マイクロコンピュータ と ワークステーション 51 － 1
（１９８８．６．２８）

日本語デスクトップパブリッシングのためのＯＳ マイクロソフトウインドウズ

北野孝夫　トーマスポーリ　中島聡
マイクロソフト株式会社　研究開発部

マイクロソフトウィンドウズ（ Microsoft Windows ）は，必ずしもデスクトップパブリッシング（ＤＴＰ）のみを意識して作られたＯＳではない．しかし，そういったアプリケーションソフトウエアを数多く走らせることなるだろうことは予測して，設計されていることは事実である．ここでは，マイクロソフトウインドウズ（ＭＳ－ウインドウズ）の概略にふれたのち，日本語デスクトップパブリッシングのアプリケーションと関連が深い，ＭＳ－ウインドウズにおけるフォントの取扱いと日本語入力の仕組みについて述べる．

Microsoft Windows as the Operating System for Japanese Desk Top Publishing Applications

Takao Kitano , Thomas Paule and Satoshi Nakajima

Microsoft Co., Ltd.

Microsoft Co., Ltd.
Sanbancho Yayoi-kan 6-2 Sanbancho
Chiyoda-ku, Tokyo 102 Japan

Microsoft Windows is not necessarily an operating system environment solely intended for Desk Top Publishing, though, it is true that it was designed knowing that it would have to run several applications of that sort. After a brief intorduction on MS-Windows, I will focus on font handling and Kanji character input system that is closely related to Japanese Desk Top Publishing Applications.

1．はじめに

ＤＴＰシステムが高価な専用しシステムから，企業の一部門，研究室，あるいは個人といった規模で使用されるようになりつつある．　この背景には，高機能なハードウェア機器の価格が下がってきたということもあるし，一般のオフィスで使われるいわゆるワープロで作成できる文書の品質が次第にあがり，かつそれと同時に使用する側の求める品質も高くなって来たということもある．このような中で，汎用機でありながらＤＴＰシステムに適したＯＳ／システムというものが登場してきた．比較的高価な専用システムのみのものだったＤＴＰを，一気に一般ユーザーのレベルで使用可能な低価格なものにしたのがＤＴＰに関してこれらが果たした役割である．

ここで注目すべきは，これらのシステムがハードウェアの機能というよりも基本ソフトウェアであるＯＳの機能によりＤＴＰの実現をサポートとしていることである．これらのマシンの代表的なものをあげるとアップル社のマッキントシュ，ソニーのＮＥＷＳといったものが挙がる．

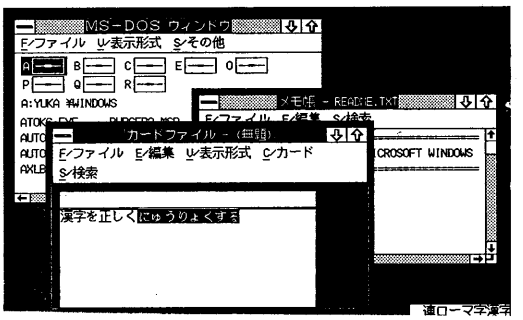
そして今回我々が取り上げるＭＳ－ウインドウズも，そういった個人ユーザーを対象としたマシン上で動作し，本来汎用的な使用を目的として作られたマシン上でＤＴＰシステムの構築をサポートするＯＳ環境である．本稿ではＭＳ－ウインドウズを例にとり，ＤＴＰシステムを構築するために必要なＯＳの条件とは何かといったことをＯＳの開発サイドから述べてゆきたい．

2．MS－ウインドウズ

2．1 MS－ウインドウズとは

マイクロソフトが開発した　ＭＳ－ウインドウズは，ＭＳ－ＤＯＳが動作するパーソナルコンピュータ上で動くように作られたウィンドウシステムである．１９８６年に最初のバージョンである１．０３を発表し，最新のバージョンは２．０となっている．

ＭＳ－ウインドウズはハードウェアとして，６４０Ｋｂｙｔｅのメモリ／インテル８６系ＣＰＵ／ビットマップディスプレイ／マウスを想定している．図２．１はＭＳ－ウインドウズ２．０の画面である．



［図２．１　ＭＳ－ウインドウズ］

2．2　ＭＳ－ウインドウズの目的

ＭＳ－ＤＯＳが，パーソナルコンピュータにおいて標準的なＯＳとなるにつれて次に示すような問題が出現した．ＭＳ－ウインドウズはこれらの問題点を解決することを目的として開発された．

