Nisca

Print Drivers for Windows™

Programmer's Reference

Version 4.15
Revised March 17, 2005

Copyright

Disclaimer

TAB Software Corp. has reviewed this manual thoroughly in order to make it an easy to use guide for your Nisca printer. All statements, technical information, and recommendations in this manual and in any guides or related documents are believed reliable, but the accuracy and completeness thereof are not guaranteed or warranted, and they are not intended to be, nor should they be understood to be, representations or warranties concerning the products described.

Your Nisca printer and software media included with your system have been sold to you subject to the limited warranties set forth in the warranty and license agreement enclosed with the respective product. All software is licensed AS IS, as described in the license agreement enclosed with the software diskette. Further, TAB Software reserves the right to make changes in the specifications of the products described in this manual at any time without notice and without obligation to notify any person of such changes.

Trademarks

The following are trademarks or registered trademarks of their respective companies: Microsoft, Windows 95, Windows NT, Nisca.

Contents

Programmer's Reference

Microsoft Windows operating systems do not inherently support functionality of card printers. Therefore, a custom programming interface must be supplied. Two interfaces have been designed to offer the most flexibility in accessing features specific to the Nisca card printers. The first interface is offered by using the WIN32 ExtEscape function. A 'C' header file has been included on the setup disk containing required definitions. The second interface provides embedded commands through the standard WIN32 text functions.

WIN32 ExtEscape Support

The WIN32 API has modified the use of the Escape function. The Escape function should only be used when passing predefined WIN32 ESC values such as QUERYESCSUPPORT. To use vendor specific ESC values, the new ExtEscape function should be used instead. The function as documented in the WIN32 Programmer's Manual:

The **ExtEscape** function allows applications to access capabilities of a particular device that are not available through GDI.

```
int ExtEscape( HDC hdc,          // handle to device context
               int nEscape,      // escape function
               int cbInput,      // number of bytes in input structure
               LPCSTR lpszInData,     // pointer to input structure
               int cbOutput,     // number of bytes in output structure
               LPSTR lpszOutData      // pointer to output structure
);
```

Parameters

hdc

Identifies the device context.

nEscape

Specifies the escape function to be performed.

cbInput

Specifies the number of bytes of data pointed to by the *lpszInData* parameter.

lpszInData

Points to the input structure required for the specified escape.

cbOutput

Specifies the number of bytes of data pointed to by the *lpszOutData* parameter.

lpszOutData

Points to the structure that receives output from this escape. This parameter must not be NULL if **ExtEscape** is called as a query function. If no data is to be returned in this structure, set *cbOutput* to 0.

Return Values

The return value specifies the outcome of the function. It is greater than zero if the function is successful, except for the QUERYESCSUPPORT printer escape, which checks for implementation only. The return value is zero if the escape is not implemented. A return value less than zero indicates an error. To get extended error information, call GetLastError.

Remarks

Use this function to pass a driver-defined escape value to a device.

Use the **Escape** function to pass one of the escape values defined by Windows to a device. **ExtEscape** might not work properly with the escapes defined by Windows. In particular, escapes in which *lpszInData* points to a structure that contains a member that is a pointer will fail.

Nisca Interface

A ‘C’ language header file has been included in the installation to provide all definitions required to interface to the print driver. To use the predefined interfaces, include “NiscaEsc.h” in any C/C++ program.

Set Document Settings

Escape Function: `ESC_SETDOCSETTINGS`
Value: `11009`

Data:

```
typedef struct
{
      WORD        dsSize;
      DWORD       dsMask;
      WORD        dsRotate180[2];
      WORD        dsDIBFormat[2];
      WORD        dsDitherMethod[2];
      WORD        dsDitherIntensity[2];
      WORD        dsGrayIntensity[2];
      WORD        dsKPanelUsage[2][4];
      WORD        dsOverlayLayer[2];
      PR5100_RECT dsPrintOverlayLayer[2];
      PR5100_RECT dsNonprintOverlayLayer[2];
      WORD        dsOverlaminate[2];
      WORD        dsOverlaminateLast;
      WORD        dsFlipCard;
      WORD        dsEjectCard;
      WORD        dsUVUsage[2];
      COLORREF    dsUVColor;
} NISCA_DOCSETTINGS;
```

