NEC

MultiCoder 500 シリーズ

MultiCoder SDK　説明書

853-810932-441-A
第2版

本文中で使用する記号の意味

このマニュアルでは本文中で次の2種類の記号を使っています。それぞれの記号について説明します。

記号	内　容
	この注意事項を守らないと、プリンターが故障するおそれがあります。また、システムの運用に影響を与えることがあります。
	この注意事項を守らないと、プリンターが正しく動作しないことがあります。

商標について

NEC、NECロゴは日本電気株式会社の登録商標です。
MultiCoderはNECエンベデッドプロダクツ株式会社の登録商標です。
LabelStar、BarStarはアイニックス株式会社の登録商標です。
Microsoft、Windows、Windows Vista、Windows Server、Visual Basic、Visual C++、SQL Serverは米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。
その他、記載の会社名および商品名は各社の商標または登録商標です。

ご注意

1. 本書の内容の一部または全部を無断転載することは禁止されています。
2. 本書の内容に関しては将来予告なしに変更することがあります。
3. NECの許可なく複製・改変などを行うことはできません。
4. 本書は内容について万全を期して作成致しましたが、万一ご不審な点や誤り、記載もれなどお気づきのことがありましたら、お買い求めの販売店にご連絡ください。
5. 運用した結果の影響については4項にかかわらず責任を負いかねますのでご了承ください。
6. 本製品を第三者に売却・譲渡する際は必ず本書も添えてください。

はじめに

本マニュアルは、NECラベルプリンター「MultiCoder 500シリーズ」を組み込んだアプリケーション開発およびシステム開発を行う際に利用することを目的とした手引き書です。
アプリケーションやシステムの設計を行う際の参照としてください。

なお、本マニュアルはアプリケーションやシステムの設計者を対象として記載しています。
アプリケーションの操作や組み込みに必要となる技術情報についてはそれぞれの説明書などを参照してください。

マニュアルの構成

本マニュアルの構成は次のとおりです。

第1章　Visual Basic環境

開発環境のプラットホームとして、Microsoft Visual Basicを使用している開発者向けの章です。
この章では、Microsoft Accessデータベースに接続し、データベースの内容を帳票印刷する手順について解説しています。

第2章　Visual C++環境

開発環境のプラットホームとして、Microsoft Visual C++を使用している開発者向けの章です。
この章では、Ainix BarStar ProおよびAinix LabelStar ProのOLEオートメーション機能を使用して帳票印刷する手順を解説しています。

第3章　NLPLコマンド送信ライブラリー

上位APからNEC Label Printer Languageコマンド（NLPLコマンド）を直接送信するためのダイナミックリンクライブラリー（DLL）がサポートする各種関数について詳しく説明しています。

目　次

メ　モ

1章 Visual Basic環境

この章では、Visual Basic 6.0およびVisual Basic .NETの環境において、Microsoft Accessデータベースに接続し、データベースの内容を帳票印刷する手順について解説しています。

1.1 対象ユーザーおよびPC環境

対象ユーザー

Visual Basic 6.0またはVisual Basic .NETの環境においてデータベースとの連携、帳票印刷プログラムの開発を検討しているユーザーを対象とします。

PC環境

- 対象OS
  - Windows 7
  - Windows Vista
  - Windows XP
  - Windows 2000
- 開発環境
  - Visual Basic 6.0
  - Visual Basic .NET
- 使用アプリケーション
  - Micrsoft Access 2000以降（別途インストールが必要です）
  - Ainix BarStar Pro（NECラベルプリンタに添付のソフトウェアCD-ROMに収録されています）
- その他
  - .NETプログラミングサポート

> チェック
>
> OfficeアプリケーションのCOMオブジェクトを使用した.NETプログラムを動作させる場合に必要となります。
> .NETプログラミングサポートのインストール方法についてはMicrosoft社のホームページをご確認ください。

1.2　開発環境別手順

ご使用の環境に合わせて以下の項番の手順を実施してください。

開発環境：Visual Basic 6.0の場合

Step 1. プリンタードライバーをインストールする

→1.3　プリンタードライバーのインストール

Step 2. データベースの接続設定をする

ADO接続の場合

→1.4.1　Visual Basic 6.0環境からのMicrosoft Accessデータベースへの接続方法(ADO接続)
→1.4.1.1　ライブラリの追加
→1.4.1.2　接続方法（コード記載による使用例）

DAO接続の場合

→1.4.2　Visual Basic 6.0環境からのMicrosoft Accessデータベースへの接続方法(DAO接続)
→1.4.2.1　ライブラリの追加
→1.4.2.2　接続方法（コード記載による使用例）

Step 3. 帳票の印刷を設定する

BarStar Proを使用する場合

→1.5.1　バーコード作成ソフトを使用した帳票印刷（BarStar Pro）
→1.5.1.1　BarStar Proのインストール
→1.5.1.2　ライブラリの追加
→1.5.1.3　バーコードの作成
→1.5.1.4　バーコードの印刷（コード記載による使用例）

Microsoft Accessバーコードオブジェクトを使用する場合

→1.5.2　Microsoft Accessバーコードオブジェクトを使用した帳票印刷
→1.5.2.1　ライブラリの追加（Microsoft BarCode Control 9.0）
→1.5.2.2　Microsoft BarCode Control 9.0が[参照設定]ダイアログにない場合
→1.5.2.3　ライブラリの追加（Microsoft Access 11.0 Object Library）
→1.5.2.4　バーコードの作成
→1.5.2.5　バーコードの生成（コード記載による使用例）

開発環境：Visual Basic .NETの場合

本手順に従って作成したプログラムを使用する場合、ご使用の環境によっては.NETプログラミングサポートをインストールする必要があります。インストール方法は、Microsoft社のホームページで確認してください。

Step 1. プリンタードライバーをインストールする

→1.3　プリンタードライバーのインストール

Step 2. データベースの接続設定をする

ADO.NET接続の場合

→1.4.3　Visual Basic .NET環境からのMicrosoft Accessデータベースへの接続方法(ADO.NET接続)
→1.4.3.1　接続方法（コード記載による使用例）

Step 3. 帳票の印刷を設定する

BarStar Proを使用する場合

→1.5.1　バーコード作成ソフトを使用した帳票印刷（BarStar Pro）
→1.5.1.1　BarStar Proのインストール
→1.5.1.2　ライブラリの追加
→1.5.1.3　バーコードの作成
→1.5.1.4　バーコードの印刷（コード記載による使用例）

Microsoft Accessバーコードオブジェクトを使用する場合

→1.5.2　Microsoft Accessバーコードオブジェクトを使用した帳票印刷
→1.5.2.1　ライブラリの追加（Microsoft BarCode Control 9.0）
→1.5.2.2　Microsoft BarCode Control 9.0が[参照設定]ダイアログにない場合
→1.5.2.3　ライブラリの追加（Microsoft Access 11.0 Object Library）
→1.5.2.4　バーコードの作成
→1.5.2.5　バーコードの生成（コード記載による使用例）

1.3 プリンタードライバーのインストール

使用するプリンターのプリンタードライバーをインストールし、プリンターに対して印刷ができることを確認します。
プリンタードライバーのインストール方法については各プリンターに同梱のマニュアルを参照してください。

1.4 データベース接続

Visual Basic環境下でのMicrosoft Accessデータベースへの接続方法について説明します。

1.4.1 Visual Basic 6.0環境からのMicrosoft Accessデータベースへの接続方法(ADO接続)

ここではADO（ActiveX Data Object）でのMicrosoft Accessデータベースへの接続方法を説明します。
ADOとはMicrosoftが提供するデータベースアクセスのためのオブジェクトであり、OLE DBを利用してアプリケーションからAccess、SQL Serverなどのさまざまなデータソースへ接続可能なプログラミングインターフェースです。

1.4.1.1　ライブラリの追加

次の手順で「Microsoft ActiveX Data Object 2.7 Library」を追加します。

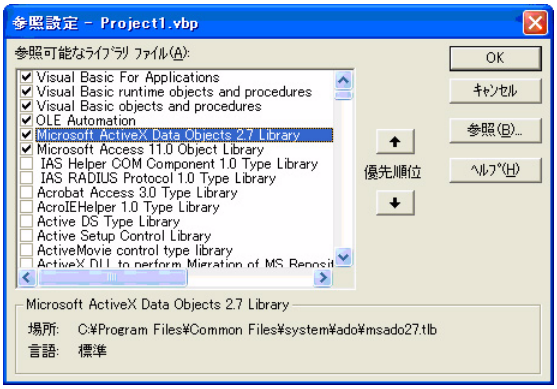
チェック

2.7はバージョンです。ご使用の環境に2.7よりも最新のバージョンがある場合は、そちらを選択してください。

❶ Visual Basic 6.0を起動します。

❷ ［プロジェクト］－［参照設定］を選択し、［参照設定]ダイアログボックスを表示します。

❸ ［参照設定]ダイアログボックスから［Microsoft ActiveX Data Object 2.7 Library］にチェックし、［OK]をクリックします。



1.4.1.2　接続方法（コード記載による使用例）

以下に、Microsoft Accessデータベースに接続するサンプルを示します。

```
'変数宣言
Dim cn As New ADODB.Connection          'Connectionオブジェクト生成
Dim rs As New ADODB.Recordset           'Recordsetオブジェクト生成

'Accessデータベースに接続
cn.Open "Driver={Microsoft Access Driver (*.mdb)};DBQ=" &
データベースのファイルパス                          'レコードセットの取得
rs.Open テーブル/クエリ名, cn, adOpenKeyset

'オブジェクトの終了処理
rs.Close
cn.Close

'メモリ開放
Set rs = Nothing
Set cn = Nothing
```

1.4.2　Visual Basic 6.0環境からのMicrosoft Accessデータベースへの接続方法(DAO接続)

ここではDAO(Data Access Object）でのMicrosoft Accessデータベースへの接続方法について説明します。
DAOとはMicrosoftが提供するデータベースアクセスのためのオブジェクトであり、ADOとは異なり、Microsoft Accessデータベースへの操作ををターゲットとしたプログラミングインターフェースです。
データベースにMicrosoft Accessを使用する場合、ADOに比べ高いパフォーマンスを得ることができます。
単一システムのアプリケーションでの使用に適した接続方法です。

1.4.2.1　ライブラリの追加

次の手順で「Microsoft DAO 3.6 Object Library」を追加します。

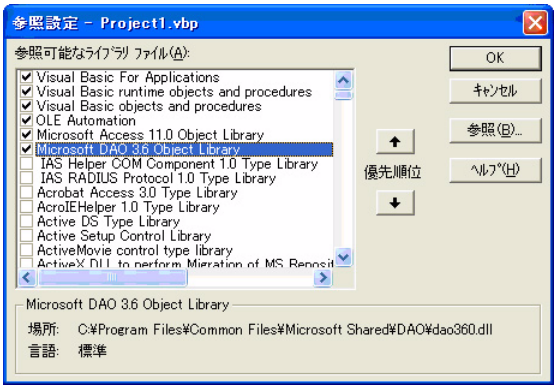
チェック

3.6はバージョンです。ご使用の環境に3.6よりも最新のバージョンがある場合は、そちらを選択してください。

❶ Visual Basic 6.0を起動します。

❷ ［プロジェクト］－［参照設定］を選択し、［参照設定]ダイアログボックスを表示します。

❸ ［参照設定]ダイアログボックスから［Microsoft DAO 3.6 Object Library］にチェックし、［OK]をクリックします。



1.4.2.2　接続方法（コード記載による使用例）

```
'変数宣言
Dim ws As DAO.Workspace                    'Workspaceオブジェクト生成
Dim db As DAO.Database                     'Databaseオブジェクト生成
Dim rs As DAO.Recordset                    'Recordsetオブジェクト生成

'デフォルトワークスペースを定義
Set ws = DBEngine.Workspaces(0)            'データベースを開く
Set db = ws.OpenDatabase(データベースのファイルパス)
                                           'テーブル名を指定してレコードセットの取得
Set rs = db.OpenRecordset(テーブル/クエリ名, dbOpenDynaset)

'オブジェクトの終了処理
rs.Close
db.Close
ws.Close

'メモリ開放
Set rs = Nothing
Set db = Nothing
Set ws = Nothing
```

