Win32 Multilingual IME Overview for IME Development

Version 1.32

04-01-1998

This documentation introduces the basics on how to develop an IME for Windows® 95, Windows® 98, and Windows NT®/Windows® 2000. It is also a supplement to the Win32® Multilingual IME API reference for IME development

The following main topics are discussed:

Overview

Beginning with Windows 95 and Windows NT® 4.0, Input Method Editors (IMEs) are provided as a dynamic-link library (DLL), in contrast to the IMEs for the Windows 3.1 Far East Edition. Each IME runs as one of the multilingual keyboard layouts. In comparison to the Windows 3.1 IME, the new Win32 Multilingual Input Method Manager (IMM) and Input Method Editor (IME) architecture provide the following advantages:

- Run as a component of the multilingual environment
- Offer multiple Input Contexts for each application task
- Keep one active IME per each application thread
- Give information to the application through message looping (no message order broken)

- Offer strong support for both IME-aware and IME-unaware applications

To fully utilize these advantages, an application needs to support the new Win32 IMM/IME application interface.

In order to maintain maximum compatibility with existing Windows 95 and Windows NT 4.0 IMEs, Windows 98 and Windows 2000 have not changed significantly in design. However, new features have been added in order to provide better system integration and to support more intelligent IMEs.

Note
IME developers must use the immdev.h in DDK, which is a superset of the imm.h in the SDK or other development tools.

Windows 98 and Windows 2000 IMM/IME

The Windows 98 and Windows 2000 IMM/IME architecture retains the Windows 95 and Windows NT 4.0 design. However, some changes have been made in order to support intelligent IME development and integration of the IME with Windows. These changes include:

- New IME functions that allow applications to communicate with the IMM/IME. These include:

  ImmAssociateContextEx

  ImmDisableIME

  ImmGetImeMenuItems
- New functions that allow the IME to communicate with IMM and applications. These include:

  ImmRequestMessage

  ImeGetImeMenuItems
- Supporting reconversion

  This is a new IME feature that allows you to reconvert a string that has already been inserted into the application's document. This function will help intelligent IMEs to get more information about the converted result and improve conversion accuracy and performance. This feature requires that the application and the IME cooperate.
- Adding IME menu items to the Context menu of the System Pen icon

  This new feature provides a way for an IME to insert its own menu items into the Context menu of the System Pen icon in the system bar and in applications.
- New bits and flags for the IME

  The following new bits support new conversion modes:

IME_CMODE_FIXED

IME_SMODE_CONVERSATION

IME_PROP_COMPLETE_ON_UNSELECT

- Edit control enhancement for the IME

  Through two new edit control messages, EM_SETIMESTATUS and EM_GETIMESTATUS, applications can manage IME status for edit controls.

- Changing IME Pen Icon and Tooltips

  Through three new messages, INDICM_SETIMEICON, INDICM_SETIMETOOLTIPS, and INDICM_REMOVEDEFAULTMENUITEMS, IME can change the system Pen Icon and Tooltips in the system task bar.

- Two new IMR messages

  IMR_QUERYCHARPOSITION and IMR_DOCUMENTFEED help the IME and application to communicate position and document information.

Win32 IME Structure

A new Win32 IME has to provide two components. One is the IME Conversion Interface and the other is the IME User Interface. The IME Conversion Interface is provided as a set of functions that are exported from the IME module. These functions are called by the IMM. The IME User Interface is provided in the form of windows. These windows receive messages and provide the user interface for the IME.

IME Aware Applications

One of the main advantages of the new Win32 IME architecture is that it provides better communication logic between the application and the IME. Following is an example of how an application could be involved with the IME:

- IME Unaware Applications

  These kinds of applications never intend to control the IME. However, as long as it accepts DBCS characters, a user can type any DBCS character to the application using IME.

- ME Half-aware Applications

  These kinds of applications typically control the various contexts of the IME, such as open and close, and composition form, but it does not display any user interface for the IME.

- IME Full-aware Applications

These kinds of applications typically want to be fully responsible for displaying any information given by the IME.

In Windows 95 and Windows NT 4.0 or later, one IME unaware application will be supported with one Default IME window and one Default Input Context.

An IME half-aware application will create its own IME window(s), also called an application IME window, using a predefined system IME class, and may or may not handle its own Input Context given to the application.

An IME fully aware application will handle the Input Context by itself and will display any necessary information given by the Input Context not using the IME window.

IME User Interface

The IME User Interface includes the IME window, the UI window, and the components of the UI window.

Features

An IME class is a predefined global class that carries out any user interface portion of the IME. The normal characteristics of an IME class are the same as with other common controls. Its window instance can be created by **CreateWindowEx**. As with static controls, the IME class window does not respond to user input by itself, but receives various types of control messages to realize the entire user interface of the IME. An application can create its own IME window(s) by using this IME class or by obtaining the Default IME window through **ImmGetDefaultIMEWnd**. In comparison to Windows 3.1, an application that wants to control the IME with these window handles (an IME-aware application) can now achieve the following benefits:

- The new IME includes candidates listing windows. Each application can have its own window instance of the UI so a user can stop in the middle of any operation to switch to another application. In the Windows 3.1 Japanese Edition, the user had to first exit an operation before switching to another application.
- Since the IME User Interface window is informed about an application's window handle, it can provide several default behaviors for the application. For example, this can include automatic repositioning of the IME window, automatic tracing of the window caret position, and mode indication for each application.

Even though the system provides only one IME class, there are two kinds of IME window. One is created by the system for the **DefWindowProc** function especially for an IME unaware program. The IME User Interface for the **DefWindowProc** function is shared by all IME unaware windows of a thread and is called the default IME window in this documentation. The other windows are created by IME aware applications and are called the application IME window.

Default and Application IME Window

The system creates a default IME window at thread initialization time, which is given to a thread automatically. This window then handles any IME user interface for an IME unaware application.

When the IME or IMM generates WM_IME_xxx messages, an IME unaware application passes them to **DefWindowProc**. Then, **DefWindowProcB** sends necessary messages to the default IME window, which provides default behavior of the IME User Interface for an unaware application. An IME aware application also uses this window when it does not hook messages from the IME. An application can use its own application IME window when it is necessary.