Data Members

dsSize
Specifies the size, in bytes, of the NISCA_DOCSETTINGS structure.

dsMask

Specifies which of the remaining members in the NISCA_DOCSETTINGS structure have been initialized. Bit 0 (defined as DSM_ROTATE180) corresponds to dsRotate180; bit 1 (defined as DSM_DIBFORMAT) specifies dsDIBFormat, and so on. dsMask can be a combination of the following values:

DSM_ROTATE180
DSM_DIBFORMAT
DSM_DITHERMETHOD
DSM_DITHERINTENSITY
DSM_GRAYINTENSITY
DSM_KPANELUSAGE
DSM_OVERLAY
DSM_OVERLAYRECTS
DSM_OVERLAMINATE
DSM_OVERLAMINATELAST
DSM_FLIPCARD
DSM_EJECTCARD
DSM_UV

dsRotate180

Rotate card side 180° before sending data to printer.

dsRotate180[0] — Card Front

dsRotate180[1] — Card Back

Possible Values

0 — Do not rotate side

1 — Rotate side 180°

dsDIBFormat

Color format for front and back of card.

dsDIBFormat[0] — Card Front

dsDIBFormat[1] — Card Back

Possible Values

DIBFMT_MONOCHROME
DIBFMT_GRAYSCALE
DIBFMT_COLOR

dsDitherMethod

Dither method if dsDIBFormat is DIBFMT_MONOCHROME.

dsDitherMethod[0] — Card Front

dsDitherMethod[1] — Card Back

Possible Values

DM_NONE
DM_COARSE
DM_ERRORDIFFUSION

dsDitherIntensity

Brightness intensity of the dithered image in terms of percentage of the current image. Default value is 150 percent brighter. Value greater than 100 will brighten the image and numbers less than 100 will darken the image. If values are too high, the resulting image will be pushed to white. Likewise if values are too low, the image will be pushed to black.

dsDitherIntensity[0] — Card Front

dsDitherIntensity[1] — Card Back

dsGrayIntensity

Brightness intensity of the gray scaled image. Default value is 100.

dsGrayIntensity[0] — Card Front

dsGrayIntensity[1] — Card Back

dsKPanelUsage

Ribbon usage for each of the four graphic components.

Components

KPU_TEXT	All printed text
KPU_MONOBITMAPS	1-bit per pixel bitmaps
KPU_PSEUDOBITMAPS	All bitmap resolutions that contain only black and white colors.
KPU_LINES	All lines and filled shapes

Resin Values

KPU_SIMULATEBLACK	Simulate black using YMC panels
KPU_RESINBLACK	Pure resin black with resin panel

Examples

```
docSettings.dsKPanelUsage[0][KPU_TEXT] = KPU_RESINBLACK;
docSettings.dsKPanelUsage[1][KPU_LINES] = KPU_SIMULATEBLACK;
```

dsOverlayLayer

Overlay settings for front and back of card.

Possible Values

PL_NONE	Do not apply overlay layer
PL_IMAGEONLY	Apply overlay to image area only
PL_IMAGEANDTEXT	Apply overlay to both image and text areas
PL_PRINTABLEAREA	Apply overlay to entire card surface
PL_USERDEFINED	Apply overlay to user-defined area

dsPrintProtectiveLayer

If dsProtectiveLayer is PL_USERDEFINED, this contains the area of the card in pixels, that will contain the protective layer.

dsNonprintProtectiveLayer

If dsProtectiveLayer is PL_USERDEFINED, this contains the area of the card in pixels, that will not contain the protective layer.

dsOverlaminate

Apply overlaminate to card.

dsOverlaminate[0] — Card Front

dsOverlaminate[1] — Card Back

Possible Values

0 — Do not apply overlaminate to card side

1 — Apply overlaminate to card side

dsOverlaminateLast

Apply the front side overlaminate after printing the back side.

Possible Values

0 — Apply front side overlaminate before printing back side

1 — Apply front side overlaminate after printing back side

dsFlipCard

Flip card prior to printing.

Possible Values

0 — Use default printer setting

1 — Do not flip card prior to printing

2— Flip card prior to printing

dsEjectCard

Eject card with specific side face up.