1.4.3　Visual Basic .NET環境からのMicrosoft Access データベースへの接続方法(ADO.NET接続)

ADO.NETとはMicrosoftが提供するデータベースアクセスのためのオブジェクトです。
ADOを.NET環境下で使用できるようにしたものであり、従来のADOと同様にさまざまなデータベース環境との接続が可能です。

1.4.3.1　接続方法（コード記載による使用例）

```
    '定数宣言
    Const CN_STRING As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
                                                    'ACCESS形式のデータベースへの接続指定パス
                                                    '変数宣言
    Dim Cn As OleDbConnection                       'OleDbConnectionオブジェクト

    On Error GoTo ErrorHandle

    'OleDbConnectionオブジェクト生成
    Cn = New OleDbConnection(CN_STRING & System.IO.Directory.GetCurrentDirectory
     & "\" & ”データベース名”)
    'DB接続
    Cn.Open()

    'オブジェクトの終了処理
    Cn.Dispose()
    Cn = Nothing

End Sub
```

1.5　帳票印刷

バーコード作成（BarStar Pro）を使用した帳票印刷とAccessバーコードオブジェクトを使用した帳票印刷の設定方法について説明します。

1.5.1　バーコード作成ソフトを使用した帳票印刷（BarStar Pro）

ここではバーコード作成ソフト（BarStar Pro）を使用して印刷する場合の設定変更を説明します。

1.5.1.1　BarStar Proのインストール

BarStar Proのインストールを行います。BarStar Proのインストール方法については、ラベルプリンタに同梱のユーザーズマニュアルまたはCD-ROM内のオンラインマニュアルを参照してください。

1.5.1.2　ライブラリの追加

BarStar Proでは、開発者向けに二つのコンポーネント、Barcodeオブジェクト(Ainix BarStar Pro Automation Server)およびBarcodeコントロール(Ainix BarStar Pro ActiveX Control)が提供されています。
ここでは、各ライブラリの追加の方法を説明します。

- Barcodeオブジェクト(Ainix BarStar Pro Automation Server)とは、VisualBasicでのシンボル生成オブジェクトを提供するオートメーションサーバーです。
- Barcodeコントロール(Ainix BarStar Pro ActiveX Control)とは、Visal Basicのフォームや各種アプリケーションソフトにバーコードオブジェクトを部品として組み込むことが可能なActiveX コントロールです。

Barcodeオブジェクト(Ainix BarStar Pro Automation Server)の追加

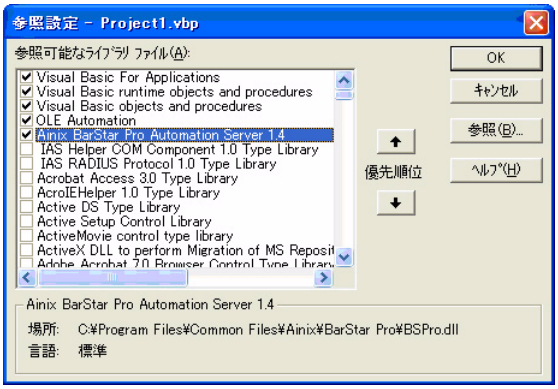
ここではVer.1.4を例に説明します。ご使用のバージョンに読み替えてください。
次の手順で「Ainix BarStar Pro Automation Server 1.4」を追加します。

Visual Basic 6.0の場合

❶ Visual Basic 6.0を起動します。

❷ [プロジェクト] ー [参照設定] を選択し、[参照設定]ダイアログボックスを表示します。

❸ [参照設定]ダイアログボックスから[Ainix BarStar Pro Automation Server 1.4]にチェックし、 [OK]をクリックします。



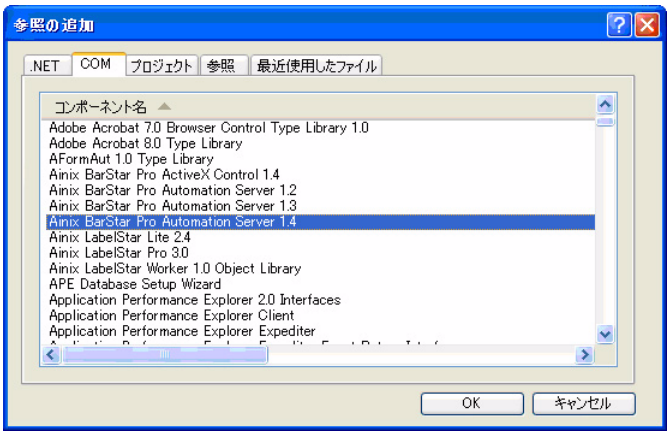
Visual Basic .NETの場合

❶ Visual Basic .NETを起動します。

❷ ［プロジェクト］－［参照の追加］を選択し、[参照の追加]ダイアログボックスを表示します。

❸ [参照の追加]ダイアログボックスで［COM］タブをクリックします。

❹ ［コンポーネント名］のリストから［Ainix BarStar Pro Automation Server 1.4］を選択し、[OK]をクリックします。



Barcodeコントロール(Ainix BarStar Pro ActiveX Control)の追加

次の手順で「Ainix BarStar Pro ActiveX Control 1.4」を追加します。

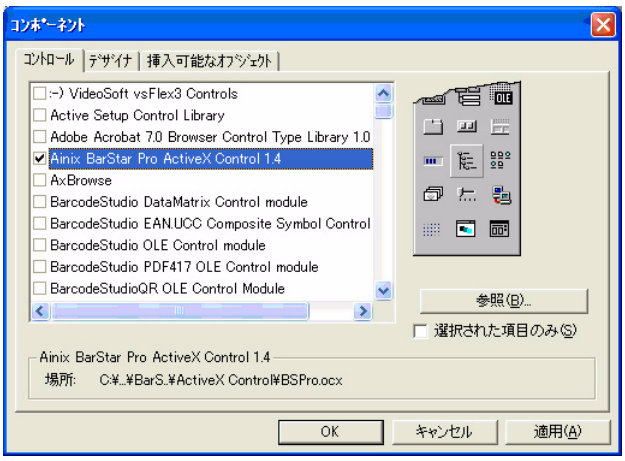
Visual Basic 6.0の場合

❶ Visual Basic 6.0を起動します。

❷ ［プロジェクト］－［コンポーネント］を選択し、[コンポーネント]ダイアログボックスを表示します。

❸ [コンポーネント]ダイアログボックスで［コントロール］タブをクリックします。

❹ ［コントロール］のリストから［Ainix BarStar Pro ActiveX Control 1.4］にチェックし、［OK]をクリックします。



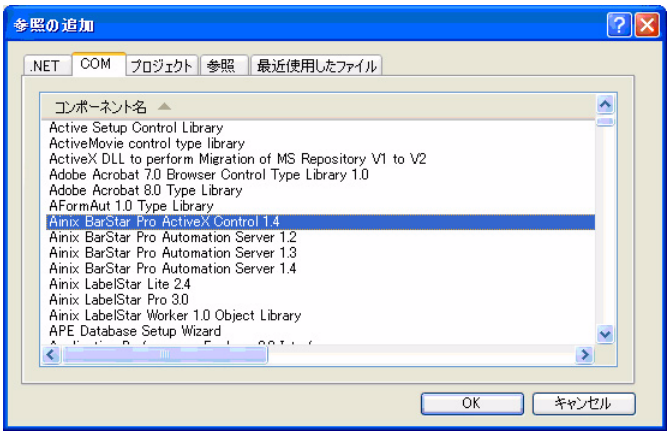
Visual Basic .NETの場合

❶ Visual Basic .NETを起動します。

❷ ［プロジェクト］－［参照の追加］を選択し、[参照の追加]ダイアログボックスを表示します。

❸ [参照の追加]ダイアログボックスで［COM］タブをクリックします。

❹ ［コンポーネント名］のリストから［Ainix BarStar Pro ActiveX Control 1.4］を選択し、[OK]をクリックします。



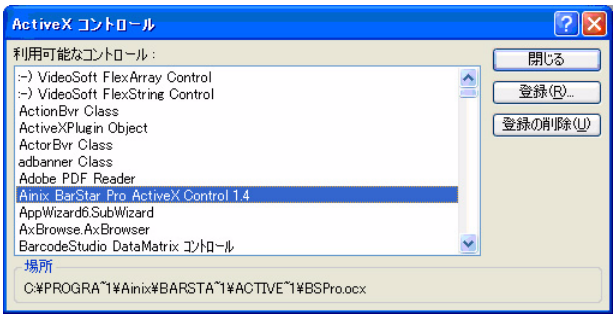
Barcodeコントロール(Ainix BarStar Pro ActiveX Control)は、その他のアプリケーションソフトにおいてもコンポーネントを追加することで使用することが可能です。
以下に、Microsoft AccessにてBarcodeコントロール(Ainix BarStar Pro ActiveX Control)を追加する方法を説明します。

Microsoft Accessの場合

❶ Microsoft Accessを起動します。

❷ ［ツール］－［ActiveX コントロール］を選択し、[ActiveX コントロール]ダイアログボックスを表示します。

❸ [利用可能なコントロール]の中から[Ainix BarStar Pro ActiveX Control 1.4]を選択し、[OK]をクリックします。



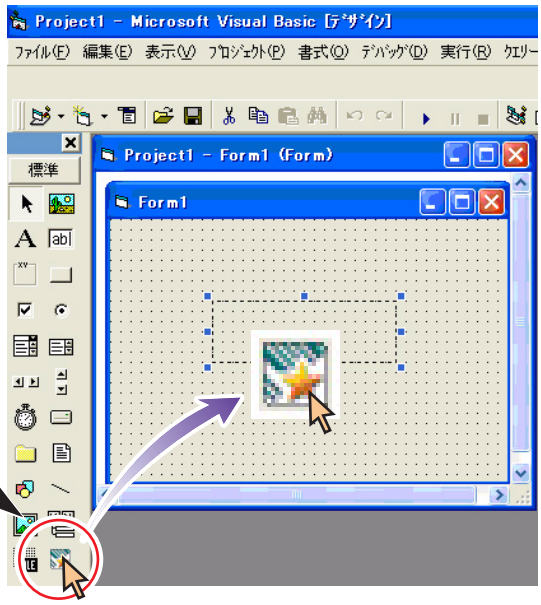
各環境にてBarcodeコントロール(Ainix BarStar Pro ActiveX Control)を追加すると、ツールボックス内にバーコードオブジェクトが追加されます。
バーコードオブジェクトを選択し、フォームやレポート上でドラック・アンド・ドロップすることでバーコードを部品として貼り付けることが可能です。

1.5.1.3　バーコードの作成

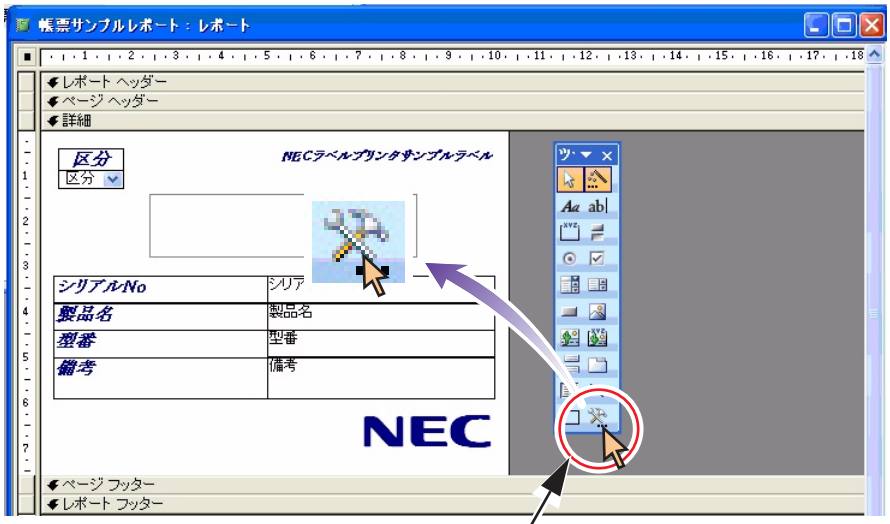
Barcodeコントロール(Ainix BarStar Pro ActiveX Control)が追加されている場合、ツールボックス内にバーコードオブジェクトが追加され、フォームやレポート上でドラック・アンド・ドロップすることでバーコードを部品として貼り付けることができます。
ここでは、VisualBasic、Microsoft Accessの環境でBarStarProバーコードオブジェクトを追加する方法を説明します。