１）アプリケーションが機種に依存して書かれている．

２）アプリケーション相互の関連が貧弱である．

３）複数のアプリケーションを同時に実行する事ができない．

４）マンマシンインターフェイスがアプリケーションごとに異なる．

2．3　ＭＳ－ウインドウズの機能

前項にてのべた問題点を踏まえ，ＭＳ－ウインドウズではつぎのような機能を提供して問題の解決をはかっている．

ハードウェアの仮想化

マルチタスク機能

オブジェクト指向のＡＰＩ

データフォーマットの共通化

グラフィカルユーザーインターフェイス

メモリ管理機能

この節ではＭＳ－ウインドウズのこれらの機能について順次述べていく．

2．3．1．ハードウェアの仮想化

アプリケーションがハードウェアに依存しないで作成されるようにするには，まずハードウェアの様々な機能を仮想化する必要がある．ＭＳ－ウインドウズではタイマー／ブロックデバイス／表示デバイス（ＣＲＴスクリーン，プリンタ）／キーボード／ポインティングデバイス／通信ポート／サウンド機能の７つを仮想化した．アプリケーションはウインドウシステムに対するファンクションコールを通してこれらの仮想化したハードウェア資源にアクセスする．ウインドウシステムはデバイスドライバーを呼び出し，デバイスドライバーが実際のアクセスを行なう．デバスドライバには６種類あり，それぞれ次に述べる役割を持つ．

１）システムドライバ

タイマー割り込みの設定，ブロックデバイスの情報の通知，コプロセッサ情報の通知

２）ディスプレイドライバ／プリンタドライバ

スクリーン及びプリンタ，プロッタへの出力

３）キーボードドライバ

キーボード入力の仮想キーへの変換と仮想キーのＡＳＣＩＩコードへの変換

４）マウスドライバ

マウス，ライトペンなどポインティングデバイスの入力処理

５）サウンドドライバ

ビープ音及び，各種サウンド機能の出力処理

６）コミュニケーションドライバ

シリアルポート／パララルポートにおける入出力処理

なおウインドウシステムの本体はハードウェアに依存しないＭＳ－ＤＯＳ上の実行ファイルであり，ハードウェアに依存した部分はすべてデバイスドライバーのモジュールに閉じ込められ，ＯＳ／２と同じダイナミックリンキングによりリンクされる．これにより，異機種間の移植に際しての手間が，非常に小さいものとなっている．

２．３．２．イベントドリブン方式のマルチタスク

ハードウェアの仮想化に伴って，アプリケーション間でハードウェアリソースを無秩序に取り合うといったことがなくなり，マルチタスクの実現が容易になった．ＭＳ－ウインドウズのマルチタスクは non-preemptive なものである．タスクの切り替えは，タスクがイベント待ちのファンクションコールをウインドウシステムに発行した時のみに起こる．そしてタスクがイベントを受け取った時点で，そのタスクの実行が再開される．イベントにはキーボード，マウス，タイマーなどの入力イベントのほかにウインドウシステムからの各種リクエスト（再描画要求など）および各種情報（ウインドウサイズの変更，ウインドウ位置の変更），他のアプリケーションからのタスク間メッセージなどがある．ＭＳ－ウインドウズ上の各タスクはこのような様々なイベントを受取りそれに応じた動作を行なうという，イベントドリブン方式のマルチタスクで動作する．

ここでアプリケーションを作成する上で重要なのは，ほとんどのイベント（ＭＳ－ウインドウズでは通常メッセージと呼んでいる）は，アプリケーションで処理する必要がないということである．アプリケーションは受け取るメッセージのほとんどを，ウインドウシステムにあらかじめ用意されているデフォルトのメッセージ処理関数を呼ぶことで処理することができる．アプリケーションが自分で処理しなければならないメッセージは，そのメッセージに対応してアプリケーション独自の処理を行ないたい場合に限られる．これは逆にアプリケーションが独自の処理を行ないたい場合は，いくらでも柔軟に対応が可能であるということでもある．