Possible Values

0 — Use default printer setting

1 — Eject card with top side up

2— Eject card with bottom side up

dsUVUsage

Enable support for UV ribbon panel usage.

dsUVUsage[0] — Card Front

dsUVUsage[1] — Card Back

Possible Values

0 — Do not use UV ribbon panel

1 — Use UV ribbon panel

dsUVColor

Color of graphic data to monitor and push to the UV panel. The default setting is magenta or Windows color RGB (255, 0, 255).

Possible Values

Any valid 32-bit color created using the WIN32 macro, RGB.

Call Position:

```
StartDoc ( );
  SetDocSettings ( );
  StartPage ( );
    . . .
  EndPage ( );
EndDoc ( );
```

Example:

```
void SetDocSettings ( HDC hdc )
{
      NISCA_DOCSETTINGS docSettings;

      docSettings.dsSize = sizeof ( NISCA_DOCSETTINGS );
      docSettings.dsMask = DSM_DIBFORMAT |
                           DSM_DITHERMETHOD |
                           DSM_DITHERINTENSITY |
                           DSM_KPANELUSAGE;
      docSettings.dsDIBFormat[0] =
         DIBFMT_COLOR;
      docSettings.dsDIBFormat[1] =
         DIBFMT_MONOCHROME;
      docSettings.dsDitherMethod[1] =
         DM_ERRORDIFFUSION;
      docSettings.dsDitherIntensity[1] = 175;
      docSettings.dsKPanelUsage[0][KPU_TEXT] =
         KPU_SIMULATEBLACK;
```

```
        docSettings.dsKPanelUsage[0][KPU_MONOBITMAPS] =
           KPU_SIMULATEBLACK;
        docSettings.dsKPanelUsage[0][KPU_PSEUDOBITMAPS] =
           KPU_SIMULATEBLACK;


        if ( ExtEscape ( hdc,
                         ESC_SETDOCSETTINGS,
                         sizeof ( NISCA_DOCSETTINGS ),
                         &docSettings,
                         sizeof ( NISCA_DOCSETTINGS ),
                         &docSettings
                       ) <= 0
           )
        {
            // ERROR
        }
}
```

Query Document Settings

Escape Function: ESC_QUERYDOCSETTINGS
Value: 11011

Data: *See ESC_SETDOCSETTINGS*

Call Position:

```
StartDoc ( );
  QueryDocSettings ( );
  StartPage ( );
    . . .
  EndPage ( );
EndDoc ( );
```

Example:

```
void QueryDocSettings ( HDC hdc )
{
      NISCA_DOCSETTINGS docSettings;

      ZeroMemory ( &docSettings,
                   sizeof ( NISCA_DOCSETTINGS )
                 );

      docSettings.dsSize = sizeof ( NISCA_DOCSETTINGS );

      // Retrieve settings for the following fields.

      docSettings.dsMask = DSM_DIBFORMAT |
                           DSM_DITHERMETHOD |
                           DSM_DITHERINTENSITY |
                           DSM_KPANELUSAGE;


      if ( ExtEscape ( hdc,
                       ESC_QUERYDOCSETTINGS,
                       sizeof ( NISCA_DOCSETTINGS ),
                       &docSettings,
                       sizeof ( NISCA_DOCSETTINGS ),
                       &docSettings
                     ) <= 0
         )
      {
          // ERROR
      }
}
```

Set Printer Settings

Escape Function: `ESC_SETPRNSETTINGS`
Value: `11010`

Data:

```
typedef struct
{
   WORD     psSize;
   DWORD    psMask;
   WORD     psICEncoder;
   WORD     psMemoryMode;
   WORD     psGammaCorrection;
   WORD     psUserDefMethod;
   COLORREF psUserGammaTable[256];
   WORD     psUserRGBGamma[3];
   WORD     psPrintTextAsGraphics;
   WORD     psForceISOEncoding;
} NISCA_PRNSETTINGS;
```

Data Members

psSize

Specifies the size, in bytes, of the PR5100_PRNSETTINGS structure.

psMask

Specifies which of the remaining members in the PR5100_PRNSETTINGS structure have been initialized. Bit 0 (defined as PSM_ICENCODER) corresponds to psICEncoder; bit 1 (defined as PSM_MEMORYMODE) specifies psMemoryMode, and so on. psMask can be a combination of the following values:

