

# **TransDisplay**

## **A TransSkel Display Window Module**

**Release 3.05**  
**26 February 1994**

### **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 can be 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.

With little effort, 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. If the destroyed window happens to be the current display output window, output is automatically turned off. Display windows not destroyed explicitly by the host are disposed of automatically by the usual TransSkel mechanism (i.e., when the host calls

`SkelCleanup()`).

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, Pascal or C string, char, short, long, boolean, and OS type (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 THINK C stdio library.

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

## Distribution Information

---

TransDisplay is 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 electronic mail at *dubois@primate.wisc.edu* or via U.S. mail at:

Paul DuBois  
Wisconsin Regional Primate Research Center  
1220 Capitol Court  
Madison, WI 53715-1299  
USA

The version of TransDisplay described in this document is written for THINK C 6.0. THINK C is a trademark of:

Symantec Corporation  
10201 Torre Avenue  
Cupertino, CA 95014 USA

This distribution of TransDisplay consists of:

### Documentation

- Release Notes
- Manual

### TransDisplay library C source code

- TransDisplay.c — TransDisplay source

### Demonstration application source code

- MiniDisplay — minimal demonstration
- EventLog — event logging demonstration

### Interface files

- TransDisplay.h — C header file
- TransDisplay.intf — Pascal interface file
- TransDisplay — binary TransDisplay library document

## **TransDisplay 3.0 Manual**

---

The remainder of this document describes the demonstration programs included in the distribution and provides a detailed specification of the TransDisplay interface. TransDisplay 3.0 requires TransSkel 3.0. Familiarity with TransSkel is assumed.

## **Demonstration Applications**

---

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**

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.

### The TransDisplay Interface — General Information

---

*TransDisplay.c* contains the source of the TransDisplay module. It can be made into a project or library document for inclusion in the your application project document, or the source can be included directly in your project.

The available calls are:

<code>NewDWindow()</code>	Create display window
<code>GetNewDWindow()</code>	Create display window from resource template
<code>SetDWindow()</code>	Set window used for output
<code>GetDWindow()</code>	Get window currently used for output
<code>GetDWindowTE()</code>	Get TextEdit record associated with window
<code>SetDWindowNotify()</code>	Install activate notification procedure
<code>SetDWindowStyle()</code>	Set window text display characteristics
<code>SetDWindowFlush()</code>	Set autoflush parameters
<code>SetDWindowPos()</code>	Scroll window to given line
<code>FlushDWindow()</code>	Flush output from display window
<code>IsDWindow()</code>	Test whether window is a display window.
<code>DisplayText()</code>	Write text
<code>DisplayString()</code>	Write Pascal string
<code>DisplayCString()</code>	Write C string
<code>DisplayChar()</code>	Write character
<code>DisplayShort()</code>	Write short integer
<code>DisplayLong()</code>	Write long integer
<code>DisplayHexChar()</code>	Write character in hex
<code>DisplayHexShort()</code>	Write short integer in hex
<code>DisplayHexLong()</code>	Write long integer in hex
<code>DisplayBoolean()</code>	Write boolean

## TransDisplay 3.0 Manual

---

DisplayOSType()

Write OS type

DisplayLn()

Write carriage return

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

The general logic of host applications using TransDisplay is:

- Initialize TransSkel, plus whatever other initialization is desired.

## **TransDisplay 3.0 Manual**

---

- For each display window to be used, call `NewDWindow()` or `GetNewDWindow()` to create it.
- Set the window to be written to with `SetDWindow()` and write to it (unnecessary unless more than one display window is used).
- To destroy display windows explicitly, call `SkelRmveWind()` for each one. Otherwise, display windows are disposed of automatically when the host calls `SkelCleanup()`.

## **The TransDisplay Interface — Procedural Specification**

---

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

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

### **Standard Interface Routines — Control Routines**

pascal `WindowPtr`

```
NewDWindow (Rect *bounds, StringPtr title, Boolean visible,  
            WindowPtr behind, Boolean goAway, 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. A pointer to the new window is the return value. If the window could not be created the return value is `nil`. The window is subject to whatever the current TransSkel window sizing defaults are. They may be changed with `SkelSetGrowBounds()` in the usual manner.

The default text display characteristics for display windows are monaco 9-point font, word wrap on, and left justification. The default autoflush values are: allow 30,000 characters maximum, flush 25,000 when that limit is exceeded. By default there is no activation/deactivation notification procedure. These values may be changed with `SetDWindowStyle()`, `SetDWindowFlush()`, and `SetDWindowNotify()`.

To destroy the display window and its data structures, pass the window pointer to `SkelRmveWind()`.

### Note

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