２．３．３．　オブジェクト指向のアプリケーションインターフェイス

このイベントドリブン方式のマルチタスクはオブジェクト指向のアプリケーションインターフェイスを実現する上で非常に効果的である．ＭＳ－ウインドウズでは全てのウインドウはある特定のクラスに属している．クラスというのは，それに属するウインドウの動作，形状を規定するもので，各アプリケーションで作成してシステムに登録するものと，あらかじめシステムに用意されているものがある．

それぞれのクラスはそのクラスのウインドウ関数というものを持つ．このウインドウ関数というのは，そのクラスに属するウインドウの動作を規定する関数であり，さきに述べたイベントはメッセージの形をとりこのウインドウ関数へと送られ処理される．例えば画面上のあるウインドウに（例えばアイコンウインドウ）に対してユーザーが何かした（例えばマウスでクリックした）という情報は，そのウインドウが属するクラスのウインドウ関数へと送られ，適切な処理が行なわれる．

アプリケーションプログラムの記述は通常，最低一つのクラスとそのクラスのウインドウ関数を記述して，そのクラスに属するものとしてウインドウを作成しウインドウの動作を規定するという形でおこなう．しかし，システムにあらかじめ用意されているクラス（例えばボタン，スクロールバー，リストボックスなど）に属するウインドウを作成したり，既存のクラスのウインドウ関数に変更／追加を加えて（これをウインドウのサブクラッシングと呼んでいる），既存のクラスをベースにして新たな動作をおこなうウインドウを作成したりするのも可能であり，柔軟なプログラムが可能となっている．

２．３．４．データフォーマットの共通化

ＭＳ－ウインドウズではアプリケーション間でのデータ通信の方法として，メッセージ通信，共有メモリ，クリップボード，ダイナミックデータエクスチェンジの４つを用意し，上に述べたマルチタスク環境でのタスク間通信を可能としている．また，テキスト以外の要素を持ったデータファイルの互換性を得るため，タグイメージファイルフォーマット（ＴＩＦＦ）というファイル形式を設定して，非テキストデータファイルの標準のファイル形式としている．

２．３．５．グラフィカルユーザーインターフェイスの提供

エンドユーザーに分かりやすく，使いやすい，統一的なインターフェイスを提供するために，メニュー／ダイアログボックス／ボタン／スイッチ，エディトコントロールなどのユーザーインターフェイスを構成するための基本的な部品となるものをウインドウシステム内であらかじめ用意している．このためアプリケーション作成者は，これらの部品を組み合わせてユーザーインターフェイスを作成することができ，開発の手間を大幅に短縮することができる．またユーザーは，全く異なる開発者が作成したアプリケーションでも統一的なインターフェイスを持つため，アプリケーションごとに全く違ったインターフェイスを学ぶ必要がないといった利点を得ることができる．

２．３．６．メモリ管理の導入

ＭＳ－ウインドウズはＭＳ－ＤＯＳをベースにしているパーソナルコンピュータをターゲットにしている．このうえでマルチタスクを実現するため，メモリ使用効率を重視したメモリマネージングを採用している．具体的には，

ダイナミックリンキング

デマンドローディング

共有ライブラリの３つがＭＳ－ウインドウズのメモリ管理の特徴となっている．

3　DTPシステム構築のためのベースとしてのMS－ウインドウズ

DTPシステムを構築する場合に，OSが提供しなければならない機能はいろいろと考えられる．MS－ウインドウズが特に日本語のDTPアプリケーションを意識して提供している機能としては次のようなものが挙げられる．

漢字フォント

日本語入力

縦書き文字の表示

ここでは，MS－ウインドウズにおけるフォントの概要，及び日本語入力の概要について，取り上げる．

3．1　フォント

DTPシステムのベースとなるシステムの条件を考えたとき，システムで，複数のフォントの取扱をサポートしているというのは，WYSIWYGの観点から言えば重要な条件である．ここではMS－ウインドウズで提供している複数フォントの取扱についてのべる．

3．1．1　ロジカルフォントとフィジカルフォント

MS－ウインドウズではフォントを2つの異なるレベルにとらえる必要がある．一つはユーザープログラムによって作成され，デバイスと関連しない仮想的な属性のみを持つロジカルフォントであり，もう一つは実際にデバイスに出力して表示することのできるフィジカルフォントである．フィジカルフォントはさらに，ディスクファイルにフォントのイメージを持ち，必要に応じてメモリにロードして使用するソフトウェアフォントと，各デバイスで固有のハードウェアとしてもつハードウェアフォントに分けられる．通常，ビットマップディスプレイに表示するためには，ソフトウェアフォントが用いられ，プリンタに出力する場合にはプリンタのハードウェアフォントが用いられる．