PSM_ICENCODER
PSM_MEMORYMODE
PSM_GAMMACORRECTION
PSM_PRINTTEXTASGRAPHICS
PSM_FORCEISOENCODING

psICEncoder

TRUE if the printer contains an IC Encoder.

psMemoryMode

Memory mode of printer frame buffers. This value should currently be set to 0.

psGammaCorrection

Device gamma correction. Values can be 0 for Device Default or 9 for user defined.

psUserDefMethod

User defined gamma correction method. Possible values are:

Possible Values

0 — User defined gamma table
1 — RGB gamma values

psUserGammaTable

Contains an array of 256 COLORREF values representing a user defined gamma table. This is commonly known as a LUT (Lookup Table).

psUserRGBGamma

Gamma values for Red, Green and Blue. Each value will generate a logarithmic table based on the value. The values have an implied four decimal places, so 1.0 is referenced as 10000.

psPrintTextAsGraphics

TRUE to send text at bitmap graphics instead of downloadable fonts. This must be set to TRUE for Arabic and Hebrew versions of Windows, otherwise, text will not appear.

psForceISOEncoding

TRUE to force track values to conform to ISO standards. Printer will not enocde JIS tracks when set to TRUE.

Call Position:

```
StartDoc ( );
  SetPrinterSettings ( hDC );
  StartPage ( );
    . . .
  EndPage ( );
EndDoc ( );
```

Example:

```
void SetDocSettings ( HDC hdc )
{
      NISCA_PRNSETTINGS prnSettings;

      prnSettings.psSize = sizeof ( NISCA_PRNSETTINGS );
      prnSettings.psMask = PSM_ICENCODER |
                           PSM_GAMMACORRECTION;
      prnSettings.psICEncoder = FALSE;
      prnSettings.psGammaCorrection = 9;
      prnSettings.psUserDefMethod = 0;
      prnSettings.psUserRGBGamma[0] = 20000; // Red
      prnSettings.psUserRGBGamma[1] = 20000; // Green
      prnSettings.psUserRGBGamma[2] = 20000; // Blue

      if ( ExtEscape ( hdc,
                       ESC_SETPRNSETTINGS,
                       sizeof ( NISCA_PRNSETTINGS ),
                       &prnSettings,
                       sizeof ( NISCA_PRNSETTINGS ),
                       &prnSettings
                     ) <= 0
         )
      {
          // ERROR
      }
}
```

Query Printer Settings

Escape Function: ESC_QUERYPRNSETTINGS
Value: 11012

Data: *See ESC_SETPRNSETTINGS*

Call Position: ***QueryPrinterSettings (hDC);***

```
StartDoc ( );
  StartPage ( );
    . . .
  EndPage ( );
EndDoc ( );
```

Example:

```
void QueryPrinterSettings ( HDC hdc )
{
      NISCA_PRNSETTINGS prnSettings;

      ZeroMemory ( &prnSettings,
                   Sizeof ( NISCA_PRNSETTINGS )
                 );

      prnSettings.psSize = sizeof ( NISCA_PRNSETTINGS );
      prnSettings.psMask = PSM_ICENCODER |
                           PSM_GAMMACORRECTION;

      if ( ExtEscape ( hdc,
                       ESC_QUERYPRNSETTINGS,
                       Sizeof ( NISCA_PRNSETTINGS ),
                       &prnSettings,
                       sizeof ( NISCA_PRNSETTINGS ),
                       &prnSettings
                     ) <= 0
         )
      {
          // ERROR
      }
}
```

Encoding

Escape Function: ESC_ENCODEDATA
Value: 11000

Data:

```
typedef struct
{
    WORD iTrack;
    WORD cBufLen;
    CHAR szBuffer[256];
} NISCA_ENCODEDATA;
```

iTrack Values:

0	JIS I, Track 1, 6 bit data, 76 maximum characters
1	JIS I, Track 1, 7 bit data, 69 maximum characters
2	JIS I, Track 2, 4 bit data, 37 maximum characters
3	JIS I, Track 3, 4 bit data, 104 maximum characters
4	JIS I, Track 3, 7 bit data, 69 maximum characters
5	JIS II, 7 bit data, 69 maximum characters

(These values are the same as outlined in the PR5100 command reference.)