IME Class

The Win32 systems provides an IME class in the system. This class is defined by the user just as the predefined Edit class is. The system IME class handles the entire UI of the IME and handles entire control messages from the IME and application, including IMM functions. An application can create its own IME User Interface by using this class. The system IME class, itself, is not replaced by any IME, but is kept as a predefined class.

This class has a window procedure that actually handles the **WM_IME_SELECT** message. This message has the *hKL* of the newly selected IME. The system IME class retrieves the name of the class defined by each IME with this *hKL*. Using this name, the system IME class creates a UI window of the currently active IME.

UI Class from IME

In this design, every IME is expected to register its own UI class for the system. The UI class provided by each IME should be responsible for IME-specific functionality. The IME may register the classes that are used by the IME itself when the IME is attached to the process. This occurs when **DllEntry** is called with DLL_PROCESS_ATTACH. The IME then has to set

the UI class name in the *lpszClassName* parameter, which is the second parameter of **ImeInquire**.

The UI class should be registered with CS_IME specified in the style field so every application can use it through the IME class. The UI class name (including the null terminator) can consist of up to 16 characters and may be increased in future versions.

The *cbWndExtra* of the UI class has to be 2 * sizeof(LONG). The purpose of this *WndExtra* is defined by the system (for example, IMMGWL_IMC and IMMGWL_PRIVATE).

The IME can register any class and create any window while working in an application.

The following sample shows how to register the IME User Interface Class:

```
BOOL WINAPI DLLEntry (
    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   = (LPSTR)NULL;
wc.lpszClassName  = (LPSTR)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   = (LPSTR)NULL;
wc.lpszClassName  = (LPSTR)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 Window

The IME windows of the IME class are created by the application or by the system. When the IME window is created, the UI window provided by the IME itself is created and owned by the IME window.

Each UI window contains the current Input Context. This Input Context can be obtained by calling **GetWindowLong** with IMMGWL_IMC when the UI window receives a WM_IME_xxx message. The UI window can refer to this Input Context and handles the messages. The Input Context from **GetWindowLong** with IMMGWL_IMC is available at any time during the UI window procedure, except when handling a WM_CREATE message.

The *cbWndExtra* of the UI windows cannot be enhanced by the IME. When the IME needs to use the extra byte of the window instance, the UI window uses **SetWindowLong** and **GetWindowLong** with IMMGWL_PRIVATE. This IMMGWL_PRIVATE provides a LONG value extra of the window instance. When the UI window needs more than one LONG value extra for private use, the UI window can place a handle for a memory block into the IMMGWL_PRIVATE area.The UI window procedure can use **DefWindowProc**, but the UI window cannot pass a WM_IME_xxx message to **DefWindowProc**. Even if the message is not handled by the UI window procedure, the UI window does not pass it to **DefWindowProc**.

The following sample demonstrates how to allocate and use a block of private memory:

```
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);
    }
}
```

The UI window must perform all tasks by referring to the Input Context that is currently selected. When a window of an application is activated, the UI window receives a message that contains the current Input Context. The UI window then uses that Input Context. Thus, the Input Context must contain all the information needed by the UI window to display the composition window, the status window, and so forth.

The UI window refers to the Input Context, but does not need to update it. However, if the UI window wants to update the Input Context, it should call the IMM functions. Because the Input Context is managed by the IMM, the IMM along with the IME should be notified when the Input Context is changed.

For example, the UI window occasionally needs to change the conversion mode of the Input Context when the user clicks the mouse. At this point, the

UI window should call **ImmSetConversionMode**. The **ImmSetConversionMode** function creates a notification for *NotifyIME* and the UI window with WM_IME_NOTIFY. If the UI window wants to change the display of the conversion mode, the UI window should wait for a **WM_IME_NOTIFY** message.

Components of the UI Window

The UI window can register and show the composition window and the status window by referring to the current Input Context. The class style of the components of the UI window must include the CS_IME bit. A window instance of the UI window gets information about the composition string, font, and position from the current Input Context.

When a window of the application is getting focused, the system gives the Input Context to this window and sets the current Input Context to the UI window. The system then sends a **WM_IME_SETCONTEXT** message with the handle of its Input Context to the application. The application then passes this message to the UI window. If current Input Context is replaced by another Input Context, the UI window should repaint the composition window. Any time the current Input Context is changed, the UI window displays a correct composition window. Thus, the status of the IME is assured.

A UI window can create a child window or pop-up window to display its status, composition string, or candidate lists. However, these windows have to be owned by the UI window and created as disabled windows. Any windows that are created by the IME should not get the focus.

IME Input Context

Each window is associated with an IME Input Context. The IMM uses the Input Context to maintain IME status, data, and so forth, and communicate with the IME and with applications.

Default Input Context

By default, the system creates a default Input Context for each thread. All IME unaware windows of the thread share this context.

Application-Created Input Context

A window of an application can associate its window handle to an Input Context to maintain any status of the IME, including an intermediate

composition string. Once an application associates an Input Context to a window handle, the system automatically selects the context whenever the window is activated. In this way, an application is free from complicated in and out focus processing.

Using the Input Context

When an application or system creates a new Input Context, the system prepares the new Input Context with the components of the IMC (IMCC). These include *hCompStr*, *hCandInfo*, *hGuideLine*, *hPrivate,* and *hMsgBuf*. Basically, the IME does not need to create the Input Context and the components of the Input Context. The IME can change the size of them and lock them to get the pointer for them.

Accessing the HIMC

When an IME accesses the Input Context, the IME has to call **ImmLockIMC** to get the pointer of the Input Context. **ImmLockIMC** increments the IMM lock count for IMC, while **ImmUnlockIMC** decrements the IMM lock count for IMC.

Accessing the HIMCC

When an IME accesses a component of the Input Context, the IME has to call **ImmLockIMCC** to get the pointer of the IMCC. **ImmLockIMCC** increments the IMM lock count for IMCC, while **ImmUnlockIMC** decrements the IMM lock count for IMCC. **ImmReSizeIMCC** can resize the IMCC to the size specified in the input parameter.

On occasion, an IME needs to create a new component in the Input Context. The IME can call **ImmCreateIMCC** to do so. To destroy a newly created component in the Input Context, the IME can call **ImmDestroyIMCC**.

The following example shows how to access the IMCC and change the size of a component:

```
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);
}
```

Generating Messages

IMEs need to generate IME messages. When an IME initiates the conversion process, the IME has to generate a **WM_IME_STARTCOMPOSITION** message. If the IME changes the composition string, the IME has to generate a **WM_IME_COMPOSITION** message.

There are two ways an IME can generate a message: one is by using the *lpdwTransKey* buffer provided by **ImeToAsciiEx**, and the other is by calling **ImmGenerateMessage**.

Using *lpdwTransBuf* to Generate Messages

Events initiated by IMEs are realized as generating messages to the window associated with the Input Context. Basically, IMEs use the *lpdwTransKey* buffer provided by the parameter of **ImeToAsciiEx** to generate the message. The IMEs put the messages into the *lpdwTransKey* buffer when **ImeToAsciiEx** is called.

The buffer specified by *lpdwTransBuf* in the **ImeToAsciiEx** function is provided by the system. This function can place messages in this buffer all at one time. The real number of messages that can be placed is given at the first double word of the buffer. However, if the **ImeToAsciiEx** function

wants to generate more messages than the given number, **ImeToAsciiEx** can put all the messages into *hMsgBuf* in the Input Context and then return the number of messages.

When the return value of **ImeToAsciiEx** is larger than the specified value in *lpdwTransBuf*, the system does not pick up the messages from *lpdwTransBuf*. Instead, the system looks up *hMsgBuf* in the Input Context, which is passed as a parameter of **ImeToAsciiEx**.

Following is the code sample for **ImeToAsciiEx** implementation:

```
UINT
    ImeToAsciiEx(
    uVirKey,
    uScanCode,
    lpbKeyState,
    lpdwTransBuf,
    fuState,
    hIMC
    )
{
    DWORD dwMyNumMsg = 0;

. . .

    // Set the messages that the IME wants to generate.
    *lpdwTransBuf++ = (DWORD)msg;
    *lpdwTransBuf++ = (DWORD)wParam;
    *lpdwTransBuf++ = (DWORD)lParam;

    // Count the number of the messages that the IME wants to
generate.
    dwMyNumMsg++;

. . .

    return dwMyNumMsg;
}
```

Using the Message Buffer to Generate Messages

Even if **ImeToAsciiEx** is not called, IMEs can still generate the message to the window associated with the Input Context by using the message buffer of the Input Context. This message buffer operates as a handle of a memory block and the IME puts the messages into this memory block. The

IME then calls the **ImmGenerateMessage** function, which sends the messages stored in the message buffer to the proper window.

Following is the code sample for **ImmGenerateMessage** implementation.

```
MyGenerateMesage(HIMC hIMC, UINT msg, WPARAM wParam, LPARAMlParam)
{
    LPINPUTCONTEXT lpIMC;
    HGLOBAL hTemp;
    LPDWORD lpdwMsgBuf;
    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(DWORD) * 3);
if (!hTemp)
// Error!
lpIMC->hMsgBuf = hTemp;
// Lock the memory for the message buffer.
lpdwMsgBuf = ImmLockIMCC(lpIMC->hMsgBuf);
if (!lpdwMsgBuf)
// Error!
lpdwNumMsgBuf += 3 * lpIMC->dwNumMsgBuf.
// Set the number of the messages.
lpIMC->dwNumMsgBuf += dwMyNumMsg;

// Set the messages that the IME wants to generate.
*lpdwMsgBuf++ = (DWORD)msg;
*lpdwMsgBuf++ = (DWORD)wParam;
*lpdwMsgBuf++ = (DWORD)lParam;

// Unlock the memory for the message buffer and the Input Context.
ImmUnlockIMCC(lpIMC->hMsgBuf);
ImmLockIMC(hIMC);