3．1．2　フォントの出力

実際の文字の表示は次のようなステップを踏んで行なわれる．

ロジカルフォントの作成

　　|

ロジカルフォントからフィジカルフォント

へのマッピング

　　|

フィジカルフォントの表示

1）ロジカルフォントの作成

MS－ウインドウズのアプリケーションがある文字列ををある書体で表示するさい，アプリケーションプログラムでは，まずその文字列を表示するためのロジカルフォントを作成する．ロジカルフォントは，実際にシステムでどのようなフォント（ソフトウェアフォント及びハードウェアフォントのフィジカルフォント）が出力可能であるかどうかとは直接関わることなく，表示したいフォントの理想的な属性を持ったものとして仮想的に作成する．図　3．1はロジカルフォント作成のさいのパラメーター一覧である．

```
typedef struct {
    short int      lfHeight;
    short int      lfWidth;
    short int      lfEscapement;
    short int      lfOrientation;
    short int      lfWeight;
    unsigned char  lfItalic;
    unsigned char  lfUnderline;
    unsigned char  lfStrikeOut;
    unsigned char  lfCharSet;
    unsigned char  lfOutPrecision;
    unsigned char  lfClipPrecision;
    unsigned char  lfQuality;
    unsigned char  lfPitchAndFamily;
    unsigned char  lfFaceName[LF_FACESIZE];
} LOGFONT;
```

[図3．1]

2）ロジカルフォントからフィジカルフォントへのマッピング

ロジカルフォントは仮想的なフォントであり，実際にデバイスに対して表示をおこなうためのフォントイメージを持たない．そのためデバイスに対して表示をおこなうためには，ロジカルフォントをフィジカルフォントへとマッピングする必要がある．この処理は出力の対象となるデバイスがソフトウェアフォントを表示するデバイスなのか，ハードウェアフォントを持つデバイスなのかで2つに分かれる．

メモリマップドビットマップディスプレイなどのデバイス固有のフォントを持たないデバイスを出力対象としている場合には，ロジカルフォントのフィジカルフォントへのマッピングはウインドウシステムがおこなう．ウインドウシステムは，アプリケーションプログラムより渡されたロジカルフォントに対して，ソフトウェアフォントの属性を調べて最も近い属性を持ったソフトウェアフォントを選び出し，実際の出力に用いることのできるフィジカルフォントとしてアプリケーションプログラムに返す．　この際，ソフトウェアフォントの属性が，ロジカルフォントによる要求と合わない場合は，整数倍の拡大（縦方向及び横方向），イタリック，ボールド，アンダーラインといったフォントイメージの加工をウインドウシステムで行なう．

なお，この過程でどのフォント（ソフトウェアフォント）がロジカルフォントに最も近いかを決めるには，それぞれのソフトウェアフォントの属性の一つ一つの項目と，ロジカルフォントの属性の対応する項目について差を取り，その加重和を計算してその値が最も小さいものを選ぶといった手順を踏んでいる．

一方プリンタのようにハードウェアフォントをもったデバイスを出力対象としている場合には，ロジカルフォントのフィジカルフォントへのマッピングは，各デバイスのデバイスドライバでお

こなう．基本的には，ウインドウシステムによるロジカルフォントのフィジカルフォントへのマッピングと同じロジックで行なわれるが，フォントイメージの加工が可能かどうかは，各デバイスドライバ（あるいは各ハードウェアデバイス）のケイパビリティに依存する．

この過程で，各アプリケーションプログラムではロジカルフォントとして要求したフォントの属性（サイズ，書体など）と厳密に同じ属性を持ったフィジカルフォントを得られるとは限らないことに注意が必要である．実際のフィジカルフォントは，ソフトウェアフォントならば，どのようなフォントファイルがディスク上に存在し，システムに登録されているかに依存するし，ハードウェアフォントならば，各デバイスの解像度や，どのようなフォントを表示用に持っているかに依存するからである．しかし，ロジカルフォントとフィジカルフォントというフォントの概念的なレベル分けは，後に述べるハードウェアに依存しないＷＹＳＩＷＹＧスタイルのアプリケーションを作成するさいに必要となる重要なものである．