Call Position:

```
StartDoc ( );
  StartPage ( );
    EncodeCard ( );
  EndPage ( );
EndDoc ( );
```

Example:

```
void EncodeCard ( HDC hdc )
{
      NISCA_ENCODEDATA encodeData;

      encodeData.iTrack  = 1;
      encodeData.cBufLen = 10;

      strcpy ( encodeData.szBuffer, "0123456789" );

      if ( ExtEscape ( hdc,
                       ESC_ENCODEDATA,
                       sizeof ( NISCA_ENCODEDATA ),
                       &endodeData,
                       0UL,
                       NULL
                     ) <= 0
         )
      {
          // ERROR
      }
}
```

Arbitrary Encoding

Escape Function: `ESC_ARBITRARYENCODE`
Value: `11014`

Data:

```
typedef struct
{
    WORD iMode;
    WORD iTrack;
    WORD iBits;
    WORD iShift;
    WORD cBufLen;
    CHAR szBuffer[256];
} NISCA_ARBITRARYENCODE;
```

`iMode` Values:

0	Standard encode data
1	Binary encode data

`iTrack` Values:

0	JIS I, Track 1, 6 bit data, 76 maximum characters
1	JIS I, Track 1, 7 bit data, 69 maximum characters
2	JIS I, Track 2, 4 bit data, 37 maximum characters
3	JIS I, Track 3, 4 bit data, 104 maximum characters
4	JIS I, Track 3, 7 bit data, 69 maximum characters
5	JIS II, 7 bit data, 69 maximum characters

(These values are the same as outlined in the PR5100 command reference.)

`iBits` Values:

1	1 bit data
2	2 bit data
3	3 bit data
4	4 bit data
5	5 bit data
6	6 bit data
7	7 bit data
8	8 bit data

iShift : Number of characters to shift start character.

Call Position:

```
StartDoc ( );
  StartPage ( );
    EncodeCard ( );
  EndPage ( );
EndDoc ( );
```

Example:

```
void EncodeCard ( HDC hdc )
{
      NISCA_ARBITRARYENCODE encodeData;

      encodeData.iMode   = 0;
      encodeData.iTrack  = 1;
      encodeData.iBits   = 5;
```

```
        encodeData.iShift  = 0;
        encodeData.cBufLen = 10;

        strcpy ( encodeData.szBuffer, “0123456789” );

        if ( ExtEscape ( hdc,
                         ESC_ARBITRARYENCODE,
                         sizeof ( NISCA_ARBITRARYENCODE ),
                         &endodeData,
                         0UL,
                         NULL
                       ) <= 0
           )
        {
            // ERROR
        }
}
```

Bar Code Condition

Escape Function: `ESC_BARCODECONDITION`
Value: `11005`

Data:

```
typedef struct
{
    WORD   width;   // Minimum Line Width
    WORD   height;  // Height Ratio
} NISCA_BARCODECONDITION;
```

`width` Values:

0	3 dots
1	4 dots
2	5 dots
3	6 dots
4	7 dots
5	8 dots
6	9 dots
7	10 dots
8	11 dots
9	12 dots

`height` Values:

0	15 %
1	20 %
2	25 %
3	30 %
4	35 %
5	40 %
6	45 %
7	50 %
8	As specified in JAN

(These values are the same as outlined in the PR5100 command reference.)

Call Position:

```
StartDoc ( );
  StartPage ( );
  SetBarCodeCondition ( );
  EndPage ( );
EndDoc ( );
```

Example:

```
void SetBarCodeCondition ( HDC hdc )
{
      NISCA_BARCODECONDITION barCode;

      barCode.width  = 3;
      barCode.height = 4;

      if ( ExtEscape ( hdc,
                       ESC_BARCODECONDITION,
                       sizeof ( NISCA_BARCODECONDITION ),
                       &barCode,
                       0UL,
                       NULL
                     ) <= 0
         )
      {
          // ERROR
      }
}
```

Bar Code

Escape Function: `ESC_BARCODE`
Value: `11006`

Data:

```
typedef struct
{
    WORD type;
    WORD direction;
    WORD x;
    WORD y;
    WORD cb;
    BYTE szBuffer[256];
} NISCA_BARCODE;
```

type Values:

0	JAN
1	JAN: abridged code
2	ITF
3	NW7 (Codabar)
4	CODE39
5	CODE128

direction Values:

0	Horizontal
1	Vertical

x: Valid card coordinate

y: Valid card coordinate

cb: Length of data pointed to by pBuffer

szBuffer: Data to be placed in bar code.