// Call ImmGenerateMessage function.
ImmGenerateMessage(hIMC);
}
```

WM_IME_COMPOSITION Message

When an IME generates a **WM_IME_COMPOSITION** message, the IME specifies *lParam* as the GCS bits. The GCS bits then inform the available members of the **COMPOSITIONSTRING** structure. Even if the IME does not update and a member is available, the IME can set the GCS bits.

When an IME generates a **WM_IME_COMPOSITION** message, the IME can also change the string attribute and clause information all at once.

ImeSetCompositionString

The **ImeSetCompositionString** function is used by applications to manipulate the IME composition string. By specifying different flags, an application can change composition string, attribute, clause, and so forth.

The second parameter of this function, *dwIndex*, specifies how the composition string should be adjusted in an IME. It includes values such as SCS_SETSTR ,SCS_CHANGEATTR, SCS_CHANGECLAUSE, SCS_QUERYRECONVERTSTRING. Each value represents a specific feature.

ImeSetCompositionString Capability

When an IME does not have the capability of **ImeSetCompositionString**, it will not specify any SCS capability in the **IMEINFO** structure. When an IME can handle **ImeSetCompositionString**, it sets the SCS_COMPSTR bit. If an IME can generate the reading string from the composition string, it will set the SCS_CAP_MAKEREAD bit.

If an IME has SCS_CAP_COMPSTR capability, **ImeSetCompositionString** will be called. In response to this call, the IME should use the new composition string generated by an application and then generate a **WM_IME_COMPOSITION** message.

SCS_SETSTR

If *dwIndex* of **ImeSetCompositionString** is SCS_SETSTR, the IME can clean up all the **COMPOSITIONSTR** structures of *hIMC*.

If necessary, the IME can also update the candidate information and generate the candidate message **WM_IME_NOTIFY** with the submessage as IMN_OPENCANDIDATE, CHANGECANDIDATE, or IMN_CLOSECANDIDATE.

An IME needs to respond to the application requirement based on different input parameters as follows:

- If the *lpRead* parameter of **ImeSetCompositonString** is available:

The IME should create the composition string from the reading string contained in *lpRead*. The IME then creates the attribute and clause information for both the new composition string and reading string of *lpRead*. The IME generates a **WM_IME_COMPOSITION** message with either GCS_COMP or GCS_COMPREAD. On occasion, an IME may finalize the conversion automatically. The IME can then generate a **WM_IME_COMPOSITION** message with either GCS_RESULT or GCS_RESULTREAD instead of GCS_COMPxxx.

- If the *lpComp* parameter of **ImeSetCompositonString** is available:

  The IME should create the composition attribute and clause information from the composition string contained in *lpComp*. The IME then generates a **WM_IME_COMPOSITION** message with GCS_COMP. If the IME has the capability of SCS_CAP_MAKEREAD, the IME should also make the new reading string at the same time. The IME then generates a **WM_IME_COMPOSITION** message with either GCS_COMP orGCS_COMPREAD. On occasion, an IME may finalize the conversion automatically. The IME can then generate a **WM_IME_COMPOSITION** message with either GCS_RESULT or GCS_RESULTREAD instead of GCS_COMPxxx.

- If both *lpRead* and *lpComp* are available:

  The IME should create the composition string and the reading string accordingly. In this case, the IME does not need to follow *lpComp* and *lpRead* completely. If an IME cannot find the relation between *lpRead* and lpComp specified by the application, it should correct the composition string. The IME can then create the attribute and clause information for both the new composition string and reading string of *lpRead*. The IME then generates a **WM_IME_COMPOSITION** message with either GCS_COMP or GCS_COMPREAD. On occasion, the IME may finalize the conversion automatically. The IME can then generate a **WM_IME_COMPOSITION** message with either GCS_RESULT or GCS_RESULTREAD instead of GCS_COMPxxx.

SCS_CHANGEATTR

SCS_CHANGEATTR effects only attribute information. The IME should not update the composition string, the clause information of the composition string, the reading of the composition string, or the clause information of the reading of the composition string.

The IME first has to check whether the new attribute is acceptable or not. It then sets the new attribute in the **COMPOSITIONSTRING** structure of *hIMC*. Last, the IME generates a **WM_IME_COMPOSITION** message.

If necessary, the IME can update the candidate information and generate the candidate messages **WM_IME_NOTIFY** with the submessage as

IMN_OPENCANDIDATE, CHANGECANDIDATE, or IMN_CLOSECANDIDATE of WM_IME_NOTIFY.

For this feature, an IME cannot finalize the composition string.

An IME needs to respond to the application requirement based on different input parameters as follows:

- If the *lpRead* parameter of **ImeSetCompositonString** is available:

  The IME should follow the new attribute in *lpRead* and then create a new attribute of the composition string for the current composition string. In this case, the clause information does not change.

  The IME generates a **WM_IME_COMPOSITION** message with either GCS_COMP or GCS_COMPREAD. If an IME cannot accept the new attribute contained in *lpRead*, it does not generate any messages and returns FALSE.
- If the *lpComp* parameter of **ImeSetCompositonString** is available:

  The IME follows the new attribute in *lpComp*. In this case, the clause information does not change.

  If the capability of the IME has SCS_CAP_MAKEREAD and the reading string is available, the IME should create a new attribute of the reading of the composition string for the current reading of the composition string.
- If both *lpRead* and *lpComp* are available:

  If the IME can accept the new information, it sets the new information in the **COMPOSITION** structure of *hIMC* and generates a **WM_IME_COMPOSITION** message with either GCS_COMP or GCS_COMPREAD.

SCS_CHANGECLAUSE

SCS_CHANGECLAUSE effects the string and attribute for both the composition string and the reading of the composition string.

If necessary, an IME can update the candidate information and generate the candidate message **WM_IME_NOTIFY** with the submessages as IMN_OPENCANDIDATE, CHANGECANDIDATE, or IMN_CLOSECANDIDATE.

An IME needs to respond to the application requirement based on different input parameters. Following are the instances in which an IME cannot finalize the composition string.

- If the *lpRead* parameter of **ImeSetCompositonString** is available:

  The IME follows the new reading clause information of *lpRead* and has to correct the attribute of the reading of the composition string. The IME can then update the composition string, the attribute, and the clause

information of the composition string. The IME generates a **WM_IME_COMPOSITION** message with either GCS_COMP or GCS_COMPREAD.

- If the *lpComp* parameter of **ImeSetCompositonString** is available:

  The IME follows the new clause information and has to correct the composition string and the attribute of the composition string. Then, the IME can update the reading attribute and the clause information of the reading attribute. The IME generates a **WM_IME_COMPOSITION** message with either GCS_COMPSTR or GCS_COMPREAD.

- If both *lpRead* and *lpComp* are available:

  If the IME can accept the new information, it sets the new information in the **COMPOSITION** structure of *hIMC* and generates a **WM_IME_COMPOSITION** message with either GCS_COMP or GCS_COMPREAD.

Soft Keyboard

Soft Keyboard is a special window displayed by the IME. Because some IMEs have special reading characters, an IME can provide a soft keyboard to display these special reading characters. In this way, the user does not have to remember the reading character for each key. For example, an IME can use bo po mo fo for its reading characters, while another IME can use radicals for its reading characters.

An IME can change the reading characters of keys and notify the user of these key changes, depending on the conversion state. For example, during candidate selection time, an IME can show just the selection keys to the user.

An IME can also create its own user interface for a soft keyboard or use the system predefined soft keyboard. If an IME wants to use the system predefined soft keyboard, it needs to specify UI_CAP_SOFTKBD in the **fdwUICaps** member of the **IMEINFO** structure when **ImeInquire** is called.

An IME needs to call **ImmCreateSoftKeyboard** to create the soft keyboard window . It can also call **ImmShowSoftKeyboard** to show or hide the soft keyboard. Because the soft keyboard window is one component of the UI window, the owner should be the UI window as well.

There are different types of soft keyboards. Each one is designed for a specific country or special purpose. An IME can change the reading characters on the soft keyboard by using one of two methods: IMC_SETSOFTKBDSUBTYPE or IMC_SETSOFTKBDDATA

Reconversion

Reconversion is a new IME function for Windows 98 and Windows 2000. It provides the capability to reconvert a string that is already inserted in an application's document. Specifically, whatever the string is, an IME is instructed to recognize the string, convert it to the reading or typing information, and then display the candidate list.

New and advanced intelligent IMEs are capable of recognizing and interpreting a complete sentence. When an IME is supplied with better information about a string, such as full sentence and string segmentation, it can accomplish better conversion performance and accuracy. For example, by supplying an entire sentence as opposed to just the reconverted strings, the IME can reconvert the string without having to first convert to reading or typing information.

The **RECONVERTSTRING** structure can store the entire sentence and point to the string that will be reconverted by *dwStartOffset* and *dwLen*. If *dwStartOffset* is zero and *dwLen* is the length of the string, the entire string is reconverted by the IME.

Simple Reconversion

The simplest reconversion is when the target string and the composition string are the same as the entire string. In this case, *dwCompStrOffset* and *dwTargetStrOffset* are zero, and *dwStrLen*, *dwCompStrLen*, and *dwTargetStrLen* are the same value. An IME will provide the composition string of the entire string that is supplied in the structure, and will set the target clause by its conversion result.

Normal Reconversion

For an efficient conversion result, the application should provide the **RECONVERTSTRING** structure with the information string. In this case, the composition string is not the entire string, but is identical to the target string. An IME can convert the composition string by referencing the entire string and then setting the target clause by its conversion result.

Enhanced Reconversion

Applications can set a target string that is different from the composition string. The target string (or part of the target string) is then included in a target clause in high priority by the IME. The target string in the **RECONVERTSTRING** structure must be part of the composition string.

When the application does not want to change the user's focus during the reconversion, the target string should be specified. The IME can then reference it.

IME Cancel Reconversion

When a user cancels the composition string generated by the reconversion, the IME should determine the original reconverted string. Otherwise, the application will loose the string.

SCS_SETRECONVERTSTRING and SCS_QUERYRECONVERTSTRING Flags

Applications can ask an IME to reconvert the string by calling **ImmSetCompositionString**. They can use either the SCS_SETSTR flag or the SCS_SETRECONVERTSTRING flag to create a new composition string. However, by using SCS_SETRECONVERTSTRING, the application can pass **RECONVERTSTRING** to the IME for better conversion efficiency.

Initially, the application should call **ImmSetCompositionString** with SCS_QUERYRECONVERTSTRING. The selected IME can then adjust the given **RECONVERTSTRING** structure for appropriate reconversion. The application then calls **ImmSetCompositionString** with SCS_SETRECONVERTSTRING to request that the IME generate a new composition string. If the application asks the IME to adjust the **RECONVERTSTRING** structure by calling SCS_QUERYRECONVERTSTRING, efficient reconversion can be accomplished.

IMR_RECONVERTSTRING and IMR_CONFIRMRECONVERTSTRING Messages

When an IME wants to reconvert, it can ask the application to provide the string to be reconverted. For example, when a user presses the **Reconversion** key or selects the **Reconversion** button in the IME status window, the IME sends a **WM_IME_REQUEST** message with IMR_RECONVERTSTRING to get the target string. Initially, the IME needs to send this with NULL *lParam* in order to get the required size of **RECONVERTSTRING**. The IME then prepares a buffer to receive the target string and sends the message again with the pointer of the buffer in *lParam*.

After the application handles IMR_RECONVERTSTRING, the IME may or may not adjust the **RECONVERTSTRING** structure given by the

application. Finally, the IME sends a **WM_IME_REQUEST** message with IMR_CONFIRMRECONVERTSTRING to confirm the **RECONVERTSTRING** structure.

If the application returns TRUE for IMR_CONFIRMRECONVERTSTRING, the IME generates a new composition string based on the **RECONERTSTRING** structure in the IMR_CONFIRMRECONVERTSTRING message. If the application returns FALSE, the IME generates a new composition string based on the original **RECONVERTSTRING** structure given by the application in the IMR_RECONVERTSTRING message. An IME will not generate a composition string for reconversion before IMR_CONFIRMRECONVERTSTRING

The given string should not be changed by SCS_QUERYRECONVERTSTRING or IMR_CONFIRMRECONVERTSTRING. An IME can change only **CompStrOffset**, **CompStrLen**, **TargetStrOffset**, and **TargetStrLen** to re-confirm it

IME Menu Functions

The purpose of this function set is to reduce the IME-related icon in the system task bar. It is a new feature for Windows 98 and Windows 2000.

The Windows system program installs two icons in the task bar when the current *hKL* is an IME. One icon is the System ML icon that indicates the current keyboard layout in the system task bar. The other is the System Pen icon that shows the IME status of the focused window. Usually, an IME places an additional icon in the task bar. The context menu for this icon is completely dependent on the IME. Having IME icons in a task bar is a quick way for a user to access an IME's special functions. However, there are three icons associated with the IME and these additional icons may be more than a user wants to deal with.

If the system provides a way for an IME to insert IME menu items into the System Pen icon, the IME then does not need to add its extra icons to the task bar.

The IMM calls the IME function **ImeGetImeMenuItems** to get the IME menu items.

An application can use **ImmGetImeMenuItems** to get an IME's special menu items, which it can add to its context menu. By calling **ImmNotify**, the selected items can be processed by the IME.

IME Menu Notification

When an application wants to handle an IME's menu items, it can call **ImmNotifyIME**. When the menu items added by the IME are selected, **NotifyIME** is called under the focused thread.

IME Help File

The IME help file is a new function added into Windows 98 and Windows NT/Windows 2000. The right click menu of the System Pen Icon has two menu items. One is the setting for the IME system and is used to change the setting of the selected IME of the focus thread. The other menu item is an online Help file that has never been enabled. Thus, this menu item is always grayed. The purpose of this menu item was to display an IME's online Help. However, because the system does not provide the IME with a way to specify the name of the IME help file, the system task bar program is not able to display it.

IME_ESC_GETHELPFILENAME

The IME_ESC_GETHELPFILENAME escape allows an IME to specify its help file name. On return from the function, the (LPTSTR)lpData is the full path file name of an IME's help file. The path name should be less than 80 * sizeof(TCHAR).

Indicator Manager for IME

By using a set of messages defined in Windows 98 and Windows 2000, an IME can change the icon and ToolTip string for the System Pen icon on the system task bar.

Indicator Window

An IME can get the window handle of the indicator by using **FindWindow** with INDICATOR_CLASS.

