Writing Reusable Code

- or -

The Disinfectant 2.0 Online Document and Help System

- or -

Abusing the List Manager for Fun and No Profit

Presented on June 14, 1990
at MacHack ’90

John Norstad
Northwestern University

jln@nuacc.bitnet
jln@acns.nwu.edu
A0173 (AppleLink)
76666,573 (CompuServe)

Outline

STR# Escape Sequences

Strings in the STR# resources may begin with one or more “escape sequences,” which all have the following common format:

byte 0 = escape code = a number between 0 and 30 (non-printable ASCII code).
byte 1 = number of bytes in the escape sequence.
remaining bytes = parameters.

Style	1/0, 1/3, 1/style (as in QuickDraw)
Justification	1/1, 1/3, 1/justification (0=left, 1=center, 2=right)
Size	1/2, 1/4, 2/percent size change
Only	1/3, 1/3, 1/flags (bit 0=screen, bit 1=print, bit 2=save)
Picture	1/4, 1/6, 2/PICT rsrc id, 2/band number
Page	1/5, 1/4, 2/page number
Keep	1/6, 1/2
Endkeep	1/7, 1/2
Insert TCON	1/8, 1/4, 2/extra space in 1/100 inch

The document has three possible destinations:

The screen, as presented to the user in the help window.
A printer.
A saved disk file.

The “only” escape sequence can be used to specify that this line is to be output to only one or two of the three possible destinations.

Many of the escape sequences have meaning for only some of the possible destinations. The following table shows the possibilities.

	Screen	Print	Save
Style	*	*	
Justification	*	*	
Size		*	
Only	*	*	*
Picture	*	*	
Page		*	
Keep		*	
Endkeep		*	
Insert TCON		*	

Note that saved documents contain plain unformatted text only, without pictures.

The “size” escape sequence has meaning only for printed documents. Why not on the screen too? Because the List Manager works only with fixed size cells, and we cannot adjust the size of those cells. We get around this restriction by using pictures in those few places where we want larger font sizes (document and chapter titles).

For pictures multiple lines are present in the STR# resource, each in the format described above. Enough lines are present to completely cover the picture in bands of the same height as the List Manager cell height. The band numbers in the escape sequences increment from 0 to the number of bands - 1.

The custom ldef keeps a one picture bitmap cache to speed up the drawing of pictures. CopyBits is used to copy individual bands from the bitmap to the cells on the screen.

The page, keep, endkeep, and itcon directives are used only for printing, and they always occur on lines by themselves, preceded by an “only” escape sequence that says the line only applies to printing. We’ll return to discuss these printing directives in more detail later.

The last line of each paragraph ends with a special “end of paragraph” marker - the special code 31. This marker is used by the printing code to tell where each paragraph ends when doing word wrap.

