IME 開発のための Win32 IME の概要

Win32 Multilingual IME Overview for IME Development

Version 1.41

日本語訳:片山博文 MZ

翻訳日:2016 年 4 月 14 日

この文書は、Windows 95、98、NT/2000 で IME を開発する方法についての基礎を紹介する。これは Win32 多言語 IME API（Win32 Multilingual IME API reference for IME development）の補足でもある。次の主なトピックが議論される。

- 概要
- IME ユーザインターフェイス
- IME 入力コンテキスト
- メッセージの生成方法
- ImeSetCompositionString
- ソフトキーボード
- 再変換
- IME メニュー関数
- IME ヘルプファイル
- Windows NT/2000 の事案
- IME のファイル形式とデータ構造

概要

Windows 95 と NT 4.0 以降では、IME（Input Method Editor; 入力方式エディタ）という入力用のソフトが、古い Windows 3.1 極東エディションの IME とは異なり、DLL 形式で提供される。それぞれの IME が多言語キーワードレイアウトの一つとして実行される。古い Windows 3.1 IME と比較して、新しい Win32 多言語入力手段マネージャ（IMM）と入力方式エディタ（IME）のアーキテクチャは以下の利点がある:

- 多言語環境のコンポーネントとして走り、
- それぞれのアプリのタスクごとに複数のインプットコンテキスト (Input Context)が提供され、
- それぞれのアプリのスレッドごとに一つのアクティブな IME が保たれ、
- （メッセージ順を壊すことなく）メッセージループを通じて、アプリに情報を提供し、
- IME を意識したアプリかそうでないかに関わらず、強いサポートを提供する。

これらの利点を完全に役立たせるために、アプリは新しい Win32 IMM/IME アプリインターフェイスをサポートする必要がある。

Windows 95 と Windows NT 4.0 の 存在する IME との最大限の互換性を保つために、Windows 98 と Windows 2000 は設計において主な変更を行っていない。しかしながら、よりよいシステム統合とより賢い IME をサポートするために、新しい特徴が追加されている。

注記

IME 開発者は、<imm.h>のスーパーセットである、DDK の<immdev.h>を使わなければならない。

Windows 98 とWindows 2000 の IMM/IME

Windows 98 とWindows 2000 の IMM/IME のアーキテクチャは、Windows 95 とWindows NT 4.0 の設計を保っている。しかしながら、賢い IME 開発とWindows とIME の統合のために、いくつかの変更点がある。それらの変更点は:

- アプリが IMM/IME と通信を行うことを可能にする新しい IME 関数。それらは::
  - ImmAssociateContextEx
  - ImmDisableIME
  - ImmGetImeMenuItems
- アプリが IMM とアプリの間で通信を行うことを可能にする新しい関数。それらは:
  - ImmRequestMessage
  - ImeGetImeMenuItems
- 再変換のサポート。これは、すでにアプリの文書に挿入した文字列を再変換することを可能にする新しい IME の機能である。この関数は、変換結果に関する情報を取得することと、変換の正確性と性能を改良することにおいて賢い IME を助ける。
- システムペンアイコンのコンテキストメニューに IME メニュー項目たちを追加すること。この新しい機能は、システムバーとアプリにおいて、システムペンアイコンのコンテキストメニューへ IME 固有のメニュー項目たちを挿入することを IME に可能にする方法を提供する。
- IME に対する新しいビットたちとフラグたち。以下の新しいビットたちが新しい変換モードをサポートする。
  - IME_CMODE_FIXED
  - IME_SMODE_CONVERSATION
  - IME_PROP_COMPLETE_ON_UNSELECT
- IME に対するエディットコントロールの拡張。2つの新しいエディットコントロールのメッセージ EM_SETIMESTATUS と EM_GETIMESTATUS を通じて、アプリは、エディットコントロールに対して、IME の状態を管理することができる。
- IME のペンアイコンとツールチップの変更。新しい3つのメッセージ、INDICM_SETIMEICON、INDICM_SETIMETOOLTIPS、そして INDICM_REMOVEDEFAULTMENUITEMS を通じて、IME は、システムタスクバーにおいてシステムぺんあいこんとツールチップを変更することができる。
- 2つの新しい IMR メッセージ。IMR_QUERYCHARPOSITION と IMR_DOCUMENTFEED は、IME とアプリに位置と文書情報と通信を行うことを助ける。
- 64 ビット準拠。2つの新しい構造体（TRANSMSG と TRANSMSGLIST）は IMM32 に追加される。それらは、IME の翻訳したメッセージを受け取るために、INPUTCONTEXT と ImeToAsciiEx で使うことができる。
- IME_PROP_ACCEPT_WIDE_VKEY。この新しいプロパティは、SendInput API によってインジェクト（inject）された Unicode を IME が扱うことができる。ImeProcessKey と ImeToAsciiEx API は、同様にインジェクトされた Unicode を扱えるように更新された。インジェクトされた Unicode は、Unicode 文字列を入力キューへ挿入するためにアプリと手書きのプログラムで使うことができる。

Win32 IME 構造体

新しい Win32 IME は、2 つのコンポーネントを提供する必要がある。一つは IME 変換インターフェイスで、もう一つは IME ユーザインターフェイス (操作系：UI) だ。IME 変換インターフェイスは、IME モジュールからエクスポートされた、関数のセットとして提供される。それらは IMM によって呼び出せる。IME UI は、ウィンドウフォームとして提供される。それらのウィンドウは、メッセージを受け取り、IME に対する UI を提供する。

IME を意識したアプリ

新しい IME アーキテクチャの主な利点の一つは、アプリと IME の間のよりよい通信ロジックを提供することだ。以下は、IME と関わることができるアプリの例である:

- IME を意識していないアプリ。この種のアプリは、IME を少しも制御しない。しかしながら、DBCS を受け入れるアプリと同じようにユーザは IME を使ってアプリに DBCS 文字を打ち込むことができる。
- IME を半分意識しているアプリ。この種のアプリは、典型的に様々な IME のコンテキスト(開く、閉じる、コンポジションフォームなど)を制御する。しかし、IME に対して UI を表示しない。
- IME を完全に意識しているアプリ。この種のアプリは、典型的に IME によって与えられた情報を表示する責任があることを完全に要求する。

Windows 95 と Windows NT 4.0 とそれ以降では、IME を意識していないアプリは、1 個の既定の IME ウィンドウと 1 個の既定の入力コンテキストでサポートされる。

IME を半分意識しているアプリは、自分自身の IME ウィンドウを作成するだろう。それはアプリケーション IME ウィンドウと呼ばれる。それは定義済みシステム IME クラスを使う。アプリに与えられた自分自身の入力コンテキストを扱うかもしれないし、扱わないかもしれない。

IME を完全に意識したアプリは、自分自身で入力コンテキストを扱い、IME ウィンドウを使うことなく入力コンテキストによって与えられた必要な情報を表示するだろう。

IME UI

IME 操作系(UI)は、IME ウィンドウ、UI ウィンドウを含み、さらに UI ウィンドウのコンポーネントも含んでいる。

特徴

IME クラスとは、定義済みのグローバルなウィンドウクラスであり、IME の操作系をすべて実現する。IME クラスの通常の特徴は、他のコントロールと同じである。そのウィンドウインスタンスは、CreateWindowEx 関数で作成できる。スタティックコントロールのように、ユーザの入力に反応させなくすることもできるが、IME のすべての操作系を実現するために、あらゆる種類のコントロールメッセージを受け取る。アプリは、IME クラスを使って、所有する IME ウィンドウを作成できるし、既定の IME ウィンドウを ImmGetDefaultIMEWnd 関数を通じて取得できる。古い Windows 3.1 とは違って、それらのウィンドウハンドルを使って今や IME を制御したいアプリ(IME を意識したアプリ)が以下の長所を実現することができる:

- 新しい IME は、候補リストウィンドウを含む。それぞれのアプリは、自分自身の UI のウィンドウインスタンスを持つことができるので、ユーザは他のアプリに切り替えるために操作を途中で中断することができる。Windows 3.1 日本語版では、ユーザは他のアプリに切り替える前に、最初に操作を終了する必要があった。
- IME UI ウィンドウは、アプリのウィンドウハンドルを知らされて、アプリにさまざまなふるまいを提供できる。例えば、IME ウィンドウの自動再配置や、ウィンドウキャレット位置の自動追跡、それぞれのアプリのためのモードの表示などが可能だ。

システムは、一つだけの IME クラスを提供するが、それらは、二種類の IME ウィンドウである。一つは、特に DefWindowProc 関数によって、システムによって作られ、一つのスレッドのすべての「IME を意識しないアプリ」で共有されるものであり、「既定の IME ウィンドウ」と呼ばれるものだ。もう一つは、IME を意識したアプリにより作成され、アプリケーション IME ウィンドウと呼ばれるものだ。

既定の IME ウィンドウとアプリケーション IME ウィンドウ

システムは、既定の IME ウィンドウをスレッドの初期化時に作成し、それをスレッドに自動的に渡す。この IME ウィンドウが、IME を意識していないアプリの IME 操作系となる。

IME や IMM が WM_IME_xxx メッセージを生成するとき、IME を意識していないアプリは、それらを DefWindowProc 関数に渡す。それから、必要なメッセージを DefWindowProcB が既定の IME ウィンドウに送信する。IME を意識していないアプリでは、既定の IME ウィンドウは IME の既定のふるまいを提供する。IME を意識しているアプリは、メッセージをフックしていないときでも IME ウィンドウを使う。アプリは、必要に応じて、アプリ所有の IME ウィンドウを使うことができる。

IME クラス

Win32 システムは、IME クラス（ウィンドウクラス）を提供する。このクラスは、定義済みの EDIT クラスのようなユーザによって定義される。システム IME クラスは、IME のすべての UI を扱い、IME とアプリ（IMM 関数を含む）からすべてのコントロールメッセージを扱う。アプリは、このクラスを使って、所有する IME UI を作成できる。システム IME クラスは、IME によっては置き換えられないが、定義済みクラスとして保たれる。

このクラスは、ウィンドウプロシージャを持つ。それは、実際に、WM_IME_SELECT メッセージを扱う。このメッセージは、新しく選択された IME の hKL (キーボードレイアウトのハンドル)を持っている。システム IME クラスは、この hKL を持ったそれぞれの IME で定義されたクラスの名前を取得する。この名前を使って、システム IME クラスは、現在アクティブな IME の UI ウィンドウを作成する。

IME からの UI クラス

設計では、すべての IME は、システムに対してそれ自身の UI クラスを登録することを想定している。それぞれの IME によって提供される UI クラスは、IME 特有の機能に対して責任がある。IME は、IME がプロセスにアタッチされるとき、IME 自身がその IME で使われるいくつかのクラスを登録するかもしれない。これは、DLL のエントリポイントが DLL_PROCESS_ATTACH で呼ばれるときに起こる。IME はそのとき、ImeInquire 関数の第二引数 lpszClassName にクラス名をセットしないといけない。

UI クラスは、すべてのアプリが IME クラスとして使えるよう、CS_IME スタイルを付けて登録されるべきだ。(ナル文字を含む) UI クラス名は、16 文字までであるが、将来のバージョンでは増えるかもしれない。

UI クラスの cbWndExtra メンバーは、2 * sizeof(LONG)でなければならない。割り増しのデータ cbWndExtra の目的はシステムによって定義される。例えば、IMMGWL_IMC と IMMGWL_PRIVATE である。