```
// 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);
```

Note
Due to the internal design requirement in the task bar manager, the IME must use **PostMessage** for INDICM_xxx messages.

Messages

IMEs can send the following messages **‹MAKE INTO JUMPS›**to the Indicator window to perform different tasks:

INDICM_SETIMEICON

INDICM_SETIMETOOLTIPS

INDICM_REMOVEDEFAULTMENUITEMS

Windows NT/Windows 2000 Issues

The following topics primarily describe the special issues related to Windows NT/Windows 2000. However, some of these issues may also apply to Windows 98.

IME and Localized Language Compatibility

Windows 2000 has full-featured IME support in any localized language version. That is, an IME can be installed and used with any Windows 2000 language. IME developers should test their IME on these environments. This new feature also requires IME developers to prepare their IME help content to include correct charset and font information so it shows up correctly on different language operating systems.

Also, IME developers should develop Unicode IME for Windows 2000. Unicode IMEs will work with Unicode applications under any system locale. For non-Unicode IMEs, the user must change the system locale to support the same language that the IME supports in order to use them.

Unicode Interface

Along with the ANSI version of the IMM/IME interface originally supported by Windows 95, Windows NT/Windows 2000 and Windows 98 support Unicode interface for the IME. To communicate with the system by Unicode interface, an IME should set the IME_PROP_UNICODE bit of the **fdwProperty** member of the **IMEINFO** structure, which is the first parameter of the **ImeInquire** function. Although **ImeInquire** is called to

initialize the IME for every thread of the application process, the IME is expected to return the same **IMEINFO** structure on a single system. Windows 98 supports all the Unicode functions, except for **ImmIsUIMessage**.

Security Concerns

There are two primary security concerns for Windows NT/Windows 2000. One involves named objects and the other involves Winlogon.

Named Objects

An IME may want to create various named objects that should be accessed from multiple processes on the local system. Such objects may include file, mutex, or event. Since a process might belong to a different user who is interactively logging onto the system, the default security attribute (created by the system when an IME creates an object with the NULL parameter as the pointer to the security attribute) may not be appropriate for all processes on the local system.

On Windows NT/Windows 2000, the first client process of the IME may be the Winlogon process that lets a user log onto the system. Since the Winlogon process belongs to the system account during the log-on session and is alive until the system shuts down, named objects created by the IME with the default security attribute during the log-on session cannot be accessed through other processes belonging to a logged-on user.

A sample IME source code that creates the security attribute appropriated for named objects is provided in the Microsoft Platform DDK. By using the sample code, IME writers can create various named objects that can be accessed from all client processes of the IME on the local system. The security attribute allocated by the sample code is per process. An IME that frequently creates named objects may want to initialize the security attribute attach time and free the security attribute at the process detach time. An IME that does not create named objects often may want to initialize the security attribute just before the creation of the named object and then free the security attribute just after the object is created.

Winlogon

Since a user in the log-on session has not been granted the access right to the system yet, information provided by an IME's configuration dialog boxes can create security problems. Even though, the system administrator can configure the system so such an IME cannot be activated on the log-on session. A well-behaved IME should not allow users to open configure dialog boxes if the client process is a Winlogon process. An IME can check

if the client process executing a log-on session is a Winlogon process by checking the IME_SYSTEMINFO_WINLOGON bit of the *dwSystemInfoFlags* parameter of **ImeInquire**.

IME File Format and Data Structures

The following topics discuss the IME file format and data structures used by the IME.

IME File Format

An IME needs to specify the following fields correctly in the version information resource. This includes the fixed file information part and the variable length information part. Please refer to the Microsoft Platform SDK for detailed information on version information resource.

Following are the specific settings the IME file should include:

dwFileOS
The *dwFileOS* should be specified in the root block of the version information. The *dwFileOS* should be VOS__WINDOWS32 for Windows 95 and Windows NT/Windows 2000 IMEs.

dwFileType
The *dwFileType* should be specified in the root block of the version information. The value is VFT_DRV.

dwFileSubtype
The *dwFileSubtype* should be specified in the root block of the version information. The value is VFT2_DRV_INPUTMETHOD.

FileDescription
The *FileDescription* is specified in the language-specific block of the version information. This should include the IME name and the version. This string is for display purposes only. Currently, the string length is 32 TCHARS, but may be extended in a future version.

ProductName
The *ProductName* is specified in the language-specific block of the version information.

Charset ID and Language ID
The code page (character set ID) and language ID are specified in the variable information block of the version information resource. If there are many code pages (character set ID) and language IDs are specified in the block, the IME uses the first code page ID to display the text and uses the first language ID for the IME language. The charset ID and language ID must match the IME language instead of the resource language. The file name of IME is 8.3.

IME Registry Contents

The IME HKEY_CURRENT_USER registry contains an Input Method key. The following table describes the contents of this Input Method.

Key	Contents
Input Method	There are four value names: Perpendicular Distance, Parallel Distance, Perpendicular Tolerance, and Parallel Tolerance. The near caret operation IME refers to these values. If these four value names are not present, a near operation IME can set a default value, depending on the IME

Value Name	Value Data
Perpendicular Distance	Distance is perpendicular to the text escapement. This is the perpendicular distance (pixels) from the caret position to the composition window without the font height and width. The near caret operation IME will adjust the composition window position according to this value and Parallel Distance. It is in REG_DWORD format.
Parallel Distance	Distance (pixels) is parallel to the text escapement. This is the parallel distance from the caret position to the composition window. It is in REG_DWORD format.
Perpendicular Tolerance	Tolerance (pixels) is perpendicular to the text escapement. This is the perpendicular distance from the caret position to the composition window. The near caret operation IME will not refresh its composition window if the caret movement is within this tolerance. It is in REG_DWORD format.
Parallel Tolerance	Tolerance (pixels) is parallel to the text escapement. This is the

	parallel distance from the caret position to the composition window. It is in REG_DWORD format.

An IME can place the per-user setting into:

Under HKEY_CURRENT_USER\Software\<Company Name>\Windows\CurrentVersion\<IME Name>.

An IME can place the per computer setting into:

Under HKEY_LOCAL_MAACHINE\Software\<Company Name>\Windows\CurrentVersion\<IME Name>.

IMM and IME Data Structures

The following structures are used for IMM and IME communication. IMEs can access these structures directly, but applications cannot.

INPUTCONTEXT

The **INPUTCONTEXT** structure is an internal data structure that stores Input Context data.