3）フィジカルフォントの表示

アプリケーションプログラムからの文字列表示の要求は，ウインドウシステムを通じて各表示デバイスのデバイスドライバへと渡される．ビットマップディスプレイなどのソフトウェアフォントを表示するデバイスでは，文字列で示された文字を，ウインドウシステムより渡されたフォントのビットマップイメージを参照しながら表示する．（このビットマップイメージはウインドウシステムによって必要に応じてさまざまな加工を施されたものである）またハードウェアフォントを持つデバイスでは，渡された文字列をそれぞれハード的に備えたフォントで表示する．

3．1．3　漢字フォント

前節では，一般的なフォントの取扱について説明し，漢字フォントについては触れなかった．ここではＭＳ－ウインドウズにおける漢字フォントの取扱について，英文字フォントの違いに留意しながら述べる．

1）ロジカルフォント

ロジカルフォントのレベルでは、漢字を表示するためのフォントと英文字のみを表示するフォントとの、構造的な区別はない．

ＭＳ－ウインドウズでは，　漢字を表示するためのロジカルフォントとして，キャラクタセットの項目にシフトＪＩＳを指定して作成すれば漢字表示用のフォントとして作成される．

2）フィジカルフォント

ＭＳ－ウインドウズのソフトウェアフォントは，漢字フォントのイメージをディスクファイルに持っていない．漢字表示用のシフトＪＩＳキャラクタセットのフォントはサイズ，フェースネーム等の属性に関する情報，及び１ｂｙｔｅ文字の領域に対応する文字（英数字，及び半角のカナ）のビットマップイメージを格納しているだけである．これは次のような理由による．

＊　ディスク容量の制限

ＭＳ－ウインドウズがターゲットとしているパーソナルコンピュータでは，ハードディスク装置を備えずにフロッピーディスクのみのシステム構成であるものも多い．ＭＳ－ウインドウズは，フロッピーディスクのみのシステムでも動作するよう考慮されている．このため漢字フォントをファイルとして持ち，Ｄｉｓｋの領域を圧迫することを避ける必要があった．

＊　メモリ容量の制限及び表示スピードの要求

ＭＳ－ＷｉｎｄｏｗｓがターゲットとしているＭＳ－ＤＯＳマシンのＣＰＵ空間は１Ｍｂｙｔｅと限られており，全ての漢字フォント情報をメモリにロードするのはメモリを非常に圧迫することになる．また，必要な部分のみのロードを繰り返して表示するのでは，実用的なスピードは到底得ることができない．

このため漢字フォントのイメージは，ウインドウシステム本体で取り扱わず，いささかトリッキーともいえる方法で，画面に表示する方法を取ることとした．次の項では漢字を含む文字列がどのようにして表示されるかを述べる．ただしこれは，ビットマップディスプレイのような，ハードウェアフォントを持たないデバイスでソフトウェアフォントを表示する時の問題である．各デバイスのハードウェアで漢字のフォントイメージをもつプリンタのような出力デバイスでは，英文字と同等の扱いで表示が行なわれる．　また，ソフトウェアフォント，ハードウェアフォントに関わらず，ロジカルフォントからフィジカルフォントへのマッピングは，英文字のフォントマッピングと同様のロジックで行なわれる．

3）漢字フォントの表示

ビットマップディスプレイに対して，アプリケーションプログラムが，漢字を含む文字列を表示する要求を発行した時，デバイスドライバには，漢字を含む文字列（シフトＪＩＳコードの文字列）及び１ｂｙｔｅ文字のフォントのビットマップイメージが渡される．デバイスドライバは，１ｂｙｔｅ文字については，ウインドウシステムより渡されたフォントのビットマップイメージを参照して表示をおこなう．２ｂｙｔｅ文字（漢字）については，パーソナルコンピュータがハードウェアとして備えている漢字ＲＯＭの内容になんらかの方法でアクセスしてフォントイメージを得る．一般に漢字ＲＯＭのフォントのイメージは，１６ｘ１６あるいは２４ｘ２４といった固定の大きさであり，デバイスドライバはそれらのフォントイメージを１ｂｙｔｅ文字の大きさに合わせて，縮小，拡大を行なって表示をおこなう．