VisualBasic環境の場合

ツールボックスから［BarCode オブジェクト］をフォーム上へドラッグ・アンド・ドロップします。
オブジェクトの設定は［プロパティ］またはソースコードで変更できます。



Microsoft Accessの場合



ツールボックスから［Anix BarStarPro ActiveX Control 1.4］をフォーム上へドラッグ・アンド・ドロップします。
オブジェクトの設定は［プロパティ］またはソースコードで変更できます。

1.5.1.4　バーコードの印刷（コード記載による使用例）

以下にバーコードを生成し、印刷するサンプルを示します。

Visual Basic 6.0の場合

```
'Windows API定義
Public Declare Function StartDoc Lib "gdi32" Alias "StartDocA" (ByVal hdc
  As Long, structDocInfo As DOCINFO) As Long
Public Declare Function StartPage Lib "gdi32" (ByVal hdc As Long) As Long
Public Declare Function EndPage Lib "gdi32" (ByVal hdc As Long) As Long
Public Declare Function EndDoc Lib "gdi32" (ByVal hdc As Long) As Long

'印刷処理
Private Sub PrintOut()
  '変数宣言
  Dim objBSPro As BSPro.Barcode          'BarStarProバーコードオブジェクト
  Dim lRet As Long                       '関数戻り値
  '印刷開始
  lRet = StartDoc(Printer.hdc, DOCINFO)
  lRet = StartPage(Printer.hdc)
  'バーコードオブジェクトの生成
  Set objBSPro = GetObject("", "BSPro.Barcode")    'オブジェクト生成
  With objBSPro
       .Symbology = &HA                  'Symbology指定
       .Text = "1234567890"              'コード設定
       .ElementSizeX = 10                '幅設定
       .ElementSizeY = 7                 '高さ設定
       .Encode Printer.hdc               'シンボルの生成
       lRet = .Draw(Printer.hdc, 0, 0)
  End With
  Set objBSPro = Nothing
  '1ページ分のジョブを印刷
  lRet = EndPage(Printer.hdc)
  '印刷終了
  lRet = EndDoc(Printer.hdc)
End Sub
```

Visual Basic .NETの場合

```
'印刷ボタン押下
Private Sub cmdPrint_Click(ByVal sender As System.Object, ByVal e As
  System.EventArgs) Handles cmdAllPrint.Click
  '印刷開始
  PrintDocument1.Print()
End Sub

'印刷処理
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object,  _
ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles
  PrintDocument1.PrintPage

  '変数宣言
  Dim objBSPro As BSPro.Barcode          'BarStar Proバーコードオブジェクト
  Dim ipHDC As IntPtr                    ' バーコードシンボル描画デバイスコンテキスト
                                           ハンドル
  Dim drawFont As New Font(" MS明朝", 7) '文字列のフォント/フォントサイズ

  'BarStar Pro バーコードオブジェクトの取得
  objBSPro = GetObject("", "BSPro.Barcode")
  'バーコード生成文字列の取得
  objBSPro.Text = &h1234657890&h
  'バーコードの生成
  objBSPro.Encode(PrintDlg.PrinterSettings.CreateMeasurementGraph-
       ics.GetHdc(), False)
  'デバイスコンテキスト取得
  ipHDC = e.Graphics.GetHdc()
  'シンボルの描画
  With objBSPro
       .Draw(CInt(ipHDC), XPOS_BAR, YPOS_BAR)'バーコードの描画
       .Symbology = &HA                  'Symbology変更
       .ElementSizeX = 10                '幅設定
       .ElementSizeY = 7                 '高さ設定
    End With
  End If
  objBSPro = Nothing
End Sub
```

1.5.2　Microsoft Accessバーコードオブジェクトを使用した帳票印刷

ここではMicrosoft Accessバーコードオブジェクトを使用して印刷する場合の設定方法を説明します。
Microsoft Accessバーコードオブジェクトを使用する場合、以下の手順が必要となります。

1.5.2.1　ライブラリの追加（Microsoft BarCode Control 9.0）

次の手順で「Microsoft BarCode Control 9.0」を追加します。

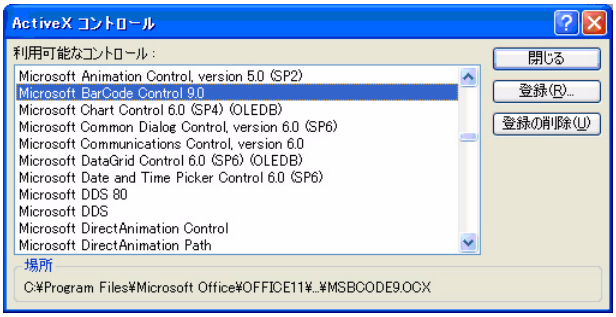
チェック

9.0はバージョンです。ご使用の環境に9.0よりも最新のバージョンがある場合は、そちらを選択してください。

❶ **Microsoft Accessを起動します。**

❷ **［ツール］－［ActiveX コントロール］を選択し、[ActiveX コントロール]ダイアログボックスを表示します。**

❸ **[利用可能なコントロール]の中から［Microsoft BarCode Control 9.0］を選択し、[OK]をクリックします。**



1.5.2.2　Microsoft BarCode Control 9.0が[参照設定]ダイアログにない場合

お使いの環境によっては[利用可能なコントロール]にMicrosoft BarCode Control 9.0が表示されない場合があります。その場合は、Microsoftのホームページよりファイルをダウンロードしてください。ファイルの保存先、インストール方法等についてはMicrosoftホームページの指示に従ってください。

1.5.2.3　ライブラリの追加（Microsoft Access 11.0 Object Library）

次の手順で「Microsoft Access 11.0 Object Library」を追加します。

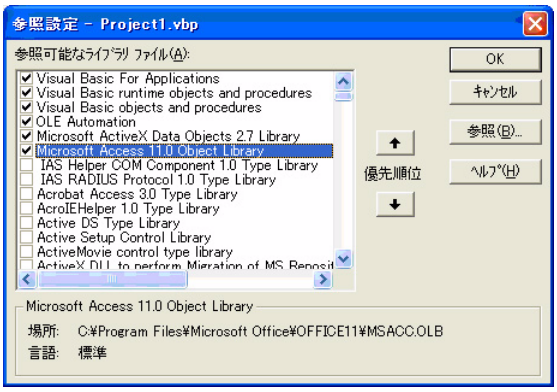
チェック

11.0はバージョンです。ご使用の環境に11.0よりも最新のバージョンがある場合は、そちらを選択してください。

❶ Visual Basic 6.0を起動します。

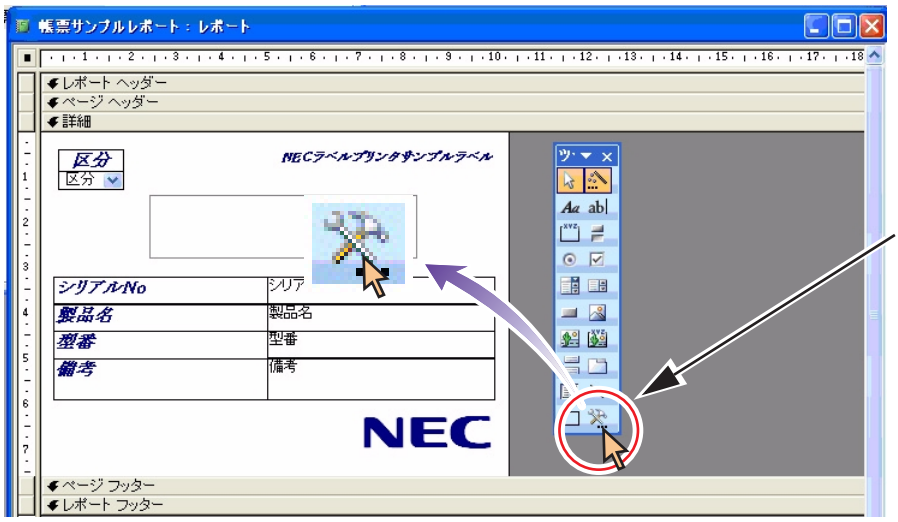
❷ ［プロジェクト］ー［参照設定］を選択し、［参照設定]ダイアログボックスを表示します。

❸ ［参照設定]ダイアログボックスから［Microsoft Access 11.0 Object Library］にチェックし、［OK]をクリックします。



1.5.2.4　バーコードの作成

Barcodeコントロール(Ainix BarStar Pro ActiveX Control)が追加されている場合、ツールボックス内にバーコードオブジェクトが追加され、フォームやレポート上でドラック・アンド・ドロップすることでバーコードを部品として貼り付けることができます。
ここでは、VisualBasic、Microsoft Accessの環境でBarStarProバーコードオブジェクトを追加する方法を説明します。



ツールボックスから［Anix BarStarPro ActiveX Control 1.4］をフォーム上へドラッグ・アンド・ドロップします。
オブジェクトの設定は［プロパティ］またはソースコードで変更できます。

1.5.2.5　バーコードの生成（コード記載による使用例）

以下にバーコードを生成し、印刷するサンプルを示します。

```
Private Sub cmdPrint_Click()

  '変数宣言
  Dim ac As New Access.Application      'Applicationオブジェクトの生成
  Dim sDir As String

  On Error GoTo ErrorHandle
  'カレントディレクトリの取得
  sDir = CurDir
  '既存のAccessデータベースをカレントデータベースとして開く
  ac.OpenCurrentDatabase sDir & "\" & "データベース名", False
  'レポートの印刷(全レコードを印刷)
  ac.DoCmd.OpenReport "レポート名", acViewNormal, "テーブル名"
  'Accessデータベースを閉じる
  ac.CloseCurrentDatabase
  'オブジェクトの終了処理
  ac.Quit
  Set ac = Nothing
  Exit Sub

ErrorHandle:
  MsgBox Err.Description, vbExclamation, "ADOサンプル"
  Resume Next

End Sub
```

2章 Visual C++環境

この章では、Microsoft Visual C++ 6.0およびMicrosoft Visual C++ .NETの環境において、Ainix BarStar ProおよびAinix LabelStar ProのOLEオートメーション機能を使用して帳票印刷する手順を解説しています。

2.1 対象ユーザーおよびPC環境

対象ユーザー

Microsoft Visual C++ 6.0またはMicrosoft Visual C++ .NETの環境においてデータベースとの連携、帳票印刷プログラムの開発を検討しているユーザーを対象とします。

PC環境

- 対象OS
  - Windows 7
  - Windows Vista
  - Windows XP
  - Windows 2000
- 開発環境*1
  - Microsoft Visual C++ 6.0
  - Microsoft Visual C++ .NET
- 使用アプリケーション
  - Ainix BarStar Pro*2
  - Ainix LabelStar Pro*3

*1 本資料はMicrosoft Visual C++ 2005の環境に則って説明しています。お使いの環境によっては若干、操作が異なる場合があります。また、この章のMicrosoft Visual C++ 2005での各手順は、MFCアプリケーション作成時の手順となります。
*2 NECラベルプリンタに添付のソフトウェアCD-ROMに収録されています
*3 60日間期間限定版がソフトウェアCD-ROMに収録されています。