The List Manager cell array and cell data array only contain entries for those lines which apply to the screen version of the document.

```
/*____________________________________________________________________

	doc.h - Report Module Interface.

	Copyright © 1988, 1989, 1990 Northwestern University.  Permission is
	granted to use this code in your own projects, provided you give
	credit to both John Norstad and Northwestern University in your about
	box or document.

	The definitions in this module are used by the rep.c, rpp.c, ldf.c,
	hlp.c, and cvrt.c modules.  They are private to these modules.
	This header file should not be included or used by any other modules.
____________________________________________________________________*/

/* For type 1 reports the userHandle field of the list record is used to
	store a handle to auxiliary information used to optimize the
	performance of the list definition function.  The following typedef
	describes this information. */

typedef struct auxInfo {
	short			cachedPictID;	/* resource id of cached picture,
								or 0 if none */
	BitMap		cachedBitMap;	/* cached picture bitmap */
	Handle		auxArray[1];	/* array of handles to STR# resources -
								expanded by rep_Init to whatever size
								is required, and terminated with a zero
								entry */
} auxInfo;

/* Escape codes used in STR# lines. */

#define	docStyle	0
#define	docJust		1
#define	docSize		2
#define	docOnly		3
#define	docPict		4
#define	docPage		5
#define	docKeep		6
#define	docEndKeep	7
#define	docITcon		8

/* End-of-paragraph marker. */

#define	docEop		31

/* Justification constants used in STR# escape sequences. */

#define	docLeft		0
#define	docCenter	1
#define	docRight	2

/* Masks for STR# "only" escape sequences. */

#define	docScreen	1
#define	docPrint	2
#define	docSave		4
```

```
/*______________________________________________________________________

	ldf.c - List Definition Procedure for the Report Module.
	
	Copyright © 1988, 1989, 1990 Northwestern University.  Permission is
	granted to use this code in your own projects, provided you give
	credit to both John Norstad and Northwestern University in your
	about box or document.
	
	This LDEF is used by type 1 reports.  It ignores all but draw
	messages.  The cell data specifies the index in the auxiliary array
	of a handle to an STR# resource, and the offset of the line within
	the resource.
_____________________________________________________________________*/


#pragma load "precompile"
#include "doc.h"


pascal void MyList (short lMessage, Boolean lSelect,
	Rect *lRect, Cell *lCell, short lDataOffset,
	short lDataLen, ListHandle lHandle)

{
	auxInfo			**aux;			/* handle to auxiliary info */
	Handle			theCells;		/* handle to cell data */
	unsigned char	*p;				/* pointer to cell data */
	Handle			theStrings;		/* handle to STR# resource */
	unsigned short	offset;			/* offset of line in STR# resource */
	unsigned char	*theLine;		/* pointer to the line */
	unsigned char	*q;				/* pointer to cur pos in the line */
	unsigned char	*qEnd;			/* pointer to end of line */
	short				nchar;			/* number of chars to draw */
	short				baseLine;		/* base line for text */
	Boolean			escStyle;		/* true if style escape sequence */
	Boolean			escJust;		/* true if just escape sequence */
	Boolean			escPict;		/* true if pict escape sequence */
	unsigned char	styleCode;		/* style */
	unsigned char	justCode;		/* justification */
	short				cellHeight;		/* cell height */
	short				cellWidth;		/* cell width */
	short				picID;			/* pict resource id */
	short				picBand;		/* pict band number */
	PicHandle		picHandle;		/* handle to pict */
	short				picWidth;		/* pict width */
	short				picHeight;		/* pict height */
	Rect				picRect;		/* pict offscren rectangle */
	Rect				picSrcRect;		/* CopyBits source rect */
	Rect				picDstRect;		/* CopyBits dest rect */
	GrafPort		picPort;		/* grafport for offscreen PICT drawing */
	BitMap			picMap;			/* bitmap for offscreen PICT drawing */
```

```
/* Get pointer to the line to be drawn. */

if (lMessage != lDrawMsg) return;
aux = (auxInfo**)(**lHandle).userHandle;
theCells = (**lHandle).cells;
p = *theCells + lDataOffset;
theStrings = (**aux).auxArray[*p++];
offset = (*p << 8) | *(p+1);
if (!*theStrings) LoadResource(theStrings);
HLock(theStrings);
theLine = *theStrings + offset;

/* Get escape sequence info. */

escStyle = escJust = escPict = false;
q = theLine+1;
qEnd = q + *theLine;
while (q < qEnd && *q < 31) {
    switch (*q) {
        case docStyle:
            escStyle = true;
            styleCode = *(q+2);
            break;
        case docJust:
            escJust = true;
            justCode = *(q+2);
            break;
        case docPict:
            escPict = true;
            picID = *(q+2)<<8 | *(q+3);
            picBand = *(q+4)<<8 | *(q+5);
            break;
    };
    q += *(q+1);
};

if (escPict) {

    /* Draw a picture. */

    cellWidth = lRect->right - lRect->left;
    cellHeight = (**lHandle).cellSize.v;

    if ((**aux).cachedPictID != picID) {

        /* This picture is not cached - we must cache it in an offscreen
            bitmap. */

        /* Dispose of any previously cached bitmap. */

        if ((**aux).cachedPictID) DisposPtr((**aux).cachedBitMap.baseAddr);
```

```
	/* Compute picRect = the bounds rectangle for the cached
		picture. */

	picHandle = GetPicture(picID);
	if (!picHandle) {
		HUnlock(theStrings);
		return;
	};
	if (!*picHandle) LoadResource((Handle)picHandle);
	HLock((Handle)picHandle);
	picWidth = (**picHandle).picFrame.right -
		(**picHandle).picFrame.left;
	picHeight = (**picHandle).picFrame.bottom -
		(**picHandle).picFrame.top;
	if (!escJust) justCode = docCenter;
	switch (justCode) {
		case docLeft:
			picRect.left = lRect->left + 4;
			break;
		case docCenter:
			picRect.left = lRect->left +
				((cellWidth - picWidth)>>1);
			break;
		case docRight:
			picRect.left = lRect->right - 4 - picWidth;
			break;
	};
	picRect.right = picRect.left + picWidth;
	picRect.top = 0;
	picRect.bottom = picHeight;

	/* Allocate and initialize the offscreen  bitmap. */

	(**aux).cachedPictID = picID;
	picMap.bounds = picRect;
	picMap.rowBytes = (((picWidth+7)>>3) + 1) & 0xfffe;
	picMap.baseAddr = NewPtr(picMap.rowBytes*picHeight);
	(**aux).cachedBitMap = picMap;

	/* Draw the picture in the offscreen bitmap. */

	OpenPort(&picPort);
	SetPortBits(&picMap);
	picPort.portRect = picRect;
	RectRgn(picPort.visRgn, &picRect);
	ClipRect(&picRect);
	EraseRect(&picRect);
	DrawPicture(picHandle, &picRect);
	HUnlock((Handle)picHandle);
	ClosePort(&picPort);
	SetPort((**lHandle).port);
};
```

```
		/* CopyBits the proper band from the offscreen cached bitmap to
			the cell. */

		picMap = (**aux).cachedBitMap;
		picSrcRect = picMap.bounds;
		picSrcRect.top = picBand*cellHeight;
		picSrcRect.bottom = picSrcRect.top + cellHeight;
		if (picSrcRect.bottom > picMap.bounds.bottom)
			picSrcRect.bottom = picMap.bounds.bottom;
		picDstRect = picSrcRect;
		picDstRect.top = lRect->top;
		picDstRect.bottom = picDstRect.top + picSrcRect.bottom - picSrcRect.top;
		CopyBits(&picMap, &(**lHandle).port->portBits, &picSrcRect,
			&picDstRect, srcCopy, nil);

	} else {

		/* Draw a text line. */

		if (!escStyle) styleCode = normal;
		if (!escJust) justCode = docLeft;
		TextFace(styleCode);
		nchar = *theLine - (q - theLine - 1);
		if (nchar && *(q+nchar-1) == docEop) nchar--;
		baseLine = lRect->bottom - 2;
		switch (justCode) {
			case docLeft:
				MoveTo(lRect->left + 4, baseLine);
				break;
			case docCenter:
				MoveTo((lRect->left + lRect->right - TextWidth(q, 0, nchar)) >> 1,
					baseLine);
				break;
			case docRight:
				MoveTo(lRect->right - 4 - TextWidth(q, 0, nchar), baseLine);
				break;
		};
		DrawText(q, 0, nchar);
		TextFace(normal);
	};

	HUnlock(theStrings);
}
```