3．1．4　ＷＹＳＩＷＹＧアプリケーションの例

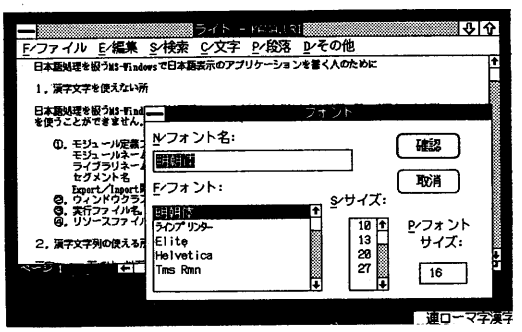
ここでは実際のフォントの出力がどのように行なわれるかを，

MS－Windows上で動作するアプリケーションソフトである ライト を例にとって説明する．

ライトはいわゆるWYSIWYG(What You See is What You Get) スタイルの文書作成アプリケーションである．最終的にプリンタによって出力されるイメージにできるだけ近いイメージを画面で確認しながら文書作成を行なうことを大きな目的としている．

1）ライトはまず起動されると，プリンタドライバに対して，どのようなフォント（ハードウェアフォント）が出力可能であるかを問い合わせる．一般にプリンタは複数のサイズ，書体のフォントを出力することができるので，それらのフォントの情報はリストとしてライトに保持される．

2）ライトはユーザーのフォント設定／変更の要求があると，ダイアログボックスを開いて先ほどのプリンタに出力可能なフォントのリストをユーザーに提示する．図3．2はライトが表示するフォント選択のためのダイアログボックスである．



［ 図3．2 ］

3）ユーザーはそれらのフォントのなかから必要なものを選択する．

4）ライトはプリンタドライバ，ディスプレイドライバに対してそれぞれ，そのデバイスの解像度の情報を問い合わせる．その情報を元にユーザーの選択したプリンターのハードウェアフォントに対して，どのような大きさでディスプレイに表示すれば，プリンターに表示する文字とディスプレイに表示される文字が同じ大きさに見えるかを調べる．

5）ライトは得られたフォントの属性をもとに，プリンタ用のロジカルフォントとディスプレイ用のロジカルフォントを，作成する．このロジカルフォントの属性（サイズ，書体など）は実際のデバイスに出力できるかどうかと無関係である．プリンタ用とディスプレイ用と，2つのロジカルフォントを作成するのは，一般にプリンタとディスプレイの解像度はまちまちであり，それぞれで見た目の同等な表示を実現するには，それぞれのデバイスに合わせたサイズのフォントで表示する必要があるからである．

6）ライトは作成したロジカルフォントをスクリーンに表示するために，フィジカルフォントに変換する．この変換を行なうために，SelectObjectというシステムファンクションをウインドウシステムに対して発行する．ウインドウシステムは前述の，フォントマッピングのロジックでフィジカルフォントへとマッピングする．　ライトは得られたフィジカルフォントでディスプレイに対して表示をおこなう．ここで得られる表示は，実際にプリンタに出力されるイメージと可能な限り似たイメージをディスプレイにおいて再現したものとなっている．

7）ユーザーは画面を見ながら編集作業を行なうことができる．

ここで例えば，ユーザーがある文字列を指定して，イタリックへの加工を指定したとする．ライトはプリンタドライバへ問い合わせて，現在処理中のフォントが，プリンタにおいてイタリックで出力できるかどうかを調べる．プリンターがイタリックでの出力をサポートしているならば，ライトはイタリックの属性をもったロジカルフォントを作成し，ディスプレイドライバ出力のためのフィジカルフォントに変換して，表示をおこなう．なおディスプレイには，そうしたいときには常に，イタリック文字を表示することができる．これはソフトウェアフォントに関しては，ウインドウシステムで，イタリック文字への加工をサポートしているためである．

イタリック処理に限らず，文字のサイズの変更，ボールド処理やアンダーライン処理などのユーザーの要求も同様の手順で処理される．すなわちライトでは，ユーザーの要求による，文字の加工，拡大，縮小などの処理と，それらのディスプレイへの表示は常にプリンタで実際に表示することができるかどうかを問い合わせたうえで，すなわちプリンタからのフィードバックをともなって，行なわれるのである．