2.2　開発環境別手順

ご使用の環境に合わせて以下の項番の手順を実施してください。

開発環境：Ainix BarStar ProのOLEオートメーション機能を使用して印刷する場合

Step 1.	プリンタードライバーをインストールする	→2.3	プリンタードライバーのインストール
Step 2.	帳票アプリケーションをインストールする	→2.4	帳票アプリケーションのインストール
Step 3.	帳票に印刷する	→2.5.1	Ainix BarStar Pro タイプライブラリの追加
		→2.5.2	タイプライブラリのインクルード（コード記載による使用例）
		→2.5.5	Ainix BarStar Proでの帳票印刷
		→2.5.5.1	バーコードの印刷（コード記載による使用例）

開発環境：Ainix LabelStar ProのOLEオートメーション機能を使用して印刷する場合

Step 1.	プリンタードライバーをインストールする	→2.3	プリンタードライバーのインストール
Step 2.	帳票アプリケーションをインストールする	→2.4	帳票アプリケーションのインストール
Step 3.	帳票に印刷する	→2.5.3	Ainix LabelStar Pro タイプライブラリの追加
		→2.5.4	タイプライブラリのインクルード（コード記載による使用例）
		→2.5.6	Ainix LabelStar Proでの帳票印刷
		→2.5.6.1	バーコードの印刷（コード記載による使用例）

2.3 プリンタードライバーのインストール

使用するプリンターのプリンタードライバーをインストールし、プリンターに対して印刷ができることを確認します。
プリンタードライバーのインストール方法については各プリンターに同梱のマニュアルを参照してください。

2.4 帳票アプリケーションのインストール

使用方法に合わせてAinix BarStar Pro、Ainix LabelStar Proをインストールします。
帳票アプリケーションのインストールは各プリンターに同梱のマニュアルをご確認ください。

2.5 帳票印刷

Visual C++環境下でAinix BarStar Pro、Ainix LabelStar ProのOLEオートメーション機能を使用して帳票印刷を行う方法について説明します。

2.5.1 Ainix BarStar Pro タイプライブラリの追加

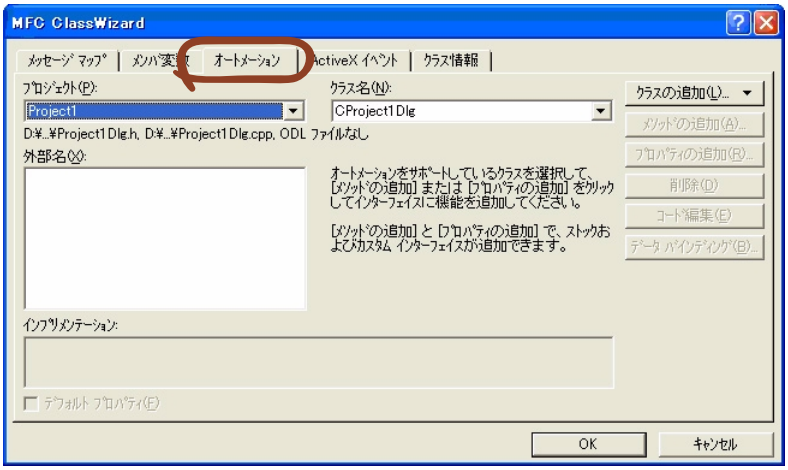
次の手順でAinix BarStar Proタイプライブラリを追加します。

2.5.1.1 Microsoft Visual C++ 6.0の場合

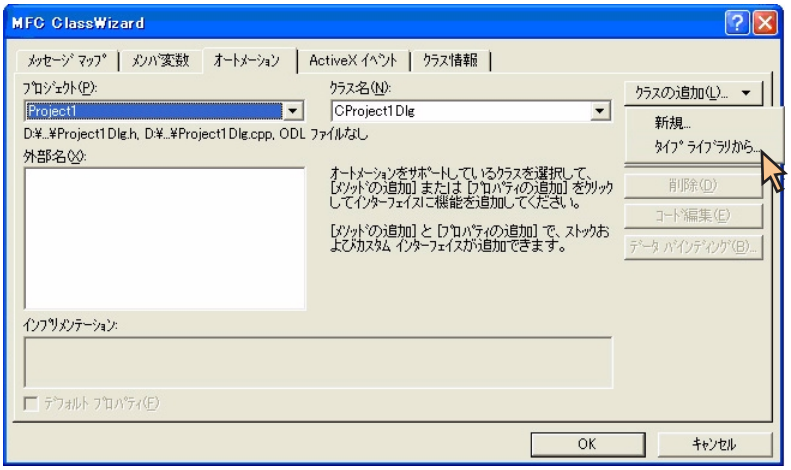
❶ Microsoft Visual C++ 6.0を起動します。

❷ [表示] ー [ClassWizard] を選択し、[MFC ClassWizard]ダイアログボックスを表示します。

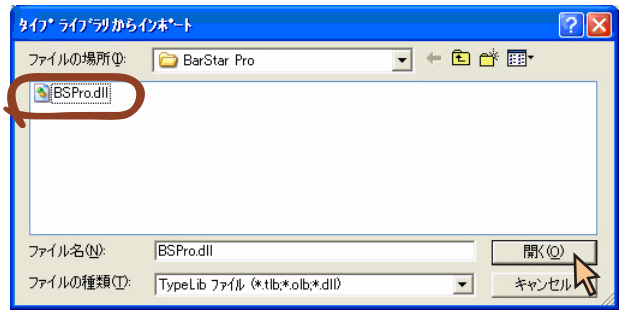
❸ [オートメーション]タブをクリックします。



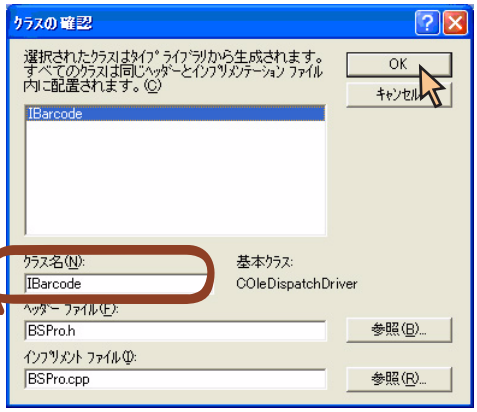
❹ [クラスの追加]をクリックすると表示されるリストボックスから、[タイプライブラリから] を選択します。
[タイプライブラリからインポート]ダイアログボックスが表示されます。



❺ **[タイプライブラリからインポート]ダイアログボックスより「BSPro.dll」を選択します。**
[クラスの確認]ダイアログボックスが表示されます。



❻ **[クラスの確認]ダイアログボックスで、任意のクラス名を設定後、[OK]をクリックします。**
「bspro.h」ファイル、「bspro.cpp」ファイルがプロジェクトに追加されます。



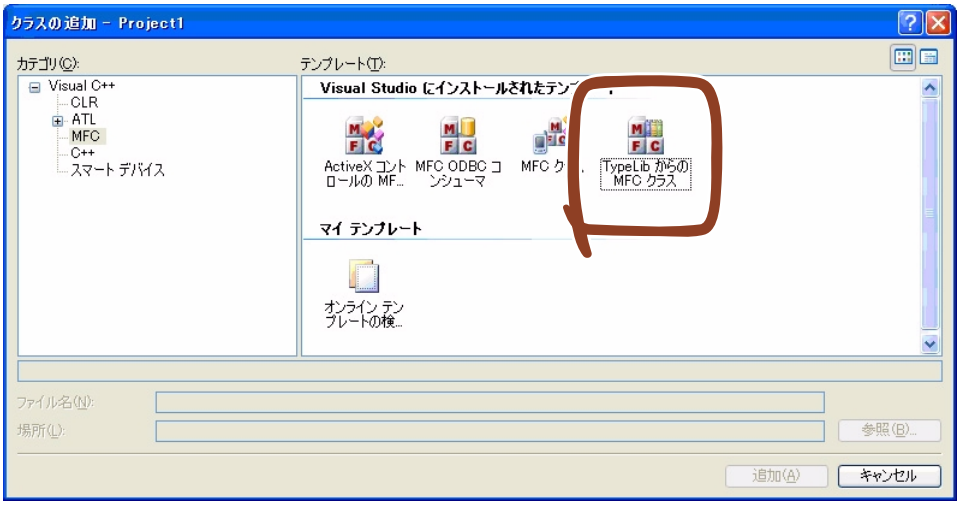
2.5.1.2　Microsoft Visual Visual C++ .NETの場合

❶ Microsoft Visual Visual C++ .NETを起動します。

❷ ［プロジェクト］ー［クラスの追加］を選択し、［クラスの追加］ダイアログボックスを表示します。

❸ ［クラスの追加］ダイアログボックスで、［カテゴリ］から［MFC］を選択します。

❹ ［テンプレート］から［TypeLibからのMFCクラス］を選択します。

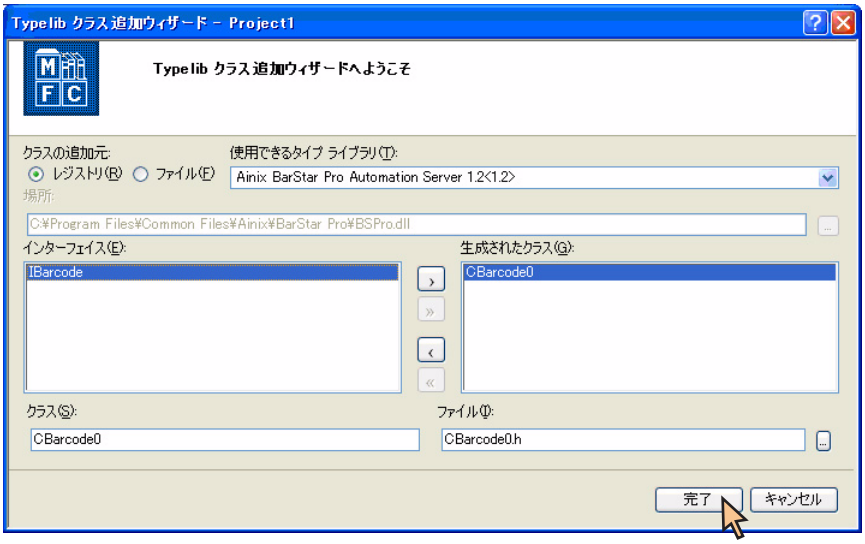


❺ [追加]をクリックします。
［Typelibクラス追加ウィザード］ダイアログボックスが表示されます。

❻ [Typelibクラス追加ウィザード]ダイアログボックスで［使用できるタイプライブラリ］から、[Ainix BarStar Pro Automation Server 1.2<1.2>］を選択します。

❼ [インタフェース]に「IBarcode」が表示されていることを確認後、「IBarcode」が選択された状態で、[>]または[>>]をクリックします。
［生成されたクラス］にタイプライブラリーの追加により作成されるクラス[CBarcode]が表示されます。

❽ [完了]をクリックします。
「CBarcode.h」ファイルがプロジェクトに追加されます。



2.5.2　タイプライブラリのインクルード（コード記載による使用例）

2.5.2.1　Microsoft Visula C++ 6.0の場合

```
// Project1Dlg.h : ヘッダー ファイル
//
#include"BSPro.h"

// CProject1Dlg ダイアログ
class CProject1Dlg : public CDialog
{
private:
  IBarcode          m_Barcode;
}
```

2.5.2.2　Microsoft Visula C++ .NETの場合

```
// Project1Dlg.h : ヘッダーファイル
//
#include"CBarcode.h"

// CProject1Dlg ダイアログ
class CProject1Dlg : public CDialog
{
public:
  CBarcode          m_Barcode;
}
```