IME は、どんなウィンドウクラスも登録でき、アプリで動作するどんなウィンドウも作成できる。次のサンプルコードは、IME UI クラスの登録方法を示す。

```
BOOL WINAPI DllMain(
      HINSTANCE hInstDLL,
      DWORD dwFunction,
      LPVOID lpNot)
{
      switch(dwFunction)
      {
      case DLL_PROCESS_ATTACH:
            hInst= hInstDLL;
            wc.style = CS_MYCLASSFLAG | CS_IME;
            wc.lpfnWndProc = MyUIServerWndProc;
            wc.cbClsExtra = 0;
            wc.cbWndExtra = 2 * sizeof(LONG);
```

```
            wc.hInstance = hInst;
            wc.hCursor = LoadCursor( NULL, IDC_ARROW );
            wc.hIcon = NULL;
            wc.lpszMenuName =NULL;
            wc.lpszClassName = szUIClassName;
            wc.hbrBackground = NULL;
            if( !RegisterClass( (LPWNDCLASS)&wc ) )
                  return FALSE;
            wc.style = CS_MYCLASSFLAG | CS_IME;
            wc.lpfnWndProc = MyCompStringWndProc;
            wc.cbClsExtra = 0;
            wc.cbWndExtra = cbMyWndExtra;
            wc.hInstance = hInst;
            wc.hCursor = LoadCursor( NULL, IDC_ARROW );
            wc.hIcon = NULL;
            wc.lpszMenuName = NULL;
            wc.lpszClassName = szUICompStringClassName;
            wc.hbrBackground = NULL;
            if( !RegisterClass( (LPWNDCLASS)&wc ) )
                  return FALSE;
            break;
      case DLL_PROCESS_DETACH:
            UnregisterClass(szUIClassName,hInst);
            UnregisterClass(szUICompStringClassName,hInst);
            break;
      }
      return TRUE;
}
```

UI ウィンドウ

IME クラスの IME ウィンドウは、アプリかシステムによって作成される。IME ウィンドウが作成されたら、IME 自身によって提供される UI ウィンドウが作成され、IME ウィンドウによって所有される。

それぞれの UI ウィンドウは、現在の入力コンテキストを所有する。この入力コンテキストは、UI ウィンドウが WM_IME_xxx メッセージを受け取ったときに、GetWindowLong 関数に IMMGWL_IMC をつけて呼び出すことで取得できる。UI ウィンドウは、この入力コンテキストを参照でき、メッセージを扱うことができる。GetWindowLong 関数に IMMGWL_IMC をつけて入力コンテキストを取得することは、UI ウィンドウプロシージャの間、いつでも可能だ（ただし WM_CREATE メッセージを除く）。

UI ウィンドウの cbWndExtra は、IME によって拡張できない。IME がウィンドウインスタンスの割り増しのデータを使う必要がある場合は、UI ウィンドウは、IMMGWL_PRIVATE をつけて SetWindowLong と GetWindowLong 関数を使う。この IMMGWL_PRIVATE は、LONG 値の割り増しのデータを提供する。UI ウィンドウのプライベートな使用目的で、さらなるデータが必要なら、IMMGWL_PRIVATE 領域にメモリブロックのハンドルを置くことができる。UI ウィンドウプロシージャは、DefWindowProc 関数が使えるが、UI ウィンドウは、WM_IME_xxx メッセージを DefWindowProc に渡すことはできない。このメッセージが UI ウィンドウプロシージャによって扱わなくても、UI ウィンドウは、DefWindowProc にそれを渡さない。

以下のサンプルコードは、どのようにプライベートなメモリーブロックを割り当てて使うかを説明する。

```
LRESULT UIWndProc (HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
      HIMC hIMC;
      HGLOBAL hMyExtra;
      switch(msg){
      case WM_CREATE:
            // Allocate the memory block for the window instance.
            hMyExtra = GlobalAlloc(GHND,size_of_MyExtra);
            if (!hMyExtra)
```

```
                MyError();
            // Set the memory handle into IMMGWL_PRIVATE
            SetWindowLong(hWnd, IMMGWL_PRIVATE, (LONG)hMyExtra);
            :
            :
            break;
      case WM_IME_xxxx:
            // Get IMC;
            hIMC = GetWindowLong(hWnd,IMMGWL_IMC);
            // Get the memory handle for the window instance.
            hMyExtra = GetWindowLong(hWnd, IMMGWL_PRIVATE);
            lpMyExtra = GlobalLock(hMyExtra);
            :
            :
            GlobalUnlock(hMyExtra);
            break;
      :
      :
      case WM_DESTROY:
            // Get the memory handle for the window instance.
            hMyExtra = GetWindowLong(hWnd, IMMGWL_PRIVATE);
            // Free the memory block for the window instance.
            GlobalFree(hMyExtra);
            break;
      default:
            return DefWindowProc(hWnd, msg, wParam, lParam);
      }
}
```

UI ウィンドウは、現在選択中の入力コンテキストを参照することによる、すべてのタスクを成し遂げなければならない。アプリのウィンドウがアクティブになったとき、UI ウィンドウは、現在の入力コンテキストを含むメッセージを受け取る。UI ウィンドウは、そのとき、入力コンテキストを使う。よって、入力コンテキストは、UI ウィンドウがコンポジションウィンドウ・状態ウィンドウなどを表示するためのすべての情報を含んでいないといけない。

UI ウィンドウは、入力コンテキストを参照するが、更新する必要はない。しかしながら、もし UI ウィンドウが入力コンテキストを更新したければ、それは、IMM 関数を呼ぶべきだ。なぜなら、入力コンテキストは、IMM によって管理され、IMM は IME と一緒に、入力コンテキストが変更されたとき、IME が通知すべきだからである。

例えば、UI ウィンドウはたまに、ユーザがマウスをクリックしたとき、入力コンテキストの変換モードを変更する必要がある。このとき、UI ウィンドウは、ImmSetConversionMode 関数を呼ぶべきである。
ImmSetConversionMode 関数は、WM_IME_NOTIFY を持った UI ウィンドウと NotifyIME へ通知を作成する。

UI ウィンドウが変換モードの表示を変更したいなら、UI ウィンドウは、WM_IME_NOTIFY メッセージを待つべきである。

UI ウィンドウのコンポーネント

UI ウィンドウは、現在の入力コンテキストを参照することにより、コンポジションウィンドウや状態ウィンドウを登録・表示できる。UI ウィンドウのコンポーネントたちのクラススタイルは、CS_IME ビットを含んでいなければならない。UI ウィンドウのウィンドウインスタンスは、現在の入力コンテキストからコンポジション文字列、フォント、そして位置の情報を得る。

アプリのウィンドウがフォーカスを得たとき、システムは、入力コンテキストをこのウィンドウに与え、現在の入力コンテキストを UI ウィンドウにセットする。システムはそのとき、WM_IME_SETCONTEXT メッセージに入力コンテキストをつけて、アプリへ送信する。すると、アプリはこのメッセージを UI ウィンドウに渡す。もし、現在の入力コンテキストが他の入力コンテキストに入れ替わっていたら、UI ウィンドウは、コンポジションウィンドウを再描画すべきだ。現在の入力コンテキストは変更されたときはいつでも、UI ウィンドウは正しいコンポジションウィンドウを表示する。したがって、IME の状態は保証される。

UI ウィンドウは、その状態やコンポジション文字列、候補リストを表示するために、子ウィンドウやポップアップウィンドウを作成できる。しかしながら、それらのウィンドウは、UI ウィンドウによって所有される必要があり、また、無効な (disabled) ウィンドウとして作成される必要がある。IME によって作成されたどんなウィンドウも、フォーカスを得てはならない。

IME 入力コンテキスト

それぞれのウィンドウは、IME 入力コンテキストに関連付けられる。IMM は、IME の状態やデータなどを管理するために、また、IME とアプリの間で通信するために、入力コンテキストを使う。

既定の入力コンテキスト

既定では、システムは各スレッドのために、既定の入力コンテキストを作成する。すべての、IME を意識しないウィンドウはこのコンテキストを共有する。

アプリ作成の入力コンテキスト

アプリのウィンドウは、未確定のコンポジション文字列を含む IME のあらゆる状態を管理するために、ウィンドウハンドルを入力コンテキストに関連付けできる。一度、アプリが入力コンテキストをウィンドウハンドルに関連付けると、システムはウィンドウがアクティブになったときに自動的にコンテキストを選択する。このように、アプリは、フォーカスの取得・解放処理における複雑さから解放されている。

入力コンテキストの使用方法

アプリまたはシステムが新しい入力コンテキストを作成するとき、システムは、新しい入力コンテキストに IMC (IMCC)のコンポーネントたちを用意する。コンポーネントたちは、hCompStr、hCandInfo、hGuideLine、hPrivate、そして hMsgBuf を含む。基本的に、IME は、入力コンテキストとそのコンポーネントたちを作成しない。IME は、それらのコンポーネントのサイズを変更でき、そのポインタを取得するために、ロックできる。

HIMC へのアクセス方法

IME が入力コンテキストにアクセスするとき、IME は、入力コンテキストのポインタを取得するために ImmLockIMC 関数を呼ぶ。ImmLockIMC 関数は、IMM のロックカウントを1つ増加させ、ImmUnlockIMC 関数は、IMM のロックカウントを1つ減少させる。

HIMCC へのアクセス方法

IME が入力コンテキストのコンポーネントにアクセスするとき、IME は IMCC のポインタを取得するために ImmLockIMCC を呼ぶ必要がある。ImmLockIMCC 関数は IMCC に対するロックカウントを1つ増加させ、一方 ImmUnlockIMC 関数は IMCC に対する IMM ロックカウントを1つ減少させる。ImmReSizeIMCC 関数は、IMCC のサイズを入力引数で指定したサイズへ変更できる

時として、IME は、入力コンテキストの新しいコンポーネントを作成する必要がある。そのため IME は、ImmCreateIMCC 関数を呼ぶことができる。入力コンテキスト中の新しく作成したコンポーネントを破棄するために、IME は ImmDestroyIMCC 関数を呼ぶことができる。

次の例は、IMCC へのアクセス方法と、コンポーネントのサイズ変更の方法を示す。

```
LPINPUTCONTEXT lpIMC;
LPCOMOSITIONSTRING lpCompStr;
HIMCC hMyCompStr;
if (hIMC) // It is not NULL context.
{
      lpIMC = ImmLockIMC(hIMC);
      if (!lpIMC)
      {
            MyError( "Can not lock hIMC");
            return FALSE;
      }
      // Use lpIMC->hCompStr.
      lpCompStr = (LPCOMPOSITIONSTRING)ImmLockIMCC(lpIMC->hCompStr);
      // Access lpCompStr.
      ImmUnlockIMCC(lpIMC->hCompStr);
      // ReSize lpIMC->hCompStr.
      if (!(hMyCompStr = ImmReSizeIMCC(lpIMC->hCompStr,dwNewSize))
      {
            MyError("Can not resize hCompStr");
            ImmUnlockIMC(hIMC);
            return FALSE;
      }
      lpIMC->hCompStr = hMyCompStr;
      ImmUnlockIMC(hIMC);
}
```

メッセージの生成

IME は IME メッセージを生成する必要がある。IME が変換プロセスを初期化したとき、IME は、WM_IME_STARTCOMPOSITION メッセージを生成する必要がある。もし、IME がコンポジション文字列を変更したら、IME は、WM_IME_COMPOSITION メッセージを生成する必要がある。

IME がメッセージを生成する方法には、二通りある。一つは、ImeToAsciiEx 関数で与えられる lpdwTransKey バッファを使うこと、もう一つは、ImmGenerateMessage 関数を呼ぶことだ。

メッセージ生成に lpdwTransKey を使う方法

IME によって開始されたイベントは、入力コンテキストに関連付けられたウィンドウへのメッセージを生成するように実現される。基本的に、IME は ImeToAsciiEx の引数として与えられた lpTransMsgList を使って、メッセージを生成する。ImeToAsciiEx が呼び出されたとき、IME は lpTransMsgList バッファへメッセージを追加する。

ImeToAsciiEx 関数の lpTransMsgList で指定されたバッファは、システムによって提供される。この関数は、一度にバッファ内のすべてのメッセージたちを置くことができる。メッセージの本当の個数は、バッファ先頭のダブルワード値で与えられる。しかしながら、もし、ImeToAsciiEx 関数が、与えられた個数よりも多くのメッセージを生成したいなら、ImeToAsciiEx 関数は、入力コンテキスト内の hMsgBuf へ、すべてのメッセージを置くことができ、そのときメッセージの個数が戻り値として返される。

ImeToAsciiEx の戻り値が lpTransMsgList で指定された値よりも大きいとき、システムは、lpTransMsgList からメッセージを取り出さない。代わりに、システムは入力コンテキスト内の hMsgBuf を見る。この hMsgBuf は、ImeToAsciiEx の引数として渡される。