Printing

Disinfectant contains a fairly sophisticated text formatter whose sole purpose in life is to print very nicely formatted copies of the document. The printed output contains a title page, table of contents, page headers, pictures, intelligent page breaks, nicely wrapped paragraphs, and appropriate use of multiple font sizes and styles.

A customized page setup dialog permits the user to specify his own font, “nominal” font size, and margins, as well as an option to print pages in reverse order (on by default for LaserWriters). See Tech Note #95 for details on how to customize the printing dialogs.

Printing is done in two passes. The first pass walks throught the document (the STR# resources), computes page breaks, and saves them in an array. The second pass prints the document. The page break array is used to compute the page numbers for the table of contents, to quickly locate the proper pages to print if the user selects a page range in the print job dialog, and to make it possible to print pages in reverse order.

Paragraphs are rewrapped to fit the margins specified by the user. The special end of paragraph markers in the STR# resources are used to tell where paragraphs begin and end.

There are five special escape sequences in the STR# resources that are used just for printing.

The “size” escape sequence specifies that the line should be printed at some percentage of the nominal font size. In Disinfectant I use five font sizes:

90%	Page headers (bold).
100%	Paragraph text (nominal size selected by user).
120%	Section titles (left justified, bold).
140%	Chapter titles (centered, bold).
200%	Main document title on title page (centered, bold).

The “page” escape sequence forces a page eject. A page number of 0 means do not put a header on this page. A page number of 0xffff means do the usual - put a header on this page, and increment the page number by 1. Any other page number n means put a header on this page, and start page numbering on this page with number n.

The “keep” and “endkeep” escape sequences delimit “keep blocks.” Keep blocks are never split across page boundaries. For example, I use this to keep a section title and the first paragraph or two of the section together. Paragraphs and pictures are “implied” keep blocks - they are never split across pages.

The “itcon” escape sequence inserts the table of contents.

Context-Sensitive Help

Disinfectant has a context sensitive help system to make it easier to locate information in the document. The user types Command-?, and the cursor turns into a question mark. The user can then click on any object on the screen to get help on that object. The help window is opened (or brought to the front if it is already open) and scrolled to the section of the document describing that object.

