWindow`. Thus the host may blithely destroy display windows with impunity. (Not that blitheness juxtaposes well with impunitance...)

```
WindowPtr GetNewDWindow (rsrcId, behind)
int      rsrcId;
WindowPtr behind;
```

`GetNewDWindow` is like `NewDWindow` except that it creates the window from the WIND resource with the given ID number.

```
SetDWindowNotify (theWind, pNotify)
WindowPtr  theWind;
ProcPtr    pNotify;
```

`SetDWindowNotify` associates a procedure with `theWind`, to be called whenever `theWind` receives an activate or deactivate event. The procedure should be declared to take one boolean parameter, which will be `true` if `theWind` is coming active, `false` if it's going inactive. When the notification procedure is called, the display window to which the event applies is the current port, and may be obtained with the QuickDraw procedure `GetPort`. This is useful if the procedure is associated with more than one window. *TransDisplay* handles activating the window properly (e.g., highlighting the scroll bar and drawing the size box appropriately), before calling the notification procedure.

If `theWind` is a display window and `pNotify` is `nil`, notification is turned off. If `theWind` is `nil`, `pNotify` becomes the default activation procedure associated with new display windows created with `NewDWindow` or `GetNewDWindow` subsequently.

### *Note*

Notification is useful mainly for applications that change enabling of menu items according to which window is frontmost. No special treatment is necessary for display windows created with a close box: The window handler simply hides the window when the box is clicked, which generates a deactivate event that can be detected with the notification procedure.

There is no notification when a display window is clobbered. If the window is clobbered at the end of application execution, the host doesn't need to know. If the window is clobbered during execution, the host must be the one telling *TransSkel* to shut down the window, and so is assumed to be the cognoscente knowing what additional actions to take.

```
SetDWindowStyle (theWind, fontNum, fontSize, wordWrap,
                 justification)
WindowPtr  theWind;
int        fontNum;
int        fontSize;
int        wordWrap;
int        justification;
```

## TransDisplay 1.0 Manual

### 6

`SetDWindowStyle` sets the text display characteristics for `theWind`. The value of `wordWrap` should be non-negative to specify wrapping on, negative to specify wrapping off. The justification values are 0, 1 or -1 to specify left, center or right justification, respectively. These are simply the justification constants found in `TextEdit.h`:

```
teJustLeft = 0
teJustCenter = 1
teJustRight = -1
```

If `theWind` is `nil`, the style parameters become the defaults for new display windows created with `NewDWindow` or `GetNewDWindow` subsequently.

**SetDWindowFlush (theWind, maxSize, flushSize)**

```
WindowPtr    theWind;
long          maxSize;
long          flushSize;
```

`SetDWindowFlush` configures the autoflush behavior of `theWind`. `maxSize` determines the maximum number of text characters allowed in the window. `flushSize` determines how many characters are flushed when the text grows beyond `maxSize` characters. Neither value may be set less than 100 characters.

If `theWind` is `nil`, the flush parameters become the defaults for new display windows created with `NewDWindow` or `GetNewDWindow` subsequently.

**FlushDWindow (theWind, byteCount)**

```
WindowPtr    theWind;
long          byteCount;
```

`FlushDWindow` removes the first `byteCount` bytes from the current text of `theWind`. If there are not that many characters of text, the effect is to empty the window.

**SetDWindow (theWind)**

```
WindowPtr    theWind;
```

`SetDWindow` makes `theWind` the current display window; subsequent output is written to that window. Pass `nil` to turn output off completely (output calls are then ignored until output is turned on again). If `theWind` is not a display window and is not `nil`, `SetDWindow` does nothing.

`SetDWindow` preserves the current port.

**GetDWindow (theWind)**

```
WindowPtr    *theWind;
```

`GetDWindow` returns the current display window in `theWind`.

### **Warning**

This value will be `nil` if output is currently turned off. *Check* the value before you pass it somewhere else!