次は、ImeToAsciiEx 実装のサンプルコードである。

```
UINT
ImeToAsciiEx(
      uVirKey,
      uScanCode,
      lpbKeyState,
```

```
      lpTransMsgList,
      fuState,
      hIMC)
{
      DWORD dwMyNumMsg = 0;
      . . .
      // Set the messages that the IME wants to generate.
      pTransMsgList->TransMsg[0].message =msg;
      pTransMsgList->TransMsg[0].wParam = wParam;
      pTransMsgList->TransMsg[0].lParam = lParam;
      // Count the number of the messages that the IME wants to generate.
      dwMyNumMsg++;
      . . .
      return dwMyNumMsg;
}
```

メッセージ生成にメッセージバッファを使う方法

もし、ImeToAsciiEx **が呼ばれなかったとしても、**IME **は、まだメッセージを、入力コンテキストのメッセージバッファを使って、入力コンテキストに関連付けられたウィンドウに生成することができる。このメッセージバッファは、メモリーブロックのハンドルとして処理され、**IME **は、メッセージをこのメモリーブロックへ置く。それから **IME **は、**ImmGenerateMessage **関数を呼び、それが適切なウィンドウへのメッセージバッファに格納されたメッセージを送信する。**
次は、**ImmGenerateMessage **実装のサンプルコードである**:

```
MyGenerateMesage(HIMC hIMC, UINT msg, WPARAM wParam, LPARAMlParam)
{
      LPINPUTCONTEXT lpIMC;
      HGLOBAL hTemp;
      LPTRANSMSG lpTransMsg;
      DWORD dwMyNumMsg = 1;
      // Lock the Input Context.
      lpIMC = ImmLockIMC(hIMC);
      if (!lpIMC) ….. // Error!
      // re-allocate the memory block for the message buffer.
      hTemp = ImmReSizeIMCC(lpIMC->hMsgBuf, (lpIMC->dwNumMsgBuf + dwMyNumMsg) *
      sizeof(TRANSMSG));
      if (!hTemp) ….// Error!
      lpIMC->hMsgBuf = hTemp;
      // Lock the memory for the message buffer.
      lpTransMsg = ImmLockIMCC(lpIMC->hMsgBuf);
      if (!lpTransMsg) …. // Error!
      // Set the messages that the IME wants to generate.
      lpTransMsg[lpIMC->dwNumMsg].message = msg;
      lpTransMsg[lpIMC->dwNumMsg].wParam = wParam;
      lpTransMsg[lpIMC->dwNumMsg].lParam = lParam;
      // Set the number of the messages.
      lpIMC->dwNumMsgBuf += dwMyNumMsg;
      // Unlock the memory for the message buffer and the Input Context.
      ImmUnlockIMCC(lpIMC->hMsgBuf);
      ImmLockIMC(hIMC);
      // Call ImmGenerateMessage function.
      ImmGenerateMessage(hIMC);
}
```

WM_IME_COMPOSITION メッセージ

IME が WM_IME_COMPOSITION メッセージを生成したとき、IME は、lParam を GCS ビットたちとして指定する。そして GCS ビットたちは、COMPOSITIONSTRING 構造体の利用可能なメンバーを通知する。IME が更新せず、メンバーが利用可能でだったとしても、IME は、GCS ビットたちをセットできる。

IME が WM_IME_COMPOSITION メッセージを生成したとき、IME はなお、文字列属性と文節情報をすべて一度で変更できる。

ImeSetCompositionString

ImeSetCompositionString 関数は、アプリが IME コンポジション文字列を操作するために使われる。別のフラグを指定することで、アプリは、コンポジション文字列、属性などを変更できる。この関数の第二引数 dwIndex は、IME においてどのようにコンポジション文字列を補正するかを指定する。それは SCS_SETSTR、SCS_CHANGEATTR、SCS_CHANGECLAUSE、SCS_QUERYRECONVERTSTRING などの値を含む。それぞれの値が特定の機能を表現する。

ImeSetCompositionString 能力

IME が ImeSetCompositionString の能力を持たないとき、IMEINFO 構造体のどんな SCS 能力も指定しないだろう。IME が ImeSetCompositionString を扱うことができるとき、それは SCS_COMPSTR ビットをセットする。もし IME がコンポジション文字列から読みの文字列を生成できるなら、それは SCS_CAP_MAKEREAD ビットをセットするだろう。

もし IME が SCS_CAP_COMPSTR 能力を持っていれば、ImeSetCompositionString 関数が呼ばれるであろう。この呼び出しの反応では、IME は、アプリによって生成された新しいコンポジション文字列を使い、そして WM_IME_COMPOSITION メッセージを生成するべきだろう。

SCS_SETSTR

ImeSetCompositionString の dwIndex が SCS_SETSTR であれば、IME は hIMC の COMPOSITIONSTRING 構造体のすべてを消去できる。

必要なら、IME は候補情報を更新したり、候補メッセージ WM_IME_NOTIFY に IMN_OPENCANDIDATE、CHANGECANDIDATE、または IMN_CLOSECANDIDATE サブメッセージをつけて生成できる。

IME は、以下のように異なる入力引数に基づくアプリの要件に反応する必要がある。

- もし、ImeSetCompositonString の引数 lpRead が利用可能であれば:
  - IME は、コンポジション文字列を、lpRead に含まれる読みの文字列から作成すべきだ。IME はそのとき、新しいコンポジション文字列と読みの文字列 lpRead の両方のために、属性と文節を作成する。IME は、WM_IME_COMPOSITION メッセージを GCS_COMP か GCS_COMPREAD のいずれかをつけて、生成する。たまに、IME は、変換を自動的にファイナライズ（finalize）しうる。IME はそのとき WM_IME_COMPOSITION メッセージを、GCS_COMPxxx の代わりに GCS_RESULT か GCS_RESULTREAD をつけて生成できる。
- もし、ImeSetCompositonString の引数 lpComp が利用可能であれば:
  - IME は、コンポジション属性と文節情報をコンポジション文字列（lpComp に含まれる）から、作成すべきだ。IME はそのとき、WM_IME_COMPOSITION メッセージを GCS_COMP をつけて生成する。もし IME が SCS_CAP_MAKEREAD の能力があれば、同時に IME は新しい読みの文字列も作成すべきだ。IME はそのとき、WM_IME_COMPOSITION メッセージを GCS_COMP か GCS_COMPREAD のいずれかをつけて生成する。ときおり、IME は変換を自動的にファイナライズしうる。IME はそのとき、WM_IME_COMPOSITION メッセージを、GCS_COMPxxx の代わりに GCS_RESULT か GCS_RESULTREAD のいずれかをつけて生成できる。
- lpRead と lpComp の両方利用可能であれば:
  - IME は、対応する、コンポジション文字列と読みの文字列を作成すべきだ。この場合、IME は、lpComp と lpRead に完全に従う必要はない。もし、IME が、アプリによって指定された lpRead と lpComp の関係がわからないときは、コンポジション文字列を修正すべきだ。IME はそのとき、新し

いコンポジション文字列と読みの文字列 lpRead の両方に対する属性と文節情報を作成できる。IME はそのとき、WM_IME_COMPOSITION メッセージを GCS_COMP か GCS_COMPREAD のいずれかをつけて生成する。ときおり、IME は、変換を自動的にファイナライズしうる。IME はそのとき、WM_IME_COMPOSITION メッセージを、GCS_COMPxxx の代わりに、GCS_RESULT か GCS_RESULTREAD のいずれかをつけて生成できる。

SCS_CHANGEATTR

SCS_CHANGEATTR は、属性情報にのみ影響する。IME は、コンポジション文字列、コンポジション文字列の文節情報、コンポジション文字列の読み、または、コンポジション文字列の読みの文節情報を更新すべきではない。

IME は最初に、新しい属性を受け入れ可能かどうかをチェックする必要がある。そして hIMC の COMPOSITIONSTRING 構造体に新しい属性をセットする。最後に IME は、WM_IME_COMPOSITION メッセージを生成する。

必要ならば、IME は、候補情報を更新でき、候補メッセージ WM_IME_NOTIFY をサブメッセージ IMN_OPENCANDIDATE、CHANGECANDIDATE、または IMN_CLOSECANDIDATE をつけて、生成できる。

この機能のために、IME は、コンポジション文字列をファイナライズできない。

IME は、以下のような異なる入力引数に基づくアプリの要件に応えないといけない。

- もし ImeSetCompositonString の引数 lpRead が利用可能であれば:
  - IME は、lpRead の新しい属性に従うべきであり、そして現在のコンポジション文字列に対するコンポジション文字列の新しい属性を作成するべきだ。この場合、文節情報は変更しない。IME は、WM_IME_COMPOSITION メッセージを GCS_COMP か GCS_COMPREAD のいずれかをつけて生成される。もし、lpRead に含まれる新しい属性が受け入れられないならば、何もメッセージを生成せず、戻り値として FALSE を返す。
- もし、ImeSetCompositonString の引数 lpComp が利用可能であれば:
  - IME は、lpComp の新しい属性に従う。この場合、文節情報は変更されない。もし IME の能力が SCS_CAP_MAKEREAD を持っていて、読みの文字列が利用可能であれば、IME は、現在のコンポジション文字列の現在の読みに対する新しいコンポジション文字列の読みの属性を作成すべきだ。
- lpRead と lpComp の両方が利用可能であれば:
  - IME が新しい情報を受け入れることができれば、hIMC の COMPOSITION 構造体に新しい情報をセットし、WM_IME_COMPOSITION メッセージを GCS_COMP か GCS_COMPREAD のいずれかをつけて生成する。

SCS_CHANGECLAUSE

SCS_CHANGECLAUSE は、コンポジション文字列とコンポジション文字列の読みの両方に対する文字列と属性に影響する。

必要であれば、IME は候補情報を更新でき、候補メッセージ WM_IME_NOTIFY をサブメッセージ IMN_OPENCANDIDATE、CHANGECANDIDATE、または IMN_CLOSECANDIDATE をつけて生成できる。

IME は、異なる入力引数に基づくアプリの要件に応える必要がある。以下は、IME がコンポジション文字列をファイナライズできない例である。

- もし ImeSetCompositonString の引数 lpRead が利用可能であれば:
  - IME は、lpRead の新しい読み文節情報に従い、コンポジション文字列の読みの属性を修正する必要がある。IME はそして、コンポジション文字列、コンポジション文字列の属性・文節情報を更新できる。IME は、WM_IME_COMPOSITION メッセージを、GCS_COMP か GCS_COMPREAD のいずれかをつけて生成する。
- もし ImeSetCompositonString の引数 lpComp が利用可能であれば:
  - IME は、新しい文節情報に従い、コンポジション文字列、コンポジション文字列の属性を修正する必要がある。そのとき、IME は、読み属性と読み属性の文節情報を更新できる。IME は WM_IME_COMPOSITION メッセージを、GCS_COMPSTR か GCS_COMPREAD をつけて生成する。
- もし lpRead と lpComp の両方が利用可能であれば:
  - IME が新しい情報を受け入れ可能であれば、hIMC の COMPOSITION 構造体に新しい情報をセットし、WM_IME_COMPOSITION メッセージを GCS_COMP か GCS_COMPREAD をつけて生成する。

ソフトキーボード

ソフトキーボードは、IME によって表示される特殊なウィンドウだ。いくつかの IME は、特殊な読みの文字を持つため、IME はそれらの特殊な読みの文字を表示するためにソフトキーボードを提供できる。この方法では、ユーザは各キーの読みの文字を覚える必要がない。例えば、IME は、読みの文字に対して「ボ」「ポ」「モ」「フォ」を使うのに対して、別の IME では読みの文字に部首を使うことができる。

IME はキーの読みの文字列を変更でき、それらのキーが変更されたことをユーザに知らせることができる(それは変換状態に依存する)。例えば、候補選択時に IME は、ユーザに選択キーのみを表示することができる。