8）ライトはユーザーからのプリントアウトの要求を受けると，プリンタ用として作成したロジカルフォントをプリンタ出力用のフィジカルフォントへと変換する．ここでのロジカルフォントからフィジカルフォントへの変換は実は形式的なものである．なぜならばライトがプリンタ用に作成したロジカルフォントの属性はプリンタドライバに問い合わせて調べた，プリンタのハードウェアフォントの属性をベースにしたものだからである．つまりライトはプリンタで表示することのできない属性を持ったロジカルフォントを作成することはないのである．

9）ライトは8）で得られたフィジカルフォントでプリンタへ文字を出力する．ここで得られる出力は，あらかじめディスプレイにて確認したイメージと同じものである．

これらの手順で write はプリンタに出力されるイメージとできるだけ近いイメージを画面で表示する．ここで注意したいのはこの手順が，異なるプリンタと異なるディスプレイの組合せでも有効であるということである．これらの手順は必要以上に手間がかかると思われるかも知れないが，アプリケーションプログラムをプリンタとディスプレイの両方に依存しないように作成するためには，必要なことである．

3．1．5　MS－ウインドウズにおけるWYSIWYGの特徴

このプリンタで実際に表示可能かどうかに重点をおいたWYSIWYGは，MS－ウインドウズの特徴であるといえる．一般にWYSIWYGというものを考えた場合，このようにプリンタの出力表示を優先する方式と，画面での編集結果を忠実にプリンタへ出力するという考えに基づいた方式に大別することができる．

前者の方式ではプリンタの持つハードウェアフォントを最大限に利用するため，比較的表示品質の低いプリンタを使用した場合に，実際の出来上りを意識しながら編集ができるという反面，ユーザーから見た編集の自由度はプリンタの能力に依存して制限を受けるという性格を持つ．また，出来上がった文書ファイルは出力対象としているプリンタのハードウェアに依存した内容を含んでしまうことがある．

これに対して，後者の方式ではユーザーの編集の自由度は非常に高い反面，システム構成の条件として，レーザビームプリンタ等の高い品質の出力が得られるプリンタを前提としており，簡易なシステムでの実現が困難であるという性格をもっている．

3．2　日本語入力

日本語ＤＴＰシステムにとって日本語入力をどのように行なうかは重要な問題である．ここではＭＳ－Ｗｉｎｄｏｗｓが提供しているかな漢字変換入力の概要について述べる．

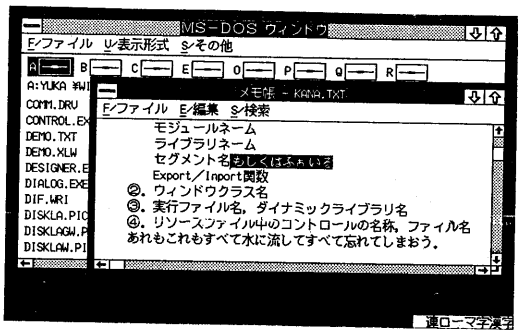
ＭＳ－Ｗｉｎｄｏｗｓでは　日本語入力専門のプロセスを通して日本語を入力し，ポップアップウインドウを作成してユーザーの入力した文字列及び変換された文字列の表示を行なう．

以下，日本語入力の方式として，かな漢字変換方式の入力方式を前提として，ユーザーからのキー入力がどのようにしてアプリケーションに渡るかを順を追って説明する．

1）ウインドウシステム起動時に，通称 kkapp と呼ばれるかな漢字変換を行なうアプリケーションを起動する．これは自動的に起動するように，あるいはユーザーの指定により明示的に起動するように設定することが可能である．

2）kkapp はウインドウシステムがキーボード入力のメッセージを，各アプリケーションに送る前に，kkapp が処理できるようにウインドウシステムに要求する．これはSetWindowHook というシステムファンクションをコールすることで行なわれる．これにより kkapp はユーザーのキー入力の "生の" 情報を得ることが可能となる．

3）kkapp はユーザーのキー入力を解釈して，漢字変換前の文字列，漢字確定の文字列など必要な文字列をポップアップウインドウで表示する．（図3．3）このポップアップウインドウの位置はキーボード入力のフォーカスを持ったアプリケーションからの位置指定があればそれにしたがう．kkapp の存在を全く意識しないアプリケーションの場合は，そのような位置指定がないので，ポップアップウインドウの位置はデフォルト（多くはスクリーン最下部）とする．