The user can also select a menu item in this “help mode” to get help on that menu item.

Clicking on an error message in the report goes to the document section describing that error message. Clicking on an infection message in the report goes to the document section describing the virus.

The system is implemented by assigning a 16 bit “tag” to each document location which we need to locate. A special TAG resource contains one longword entry for each assigned tag. The entry specifies the tag and the line number (cell number) in the document of the associated first line of text for that tag.

Implementing help for screen objects and menu commands is quite simple. If the user clicks on a screen object or selects a menu command in help mode, we simply call the help module with the fixed tag for that object specified as a parameter. The help module opens the help window or brings it to the front, looks up the line number for the specified tag in the TAG resource, and calls the List Manager to scroll to that cell.

Implementing help for error and infection messages in the report is a bit more complicated, since we must keep track of which tags are associated with which lines in the report. Whenever an error or infection message is issued by the scanning code, it records the tag associated with that message and the current report line number in a table. When the user clicks in the report rectangle in help mode, the List Manager is called to determine which line the user clicked on. The table is then searched to discover which tag (if any) is associated with this line. Then the help module is called and passed the tag.

The Wrap and Cvrt Tools

The source for the Disinfectant document is maintained with Microsoft Word as a plain text file (no formatting), with each paragraph stored on the file as one long line (carriage returns only at the ends of paragraphs).

The text for the document is intermixed with special formatting directives. Directives always occur on lines by themselves, and they always begin with a backslash (\) in column 1.

This source file is processed by two MPW tools named “wrap” and “cvrt”. Their job is to convert the text into the sequence of STR# resources used by the custom ldef, the printing code, and the saving code.

Here’s a list of all the directives:

\str#
: Start a new STR# resource.

\tcon title
: Table of contents entry. The title and current line number are saved in a special TCON resource. The TCON resource is used both to display the interactive table of contents in the help window, and to produce the printed table of contents. Table of contents entries can be directed to only one of the two possible destinations with a preceeding \only directive.

\tag nnn
: Record a tag in the TAG resource.

\style xxx xxx xxx
: The next line is drawn/printed in the specified style or styles. xxx may be any of the usual styles: normal, bold, italic, underline, etc.

\just xxx
: The next line or picture is drawn/printed with the specified justification. xxx may be left, center, or right. Left is the default for text, while center is the default for pictures.

\size xxx
: The next line is printed xxx% larger than normal. Printing only.

\only xxx xxx
: The next line or picture or tcon entry should only be output to the specified destination. xxx may be screen, print, or save.

\pict id
    Insert a copy of the PICT resource with the specified id.
\page nnn
    Start a new page. Printing only.
\keep
    Begin keep block. Printing only.
\endkeep
    End keep block. Printing only.
\itcon xspace
    Insert table of contents. Printing only. Xspace is extra space between lines, in 1/100 inch. The table of contents is centered horizontally.

The wrap tool has one primary job - to wrap the paragraphs to fit the margins in the help window. It is a “filter” - it reads standard input, and writes standard output. It has two parameters: the right margin in pixels, and a flag specifying that the special end of paragraph marker should be appended to the last line of each paragraph.

The output from the wrap tool is piped into the cvrt tool. Cvrt is more complicated than wrap. Cvrt’s jobs are to process the formatting directives, build the escape sequences, and output the STR#, TCON, and TAG resources.

A significant performance improvement is achieved by having cvrt precompute the List Manager cell data array and output it as a special CELL resource. At run time, when the help window list record is initialized, a handle to this CELL resource is simply “plugged into” the list manager record, instead of laboriously recomputing it. This is the optimization which makes the help window appear almost instantaneously.

The wrap and cvrt tools are called by the MPW makefile for Disinfectant. When the MS Word source file is modified, the makefile automatically calls the tools to rebuild the STR#, TCON, TAG, and CELL resources.

As an example, here's the beginning of the Disinfectant document source file. This is complicated stuff - the rest of the source file uses the special directives much more sparingly.

```
\page 0
\only screen
\tcon Disclaimer
\size 300

\only screen
\pict 204
\only print save
\style bold
\just center
\size 200
Disinfectant 2.0b1
\size 200

\just center
June 12, 1990
\size 200

\pict 200

\only print

\keep
\only print
\style bold
\just center
\size 140
Table of Contents
\only print

\itcon 0
\endkeep
```

\page 1
\only print
\tcon Disclaimer and Copyright Notice
\style bold
\size 120
Disclaimer and Copyright Notice