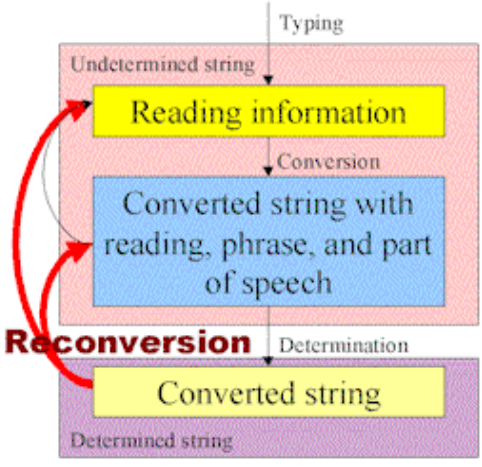
IME は、ソフトキーボードに自分自身の UI を作成できるし、システム定義済みのソフトキーボードを使うことができる。もし、IME がシステム定義済みのソフトキーボードを使いたければ、ImeInquire が呼ばれたときに、IMEINFO 構造体の fdwUICaps メンバーに UI_CAP_SOFTKBD を指定する必要がある。

IME は、ソフトキーボードウィンドウを作成するのに ImmCreateSoftKeyboard を呼び出す必要がある。それはソフトキーボードの表示・非表示を切り替えるために ImmShowSoftKeyboard も呼び出すことができる。ソフトキーボードは UI ウィンドウの1つのコンポーネントであるから、所有者は UI ウィンドウでなければならない。

ソフトキーボードの種類はいくつかある。それぞれは、特定の国や特殊な目的で設計される。IME は、次のいずれかの手段を使って、ソフトキーボードの読み文字を変更できる:IMC_SETSOFTKBDSUBTYPE または IMC_SETSOFTKBDDATA。

再変換

再変換は、Windows 98 と Windows 2000 で登場した IME の新しい機能である。それは、アプリの文書にすでに挿入された文字列を再変換する能力を提供する。特に文字列が何であれ、IME は文字列を認識するよう指示され、読みや入力情報に変換し、そして候補リストを表示する。



新しく、強化された賢い IME は、完全な文の認識や解釈の機能がある。IME が文字列について、よりよい情報を提供されたら(完全な文や文の区切り)、それはより良い変換パフォーマンスと正確性を遂行できる。例えば、再変換されると仮定した、完全な文を供給することで、IME は、最

初に読みや入力情報がなくても、文字列を再変換できる。

以前は、確定の後に編集することは困難であった。未確定の文字列が持つ情報（読み、句、音素）が失われるからだ。再変換は、それらの情報を取り戻そうとし、確定の後の編集が確定の前と同じように簡単になる。ユーザは、候補から別の同音語を選択でき、句の区切りを変更でき、IME に再び分析させることなどができる。

RECONVERTSTRING 構造体は、すべての文を格納でき、 dwStartOffset と dwLen によって再変換される文字列を指し示すことができる。もし dwStartOffset がゼロで、dwLen が文字列の長さであれば、文字列全体が IME によって再変換される。

単純な再変換

単純な再変換は、ターゲットの文字列とコンポジション文字列が文字列全体で同じであるときだ。この場合、dwCompStrOffset と dwTargetStrOffset がゼロであり、dwStrLen、dwCompStrLen、そして dwTargetStrLen が同じ値である。IME は、構造体で与えられた文字列全体のコンポジション文字列を提供し、変換結果によりターゲットの文節をセットする。

普通の再変換

有能な変換結果に対しては、アプリは、情報文字列をつけた RECONVERTSTRING 構造体を提供すべきである。この場合、コンポジション文字列は、文字列全体ではなく、ターゲットの文字列と同じである。IME は全体の文字列を参照することでコンポジション文字列を変換でき、変換結果によりターゲットの文節をセットする。

強化された再変換

アプリは、コンポジション文字列とは異なるターゲットの文字列をセットできる。ターゲット文字列（またはその一部）はそのとき、IME によって高い優先順位でターゲットの文節に入る。RECONVERTSTRING 構造体のターゲットの文字列は、コンポジション文字列の一部でなければならない。アプリが再変換の間、ユーザのフォーカスを変更したくないときは、ターゲットの文字列が指定されるべきだ。IME はそのとき、それを参照できる。

IME キャンセルの再変換

再変換によって生成されたコンポジション文字列をユーザがキャンセルしたとき、IME は、オリジナルの再変換文字列を確定すべきである。さもなければアプリは文字列を失うだろう。

SCS_SETRECONVERTSTRING と SCS_QUERYRECONVERTSTRING フラグ

アプリは、ImmSetCompositionString を呼び出すことで、IME に文字列を再変換するか尋ねることができる。新しいコンポジション文字列を作るために SCS_SETSTR フラグか、SCS_SETRECONVERTSTRING フラグのいずれかを使うことができる。しかしながら、 SCS_SETRECONVERTSTRING を使えば、アプリはより効率の良い変換のために IME に RECONVERTSTRING を渡すことができる。

最初に、アプリは SCS_QUERYRECONVERTSTRING をつけて ImmSetCompositionString を 呼び出すべきだ。選択中の IME はそのとき、より好ましい再変換のために与えられた RECONVERTSTRING 構造体を調整できる。アプリはそして、SCS_SETRECONVERTSTRING をつけて ImmSetCompositionString を呼び出す。

それは IME に新しいコンポジション文字列を生成させるためだ。もし、アプリが SCS_QUERYRECONVERTSTRING を呼ぶことで IME に RECONVERTSTRING 構造体を調整するかをたずねたら、効率の良い再変換が遂行できる。

SCS_SETRECONVERTSTRING か SCS_QUERYRECONVERTSTRING は、
SCS_CAP_SETRECONVERTSTRING プロパティを持った IME に対してのみ使用できる。このプロパティは、ImmGetProperty 関数を使うことで取得できる。

IMR_RECONVERTSTRING と IMR_CONFIRMRECONVERTSTRING メッセージ

IME が再変換したいとき、アプリに、再変換される文字列の提供を問い合わせることができる。例えば、ユーザが再変換キーを押したときや、IME 状態ウィンドウで再変換ボタンを選択したとき、IME は、ターゲット文字列を取得するために、WM_IME_REQUEST メッセージを IMR_RECONVERTSTRING をつけて送信する。最初に IME は、RECONVERTSTRING の要求サイズを取得するために、これに NULL の lParam とともに送信する必要がある。IME はそのとき、ターゲット文字列を受け取るバッファを用意し、再び、lParam にバッファのポインタをセットしてメッセージを送信する。

アプリが IMR_RECONVERTSTRING を扱った後で、IME はアプリによって与えられた RECONVERTSTRING を調整するかもしれないし、しないかもしれない。最後に IME は、RECONVERTSTRING 構造体を確認するために、WM_IME_REQUEST メッセージを IMR_CONFIRMRECONVERTSTRING をつけて送信する。

もし、アプリが IMR_CONFIRMRECONVERTSTRING に対して TRUE を戻り値として返せば、IME は、IMR_CONFIRMRECONVERTSTRING メッセージの中で RECONERTSTRING 構造体 に基づいて新しいコンポジション文字列を生成する。もしアプリが FALSE を返せば、IME は、IMR_RECONVERTSTRING メッセージの中でアプリによって与えられたオリジナルの RECONVERTSTRING 構造体に基づいて新しいコンポジション文字列を生成する。IME は、IMR_CONFIRMRECONVERTSTRING の前に再変換用のコンポジション文字列を生成しようとしないだろう。

与えられた文字列は、SCS_QUERYRECONVERTSTRING や IMR_CONFIRMRECONVERTSTRING によって変更されるべきではない。IME は、再確認のため、CompStrOffset、CompStrLen、TargetStrOffset、そして TargetStrLen のみ変更できる。

IME メニュー関数たち

この関数集合の目的は、システムタスクバーの IME 関連のアイコンを減らすためにある。これは、Windows 98 と Windows 2000 で登場した新しい機能だ。

Windows システムプログラムは、現在の hKL（キーボードレイアウト）が IME であれば、タスクバーに二つのアイコンを導入する。一つのアイコンは、システムタスクバーにある、現在のキーボードレイアウトを示すシステム ML アイコンである。もう一方は、フォーカスされたウィンドウの IME 状態を表すシステムペンアイコンである。たいてい、IME は、追加的なアイコンをタスクバーに置く。このアイコンに対するコンテキストメニューは、完全に IME に依存している。IME アイコンをタスクバーに持つことは、ユーザにとって IME の特別な機能にアクセスする素早い方法だ。しかしながら、IME に関連付けられた三つのアイコンが存在し、それらの追加的なアイコンは、ユーザが欲する以上のものであるかもしれない。

もしシステムが IME に対して IME メニューアイテムをシステムペンアイコンに挿入する方法を提供すれば、IME は余分なアイコンをタスクバーに追加する必要はない。

IME は、IME メニュー項目を取得するために、IME 関数 ImeGetImeMenuItems を呼ぶ。

アプリは、IME の特別なメニュー項目を取得するために、ImmGetImeMenuItems を使うことができる。それらの項目は、コンテキストメニューに追加できる。ImmNotify を呼び出すことで、選択項目が IME によって処理で

きる。

IME メニュー通知

アプリが IME のメニュー項目を扱いたいとき、ImmNotifyIME 関数を呼ぶことができる。IME によって追加されたメニュー項目が選択されたとき、フォーカスされたスレッドのもとで ImmNotifyIME が呼ばれる。

IME ヘルプファイル

IME ヘルプファイルは、Windows 98 と Windows NT に追加された新しい機能である。システムペンアイコンの右クリックメニューは二つのメニュー項目を持つ。一つは、IME システムの設定であり、フォーカススレッドの選択中の IME の設定を変更するために使われる。もう一つのメニュー項目は、決して有効にならないオンラインヘルプファイルである。よってこのメニュー項目は常に灰色である。このメニュー項目の目的は、IME のオンラインヘルプを表示するためであったが、システムが IME ヘルプファイルの名前を指定する方法を持った IME を提供しなかったため、システムタスクバープログラムは、それを表示できない。

IME_ESC_GETHELPFILENAME

IME_ESC_GETHELPFILENAME エスケープは、IME にヘルプファイル名を指定することを可能にする。この関数から戻るとき、(LPTSTR)lpData は IME のヘルプファイルのフルパスファイル名である。

もしヘルプの内容が HTML ヘルプ形式であれば、ヘルプファイルの拡張子が.chm であることを確認して下さい。そうすると Windows ユーザはどのヘルプエンジンを開始するかがわかる。

IME に対するインディケータ管理

Windows 98 と Windows 2000 で定義されたメッセージ集合を使えば、IME は、システムタスクバー上のシステムペンアイコンに対するアイコンとツールチップを変更できる。

インディケータウィンドウ

IME は、FindWindow 関数を INDICATOR_CLASS と共に使って、ウィンドウハンドルを取得できる。

```
// Get the window handle of Indicator.
hwndIndicator = FindWindow(INDICATOR_CLASS, NULL);
if (!hwndIndicator)
{
      // There is no indicator window. Tray does not have System Pen icon.
      return FALSE;
}
// Call PostMessage to change Pen icon.
PostMessage(hwndIndicator, INIDCM_SETIMEICON, nIconIndex, hMyKL);
```

注記

タスクバー管理の内部設計の要求のため、IME は INDICM_xxx メッセージに対して PostMessage を使わねばならない。

メッセージ

IME は、異なるタスクを行うために、以下のメッセージをインディケータウィンドウに送信できる。

- INDICM_SETIMEICON
- INDICM_SETIMETOOLTIPS
- INDICM_REMOVEDEFAULTMENUITEMS

Windows NT とWindows 2000 の事案

以下のトピックスは、本質的に、Windows NT/2000 に関連する特別な事案を述べる。しかしながら、いくつかは Windows 98 にも適用されうる。

IME とローカライズされた言語互換性

Windows 2000 は、すべてのローカライズされた言語バージョンにおける完全に組み込まれた IME サポートを持つ。すなわち、Windows 2000 のどんな言語についても IME をインストールして使うことができる。IME 開発者は、それらの IME をそれらの環境でテストすべきだ。この新しい特徴はまた、そして、正しい文字セットとフォント情報を含み、別の言語の OS においても正しく見られるような、IME ヘルプの内容を準備させることを IME 開発者に要求する。