［ 図3．3 ］

4）kkapp はユーザーからの文字列確定指定，により，カレントのキーボード入力のフォーカスを持ったアプリケーションのメッセージキューへその文字列を送り込む．

5）アプリケーションは通常のキーボードからの文字入力と同じ要領で，kkapp から送られた日本語文字列を処理する．また必要があれば現在のカーソル位置を kkapp へ通知する．

この一連の手順で，アプリケーションは日本語文字列を受け取ることができる．重要なのは，アプリケーションがかな漢（ kkapp ）を全く意識しないでもよい点である．これはアプリケーションがどのハードウェア上でも，どのかな漢（ kkapp ）との組合せでも動作するようにするためには必要なことである．また唯一ある，カーソル位置の通知という，アプリケーションと kkapp とのインターフェイスも非常にシンプルであり，それぞれのかな漢字モジュールすべてで統一が可能である．

なお実際の kkapp のかな漢字変換の仕組みについて触れておく'かな'から漢字への変換の方法は kkappに任されているが，通常ＭＳ－ＤＯＳ上で動作している，いわゆる日本語ＦＥＰ（フロントエンドプロセッサ）と通信を行なって実際の変換を行なう．ＭＳ－ＤＯＳマシンにおける日本語ＦＥＰは，それぞれのハードウェアに密着して独自の発達を遂げたため，統一したインターフェイスをもっていない．このため 日本語ＦＥＰと通信する kkapp もまたＭＳ－Ｗｉｎｄｏｗｓ上のアプリケーションとしては例外的に，ハードウェアに依存するモジュールとなっている．

4．まとめ

ＭＳ－ウインドウズの目的で最も大きなものは，ハードウェアに依存しないアプリケーションの動作環境を，提供することである．ここで述べた，フォント表示の機構や日本語入力の仕組みはそれらのうちの一つであるといえる．

最後に我々が，ＭＳ－ウインドウズに次ぐものとして開発中の

プレゼンテイションマネージャーについて簡単にふれよう．ＯＳ／２プレゼンテイションマネージャーにおいては，ＭＳ－ＤＯＳとＭＳ－ウインドウズの組合せでは，様々な制約により実現ができなかった機能が可能になる．以下に挙げるものはその代表的なものである．

1）コードページのサポート

ＭＳ－ウインドウズではディスプレイドライバが処理するコードはシフトＪＩＳコードに固定であった．ＯＳ／２ではコードページという概念が導入されており，ユーザーはシステムのコードを変更することが可能である．プレゼンテイションマネージャーにおいてはコードページのサポートにより，日本語版と言うよりはＤＢＣＳ（Double Byte Character Set）版といった形態になる予定である．

2）漢字フォントファイル

ＭＳ－ウインドウズではアプリケーションと共に漢字フォントを提供するといった形態がとれなかった．アプリケーション開発者にとっては，複数の漢字フォントを自由にシステムに取り入れるのは事実上不可能であった．プレゼンテーションマネージャーでは漢字フォントファイルを取り扱えるような仕組みを導入し，豊かな日本語の表示を行なう方法を提供する予定である．

3）かな漢字変換

ＭＳ－ウインドウズでは，ＭＳ－ＤＯＳ用のＦＥＰと通信して漢字変換を行なうという方法を採っていたため，かな漢字変換モジュールはハードウェア依存となっていた．ＯＳ／２においてはこの面を改良し，ハードウェアに依存しない日本語入力を実現する方法を研究中である．

参考文献

1) 中島聡：Microsoft(R) Windowsの三つのインターフェイス（ユーザー，アプリケーション，ハードウェア），(1987)

2) 萩谷昌巳他：特集 ウインドウシステム，bit, Vol.19,No.3 (1987)

3) 萩谷昌巳 森島晃年 ワークステーション上のウインドウシステムについて，コンピュータソフトウェア Vol.5 No.2 April (1988)

4) 井上尚司他：ＵＮＩＸワークステーション ＮＥＷＳ，アスキー，(1987)

5) Adobe Systems Incorporated : PostScript(R) Language Reference Manual, (1985)

6) Apple Documentation Group: Inside Macintosh, (1985)

7) Microsoft Corporation : Microsoft(R) Windows Programmer' Reference, (1987)

8) Gordon Letwin : Inside OS/2, Microsoft Press, (1988)

9) Charles Petzold : Programming Windows, Microsoft Press, (1988)