2.5.3　Ainix LabelStar Pro タイプライブラリの追加

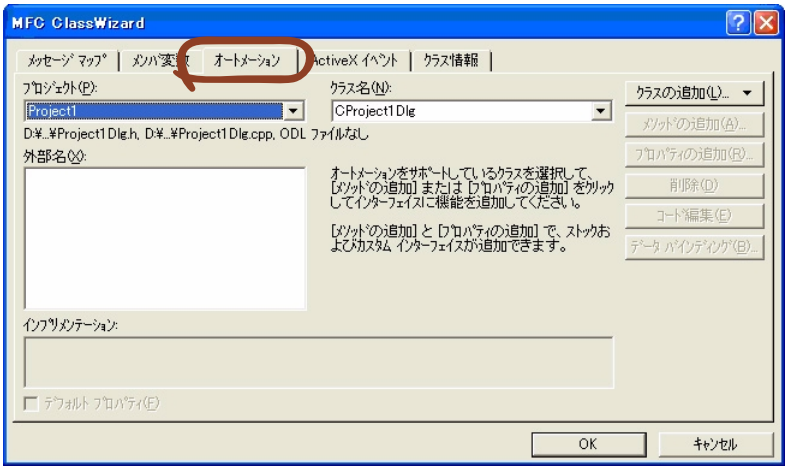
次の手順でAinix LabelStar Proタイプライブラリを追加します。

2.5.3.1　Microsoft Visual C++ 6.0の場合

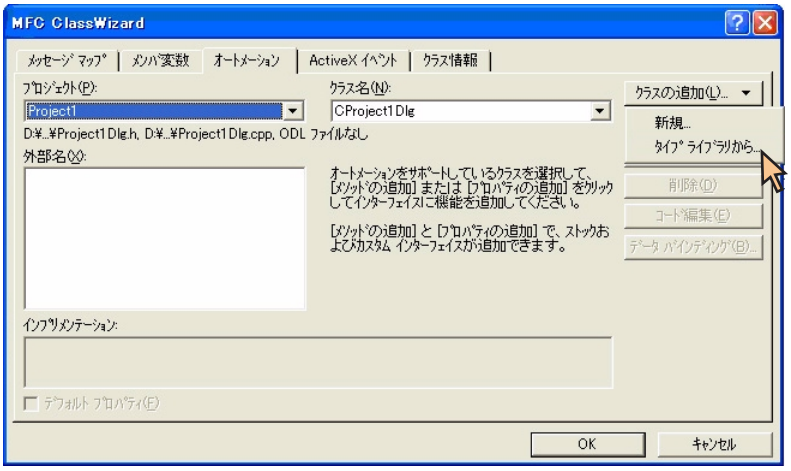
❶ Microsoft Visual C++ 6.0を起動します。

❷ ［表示］－［ClassWizard］を選択し、[MFC ClassWizard]ダイアログボックスを表示します。

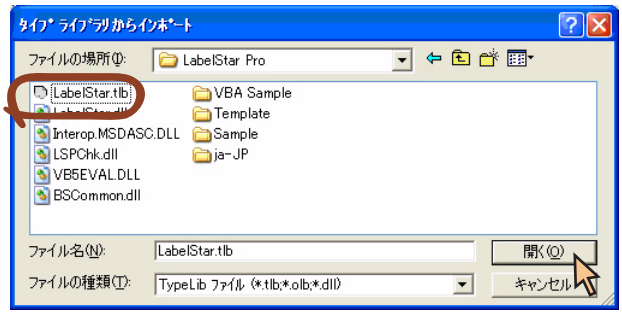
❸ [オートメーション]タブをクリックします。



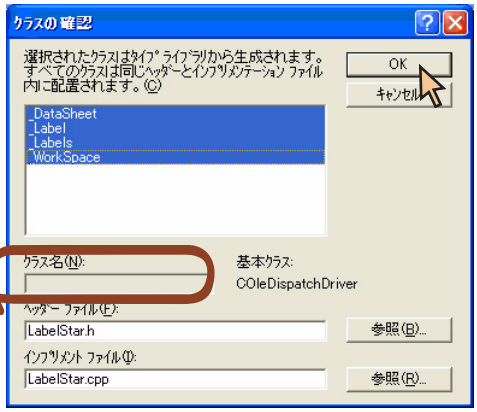
❹ [クラスの追加]をクリックすると表示されるリストボックスから、［タイプライブラリから］を選択します。
［タイプライブラリからインポート]ダイアログボックスが表示されます。



❺ **[タイプライブラリからインポート]ダイアログボックスより「LabelStar.tlb」を選択します。**
[クラスの確認]ダイアログボックスが表示されます。



❻ **[クラスの確認]ダイアログボックスで、任意のクラス名を設定後、[OK]をクリックします。**
「labelstar.h」ファイル、「labelstar.cpp」ファイルがプロジェクトに追加されます。



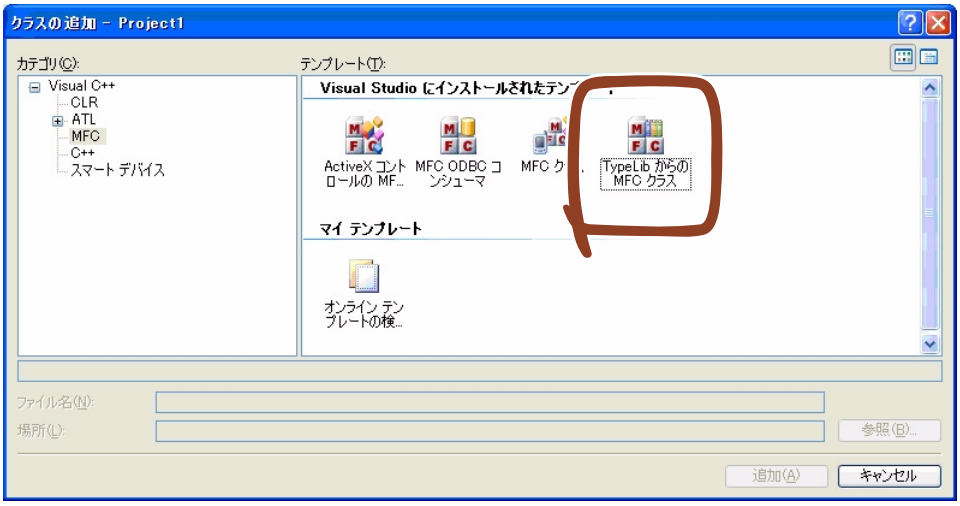
2.5.3.2　Microsoft Visual Visual C++ .NETの場合

❶ Microsoft Visual Visual C++ .NETを起動します。

❷ ［プロジェクト］－［クラスの追加］を選択し、［クラスの追加］ダイアログボックスを表示します。

❸ ［クラスの追加］ダイアログボックスで、［カテゴリ］から［MFC］を選択します。

❹ ［テンプレート］から［TypeLibからのMFCクラス］を選択します。



❺ ［追加］をクリックします。

［Typelibクラス追加ウィザード］ダイアログボックスが表示されます。

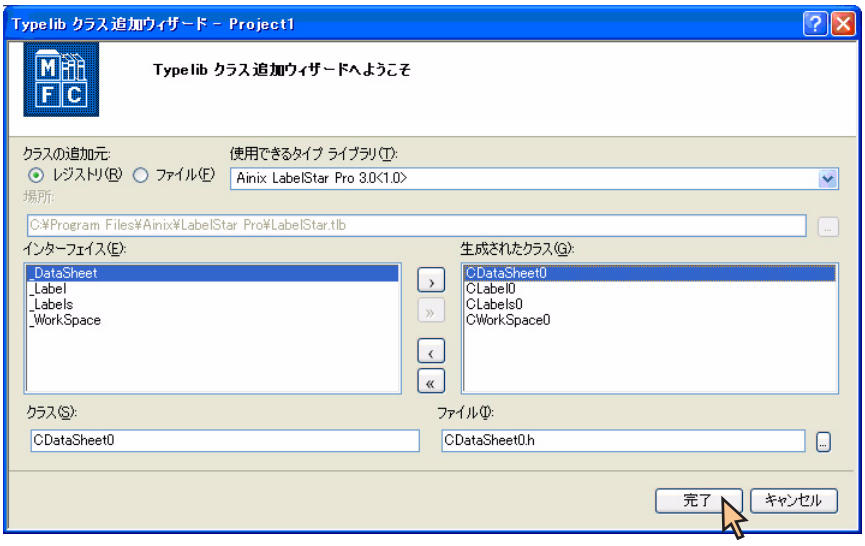
❻ ［Typelibクラス追加ウィザード］ダイアログボックスで［使用できるタイプライブラリ］から、［Ainix　LabelStar Pro 3.0<1.0>］を選択します。

❼ ［インタフェース］に「_DataSheet」、「_Label」、「_Labels」、「_WorkSpace」が表示されていることを確認後、「IBarcode」が選択された状態で、［>>］をクリックします。

［生成されたクラス］にタイプライブラリーの追加により作成されるクラス［CDataSheet］、［CLabel］、［CLabels］、［CWorkSpace］が表示されます。

❽ ［完了］をクリックします。

「CDataSheet.h」、「CLabel.h」、「CLabels.h」、「CWorkSpace.h」ファイルがプロジェクトに追加されます。



2.5.4 タイプライブラリのインクルード（コード記載による使用例）

2.5.4.1 Microsoft Visula C++ 6.0の場合

```
// Project1Dlg.h : ヘッダー ファイル
//
#include             "labelstar.h"

// CProject1Dlg ダイアログ

class CProject1Dlg : public CDialog
{
private:
  _WorkSpace         m_WorkSpace;
  _Labels            m_Labels;
  _Label             m_Label;
  _DataSheet         m_DataSheet;
}
```

2.5.4.2 Microsoft Visula C++ .NETの場合

```
// Project1Dlg.h : ヘッダーファイル
//
#include"CWorkSpace.h"
#include"CLabels.h"
#include"CLabel.h"
#include"CDataSheet.h"

// CProject1Dlg ダイアログ
class CProject1Dlg : public CDialog
{
private:
  CWorkSpace         m_WorkSpace;
  CLabels            m_Labels;
  CLabel             m_Label;
  CDataSheet         m_DataSheet;
};
```

2.5.5 Ainix BarStar Proでの帳票印刷

ここでは、Ainix BarStar ProのOLEオートメーション機能を使用してバーコードを生成・印刷する方法について説明します。

以下の印刷準備が完了していることを確認してください。

- プリンタドライバのインストール
  21 ページの「プリンタードライバーのインストール」を参照し、プリンタドライバがインストールされていることを確認してください。
- タイプライブラリの追加
  以下の項を参照し、タイプライブラリが追加されていることを確認してください。
  - Ainix BarStar Pro タイプライブラリの追加（22ページ）
  - タイプライブラリのインクルード（コード記載による使用例）（25ページ）

上記の準備完了後、次項のAinix BarStar Proバーコード生成・印刷の使用例のようにソースコードのコーディングを行います。

2.5.5.1 バーコードの印刷（コード記載による使用例）

以下にOLEオートメーション機能を使用し、バーコードを生成・印刷するサンプルを示します。

```
//変数宣言
CDC cdc;                                    //デバイスコンテキスト
VARIANT vRefDC;                             //Encodeメソッド第引数
VARIANT vPattern;                           //Encodeメソッド第引数

//印刷開始
if(StartDoc(cdc.m_hDC, &docInfo) <= 0)
{
   AfxMessageBox(_T("StartDoc処理に失敗しました。"));
   cdc.DeleteDC();
   return;
}
//1ページ分のジョブの印刷開始
if(StartPage(cdc.m_hDC) <= 0)
{
   AfxMessageBox(_T("StartPage処理に失敗しました。"));
   cdc.DeleteDC();
   return;
}

//バーコードオブジェクトの生成
m_Barcode.SetSymbology(“シンボルの種類”);
m_Barcode.SetText(“エンコードデータ”);
m_Barcode.SetElementSizeX(“バーの幅”);
m_Barcode.SetElementSizeY(“バーの高さ”);
m_Barcode.SetFontSize(“フォントサイズ”);
```

```
//Encodeメソッドの引数の指定
VariantInit(&vRefDC);
VariantInit(&vPattern);
vRefDC.byref = cdc.m_hDC;
vPattern.boolVal = FALSE;


//Encodeの実行
m_Barcode.Encode(vRefDC,vPattern);

//Encodeデータの描画
m_Barcode.Draw((long)cdc.m_hDC, XPOS_BAR, YPOS_BAR);

//1ページ分のジョブを印刷
if(EndPage(cdc.m_hDC) <= 0)
{
  AfxMessageBox(_T("EndPage処理に失敗しました。"));
  cdc.DeleteDC();
  return;
}

//印刷終了
if EndDoc(cdc.m_hDC) <= 0)
{
  AfxMessageBox(_T("Enddoc処理に失敗しました。"));
  cdc.DeleteDC();
  return;
}

//ハンドル開放
DeleteDC(cdc);
```