```
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;
```

Members

hWnd
Window handle that uses this Input Context. If this Input Context has shared windows, this must be the handle of the window that is activated. It can be reset with **ImmSetActiveContext**.

fopen
Present status of opened or closed IME.

ptStatusWndPos
Position of the status window.

ptSoftKbdPos
Position of the soft keyboard.

fdwConversion
Conversion mode that will be used by the IME composition engine.

fdwSentence
Sentence mode that will be used by the IME composition engine.

lfFont
LogFont structure to be used by the IME User Interface when it draws the composition string.

cfCompForm
COMPOSITIONFORM structure that will be used by the IME User Interface when it creates the composition window.

cfCandForm[4]
CANDIDATEFORM structures that will be used by the IME User Interface when it creates the candidate windows. This IMC supports four candidate forms.

hCompStr
Memory handle that points to the **COMPOSITIONSTR** structure. This handle is available when there is a composition string.

hCandInfo
Memory handle of the candidate. This memory block has the **CANDIDATEINFO** structure and some **CANDIDATELIST** structures. This handle is available when there are candidate strings.

hGuideLine
Memory handle of **GuideLine**. This memory block has the **GUIDELINE** structure. This handle is available when there is guideline information.

hPrivate
Memory handle that will be used by the IME for its private data area.

dwNumMsgBuf
Number of messages that are stored in the **hMsgBuf**.

hMsgBuf
Memory block that stores the messages. The format of this memory block is [Message1] [wParam1] [lParam1] {[Message2] [wParam2] [lParam2]{...{...{...}}}}. All values are double word.

fdwinit
Initialize flag. When an IME initializes the member of the **INPUTCONTEXT** structure, the IME has to see the bit of this member. The following bits are provided.

Bit	Description
INIT_STATUSWNDPOS	Initialize **ptStatusWndPos**.
INIT_CONVERSION	Initialize **fdwConversion**.
INIT_SENTENCE	Initialize **fdwSentence**.
INIT_LOGFONT	Initialize **lfFont**.
INIT_COMPFORM	Initialize**cfCompForm**.
INIT_SOFTKBDPOS	Initialize **ptSoftKbdPos**.

dwReserve[3]
Reserved for future use.

Note
During a call to **ImeToAsciiEx**, an IME can generate the messages into the **lpdwTransKey** buffer. However, if an IME wants to generate the messages to the application, it can store the messages in **hMsgBuf** and call **ImmGenerateMessage**. The **ImmGenerateMessage** function then sends the messages in **hMsgBuf** to the application.

COMPOSITIONSTR

The **COMPOSITIONSTR** structure contains the composition information. During conversion, an IME places conversion information into this structure.