また、IME 開発者は、Windows 2000 に対しては Unicode IME を開発すべきだ。Unicode IME は、どんなシステムロケールでも Unicode アプリで動作するだろう。Unicode ではない IME については、それらを使うために、IME がサポートする言語と同じ言語をサポートするため、ユーザはシステムロケールを変更しなければならない。

Unicode インターフェイス

Windows 95 でサポートされた IMM/IME インターフェイスの ANSI 版と共に、Windows NT と Windows 98 は IME に対して Unicode インターフェイスをサポートする。Unicode インターフェイスによってシステムと通信するため、IME は、IMEINFO 構造体の fdwProperty メンバーの IME_PROP_UNICODE ビットをセットするべきだ。

fdwProperty は、ImeInquire 関数の最初の引数である。ImeInquire は、アプリプロセスのすべてのスレッドに対して IME を初期化するために呼び出される。IME は、単一のシステムにおいて同じ IMEINFO 構造体を返すことを想定している。Windows 98 は、ImmIsUIMessage を除くすべての Unicode 関数をサポートする。

セキュリティ関連

Windows NT に対して、二つの主なセキュリティの関心事がある。一つは、名前付きオブジェクトで、もう一つは Winlogon である。

名前付きオブジェクト

IME は、ローカルシステムにおいて、複数のプロセスからアクセスされる様々な名前付きオブジェクトを作成したいかもしれない。そのようなオブジェクトは、ファイル、ミューテックス、イベントを含むかもしれない。プロセスは、対話的にログオンしている別のユーザに属するので、(セキュリティ属性へのポインタとして NULL を指定して、IME がオブジェクトを作成したときにシステムによって作成された)既定のセキュリティ属性は、すべてのプロセスにとって適切ではないかもしれない。

Windows NT では、IME の最初のクライアントプロセスは、Winlogon プロセスかもしれない。Winlogon プロセスとは、ユーザがシステムにログオンするプロセスである。Winlogon プロセスは、ログオンセッションの間、

システムアカウントに属するのだが、それがシステムがシャットダウンするまで生きていて、既定のセキュリティ属性でIMEによって作成された名前付きオブジェクトは、ログオンセッションの間、他のログオンしたユーザに属する他のプロセスを通じてはアクセスできない。

名前付きオブジェクトに対して適切に設定されたセキュリティ属性を作成するサンプルIMEソースコードは、Microsoft Platform DDKで提供されている。サンプルコードを使えば、IMEを書く人は、ローカルシステム上のIMEのすべてのクライアントプロセスからアクセスされる、様々な名前付きオブジェクトを作成できる。サンプルコードによって割り当てられるセキュリティ属性は、プロセスごとである。IMEは、取り付け時にセキュリティ属性を初期化、または取り外し時にセキュリティ属性を解放するために、名前付きオブジェクトをしばしば作成したいかもしれない。名前付きオブジェクトを作成しないIMEは、しばしば、名前付きオブジェクトの作成の直前に、セキュリティ属性を初期化したいかもしれないし、オブジェクトが作成された直後にセキュリティ属性を解放したいかもしれない。

Winlogon

ログオンセッションのユーザは、まだシステムへのアクセス権が許されないので、IMEの設定ダイアログで提供された情報は、セキュリティ問題を生じかねない。だが、システム管理者は、ログオンセッション上でIMEがアクティブにならないようにシステムの設定を変更できる。うまくふるまうIMEは、IMEのクライアントプロセスがWinlogonプロセスだったなら、ユーザに設定ダイアログを開くことを許すべきではない。IMEは、もし、ログオンセッションを実行しているクライアントプロセスがWinlogonプロセスかどうかを、ImeInquire関数の引数dwSystemInfoFlagsのIME_SYSTEMINFO_WINLOGONビットにより、チェックできる。

IMEファイル形式とデータ構造

次のトピックスは、IMEファイル形式とIMEに使われているデータ構造を議論する。

IMEファイル形式

IMEは、次のフィールドをバージョン情報リソースに正しく指定する必要がある。これは、固定ファイル情報の部分と、可変長情報の部分を含む。バージョン情報リソースにおける詳しい情報については、Microsoft Platform SDKを参照されたい。

以下が、IMEファイルが含むべき特定の設定である:

- dwFileOS。dwFileOSは、VOS__WINDOWS32であるべきだ。
- dwFileType。IMEでは、VFT_DRVを指定するべきだ。
- dwFileSubtype。IMEでは、VFT2_DRV_INPUTMETHODを指定すべきだ。
- FileDescription。FileDescriptionは、言語固有のブロックに指定される。これはIMEの名前とバージョンであるべきだ。この文字列は表示目的で使われる。長さは32個分のTCHARである。
- ProductName。言語固有のブロックで指定される。文字通り製品名である。
- 文字セットIDと言語ID。コードページ(文字セットID; CharSet ID)と言語ID (Language ID)の対である。複数のコードページが指定されるとIMEは最初の対を使用する。

ファイル名は8.3形式でなければならない。

IMEレジストリ内容

HKEY_CURRENT_USERレジストリは入力メソッドのキーを含む。次の表はこの入力メソッドの内容を説明す

る。

(表は省略)

IMM と IME のデータ構造

以下の構造体は IMM と IME の通信で使われる。IME は、これらの構造体に直接アクセスできるが、アプリはできない。

INPUTCONTEXT

INPUTCONTEXT 構造体は、入力コンテキストのデータを格納する内部のデータ構造体である。

```
typedef struct tagINPUTCONTEXT {
      HWND hWnd;
      BOOL fOpen;
      POINT ptStatusWndPos;
      POINT ptSoftKbdPos;
      DWORD fdwConversion;
      DWORD fdwSentence;
      union {
            LOGFONTA A;
            LOGFONTW W;
      } lfFont;
      COMPOSITIONFORM cfCompForm;
      CANDIDATEFORM cfCandForm[4];
      HIMCC hCompStr;
      HIMCC hCandInfo;
      HIMCC hGuideLine
      HIMCC hPrivate;
      DWORD dwNumMsgBuf;
      HIMCC hMsgBuf;
      DWORD fdwInit
      DWORD dwReserve[3];
} INPUTCONTEXT;
```

メンバー

メンバーの名前	説明
hWnd	入力コンテキストを使うウィンドウハンドル。入力コンテキストが共有ウィンドウを持っていれば、これはアクティブなウィンドウのハンドルである。 ImmSetActiveContext で入力コンテキストをリセットできる。
fOpen	IME が開いているか、閉じているかの現在の状態。
ptStatusWndPos	状態ウィンドウの位置。
ptSoftKbdPos	ソフトキーボードの位置。
fdwConversion	変換モード。
fdwSentence	センテンスモード。
lfFont	コンポジション文字列を描画するための論理フォント。LOGFONT 構造体。
cfCompForm	COMPOSITIONFORM 構造体。コンポジションウィンドウを作成するときに使われる。
cfCandForm[4]	CANDIDATEFORM 構造体。候補ウィンドウを作成するときに、使われる。

メンバーの名前	説明
hCompStr	COMPOSITIONSTRING 構造体を指すメモリーハンドル。コンポジション文字列があるとき利用可能。
hCandInfo	CANDIDATEINFO 構造体を指す候補のメモリーハンドル。候補文字列があるとき、利用可能。
hGuideLine	ガイドラインを表す GUIDELINE 構造体を指すメモリーハンドル。ガイドラインがあるとき利用可能。
hPrivate	IME のプライベートなメモリーハンドル。
dwNumMsgBuf	hMsgBuf に格納されたメッセージの個数。
hMsgBuf	TRANSMSG 構造体で表されたメッセージ列を指すメモリーブロック。
fdwInit	初期化フラグ。INPUTCONTEXT 構造体を初期化するときに、このビット列を見なければならない。 INIT_STATUSWNDPOS は ptStatusWndPos を初期化する。 INIT_CONVERSION は fdwConversion を初期化する。 INIT_SENTENCE は fdwSentence を初期化する。 INIT_LOGFONT は lfFont を初期化する。 INIT_COMPFORM は cfCompForm を初期化する。 INIT_SOFTKBDPOS は ptSoftKbdPos を初期化する。
dwReserve[3]	将来のため予約済み。

注記

ImeToAsciiEx を呼び出す間、IME は、lpdwTransKey バッファ内へメッセージ列を生成できる。しかしながら、もし IME がアプリへメッセージを生成したければ、それは、メッセージ列を hMsgBuf 内部へ格納して、ImmGenerateMessage を呼ぶことができる。ImmGenerateMessage 関数はそのとき、hMsgBuf 内のメッセージをアプリへ送る。

COMPOSITIONSTRING 構造体

COMPOSITIONSTRING 構造体は、コンポジション文字列の情報を持つ。変換の間、IME は変換情報をこの構造体へ格納する。

```
typedef struct tagCOMPOSITIONSTR {
      DWORD dwSize;
      DWORD dwCompReadAttrLen;
      DWORD dwCompReadAttrOffset;
      DWORD dwCompReadClsLen;
      DWORD dwCompReadClsOffset;
      DWORD dwCompReadStrLen;
      DWORD dwCompReadStrOffset;
      DWORD dwCompAttrLen;
      DWORD dwCompAttrOffset;
      DWORD dwCompClsLen;
      DWORD dwCompClsOffset;
      DWORD dwCompStrLen;
      DWORD dwCompStrOffset;
      DWORD dwCursorPos;
      DWORD dwDeltaStart;
      DWORD dwResultReadClsLen;
      DWORD dwResultReadClsOffset;
```

```
        DWORD dwResultReadStrLen;
        DWORD dwResultReadStrOffset;
        DWORD dwResultClsLen;
        DWORD dwResultClsOffset;
        DWORD dwResultStrLen;
        DWORD dwResultStrOffset;
        DWORD dwPrivateSize;
        DWORD dwPrivateOffset;
} COMPOSITIONSTR;
```

メンバー

メンバーの名前	説明
dwSize	この構造体のメモリーブロックのサイズ。
dwCompReadAttrLen	コンポジション文字列の読みの文字列の属性情報の長さ。
dwCompReadAttrOffset	この構造体の開始位置から属性へのオフセット。属性情報はここに格納される。
dwCompReadClsLen	コンポジション文字列の読みの文字列の文節情報の長さ。
dwCompReadClsOffset	この構造体の開始位置から文節へのオフセット。文節情報はここに格納される。
dwCompReadStrLen	コンポジション文字列の読みの文字列の長さ。
dwCompReadStrOffset	この構造体の開始位置から読みの文字列へのオフセット。コンポジション文字列の読みの文字列は、ここに格納される。
dwCompAttrLen	コンポジション文字列の属性情報の長さ。
dwCompAttrOffset	この構造体の開始位置から属性へのオフセット。属性情報はここに格納される。
dwCompClsLen	コンポジション文字列の文節情報の長さ。
dwCompClsOffset	この構造体の開始位置から文節へのオフセット。文節情報はここに格納される。
dwCompStrLen	コンポジション文字列の長さ。
dwCompStrLen	この構造体の開始位置からコンポジション文字列へのオフセット。コンポジション文字列はここに格納される。
dwCursorPos	コンポジション文字列におけるカーソルの位置。
dwDeltaStart	コンポジション文字列における変更部分の開始位置。もし、コンポジション文字列が以前の状態から変更されていれば、その変更点の最初の位置がここに格納される。
dwDeltaStart	結果文字列の読みの文字列の文節情報の長さ。
dwResultReadClsOffset	この構造体の開始位置から結果読み文節へのオフセット。文節情報はここに格納される。
dwResultReadStrLen	結果文字列の読みの文字列の長さ。
dwResultReadStrOffset	この構造体の開始位置から読みの文字列へのオフセット。結果文字列の読みの文字列がここに格納される。
dwResultClsLen	結果文字列の文節情報の長さ。

メンバーの名前	説明
dwResultClsOffset	この構造体の開始位置から結果文節へのオフセット。文節情報はここに格納される。
dwResultStrLen	結果文字列の長さ。
dwResultStrLen	この構造体の開始位置から結果文字列へのオフセット。結果文字列はここに格納される。
dwPrivateSize	このメモリーブロックのプライベート領域のサイズ。
dwPrivateOffset	この構造体の開始位置からプライベート領域へのオフセット。プライベート領域はここに格納される。