2.5.6 Ainix LabelStar Proでの帳票印刷

ここでは、Ainix LabelStar ProのOLEオートメーション機能を使用して帳票データ(lblファイル)を印刷する方法について説明します。

以下の印刷準備が完了していることを確認してください。

- プリンタドライバのインストール
  21 ページの「プリンタードライバーのインストール」を参照し、プリンタドライバがインストールされていることを確認してください。
- タイプライブラリの追加
  以下の項を参照し、Visual C++ の開発環境にタイプライブラリが追加されていることを
  - Ainix LabelStar Pro タイプライブラリの追加（26ページ）
  - タイプライブラリのインクルード（コード記載による使用例）（29ページ）
- 印刷する帳票データ(lblファイル)
  Ainix LabelStar ProのOLEオートメーション印刷では、予め印刷する帳票データ(lblファイル)を作成しておかなければなりません。
  Ainix LabelStar Proのマニュアル、ヘルプを参照し、帳票データを作成してください。

上記の準備完了後、次項の使用例のように帳票データ印刷のソースコードをコーディングします。

2.5.6.1 バーコードの印刷（コード記載による使用例）

以下にOLEオートメーション機能を使用し、Ainix LabelStar Pro の帳票データ(lblファイル)を印刷するサンプルを示します。

```
//変数宣言
COleException oleError;                    //OLEの例外
VARIANT       vIndex;                      //ラベルのインデックス番号
VARIANT       vLabels;                     //Labelsコレクション
VARIANT       vCopies;                     //印刷部数
VARIANT       vTestPrint;                  //テストプリントするorしない

//オブジェクト生成
m_WorkSpace.CreateDispatch(("LabelStar.Workspace"), & oleError);

//ファイルを開く
::VariantInit(&vIndex);
Index.vt = VT_EMPTY;
vLabels  = m_WorkSpace.Labels(vIndex);
m_Labels.AttachDispatch(vLabels.pdispVal);
m_Label.AttachDispatch(m_Labels.OpenLabel("ファイルパス"));

//DataSheetオブジェクトを取得
m_DataSheet.AttachDispatch(m_Label.DataSheet());

//印刷部数指定
::VariantInit(&vCopies);
vCopies.vt   = VT_I2;
vCopies.iVal = 1;
```

```
//テスト印刷する/しない
::VariantInit(&vTestPrint);
vTestPrint.vt   = VT_BOOL;
vTestPrint.iVal = FALSE;


//ラベルの印刷
m_Label.PrintOut(0, vCopies, vTestPrint);

//終了処理
m_DataSheet.ReleaseDispatch();

//ラベルのクローズ
m_Labels.CloseLabel(m_Label);
m_Label.ReleaseDispatch();
m_Labels.ReleaseDispatch();

//LabelStarProの終了
m_WorkSpace.Quit();
m_WorkSpace.ReleaseDispatch();
```

メ　モ

3章 NLPLコマンド送信ライブラリー

この章では、NLPLMC32.dllおよびNLPLMC32.dll.lib（以下、「NLPLMC32」と呼びます）に関して説明します。

3.1 概　要

NLPLMC32は、上位APからNEC Label Printer Languageコマンド（NLPL：プリンターが認識するコマンド）を直接、プリンターに送信する機能を提供します。
DLLの形式は、レジストリー登録が不要であり、公開インターフェース関数をエクスポートし、Win32APIのGetProcAddress()により関数アドレスが取得可能なダイナミックリンクライブラリー（DLL）とします。
Libの形式は、レジストリ登録が不要であり、上位APに組み込み可能とします。

チェック

NLPLコマンドに関する詳しい説明は、プリンターに添付のソフトウェアCD-ROMに収録されている「NLPLコマンドマニュアル」を参照してください。

3.1.1　動作環境

サポートOSを以下に示します。
OSの動作環境は、OSのReadme等で確認してください。

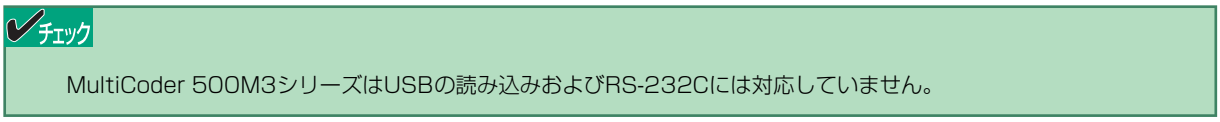
- Windows 7 日本語版
- Windows Vista 日本語版
- Windows XP 日本語版
- Windows 2000 日本語版
- Windows Server 2008 日本語版
- Windows Server 2008 R2 日本語版
- Windows Server 2003 日本語版
- Windows Server 2003 R2 日本語版

ハードウェア環境は、OSの動作環境に準じます。
64bit OSで使用する場合は、32bitモードで使用してください。

3.1.2　接続方法

NLPLMC32の動作をサポートする接続方法を以下に示します。

- LPT（書き込みのみ）
- USB（書き込み/読み込み）
- RS-232C（書き込みのみ）
- TCP/IP（書き込み/読み込み）



通信機能概要

接続方法	印　刷		通信方法	
	印刷データ送信	通信ポート （プリンタードライバーのポート）	送信	応答受信
LPT	プリンタードライバー	LPT	○	×
USB	プリンタードライバー	USB	○	○*1
RS-232C*1	プリンタードライバー	RS-232C	○	×
TCP/IP	プリンタードライバー	• Standard TCP/IP • NEC Print Server Port*1, *2	○	○

○：サポート
×：未サポート

＊1　MultiCoder 500M3 シリーズは、サポートしていません。

＊2　指定ポートが IPP の場合、NLPLMC32 とプリンター間の暗号化通信が可能です。

以下に、プリンターごとでサポートしているインターフェースを示します。

プリンターごとでサポートしているインターフェース

プリンター	接続方法			
	LPT	USB	RS-232C	TCP/IP
MultiCoder 500M3シリーズ	○	○	×	○
MultiCoder 502Lシリーズ	○	○	○	○

○：サポート
×：未サポート

3.1.3　プリンタードライバー

NLPLコマンド送信にプリンタードライバーを使用します。接続しているMultiCoderシリーズ用のプリンタードライバーを必ずインストールしてください。

3.1.4　プリンター

対応するプリンターを以下に示します。

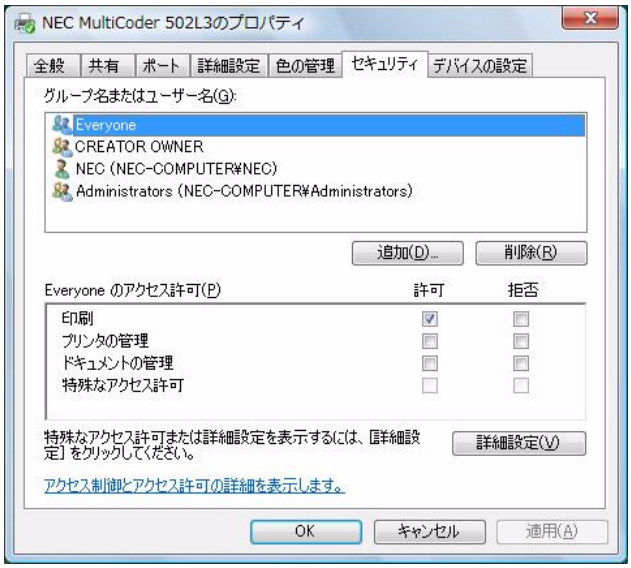
- MultiCoder 500M3シリーズ
- MultiCoder 502Lシリーズ

3.1.5　運用上の制限

- NLPLMC32のAPIは同期関数であり、処理終了まで上位APに処理が戻りません。
- 上位APから同時に複数のプリンターを制御する場合、上位APでのスレッド処理等が必要になります。
- プリンタードライバーを一時停止にした状態で、NLPLMC32のAPIをコールしないでください。

> チェック
>
> プリンタードライバーを一時停止にした状態で、APIをコールすると、NLPLMC32で通信タイムアウトが発生します。

- USER権限で使用する場合は、権限を与える必要があります。プリンタードライバーの［セキュリティ］シートから［プリンタの管理］を許可してください。



3.1.6　モジュール名

NLPLMC32は、ユーザープログラムからAPIで呼び出され、NLPLコマンドを送信するモジュールとして動作します。NLPLMC32のモジュール名を以下に示します。

- NLPLMC32.dll
- NLPLMC32.lib
- NLPLMC32.h

3.2 API関数

以下のAPI関数をサポートします。

関数名	名　称	機能説明	参照先	備考
NLPLPrinterOpen	プリンター接続処理	指定した論理プリンターに接続する際にコールします。 処理が成功すると、プリンタードライバーのハンドルを取得します。	40ページ	LAN以外
NLPLSetJobName	NLPL コマンド送信ジョブ名設定処理	印刷ジョブのジョブ名を指定します。 NLPL コマンドを送信する際のジョブ名を設定する際にコールします。	41ページ	
NLPLSetCommand	NLPL コマンド生成処理	NLPLコマンドを生成します。 NLPL コマンドを生成する際にコールします。本APIにて生成されたNLPLコマンドは、NLPL コマンド送信処理(NLPLSendCommand)時にプリンターへ送信されます。	42ページ	
NLPLReceiveCommand	プリンター応答受信処理	プリンターからの応答を受信し、処理結果を上位APに返します。 プリンター応答を受信する際にコールします。NLPL コマンド送信処理(NLPLSendCommand)にて送信されたNLPLコマンドにプリンター応答がある場合などに本APIにてプリンター応答を取得します。	43ページ	
NLPLSendCommand	NLPL コマンド送信処理	NLPLSetCommandにて設定したNLPL コマンドを送信します。 NLPLコマンドを送信する際にコールします。上位APにて生成されたNLPL コマンドをプリンターに送信します。	44ページ	
NLPLPrinterColse	プリンター接続終了処理	論理プリンターとの接続を終了します(プリンターハンドルの開放)。	45ページ	
NLPLPrinterOpenL	プリンター接続処理	設定したネットワークプリンター名/IPアドレスおよびソケットポート番号に対応するプリンターに接続します。 指定したプリンターに接続する際にコールします。ソケット（Socket Despriptor）を取得します。	46ページ	LAN用
NLPLSendCommandL	NLPL コマンド送信処理	NLPLコマンドを生成します。続いて、NLPLコマンドをプリンターへ送信します。 NLPLコマンドを送信する際にコールします。上位APから転送されたNLPL コマンドをプリンターに送信します。プリンターからの応答を受信し、処理結果を上位APに返します。NLPL コマンド送信処理は任意のタイミングで実行可能です。	47ページ	
NLPLReceiveCommandL	NLPL コマンド送信処理	プリンター応答を受信し、処理結果を上位APに返します。 NLPLコマンドを送信する際にコールします。上位APから転送されたNLPL コマンドをプリンターに送信します。プリンターからの応答を受信し、処理結果を上位APに返します。NLPL コマンド送信処理は任意のタイミングで実行可能です。	48ページ	
NLPLPrinterCloseL	NLPL コマンド送信処理	プリンターとの接続を終了します。	49ページ	