```
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;
```

Members

dwSize
Memory block size of this structure.

dwCompReadAttrLen
Length of the attribute information of the reading string of the composition string.

dwCompReadAttrOffset
Offset from the start position of this structure. Attribute information is stored here.

dwCompReadClsLen
Length of the clause information of the reading string of the composition string.

dwCompReadClsOffset
Offset from the start position of this structure. Clause information is stored here.

dwCompReadStrLen
Length of the reading string of the composition string.

dwCompReadStrOffset
Offset from the start position of this structure. Reading string of the composition string is stored here.

dwCompAttrLen
Length of the attribute information of the composition string.

dwCompAttrOffset
Offset from the start position of this structure. Attribute information is stored here.

dwCompClsLen

Length of the clause information of the composition string.

dwCompClsOffset
Offset from the start position of this structure. Clause information is stored here.

dwCompStrLen
Length of the composition string.

dwCompStrOffset
Offset from the start position of this structure. The composition string is stored here.

dwCursorPos
Cursor position in the composition string.

dwDeltaStart
Start position of change in the composition string. If the composition string has changed from the previous state, the first position of such a change is stored here.

dwResultReadClsLen
Length of the clause information of the reading string of the result string.

dwResultReadClsOffset
Offset from the start position of this structure. Clause information is stored here.

dwResultRieadStrLen
Length of the reading string of the result string.

dwResultReadStrOffset
Offset from the start position of this structure. Reading string of the result string is stored at this point.

dwResultClsLen
Length of the clause information of the result string.

dwResultClsOffset
Offset from the start position of this structure. Clause information is stored here.

dwResultStrLen
Length of the result string.

dwResultStrOffset
Offset from the start position of this structure. Result string is stored here.

dwPrivateSize
Private area in this memory block.

dwPrivateOffset
Offset from the start position of this structure. Private area is stored here.

Note

For Unicode: All **dw*StrLen** members contain the size in Unicode characters of the string in the corresponding buffer. Other **dw*Len and dw*Offset** members contain the size in bytes of the corresponding buffer.

The format of the attribute information is a single-byte array and specifies the attribute of string. The following values are provided. Those not listed are reserved.

Value	Content
ATTR_INPUT	Character currently being entered.
ATTR_TERGET_CONVERTED	Character currently being converted (already converted).
ATTR_CONVERTED	Character given from the conversion.
ATTR_TERGET_NOTCONVERTED	Character currently being converted (yet to be converted).
ATTR_FIXEDCONVERTED	Characters will not be converted anymore.
ATTR_INPUT_ERROR	Character is an error character and cannot be converted by the IME.

Following is a description of the content for the values provided in the preceding table.

Content	Description
Character currently being entered.	Character that the user is entering. If this is Japanese, this character is a hiragana, katakana, or alphanumeric character that has yet to be converted by the IME.
Character currently being converted (already converted).	Character that has been selected for conversion by the user and converted by the IME.
Character given from conversion.	Character which the IME has converted.
Character currently being converted (yet to be converted).	Character that has been selected for conversion by the user and not yet converted by the IME. If this is Japanese, this character is a hiragana, katakana, or alphanumeric

	character that the user has entered.
Character is an error character and cannot be converted by the IME.	Character is an error character and the IME cannot convert this character. For example, some consonants cannot be combined.

Comments

The length of the attribute information is the same as the length of the string. Each byte corresponds to each byte of the string. Even if the string includes DBCS characters, the attribute information has the information bytes of both the lead byte and the second byte.

For Windows NT/Windows 2000 Unicode, the length of the attribute information is the same as the length in Unicode character counts. Each attribute byte corresponds to each Unicode character of the string.

The format of the clause information is a double-word array and specifies the numbers that indicate the position of the clause. The position of the clause is the position of the composition string, with the clause starting from this position. At the least, the length of information is two double words. This means the length of the clause information is 8 bytes. The first double word has to be zero and is the start position of the first clause. The last double word has to be the length of this string. For example, if the string has three clauses, the clause information has four double words. The first double word is zero. The second double word specifies the start position of the second clause. The third double word specifies the start position of the third clause. The last double word is the length of this string.

For Windows NT/Windows 2000 Unicode, the position of each clause and the length of the string is counted in Unicode characters.

The **dwCursorPos** member specifies the cursor position and indicates where the cursor is located within the composition string, in terms of the count of that character. The counting starts at zero. If the cursor is to be positioned immediately after the composition string, this value should be equal to the length of the composition string. In the event there is no cursor, a value of -1 is specified here. If a composition string does not exist, this member is invalid.

For Windows NT/Windows 2000 Unicode, the cursor position is counted in Unicode characters.

CANDIDATEINFO

The **CANDIDATEINFO** structure is a header of the entire candidate information. This structure can contain 32 candidate lists at most, and these candidate lists have to be in the same memory block.

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

Members

dwSize
Memory block size of this structure.

dwCount
Number of the candidate lists that are included in this memory block.

dwOffset[32]
Contents are the offset from the start position of this structure. Each offset specifies the start position of each candidate list.

dwPrivateSize
Private area in this memory block.

dwPrivateOffset
Offset from the start position of this structure. The private area is stored here.

GUIDELINE

The **GUIDELINE** structure contains the guideline information that the IME sends out.