# TransDisplay 1.0 Manual

## 7

```
Boolean IsDWindow (theWind)
WindowPtr    theWind;
```

IsDWindow returns `true` if `theWind` is a display window, `false` otherwise.

```
Boolean SetDWindowPos (theWind, lineNum)
WindowPtr    theWind;
int          lineNum;
```

SetDWindowPos scrolls the text in `theWind` so that the given line is at the top of the window, if possible. This is useful mainly for scrolling to the top of the text (`lineNum = 0`), or the bottom (`lineNum = some large number, like 32767`).

```
TEHandle GetDWindowTE (theWind)
WindowPtr    theWind;
```

GetDWindowTE returns a handle to the TextEdit record associated with `theWind`, or `nil` if it's not a display window. This call allows the host to perform arbitrary text operations not supported by the standard *TransDisplay* calls.

## Standard Interface Routines—Output Routines

All display routines write to the current display window, and scroll the new output into view if necessary. No output is written if the current display window has been set to `nil` with `SetDWindow`, or if no call has been made to `NewDWindow`. By implication, you can make disable all output calls by deleting display window creation calls from the host. If they are replaced later, output capability is restored without the need for deleting and replacing the output calls themselves.

The current port is preserved across all output calls.

```
DisplayText (t, len)
Ptr         t;
long        len;
```

DisplayText writes arbitrary text.

```
DisplayString (s)
StringPtr     s;
```

DisplayString writes the string, which should be in Pascal style.

```
DisplayChar (c)
char        c;
```

DisplayChar writes the character.

# TransDisplay 1.0 Manual

## 8

**DisplayHexChar (c)**  
**char c;**

DisplayHexChar writes the value of the character as 2-digit hex number.

**DisplayInt (i)**  
**int i;**

DisplayInt writes the value of the integer.

**DisplayHexInt (i)**  
**int i;**

DisplayHexInt writes the value of the integer as a 4-digit hex number.

**DisplayLong (l)**  
**long l;**

DisplayLong writes the value of the long integer.

**DisplayHexLong (l)**  
**long l;**

DisplayHexLong writes the value of the long integer as an 8-digit hex number.

**DisplayBoolean (b)**  
**Boolean b;**

DisplayBoolean writes the string "\ptrue" if b is true, "\pfalse" otherwise.

**DisplayLn ()**

DisplayLn writes a carriage return to the display window.

## Using Notification Procedures

The notification procedure for a display window, if one is installed, is called whenever a display window is activated or deactivated. It is also called whenever the user clicks in the close box. *TransDisplay* installs window handlers with `nil` close procedures, so if the window is created with a close box, clicking in the close box causes *TransSkel* simply to hide the window. Since hiding a window generates a deactivate event, the notification procedure is called.

Generally, Macintosh applications allow the user the option of closing windows via a Close item in the File menu. If the host provides such an option, it should simply hide any display window that is frontmost when Close is selected.



```
if (IsDisplayWindow (FrontWindow ()))  
    HideWindow (FrontWindow ());
```

This generates a deactivate event, and the notification procedure will be called in the usual manner.

The question of what the notification procedure should do is a bit different. Typically, certain menu items are enabled or disabled. The host may also wish to destroy the display window altogether (perhaps to free up memory), rather than just leave it hidden. This can be done as follows:

```
Notify (active)  
Boolean active;  
{  
WindowPeek w;  
  
    if (!active)          /* check if invisible on deactivate */  
    {  
        GetPort (&w);  
        if (w->visible != 0)  
            SkelRmveWind (w);    /* destroy window */  
    }  
  
    /* set menu items appropriately here */  
}
```

The current port is obtained with `GetPort` since the port when a notification procedure is called always corresponds to the window being deactivated or activated. (If the host only uses one display window, or maps each display window onto a different notification procedure, then of course it does not need to find out which one is current, since it will know implicitly.)