3.2.1　NLPLPrinterOpen

指定した論理プリンターのハンドルを取得します。

書式

```
HANDLE NLPLPrinterOpen(
  LPSTR             pPrinterName);
```

引数

```
LPSTR             pPrinterName      //論理プリンター名
```

戻り値

処理成功（プリンターハンドル）、その他（マイナス値）

チェック

関数の引数に不正値が確認された場合や、関数内部でエラーが発生した場合は、マイナス値を返します。詳細は50 ページの「関数戻り値一覧」を参照してください。

設定値

pPrinterName：論理プリンター名

NLPLコマンドを送信する論理プリンターの名称を指定します。
空文字列、またはNULLが設定されると、NLPLMC32の内部エラーとして上位APに処理を戻します。
設定可能な文字数は2～220（1バイト文字）です。

3.2.2　NLPLSetJobName

印刷ジョブのジョブ名を指定します。ジョブ名は必ず指定する必要があります。

書式

```
int NLPLSetJobName(
  HANDLE            hPrinter,
  LPSTR             pJobName);
```

引数

```
HANDLE              hPrinter,         //プリンターハンドル
LPSTR               pJobName          //ジョブ名
```

戻り値

処理成功（0）、その他（マイナス値）

チェック

関数の引数に不正値が確認された場合や、関数内部でエラーが発生した場合は、マイナス値を返します。詳細は50 ページの「関数戻り値一覧」を参照してください。

設定値

hPrinter：プリンターハンドル

NLPLコマンドを送信するプリンターのハンドルを指定します。
NLPLPrinterOpenにて取得したプリンターハンドルを本設定値として指定します。

pJobName：ジョブ名

ジョブ名を指定します。
NLPLSendCommandにてNLPLコマンド送信時のジョブ名として本設定値が使用されます。
設定可能な文字数は2～220（1バイト文字）です。

3.2.3 NLPLSetCommand

NLPLコマンドを生成します。
本APIで生成されたNLPLコマンド文字列は、NLPLSendCommandをコールすることでプリンターに送信されます。

チェック

NLPLコマンドはバイナリーとして指定してください。

書式

```
int NLPLSetCommand(
  HANDLE            hPrinter,
  DWORD             dwInDataSize,
  LPBYTE            pInData);
```

引数

```
HANDLE            hPrinter,          //プリンターハンドル
DWORD             dwInDataSize,      //NLPLコマンドサイズ
LPBYTE            pInData            //NLPLコマンド
```

戻り値

処理成功（0）、その他（マイナス値）

チェック

関数の引数に不正値が確認された場合や、関数内部でエラーが発生した場合は、マイナス値を返します。詳細は50 ページの「関数戻り値一覧」を参照してください。

設定値

hPrinter：プリンターハンドル
NLPLコマンドを送信するプリンターのハンドルを指定します。
NLPLPrinterOpenにて取得したプリンターハンドルを本設定値として指定します。

dwInDataSize：NLPLコマンドサイズ
pInDataのバイト数を指定します。
pInDataで指定されたポインターからdwInDataSizeのバイト数だけ、プリンターに送信します。設定できる値の範囲は0～65535です。

pInData：NLPLコマンド
上位APからは出力コマンド文字列のデータをバイトのポインターとして受け取ります。
指定されたバイトデータは、プリンターにそのまま送信されます。
NULLの場合、NLPLMC32の内部エラーとして上位APに処理を戻します。

3.2.4　NLPLReceiveCommand

プリンターからの応答を受信し、処理結果を上位APに返します。

書式

```
int NLPLReceiveCommand(
  HANDLE            hPrinter,
  DWORD             dwOutDataSize,
  LPBYTE            pOutData);
```

引数

```
HANDLE            hPrinter,          //プリンターハンドル
DWORD             dwOutDataSize      //プリンター応答データサイズ
LPBYTE            pOutData           //プリンター応答データ
```

戻り値

処理成功（0）、その他（マイナス値）

チェック

関数の引数に不正値が確認された場合や、関数内部でエラーが発生した場合は、マイナス値を返します。詳細は50 ページの「関数戻り値一覧」を参照してください。

設定値

`hPrinter`：プリンターハンドル

NLPLコマンドを送信するプリンターのハンドルを指定します。
`NLPLPrinterOpen()`にて取得したプリンターハンドルを本設定値として指定します。

`dwOutDataSize`：プリンター応答データサイズ

上位APで確保した`pResponse`のサイズをバイト単位で設定します。
プリンターからの応答文字列のバイト数が`dwOutDataSize`より大きい場合、`pOutData`には何も書き込まずにNLPLMC32の内部エラーとして上位APに処理を戻します。
設定できる値の範囲は0～1024です。

チェック

`pOutData`はNLPLコマンドの応答データのバイト数以上のメモリ領域を、上位APで確保する必要があります。

出力情報

`pOutData`：プリンター応答データ

プリンターからの応答データをそのまま出力します。

チェック

`NLPLSendCommand`にてプリンターからの装置応答がないNLPLコマンドを送信した場合は、本APIをコールしないでください。

3.2.5　NLPLSendCommand

NLPLSetCommandにて設定したNLPLコマンドを送信します。

書式

```
int NLPLSendCommand(
  HANDLE              hPrinter);
```

引数

```
HANDLE              hPrinter          //プリンターハンドル
```

戻り値

処理成功（0）、その他（マイナス値）

チェック

関数の引数に不正値が確認された場合や、関数内部でエラーが発生した場合は、マイナス値を返します。詳細は50 ページの「関数戻り値一覧」を参照してください。

設定値

hPrinter：プリンターハンドル

NLPLコマンドを送信するプリンターのハンドルを指定します。

NLPLPrinterOpen()にて取得したプリンターハンドルを本設定値として指定します。

3.2.6　NLPLPrinterColse

論理プリンターとの接続を終了します。

書式

```
int NLPLPrinterColse(
  HANDLE              hPrinter);
```

引数

```
HANDLE              hPrinter          //プリンターハンドル
```

戻り値

処理成功（0）、その他（マイナス値）

チェック

関数の引数に不正値が確認された場合や、関数内部でエラーが発生した場合は、マイナス値を返します。詳細は50 ページの「関数戻り値一覧」を参照してください。

設定値

hPrinter：プリンターハンドル

NLPLコマンドを送信するプリンターのハンドルを指定します。

NLPLPrinterOpenにて取得したプリンターハンドルを本設定値として指定します。

3.2.7　NLPLPrinterOpenL

設定したネットワークプリンター名/IPアドレスおよびソケットポート番号に対応するプリンターに接続します。

書式

```
SOCKET NLPLPrinterOpenL(
  LPSTR             pPrinterName,
  LPSTR             pHostName,
  LPSTR             pSocketPort);
```

引数

```
LPSTR             pPrinterName      //論理プリンター名
LPSTR             pHostName         //ネットワークプリンター名/IPアドレス
LPSTR             pSocketPort       //ソケットポート番号
```

戻り値

処理成功（ソケット番号(Soket Descriptor)）、その他（エラー値）

チェック

関数の引数に不正値が確認された場合や、関数内部でエラーが発生した場合は、マイナス値を返します。詳細は50 ページの「関数戻り値一覧」を参照してください。

戻り値

処理成功（ソケット）、その他（マイナス値）

チェック

関数の引数に不正値が確認された場合や、関数内部でエラーが発生した場合は、マイナス値を返します。詳細は50 ページの「関数戻り値一覧」を参照してください。

設定値

`pPrinterName`：論理プリンター名
　NLPLコマンドを送信する論理プリンターの名称を指定します。
　空文字列、またはNULLが設定されると、NLPLMC32の内部エラーとして上位APに処理を戻します。
　設定可能な文字数は2～220（1バイト文字）です（論理プリンター名の入力可能文字数が220文字になります）。

`pHostName`：ネットワークプリンター名/IPアドレス
　ネットワークプリンター名またはIPアドレスを設定します。
　空文字列、またはNULLが設定されると、NLPLMC32の内部エラーとして上位APに処理を戻します。

`pSocketPort`：ソケットポート番号
　ソケットポート番号を設定します。
　設定できる値の範囲は0～65535です。

3.2.8　NLPLSendCommandL

NLPLコマンドを生成し、LANで接続されているプリンターに送信します。

チェック

NLPLコマンドはバイナリーとして指定してください。

書式

```
int NLPLSendCommandL(
  SOCKET            Socket,
  DWORD             dwInDataSize,
  LPBYTE            pInData);
```

引数

```
SOCKET            Socket            //ソケット（Soket Descriptor）
DWORD             dwInDataSize      //NLPLコマンドサイズ
LPBYTE            pInData           //NLPLコマンド
```

戻り値

処理成功（0）、その他（マイナス値）

チェック

関数の引数に不正値が確認された場合や、関数内部でエラーが発生した場合は、マイナス値を返します。詳細は50 ページの「関数戻り値一覧」を参照してください。

設定値

`Socket`：ソケット（Soket Descriptor）

ソケット（Soket Descriptor）を指定します。

`NLPLPrinterOpenL()`の戻り値を本値に指定します。

`dwInDataSize`：NLPLコマンドサイズ

`lpInData`のバイト数を指定します。

`lpInData`で指定されたポインターから`dwInDataSize`のバイト数だけ、プリンターに送信します。

設定できる値の範囲は0～65535です。

`pInData`：NLPLコマンド

上位APからは出力コマンド文字列のデータをバイトのポインターとして受け取ります。

指定されたバイトデータは、プリンターにそのまま送信されます。

NULLの場合、NLPLMC32の内部エラーとして上位APに処理を戻します。

3.2.9　NLPLReceiveCommandL

プリンター応答を受信し、処理結果を上位APに返します。

書式

```
int NLPLReceiveCommandL(
  SOCKET            Socket,
  DWORD             dwOutDataSize,
  LPBYTE            pOutData);
```

引数

SOCKET	Socket	//ソケット（Soket Descriptor）
DWORD	dwOutDataSize	//NLPLコマンドの応答データサイズ
LPBYTE	pOutData	//NLPLコマンドの応答データ

戻り値

処理成功（0）、その他（マイナス値）

チェック

関数の引数に不正値が確認された場合や、関数内部でエラーが発生した場合は、マイナス値を返します。詳細は50 ページの「関数戻り値一覧」を参照してください。

設定値

Socket：ソケット（Soket Descriptor）
　ソケット（Soket Descriptor）を指定します。
　NLPLPrinterOpenL()の戻り値を本値に指定します。

dwOutDataSize：NLPLコマンドの応答データサイズ
　上位APで確保したlpOutDataのサイズをバイト単位で設定します。
　プリンターからの応答文字列のバイト数がdwOutDataSizeより大きい場合、pOutDataには何も書き込まずにNLPLMC32の内部エラーとして上位APに処理を戻します。
　設定できる値の範囲は0～1024です。

チェック

pOutDataはNLPLコマンドの応答データのバイト数以上のメモリ領域を、上位APで確保する必要があります。

出力情報

pOutData：NLPLコマンドの応答データ
　プリンターからの応答データを、そのまま出力します。

チェック

NLPLSendCommandLにてプリンターからの装置応答がないNLPLコマンドを送信した場合は、本APIをコールしないでください。

3.2.10　NLPLPrinterCloseL

プリンターとの接続を終了します。

書式

```
int NLPLPrinterCloseL(
  SOCKET            Socket);
```

引数

```
SOCKET              Socket              //ソケット（Soket Descriptor）
```

戻り値

処理成功（0）、その他（マイナス値）

チェック

関数の引数に不正値が確認された場合や、関数内部でエラーが発生した場合は、マイナス値を返します。詳細は50 ページの「関数戻り値一覧」を参照してください。

設定値

Socket：ソケット（Soket Descriptor）
　ソケット（Soket Descriptor）を指定します。
　NLPLPrinterOpenL()の戻り値を本値に指定します。