Unicode の場合の注記：すべての dw*StrLen メンバーは、対応するバッファ内の Unicode 文字列のサイズである。その他の dw*Len や dw*Offset メンバーは、対応するバッファ内のバイト数である。

属性（attribute）情報の形式は、各文字に対応するバイトの配列である。次の値が提供される。リストにないものは予約済みである。

値	内容
ATTR_INPUT	現在入力されている文字。
ATTR_TARGET_CONVERTED	変換中ですでに変換済みの文字。
ATTR_CONVERTED	IME により変換された文字。
ATTR_TARGET_NOTCONVERTED	変換中で選択されたがまだ変換済みではない文字。
ATTR_FIXEDCONVERTED	これ以上変換されない文字。
ATTR_INPUT_ERROR	エラー文字で変換できない。

以下は上の表で述べられた内容の説明である。

内容	説明
現在入力されている文字。	ユーザが入力中の文字。日本語ならこの文字はまだ IME によって変換されていないひらがな、カタカナ、英数字になる。
変換中ですでに変換済みの文字。	ユーザによって選択され、IME によって選択された文字。
IME により変換された文字。	IME が変換した文字。
変換中で選択されたがまだ変換済みではない文字。	ユーザによって変換のために選択されていて、まだ IME によって変換されていない文字。日本語ならこの文字は、ユーザが入力したひらがな、カタカナ、英数字になる。
エラー文字で変換できない。	文字がエラー文字で、IME がこの文字を変換できない。例えばいくつかの子音は連結できない。

コメント

属性情報の長さは、文字列の長さと同じ。それぞれのバイトが文字列の1バイトに対応する。文字列が DBCS 文字を含んでいたとしても、属性情報は、先行（lead）バイトと二番目のバイトの両方の情報バイトを持つ。

Windows NT Unicode では、属性情報の長さは、Unicode 文字の個数での長さと同じだ。それぞれの属性バイトは、それぞれの Unicode 文字に対応する。

文節情報の形式は、ダブルワードの配列であり、文節の位置を表す数値を指定する。文節の位置は、そのコンポジション文字列の位置であり、それはこの位置から始まる文節を伴う。少なくとも、情報の長さは2ダブルワード以上である。これは、1つの文節情報の長さは 8 バイトであることを意味する。最初のダブルワード値は、ゼロであり、それは最初の文節の位置である。最後のダブルワード値は、この文字列の長さでなければならない。例えば、文字列が 3 つの文節を持てば、文節情報は 4 ダブルワードを持つ。最初のダブルワードはゼロだ。二番目のダブルバイトは、二番目の文節の開始位置を表す。三番目のダブルバイトは、三番目の文節の開始位置を指定する。最後のダブルバイトは、この文字列の長さである。

Windows NT Unicode では、それぞれの文節の位置と、その文字列の長さは、Unicode 文字でカウントされる。

dwCursorPos メンバーは、カーソル位置を指定し、文字列の個数でコンポジション文字列のどこにカーソルが位置するのかを示す。カウントはゼロから始まる。もしカーソルがコンポジション文字列の直後に位置していれば、この値は、コンポジション文字列の長さに等しいはずだ。もしもカーソルがなければ、-1 の値が指定される。コンポジション文字列が存在しなければ、このメンバーは無効である。

Windows NT Unicode では、カーソル位置は Unicode 文字で数えられる。

CANDIDATEINFO 構造体

CANDIDATEINFO 構造体は、候補情報のヘッダーである。この構造体は、最大で 32 個の候補リストを所持でき、これらの候補リストは、同じメモリーブロックの中にないといけない。

```
typedef struct tagCANDIDATEINFO {
      DWORD dwSize;
      DWORD dwCount;
      DWORD dwOffset[32];
      DWORD dwPrivateSize;
      DWORD dwPrivateOffset;
} CANDIDATEINFO;
```

メンバー

メンバーの名前	説明
dwSize	このメモリーブロックのサイズ。
dwCount	このメモリーブロックに含まれる候補リストの個数。
dwOffset[32]	中身は、この構造体の開始位置からのオフセットであり、各オフセットは、それぞれの候補リストの開始位置を指定する。
dwPrivateSize	このメモリーブロックのプライベート領域のサイズ。
dwPrivateOffset	この構造体の開始位置からプライベート領域の開始位置へのオフセット。

GUIDELINE 構造体

GUIDELINE 構造体は、IME が送り出すガイドライン情報を所持する。

```
typedef struct tagGUIDELINE {
      DWORD dwSize;
      DWORD dwLevel; // the error level.
      // GL_LEVEL_NOGUIDELINE,
      // GL_LEVEL_FATAL,
      // GL_LEVEL_ERROR,
      // GL_LEVEL_WARNNING,
      // GL_LEVEL_INFORMATION
      DWORD dwIndex; // GL_ID_NODICTIONARY and so on.
      DWORD dwStrLen; // Error Strings, if this is 0, there
```

```
        // is no error string.
        DWORD dwStrOffset;
        DWORD dwPrivateSize;
        DWORD dwPrivateOffset;
    } GUIDELINE;
```

注記

Unicode では、dwStrLen メンバーは Unicode 文字で数えたときのエラー文字列のサイズである。dwSize、dwStrOffset、dwPrivateSize などの他のサイズのパラメータは、バイト数で数えたときのサイズである。

メンバー

<table>
<tr><th>メンバーの名前</th><th>説明</th></tr>
<tr><td>dwLevel</td><td>以下のようなエラーレベルを指定する。

<table><tr><th>値</th><th>説明</th></tr><tr><td>GL_LEVEL_NOGUIDELINE</td><td>何もガイドラインがないことを意味し、UI は、古いガイドラインは非表示にするべきだ。</td></tr><tr><td>GL_LEVEL_FATAL</td><td>致命的なエラーが発生したことを意味する。</td></tr><tr><td>GL_LEVEL_ERROR</td><td>通常のエラーが発生したことを意味し、処理が止まるかもしれない。</td></tr><tr><td>GL_LEVEL_WARNING</td><td>警告が発生したことを意味し、処理は続行される。</td></tr><tr><td>GL_LEVEL_INFORMATION</td><td>ユーザに対する情報を意味する。</td></tr></table></td></tr>
<tr><td>dwIndex</td><td>次の値を指定する。

<table><tr><th>値</th><th>説明</th></tr><tr><td>GL_ID_UNKNOWN</td><td>未知のエラー。</td></tr><tr><td>GL_ID_NOMODULE</td><td>IME が必要なモジュールが見つからなかったときに指定する。</td></tr><tr><td>GL_ID_NODICTIONARY</td><td>辞書が見つからないか、辞書がおかしいときに指定する。</td></tr><tr><td>GL_ID_CANNOTSAVE</td><td>辞書や統計情報が保存できないときに指定する。</td></tr><tr><td>GL_ID_NOCONVERT</td><td>少しも変換できないときに指定する。</td></tr><tr><td>GL_ID_TYPINGERROR</td><td>IME が処理できないタイプエラーのとき指定する。</td></tr><tr><td>GL_ID_TOOMANYSTROKE</td><td>一文字や一つの文節に多すぎるキーストロークがあったときに指定する。</td></tr><tr><td>GL_ID_READINGCONFLICT</td><td>読みの衝突が発生したときに指定する。例えば、子音が連結できないときである。</td></tr><tr><td>GL_ID_INPUTREADING</td><td>IME がユーザに読みの文字入力状態にあることを伝える。</td></tr><tr><td>GL_ID_INPUTRADICAL</td><td>IME がユーザに部首文字入力状態にあることを伝える。</td></tr></table></td></tr>
</table>

<table>
<tr><th>メンバーの名前</th><th>説明</th></tr>
<tr><td></td><td>
<table>
<tr><th>値</th><th>説明</th></tr>
<tr><td>GL_ID_INPUTCODE</td><td>ユーザに文字コード入力状態にあることを伝える。</td></tr>
<tr><td>GL_ID_CHOOSECANIDATE</td><td>ユーザに、候補文字列を選択する状態であることを伝える。</td></tr>
<tr><td>GL_ID_REVERSECONVERSION</td><td>逆変換の情報をユーザーに伝える。逆変換は、ImmGetGuideLine(hIMC, GGL_PRIVATE, lpBuf, dwBufLen)を通じて取得できる。lpBuf の情報は、CANDIDATELIST 形式である。</td></tr>
<tr><td>GL_ID_PRIVATE_FIRST</td><td>GL_ID_PRIVATE_FIRST と GL_ID_PRIVATE_LAST の間に位置する ID は、IME 用に予約済みである。IME はこれらの ID を自分自身の GUIDELINE 用に使うことができる。</td></tr>
<tr><td>GL_ID_PRIVATE_LAST</td><td>GL_ID_PRIVATE_FIRST と GL_ID_PRIVATE_LAST の間に位置する ID は、IME 用に予約済みである。IME はこれらの ID を自分自身の GUIDELINE 用に使うことができる。</td></tr>
</table>
</td></tr>
<tr><td>dwPrivateSize</td><td>プライベート領域のサイズ。</td></tr>
<tr><td>dwPrivateOffset</td><td>この構造体からプライベート領域の開始位置へのオフセット。プライベート領域はここに格納される。</td></tr>
</table>

IME を管理する構造体

IME を管理するための構造体について取り上げる。

IMEINFO 構造体

IMEINFO 構造体は、IMM または IME インターフェイスにより、内部で使われる。

```
typedef struct tagIMEInfo {
      DWORD dwPrivateDataSize; // The byte count of private data
      // in an IME context.
      DWORD fdwProperty; // The IME property bits. See
      // description below.
      DWORD fdwConversionCaps; // The IME conversion mode
      // capability bits.
      DWORD fdwSentenceCaps; // The IME sentence mode
      // capability.
      DWORD fdwUICaps; // The IME UI capability.
      DWORD fdwSCSCaps; // The ImeSetCompositionString
      // capability.
      DWORD fdwSelectCaps; // The IME inherit IMC capability.
} IIMEINFO;
```

メンバー

メンバーの名前	説明
dwPrivateDataSize	プライベート領域のバイトサイズ。

<table>
<tr><th>メンバーの名前</th><th>説明</th></tr>
<tr><td>fdwProperty</td><td>fdwProperty の上位ワードは、次のようなビットたちを所持し、それらはアプリによって使われる。

<table>
<tr><th>ビット</th><th>説明</th></tr>
<tr><td>IME_PROP_AT_CARET</td><td>IME コンポジションウィンドウがキャレット位置にあることを意味する。</td></tr>
<tr><td>IME_PROP_SPECIAL_UI</td><td>IME が特別な UI を有することを示す。IME は非標準の UI に対してこのビットを指定すべきだ。</td></tr>
<tr><td>IME_PROP_CANDLIST_START_FROM_1</td><td>候補リストの文字列が 1 から始めるか、またはゼロから始まるかを指定する。</td></tr>
<tr><td>IME_PROP_UNICODE</td><td>セットされれば、IME は Unicode IME として見なされる。システムと IME は Unicode IME インターフェイスを通じて通信するだろう。</td></tr>
<tr><td>IME_PROP_COMPLETE_ON_UNSELECT</td><td>Windows 98 と Windows 2000 で定義された。このビットがセットされれば、IME が非アクティブになったときにコンポジション文字列を確定するだろう。</td></tr>
</table>

fdwProperty の下位ワードは、次のようなビットたちを所持し、システムによって使われる。

<table>
<tr><th>ビット</th><th>説明</th></tr>
<tr><td>IME_PROP_END_UNLOAD</td><td>IME が誰にも使われていないときにアンロードすることを示す。</td></tr>
<tr><td>IME_PROP_KBD_CHAR_FIRST</td><td>これをつけると、DBCS 文字列を変換する前に、システムがキーボードによる文字を変換する。この文字は情報の助けとして IME に渡される。</td></tr>
<tr><td>IME_PROP_NEED_ALTKEY</td><td>これをつけると、IME は、ImeProcessKey に渡すのに ALT キーを必要とする。</td></tr>
<tr><td>IME_PROP_IGNORE_UPKEYS</td><td>これを指定すると、上向きキーを必要としない。</td></tr>
<tr><td>IME_PROP_ACCEPT_WIDE_VKEY</td><td>Windows 2000 でサポートされる。これをつけると、IME は、VK_PACKET を使って SendInput 関数から来たインジェクトされた Unicode を処理する。Unicode とインジェクトされた Unicode はアプリに直接送信される。</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>メンバーの名前</th><th>説明</th></tr>
<tr><td>fdwConversionCaps</td><td>変換モードと同じ定義。もし関係するビットが OFF なら、IME は、変換モードの対応するビットが ON か OFF に関わらず、変換モードを扱うための能力を持たない。