(These values are the same as outlined in the PR5100 command reference.)

Call Position:

```
StartDoc ( );
  StartPage ( );
  BarCode ( );
  EndPage ( );
EndDoc ( );
```

Example:

```
void BarCode ( HDC hdc )
{
      NISCA_BARCODE barCode;

      barCode.type      =   4;
      barCode.direction =   0;
      barCode.x         = 100;
      barCode.y         = 100;
      barCode.cb        =  10;

      strcpy ( barCode.szBuffer, “0123456789” );

      if ( ExtEscape ( hdc,
                       ESC_BARCODE,
                       sizeof ( NISCA_BARCODE ),
                       &barCode,
                       0UL,
                       NULL
                     ) <= 0
         )
      {
          // ERROR
      }
}
```

WIN32 Text Support

By supporting embedded commands in the WIN32 text functions, you are able to quickly print cards that can be encoded or contain bar codes generated by the Nisca printer. The print driver monitors all text commands and if it finds any of the embedded commands beginning a line, it will generate the specific Nisca print command and not actually print the text on the card. Currently, only Encoding and Bar Code commands are supported through the Text interface.

All text commands will begin with the two character combination, '~@', followed by a numeric command. The text after the numeric command will be specific to each command.

NOTE: Some applications convert text to graphics prior to printing. The driver will not see these text commands.

Encoding

Command: `1`
Syntax: `~@1,`*track*`,`*length*`,`*data*

Fields:		
	track	0 – JIS I, track 1, 7 bit data 1 – JIS I, track 1, 8 bit data 2 – JIS I, track 2, 5 bit data 3 – JIS I, track 3, 5 bit data 4 – JIS I, track 3, 8 bit data 5 – JIS II, 8 bit data
	length	Length of data to encode
	data	Character data to encode

Sample: To encode the ten numeric digits on track 1 with 8 bits per character type:
`~@1,1,10,0123456789`

To encode the 26-character alphabet on track 3 with 8 bits per character type:
`~@1,4,26,ABCDEFGHIJKLMNOPQRSTUVWXYZ`

Arbitrary Encoding

Command: 4
Syntax: ~@4,*track*,*bits*,*length*,*data*

Fields:		
	track	0 – Track 1 2 – Track 2 3 – Track 3
	bits	*Bit density, 1 through 8*
	length	Length of data to encode
	data	Character data to encode

Sample: To encode the ten numeric digits on track 1 with 5 bits per character type:
```
~@4,2,5,10,0123456789
```

To encode the 26-character alphabet on track 3 with 8 bits per character type:
```
~@4,3,8,26,ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Bar Code Condition

Command: 2
Syntax: ~@2,*width*,*height*

Fields:		
	width	Minimum dots per line 0 – 3 dots 1 – 4 dots 2 – 5 dots 3 – 6 dots 4 – 7 dots 5 – 8 dots 6 – 9 dots 7 – 10 dots 8 – 11 dots 9 – 12 dots
	height	Height to width ratio 0 – 15% 1 – 20 % 2 – 25% 3 – 30% 4 – 35% 5 – 40% 6 – 45% 7 – 50% 8 – As specified in JAN

Sample: To set the minimum dots per line to 6 and height ratio to 35% type:
~@2,3,4

Bar Code

Command: 3
Syntax: ~@3,*type*,*direction*,*x*,*y*,*length*,*data*

Fields:		
	type	0 –JAN 1 – JAN: abridged code 2 – ITF 3 – NW7 (Codabar) 4 – CODE39 5 – CODE128
	direction	0 – Horizontal 1 – Vertical

Sample: To print a CODE39 barcode horizontally containing the ten digits type:
~@3,4,0,100,100,10,0123456789