3.2.11　関数戻り値一覧

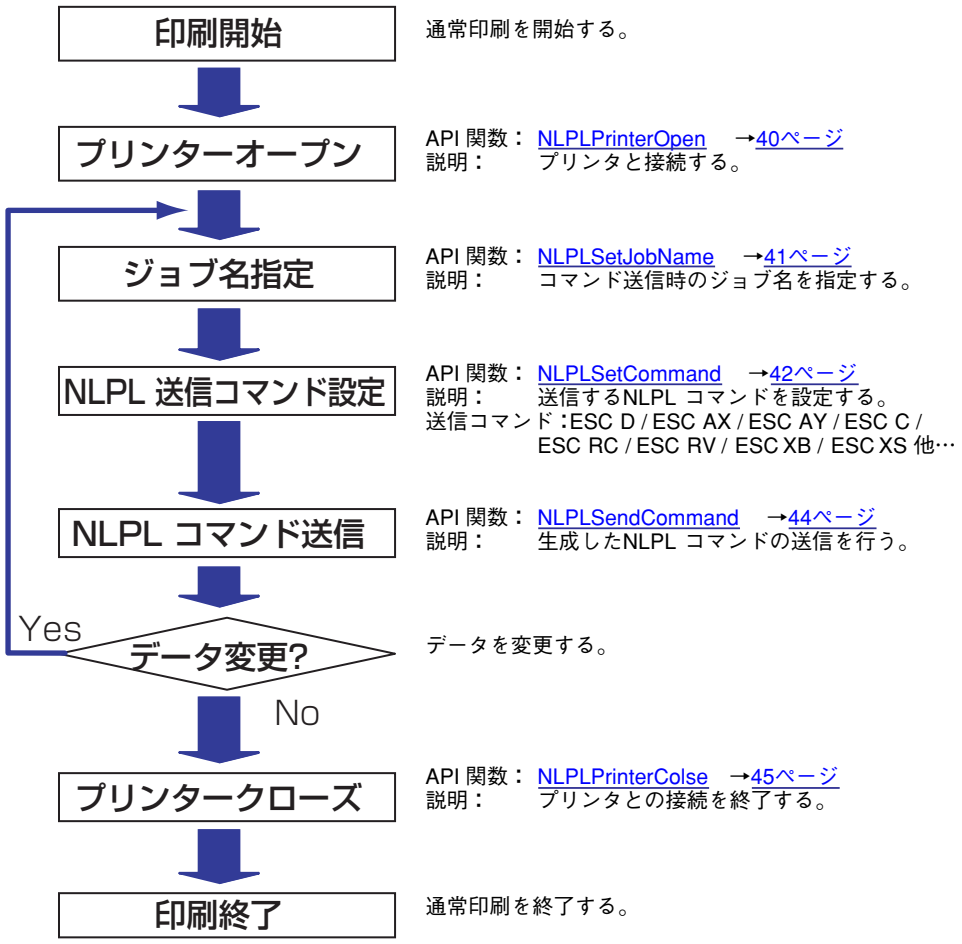
関数は処理成功時、「0」を戻します。
関数の引数に不正値が確認された場合や、関数内部でエラーが発生した場合は、下表に示す値を戻します。

戻り値	定義名	エラー発生原因
0	NLPL_SUCCESS	処理成功
-10	ERROR_PRINTERHANDLE_NULL	パラメーターエラー（プリンターハンドルがNULL）
-20	ERROR_SOCKETPORT_NULL	パラメーターエラー（ソケットがNULL）
-30	ERROR_PRINTERMODEL	パラメーターエラー（プリンターモデル名不正）
-100	ERROR_NLPL_PRINTEROPEN	関数エラー（NLPLPrinterOpen）
-101	ERROR_PRINTERNAME_NULL	パラメーターエラー（プリンタ名がNULL）ー
-102	ERROR_PRINTERNAME_BLANK	パラメーターエラー（プリンタ名が空文字）ー
-103	ERROR_PRINTERNAME_FAILSET	パラメーターエラー（プリンター名に指定不可の文字がある）
-110	ERROR_NLPL_SETJOBNAME	関数エラー（NLPLSetJobName）
-111	ERROR_JOBNAME_NULL	パラメーターエラー（ジョブ名がNULL）
-112	ERROR_JOBNAME_BLANK	パラメーターエラー（ジョブ名が空文字）
-113	ERROR_JOBNAME_FAILSET	パラメーターエラー（ジョブ名に指定不可の文字がある）
-120	ERROR_NLPL_SET_COMMAND	関数エラー（NLPLSetCommand）
-121	ERROR_INDATA_NULL	パラメーターエラー（NLPLコマンドがNULL）
-122	ERROR_INDATA_BLANK	パラメーターエラー（NLPLコマンドが空文字）
-124	ERROR_INDATA_OVERSIZE	パラメーターエラー（NLPLコマンドが指定したNLPLコマンドサイズを超えている）
-125	ERROR_INDATASIZE_OUTOF	パラメーターエラー（NLPLコマンドサイズが範囲外）
-130	ERROR_NLPL_SENDCOMMAND	関数エラー（NLPLSendCommand）
-140	ERROR_NLPL_RECEIVECOMMAND	関数エラー（NLPLReceiveCommand）
-145	ERROR_OUTDATASIZE_OUTOF	パラメーターエラー（受信データサイズが範囲外）
-150	ERROR_NLPL_PRINTERCLOSE	関数エラー（NLPLPrinterClose）
-200	ERROR_NLPL_PRINTEROPENL	関数エラー（NLPLPrinterOpenL）
-201	ERROR_HOSTNAME_NULL	パラメーターエラー（ホスト名がNULL）
-202	ERROR_HOSTNAME_BLANK	パラメーターエラー（ホスト名が空文字）
-205	ERROR_SOCKETPORTSIZE_OUTOF	パラメーターエラー（ソケットポート番号が範囲外）
-210	ERROR_NLPL_SENDCOMMANDL	関数エラー（NLPLSendCommandL）
-211	ERROR_SOCKETINDATA_NULL	パラメーターエラー（NLPLコマンドがNULL）
-212	ERROR_SOCKETINDATA_BLANK	パラメーターエラー（NLPLコマンドが空文字）
-214	ERROR_SOCKETINDATA_OVERSIZE	パラメーターエラー（NLPLコマンドが指定したNLPLコマンドサイズを超えている）
-215	ERROR_SOCKETINDATASIZE_OUTOF	パラメーターエラー（NLPLコマンドサイズが範囲外）
-220	ERROR_NLPL_RECEIVECOMMANDL	関数エラー（NLPLReceiveCommandL）
-225	ERROR_SOCKETOUTDATA_NULL	パラメーターエラー（受信データサイズが範囲外）
-230	ERROR_NLPL_PRINTERCLOSEL	関数エラー（NLPLPrinterCloseL）
-315	ERROR_BIDIRECTIONAL	双方向サポートが無効になっている
-320	ERROR_COMMUNICATION	LAN通信エラー
-900	ERROR_OTHERS	その他エラー

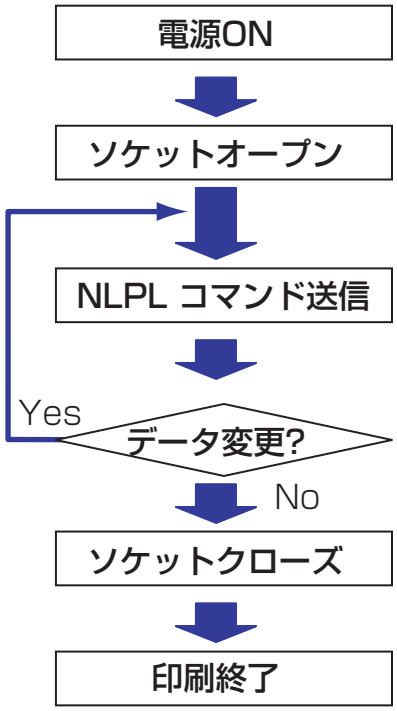
3.2.12　APIコールによるNLPLコマンド送信例

NLPLコマンドの送信例を以下に示します。

3.2.12.1　印刷（LAN接続以外）



3.2.12.2　印刷（LAN接続）



通常印刷を開始する。

API 関数： NLPLPrinterOpenL　→46ページ
説明：　プリンタに接続する。

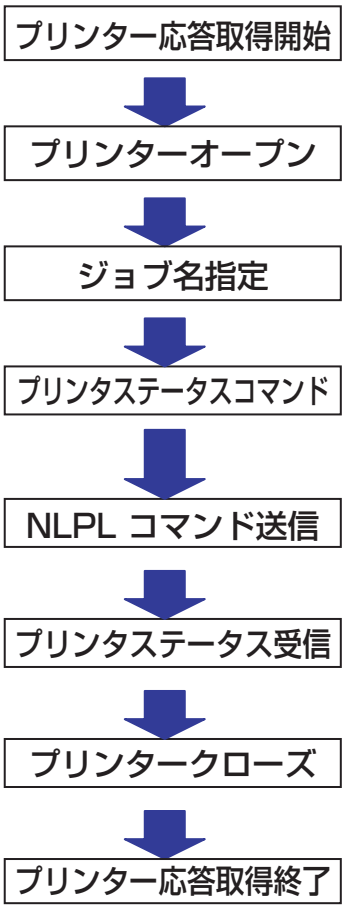
API 関数： NLPLSendCommandL　→47ページ
説明：　NLPL コマンドの設定、送信を行う。
送信コマンド：ESC D / ESC AX / ESC AY / ESC C /
ESC RC / ESC RV / ESC XB / ESC XS 他…

データを変更する。

API 関数： NLPLPrinterCloseL　→49ページ
説明：　プリンターとの接続を終了する。

通常印刷を終了する。

3.2.12.3　プリンター応答受信（USB接続）



プリンター応答取得処理を開始する。

API 関数： NLPLPrinterOpen　→40ページ
説明：　プリンターと接続する。

API 関数： NLPLSetJobName　→41ページ
説明：　コマンド送信時のジョブ名を指定する。

API 関数： NLPLSetCommand　→42ページ
説明：　プリンターステータス取得コマンド
送信コマンド :ESC WS

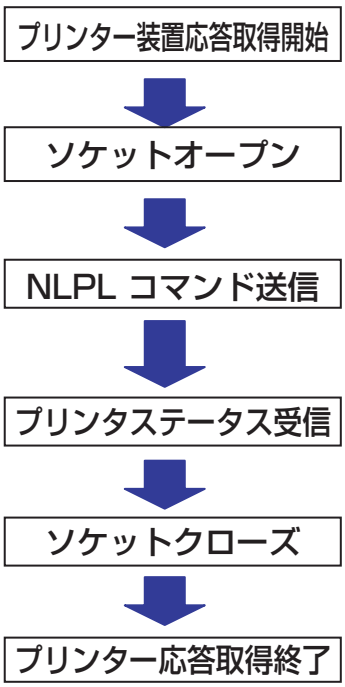
API 関数： NLPLSendCommand　→44ページ
説明：　生成したNLPL コマンドの送信を行う。

API 関数： NLPLReceiveCommand　→43ページ
説明：　プリンター応答を受信する。

API 関数： NLPLPrinterColse　→45ページ
説明：　プリンターとの接続を終了する。

プリンター応答取得処理を終了する。

3.2.12.4　プリンター応答受信（LAN接続）



プリンター応答取得処理を開始する。

API 関数：NLPLPrinterOpenL　→46ページ
説明：　プリンターに接続する。

API 関数：NLPLSetCommand　→42ページ
説明：　プリンターステータス取得コマンド。
送信コマンド：ESC WS

API 関数：NLPLReceiveCommandL　→48ページ
説明：　プリンター応答を受信する。

API 関数：NLPLPrinterCloseL　→49ページ
説明：　プリンターとの接続を終了する。

プリンター応答取得処理を終了する。