```
pascal WindowPtr
```

```
GetNewDWindow (short resourceNum, WindowPtr behind);
```

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

```
pascal void
```

```
SetDWindowNotify (WindowPtr w, TDispActivateProcPtr p);
```



`SetDWindowNotify()` associates a procedure `p` with `w`, to be called whenever `w` receives an activate or deactivate event. The procedure should be declared to take one boolean parameter, like so:

```
pascal void
MyNotify (Boolean active)
{
}
```

The parameter will be `true` if `w` 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 you associate the procedure 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 `w` is a display window and `p` is `nil`, notification is turned off. If `w` is `nil`, `p` becomes the default activation procedure associated with new display windows created with subsequent calls to `NewDWindow()` or `GetNewDWindow()`.

### 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. Thus, if you want to destroy a display window when it's closed, check whether the window is still visible when it goes inactive. If not, call `SkelRmveWind()` to remove it.

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 know what additional actions to take, if any.

```
pascal void
SetDWindowStyle (WindowPtr w,
                 short font,
                 short size,
                 short wrap,
                 short just);
```

`SetDWindowStyle()` sets the text display characteristics for `w`. The value of `wordWrap` should be non-negative to specify wrapping on, negative to specify wrapping off. The justification values are the usual TextEdit justification constants `teJustLeft`, `teJustCenter`, and `teJustRight` to specify left, center or right justification, respectively.

If `w` is `nil`, the style parameters become the defaults for new display windows created with subsequent calls to `NewDWindow()` or `GetNewDWindow()`.

```
pascal void
SetDWindowFlush (WindowPtr w, long maxText, long flushAmt);
```

## TransDisplay 3.0 Manual

---

`SetDWinFlush()` configures the autoflush behavior of `w`. `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 `w` is `nil`, the flush parameters become the defaults for new display windows created with subsequent calls to `NewDWindow()` or `GetNewDWindow()`.

```
pascal void
FlushDWindow (WindowPtr w, long byteCount);
```

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

```
pascal void
SetDWindow (WindowPtr w);
```

`SetDWindow()` select `w` as 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 `w` is not a display window and is not `nil`, `SetDWindow()` does nothing.

`SetDWindow()` preserves the current port.

```
pascal WindowPtr
GetDWindow (void);
```

`GetDWindow()` returns the current display window.

### Warning

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

```
pascal Boolean
IsDWindow (WindowPtr w);
```

`IsDWindow()` returns `true` if `w` is a display window, `false` otherwise.

```
pascal void
SetDWindowPos (WindowPtr w, short lineNum);
```

`SetDWindowPos()` scrolls the text in `w` 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`).

```
pascal TEHandle
GetDWindowTE (WindowPtr w);
```

`GetDWindowTE()` returns a handle to the TextEdit record associated with `w`, 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 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.

```
pascal void
DisplayText (Ptr theText, long len);
```

`DisplayText()` writes arbitrary text.

```
pascal void
DisplayString (StringPtr str);
```

`DisplayString()` writes the string, which should be a Pascal string.

```
pascal void
DisplayCString (char *str);
```

`DisplayCString()` writes the string, which should be a C string.

This function isn't very useful from within Pascal applications.

```
pascal void
DisplayChar (short c);
```

`DisplayChar()` writes the character.

```
pascal void
DisplayHexChar (short c);
```

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

```
pascal void
DisplayShort (short i);
```

`DisplayShort()` writes the value of the short integer.

```
pascal void  
DisplayHexShort (short i);
```

`DisplayHexShort()` writes the value of the short integer as a 4-digit hex number.

```
pascal void  
DisplayLong (long l);
```

`DisplayLong()` writes the value of the long integer.

```
pascal void
```

## TransDisplay 3.0 Manual

---

```
DisplayHexLong (long l);
```

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

```
pascal void  
DisplayBoolean (Boolean b);
```

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

```
pascal void  
DisplayOSType (OSType type);
```

`DisplayOSType()` writes the four-character value of the given OS type. It doesn't print any single-quotes around the value.

```
pascal void  
DisplayLn (void);
```

`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:

## TransDisplay 3.0 Manual

---

```
pascal void
Notify (Boolean active)
{
    WindowPtr    w;

    if (!active)                /* check if invisible on deactivate */
    {
        GetPort (&w);
        if (((WindowPeek) w)->visible == 0)
            SkelRmveWind (w);    /* destroy window */
    }

    /* set menu items appropriately here */
}
```

## TransDisplay 3.0 Manual

---

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.)

An alternative to setting menu items when a display window becomes active or inactive is to install a menu hook to be called whenever the mouse is clicked in the menu bar. The hook procedure can check what sort of window is frontmost and set menu items accordingly. You can use `SkelSetMenuHook()` to install such a hook.