<table>
<tr><th>変換モード</th><th>説明</th></tr>
<tr><td>IME_CMODE_KATAKANA</td><td>IME はカタカナをサポートする。</td></tr>
<tr><td>IME_CMODE_NATIVE</td><td>IME はネイティブモードをサポートする。</td></tr>
<tr><td>IME_CMODE_FULLSHAPE</td><td>IME は全角モードをサポートする。</td></tr>
<tr><td>IME_CMODE_ROMAN</td><td>IME はローマ字入力をサポートする。</td></tr>
<tr><td>IME_CMODE_CHARCODE</td><td>IME は文字コード入力をサポートする。</td></tr>
<tr><td>IME_CMODE_HANJACONVERT</td><td>IME は Hanja 変換モードをサポートする。</td></tr>
<tr><td>IME_CMODE_SOFTKBD</td><td>IME はソフトキーボードをサポートする。</td></tr>
<tr><td>IME_CMODE_NOCONVERSION</td><td>IME は変換なしモードをサポートする。</td></tr>
<tr><td>IME_CMODE_EUDC</td><td>IME はエンドユーザ定義文字（EUDC）モードをサポートする。</td></tr>
<tr><td>IME_CMODE_SYMBOL</td><td>IME はシンボルモードをサポートする。</td></tr>
<tr><td>IME_CMODE_FIXED</td><td>IME は固定変換モードをサポートする。このモードは前変換を可能にするが、すべての変換ではない。例としては、DBCS のひらがなローマ字の固定変換モードがある。このモードでは、IME はキー入力文字列をローマ字変換により DBCS ひらがなに変換できるが、ひらがなから漢字への変換は防止される。</td></tr>
</table>
</td></tr>
<tr><td>fdwSentenceCaps</td><td>センテンスモードと同じ定数定義である。もし関係するビットが OFF なら、IME は、センテンスモードの対応するビットが ON か OFF に関わらず、センテンスモードを扱うための能力を持たない。

<table>
<tr><th>センテンスモード</th><th>説明</th></tr>
<tr><td>IME_SMODE_PLAURALCLAU</td><td>IME は、複数文節センテンスモード</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>メンバーの名前</th><th>説明</th></tr>
<tr><td></td><td>
<table>
<tr><th>センテンスモード</th><th>説明</th></tr>
<tr><td>SE</td><td>をサポートする。</td></tr>
<tr><td>IME_SMODE_SINGLECONVERT</td><td>IME は、単一文節センテンスモードをサポートする。</td></tr>
<tr><td>IME_SMODE_AUTOMETIC</td><td>IME は、自動センテンスモードをサポートする。</td></tr>
<tr><td>IME_SMODE_PHRASEPREDICT</td><td>IME は、フレーズ予測センテンスモードをサポートする。</td></tr>
<tr><td>IME_SMODE_CONVERSATION</td><td>IME は対話モードを使う。これはチャットアプリで便利である。チャットアプリは、IME のセンテンスモードを対話スタイルに変更できる。これは Windows 98 と Windows 2000 で新しく登場する。</td></tr>
</table>
</td></tr>
<tr><td>fdwUICaps</td><td>これは UI の能力を指定するビット列である。
<table>
<tr><th>ビット</th><th>説明</th></tr>
<tr><td>UI_CAP_2700</td><td>UI は、LogFont エスケープがゼロか 2700 のときの UI をサポートする。</td></tr>
<tr><td>UI_CAP_ROT90</td><td>UI は、LogFont エスケープは、ゼロか、900、1800 または 2700 をサポートする。</td></tr>
<tr><td>UI_CAP_ROTANY</td><td>UI は、すべての LogFont エスケープをサポートする。</td></tr>
<tr><td>UI_CAP_SOFKBD</td><td>システムから提供されるソフトキーボードを使う。</td></tr>
</table>
</td></tr>
<tr><td>fdwSCSCaps</td><td>IME が持つ ImmSetCompositionString の能力を指定する。
<table>
<tr><th>ビット</th><th>説明</th></tr>
<tr><td>SCS_CAP_COMPSTR</td><td>IME は SCS_SETSTR によるコンポジション文字列を生成できる。</td></tr>
<tr><td>SCS_CAP_MAKEREAD</td><td>SCS_SETSTR をつけて ImmSetCompositionString を呼び出したとき、IME は lpRead を使うことなくコンポジション文字列の読みを生成できる。IME がこの能力 SCS_CAP_MAKEREAD を持つとき、SCS_SETSTR に対してアプリは、lpRead をセットする必要はない。</td></tr>
</table>
</td></tr>
<tr><td>fdwSelectCaps</td><td>アプリ用の能力を指定する。ユーザが IME を変換したとき、アプリはこの能力をチェックすることで、変換モードを継承するか、しないかを決定できる。もし新しく選択した IME がこの能力を持たなければ、アプリは新</td></tr>
</table>

<table>
<tr><th>メンバーの名前</th><th>説明</th></tr>
<tr><td></td><td>しいモードを受け入れないだろう。そしてモードを再び取得する必要があるだろう。以下のビットが与えられる。
<table><tr><th>ビット</th><th>説明</th></tr><tr><td>SELECT_CAP_CONVMODE</td><td>ImeSelect において、IME は、変換モードを継承する能力を持っている。</td></tr><tr><td>SELECT_CAP_SENTENCE</td><td>ImeSelect において、IME は、センテンスモードを継承する能力を持っている。</td></tr></table></td></tr>
</table>

IME 通信で使われる構造体

CANDIDATELIST 構造体は、候補リストに関する情報を持つ。

```
typedef struct tagCANDIDATELIST {
      DWORD dwSize; // the size of this data structure.
      DWORD dwStyle; // the style of candidate strings.
      DWORD dwCount; // the number of the candidate strings.
      DWORD dwSelection; // index of a candidate string now selected.
      DWORD dwPageStart; // index of the first candidate string show in
      // the candidate window. It maybe varies with
      // page up or page down key.
      DWORD dwPageSize; // the preference number of the candidate
      // strings shows in one page.
      DWORD dwOffset[]; // the start positions of the first candidate
      // strings. Start positions of other
      // (2nd, 3rd, ..) candidate strings are
      // appened after this field. IME can do this
      // by reallocating the hCandInfo memory
      // handle. So IME can access dwOffset[2] (3rd
      // candidate string) or dwOffset[5] (6st
      // candidate string).
      // TCHAR chCandidateStr[]; // the array of the candidate strings.
} CANDIDATELIST;
```

メンバー

<table>
<tr><th>メンバーの名前</th><th>説明</th></tr>
<tr><td>dwSize</td><td>この構造体、オフセット配列、そしてすべての候補文字列を合わせたバイトサイズ。</td></tr>
<tr><td>dwStyle</td><td>候補スタイル値。以下の 1 個以上の値を指定できる。
<table><tr><th>値</th><th>意味</th></tr><tr><td>IME_CAND_UNKNOWN</td><td>未知の候補スタイル。</td></tr><tr><td>IME_CAND_READ</td><td>候補は同じ読みを持つ。</td></tr><tr><td>IME_CAND_CODE</td><td>候補は一つのコード範囲にある。</td></tr><tr><td>IME_CAND_MEANING</td><td>候補は同じ意味を持つ。</td></tr><tr><td>IME_CAND_RADICAL</td><td>候補は同じ部首から構成される。</td></tr></table></td></tr>
</table>

<table>
<tr><th>メンバーの名前</th><th>説明</th></tr>
<tr><td></td><td>
<table>
<tr><th>値</th><th>意味</th></tr>
<tr><td>IME_CAND_STROKES</td><td>候補は同じ画数から構成される。</td></tr>
</table>

IME_CAND_CODE スタイルについては、候補リストは、dwCount メンバーに依存する特殊な構造体を持つ。もし dwCount が 1 なら、dwOffset はオフセットでなく単一の文字を持ち、候補文字列は提供されない。dwCount が 1 より大きければ、dwOffset は、有効なオフセットを持ち、候補文字列は個別の文字列のテキストである。
</td></tr>
<tr><td>dwCount</td><td>候補文字列の個数。</td></tr>
<tr><td>dwSelection</td><td>選択された候補文字列のインデックス。</td></tr>
<tr><td>dwPageStart</td><td>候補ウィンドウで最初の候補文字列のインデックス。これは PageUp と PageDown キーで変化する。</td></tr>
<tr><td>dwPageSize</td><td>候補ウィンドウの 1 ページに表示される候補文字列の個数。もしこの数がゼロなら、アプリは適当な値を定義できる。</td></tr>
<tr><td>dwOffset</td><td>この構造体の開始位置から、候補文字列の開始位置へのオフセットの配列。</td></tr>
</table>

コメント

CANDIDATELIST 構造体は、ImmGetCandidateList の結果を返すのに使われる。候補文字列は、dwOffset 配列の最後のオフセットの後に続く。

COMPOSITIONFORM 構造体

COMPOSITIONFORM 構造体は、IMC_SETCOMPOSITIONWINDOW と IMC_SETCANDIDATEPOS メッセージに使われる。

```
typedef tagCOMPOSITIONFORM {
      DWORD dwStyle;
      POINT ptCurrentPos;
      RECT rcArea;
}COMPOSITIONFORM;
```

メンバー

<table>
<tr><th>メンバーの名前</th><th>説明</th></tr>
<tr><td>dwStyle</td><td>位置スタイル。
<table>
<tr><th>値</th><th>意味</th></tr>
<tr><td>CFS_DEFAULT</td><td>コンポジションウィンドウは、既定の位置に移動する。IME ウィンドウは、クライエント領域の外側で浮動ウィンドウとしてコンポジションウィンドウを表示できる。</td></tr>
<tr><td>CFS_FORCE_POSITION</td><td>ptCurrentPos で指定された位置に正確にコンポジションウィンドウの左上端を表示する。座標は、コ</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>メンバーの名前</th><th>説明</th></tr>
<tr><td></td><td>
<table>
<tr><th>値</th><th>意味</th></tr>
<tr><td></td><td>ンポジションウィンドウを含んでいるウィンドウの左上端から相対的であり、この位置は IME から調整されない。</td></tr>
<tr><td>CFS_POINT</td><td>ptCurrentPos で指定された位置にコンポジションウィンドウの左上端を表示する。座標は、コンポジションウィンドウを含むウィンドウの左上端から相対的であり、位置は IME により補正を受ける。</td></tr>
<tr><td>CFS_RECT</td><td>rcArea で指定された位置にコンポジションウィンドウを表示する。座標は、コンポジションウィンドウを含むウィンドウの左上から相対的である。</td></tr>
</table>
</td></tr>
<tr><td>ptCurrentPos</td><td>コンポジションウィンドウの左上端の座標。</td></tr>
<tr><td>rcArea</td><td>コンポジションウィンドウの左上端と右下端の座標。</td></tr>
</table>

コメント

もし、COMPOSITIONFORM 構造体のスタイルが、CFS_POINT か CFS_FORCE_POINT であれば、IME は、アプリによって与えられた ptCurrentPos で指定した位置からコンポジション文字列を描画するだろう。もし、スタイルが CFS_RECT であれば、コンポジション文字列は、rcArea で指定された長方形の内側になるだろう。そうでなければ、rc はアプリウィンドウのクライアント長方形になる。

アプリが変換フォントを指定したときは、コンポジションウィンドウは、コンポジション文字列のエスケープメント（escapement）に従って回転される。コンポジション文字列の方向は、コンポジションウィンドウのフォントのエスケープメントに従う。IME はそのとき、コンポジションウィンドウを描画する。次は、変換フォントのエスケープメントに対して様々な値を使ったこのプロセスの例である。

- コンポジション文字列のエスケープメントがゼロ。
  - たいていは、コンポジションウィンドウのエスケープメントがゼロである。この場合、変換フォーム構造体の ptCurrentPos は、文字列の左上端を指す。すべての IME はこのタイプをサポートする。
- コンポジション文字列のエスケープメントが 2700。
  - これは縦書きの典型例。アプリが縦書きを提供するとき、アプリは、ImmSetCompositionFont に 2700 のエスケープメントを指定する。コンポジション文字列は下向きになる。UI_CAP_2700、UI_CAP_ROT90、または UI_CAP_ROTANY の能力を持つ IME は、このタイプをサポートするだろう。
- コンポジション文字列のエスケープメントが 900 か 1800。
  - UI_CAP_ROT90 や UI_CAP_ROTANY の能力を持つ IME はこれをサポートするだろう。
- コンポジション文字列のエスケープメントが任意の値。
  - UI_CAP_ROTANY の能力を持つ IME はこれをサポートするだろう。

補注

UI_CAP_ROT90 と UI_CAPS_ANY は、IME の拡張としてオプションである。UI_CAP_2700 が推奨される。

CANDIDATEFORM 構造体

CANDIDATEFORM 構造体は、IMC_GETCANDIDATEPOS と IMC_SETCANDIDATEPOS メッセージに使われる。

```
typedef tagCANDIDATEFORM {
      DWORD dwIndex;
      DWORD dwStyle;
      POINT ptCurrentPos;
      RECT rcArea;
} CANDIDATEFORM;
```

メンバー

メンバーの名前	説明
dwIndex	候補リストの ID を指定する。ゼロは最初の候補リスト、1 は二番目、そして 3 まで続く。
dwStyle	CFS_CANDIDATEPOS か CFS_EXCLUDE を指定する。キャレット近傍 IME については、CFS_DEFAULT も指定できる。dwStyle が CFS_DEFAULT ならば、キャレット近傍 IME は、他の UI コンポーネントに基づく候補位置を修正するだろう。
ptCurrentPos	dwStyle に依存する。dwStyle が CFS_CANDIDATEPOS のとき、ptCurrentPos は、候補リストウィンドウが表示すべき場所の推奨される位置を指定する。dwStyle が CFS_EXCLUDE のとき、ptCurrentPos は、キャレット位置などの関心のある現在位置を指定する。
rcArea	CFS_EXCLUDE の場合、候補ウィンドウに認められない長方形を指定する。

STYLEBUF 構造体

STYLEBUF 構造体は、識別子とスタイルの名前を含む。

```
typedef struct tagSTYLEBUF {
      DWORD dwStyle;
      TCHAR szDescription[32]
} STYLEBUF;
```

メンバー

メンバーの名前	説明
dwStyle	登録単語のスタイルを指定する。
szDescription	このスタイルの説明文字列。

補注

IME_REGWORD_STYLE_EUDC を含む登録文字列のスタイルである。この文字列は、EUDC 範囲:

IME_REGWORD_STYLE_USER_FIRST から IME_REGWORD_STYLE_USER_LAST までである。
定数は、IME_REGWORD_STYLE_USER_FIRST から IME_REGWORD_STYLE_USER_LAST までの範囲で、プライベートな IME ISV スタイルのためにある。IME ISV は自分自身のスタイルを自由に定義できる。

SOFTKBDDATA 構造体

SOFTKBDDATA 構造体は、各仮想キーに対する DBCS コードを定義する。

```
typedef struct tagSOFTKBDDATA {
      UINT uCount;
      WORD wCode[][256]
} SOFTKBDDATA;
```

メンバー

メンバーの名前	説明
uCount	内部コード配列への 256 ワードの仮想キーマッピングの個数。
wCode[][256]	内部コード配列への 256 ワードの仮想キーマッピング。これは 1 個の 256 ワードの配列より多いかもしれない。

補注

1 種類のソフトキーボードに対して 2 つの 256 ワード配列を使うことはあり得る。一つは非シフト状態、もう一つは、シフト状態である。ソフトキーボードは 1 個の仮想キーを表示するのに2つの内部コードを使うことができる。

RECONVERTSTRING 構造体

RECONVERTSTRING 構造体は、IME の再変換のための文字列たちを定義する。それは再変換のための文字列たちを含むメモリブロックにおいて最初の項目である。

```
typedef struct _tagRECONVERTSTRING {
      DWOPD dwSize;
      DWORD dwVersion;
      DWORD dwStrLen;
      DWORD dwStrOffset;
      DWORD dwCompStrLen;
      DWORD dwCompStrOffset;
      DWORD dwTargetStrLen;
      DWORD dwTargetStrOffset;
} RECONVERTSTRING;
```

メンバー

メンバーの名前	説明
dwSize	この構造体のメモリーブロックのサイズ。
dwVersion	システムで予約済み。ゼロでなければならない。
dwStrLen	コンポジション文字列を含む文字列の長さ。
dwStrOffset	この構造体の開始位置からのオフセット。再変換単語を含む文字列はここに格納される。
dwCompStrLen	コンポジション文字列になる文字列の長さ。
dwCompStrOffset	コンポジション文字列になる文字列のオフセット。
dwTargetStrLen	コンポジション文字列においてターゲットの文節に関連する文字列の長さ。
dwTargetStrOffset	コンポジション文字列においてターゲットの文節に関連する文字列のオフセット。

補注

RECONVERTSTRING 構造体は、Windows 98 と Windows 2000 で新しく登場した。dwCompStrOffset と dwTargetOffset メンバーは、dwStrOffset の相対的な位置となる。Windows NT Unicode については、dwStrLen、dwCompStrLen、そして dwTargetStrLen は TCHAR の個数で、dwStrOffset、dwCompStrOffset、

そして dwTargetStrOffset はバイトオフセットである。

コメント

SCS_SETRECONVERTSTRING と SCS_QUERYRECONVERTSTRING をつけて ImmSetCompositionString を呼ぶことによって、アプリが再変換プロセスを開始したら、アプリは、コンポジション文字列バッファと同様にこの構造体に対する必要なメモリーを確保する責任がある。IME は後になってこのメモリを使うべきではない。もし IME がプロセスを開始したら、IME は構造体とコンポジション文字列バッファのために必要なメモリーを確保すべきである。

IMEMENUITEMINFO 構造体

IMEMENUITEMINFO 構造体は、IME メニュー項目たちに関する情報を含む。

```
typedef _tagIMEMENUITEMINFO {
      UINT cbSize;
      UINT fType;
      UINT fState;
      UINT wID;
      HBITMAP hbmpChecked;
      HBITMAP hbmpUnchecked;
      DWORD dwItemData;
      TCHAR szString[48];
      HBITMAP hbmpItem;
} IMEMENUITEMINFO;
```

メンバー

<table>
<tr><th>メンバーの名前</th><th>説明</th></tr>
<tr><td>cbSize</td><td>構造体のサイズ。</td></tr>
<tr><td>fType</td><td>メニュー項目の種類。次の 1 個以上の値を指定できる。
<table>
<tr><th>値</th><th>意味</th></tr>
<tr><td>IMFT_RADIOCHECK</td><td>hbmpChecked が NULL であれば、チェックマークではなくラジオボタンマークを使ってメニュー項目をチェックする。</td></tr>
<tr><td>IMFT_SEPARATOR</td><td>メニュー項目が区分線であることを指定する。区分線は、横線として表示される。hbmpItem と szString は無視される。</td></tr>
<tr><td>IMFT_SUBMENU</td><td>メニュー項目がサブメニューであることを指定する。</td></tr>
</table></td></tr>
<tr><td>fState</td><td>メニュー項目の状態。次の 1 個以上の値を指定できる。
<table>
<tr><th>値</th><th>意味</th></tr>
<tr><td>IMFS_CHECKED</td><td>メニュー項目にチェックをつける。hbmpChecked を参照。</td></tr>
<tr><td>IMFS_DEFAULT</td><td>既定のメニュー項目であることを指定する。既定のメニュー項目は太字で表示される。</td></tr>
<tr><td>IMFS_DISABLED</td><td>メニュー項目を無効にする。ただし、灰色にはなら</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>メンバーの名前</th><th>説明</th></tr>
<tr><td></td><td>
<table>
<tr><th>値</th><th>意味</th></tr>
<tr><td></td><td>ない。灰色にしたいときは IMFS_GRAYED を指定しなければならない。</td></tr>
<tr><td>IMFS_ENABLED</td><td>メニュー項目を有効にする。これが既定の状態。</td></tr>
<tr><td>IMFS_GRAYED</td><td>メニュー項目を無効にし、灰色にする。</td></tr>
<tr><td>IMFS_HILITE</td><td>メニュー項目をハイライトにする。</td></tr>
<tr><td>IMFS_UNCHECKED</td><td>メニュー項目のチェックを解除する。
hbmpUnchecked を参照。</td></tr>
<tr><td>IMFS_UNHILITE</td><td>メニュー項目のハイライトを解除する。これが既定の状態。</td></tr>
</table>
</td></tr>
<tr><td>wID</td><td>メニュー項目を識別するために、アプリ定義の 16 ビットの値。</td></tr>
<tr><td>hbmpChecked</td><td>項目がチェックされたときに隣に表示するビットマップのハンドル。NULL ならば、既定のビットマップが使われる。種類が IMFT_RADIOCHECK なら、既定のビットマップは弾丸になる。それ以外なら、既定のビットマップはチェックマークになる。</td></tr>
<tr><td>hbmpUnchecked</td><td>項目がチェックされていないときに隣に表示するビットマップのハンドル。NULL ならビットマップを使わない。</td></tr>
<tr><td>dwItemData</td><td>メニュー項目に関連付けられたアプリ定義の値。</td></tr>
<tr><td>szString</td><td>メニュー項目の中身。これはナル終端の文字列。</td></tr>
<tr><td>hbmpItem</td><td>表示するビットマップハンドル。</td></tr>
</table>

注記

IMEMENUITEMINFO 構造体は Windows 98 と Windows 2000 で登場した新しい構造体である。Unicode 版は szString に WCHAR を使う。

TRANSMSG 構造体

TRANSMSG 構造体は、IME 生成のメッセージを受け取るために ImeToAsciiEx で使われる、転送メッセージを含む。

```
typedef _tagTRANSMSG {
      UINT message;
      WPARAM wParam;
      LPARAM lParam;
} TRANSMSG;
```

メンバー

メンバーの名前	説明
message	メッセージ識別子を指定する。
wParam	このメッセージに関する追加的な情報を指定する。この正確な意味は、message メンバーの値に依存する。
lParam	このメッセージに関する追加的な情報を指定する。この正確な意味は、message

メンバーの名前	説明
	メンバーの値に依存する。

注記

この構造体は、未来の 64 ビット Windows のために追加された。この構造体は、以前に用いられた LPDWORD lpdwTransBuf を置き換えるために、ImeToAsciiEx によって、TRANSMSGLIST 構造体と一緒に用いられるだろう。この構造体を使うことで、同じオフセットでメモリーのデータを保持し、下位互換性を保つ。

TRANSMSGLIST 構造体

TRANSMSGLIST 構造体は、ImeToAsciiEx から返却された転送メッセージリストを含む。

```
typedef _tagTRANSMSGLIST {
      UINT uMsgCount;
      TRANSMSG TransMsg[1];
} TRANSMSGLIST;
```

メンバーの名前	説明
uMsgCount	TransMsg 配列のメッセージの個数。
TransMsg	メッセージを表す TRANSMSG 構造体の配列。

注記

この構造体は、未来の 64 ビット Windows のために追加された。この構造体は、以前に用いられた LPDWORD lpdwTransBuf を置き換えるために、ImeToAsciiEx によって、TRANSMSGLIST 構造体と一緒に用いられるだろう。この構造体を使うことで、同じオフセットでメモリーのデータを保持し、下位互換性を保つ。