```
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;
```

Note
For Unicode, the **dwStrLen** member specifies the size in Unicode characters of the error string. Other size parameters such as **dwSize** **dwStrOffset**, **dwPrivateSize** contain values counted in bytes.

Members

dwLevel
The **dwLevel** specifies error level. The following values are provided.

Value	Description
GL_LEVEL_NOGUIDELINE	No guideline present. If the old guideline is shown, the UI should hide the old guideline.
GL_LEVEL_FATAL	Fatal error has occurred. Some data may be lost.
GL_LEVEL _ERROR	Error has occurred. Handling may not be continued.
GL_LEVEL _WARNING	IME warning to user. Something unexpected has occurred, but the IME can continue handling.
GL_LEVEL _INFORMATION	Information for the user.

dwIndex
The following values are provided.

Value	Description
GL_ID_UNKNOWN	Unknown error. The application should refer to the error string.
GL_ID_NOMODULE	IME cannot find the module that the IME needs.
GL_ID_NODICTIONARY	IME cannot find the dictionary or the dictionary looks strange.
GL_ID_CANNOTSAVE	Dictionary or the statistical data cannot be saved.
GL_ID_NOCONVERT	IME cannot convert anymore.
GL_ID_TYPINGERROR	Typing error. The IME cannot handle this typing.
GL_ID_TOOMANYSTROKE	Two many keystrokes for one

	character or one clause.
GL_ID_READINGCONFLICT	Reading conflict has occurred. For example, some vowels cannot be combined.
GL_ID_INPUTREADING	IME prompts the user now it is in inputting reading character state.
GL_ID_INPUTRADICAL	IME prompts the user now it is in inputting radical character state.
GL_ID_INPUTCODE	IME prompts the user to input the character code state.
GL_ID_CHOOSECANIDATE	IME prompts the user to select the candidate string state.
GL_ID_REVERSECONVERSION	IME prompts the user to provide the information of the reverse conversion. The information of reverse conversion can be obtained through ImmGetGuideLine(*hIMC*, GGL_PRIVATE. *lpBuf*, *dwBufLen*).The information contained in *lpBuf* is in CANDIDATELIST format.
GL_ID_PRIVATE_FIRST	ID located between GL_ID_PRIVATE_FIRST and GL_ID_PRIVATE_LAST is reserved for the IME. The IME can freely use these IDs for its own **GUIDELINE**.
GL_ID_PRIVATE_LAST	ID located between GL_ID_PRIVATE_FIRST and GL_ID_PRIVATE_LAST is reserved for the IME. The IME can freely use these IDs for its own **GUIDELINE**.

dwPrivateSize
Private area in this memory block.

dwPrivateOffset
Offset from the start position of this structure. The private area is stored here.

IME Management Structures

The following topics describe the structures used to manage IMEs.

IMEINFO

The **IMEINFO** structure is used internally by IMM and IME interfaces.

```
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;
```

Members

dwPrivateDataSize
Byte count of the structure.

fdwProperty
HIWORD of **fdwProperty** contains the following bits, which are used by the application.

Bit	Description
IME_PROP_AT_CARET	Bit On indicates that the IME conversion window is at caret position. Bit Off indicates a near caret position operation IME.
IME_PROP_SPECIAL_UI	Bit On indicates that the IME has a special UI. The IME should set this bit only when it has an nonstandard UI that an application cannot display. Typically, an IME will not set this flag.
IME_PROP_CANDLIST_START_FROM_1	Bit ON indicates that the UI of the candidate list string starts from zero or 1. An application can draw the candidate list string by adding the 1, 2, 3, and so on in front of the

candidate string.

IME_PROP_UNICODE	Bit ON indicates that the string content of the Input Context is or is not in UNICODE.

The LOWORD of **fdwProperty** contains the following bits, which are used by the system.

Bit	Description
IME_PROP_END_UNLOAD	Bit On indicates that the IME will unload when there is no one using it.
IME_PROP_KBD_CHAR_FIRST	Before the IME translates the DBCS character, the system first translates the characters by keyboard. This character is passed to the IME as an information aid. No aid information is provided when this bit is off.
IME_PORP_NEED_ALTKEY	IME needs the ALT key passed to **ImeProcessKey**.
IME_PROP_IGNORE_UPKEYS	IME does not need the UP key passed to **ImeProcessKey**.
IME_PROP_COMPLETE_ON_ UNSELECT	New property bit defined for Windows 98 and Windows 2000. If set, the IME will complete the composition string when the IME is deactivated. If clear, the IME will cancel the composition string when the IME is deactivated (such as from a keyboard layout change).

fdwConversionCaps

Same definition as the conversion mode. If the relative bit is off, the IME does not have the capability to handle the conversion mode no matter whether the corresponding bit of the conversion mode is on or off.

Conversion mode	Description
IME_CMODE_KATAKANA	Bit On indicates that the IME supports KATAKANA mode. Otherwise, it does not.
IME_CMODE_NATIVE	Bit On indicates that the IME supports NATIVE mode. Otherwise, it does not.
IME_CMODE_FULLSHAPE	Bit On indicates that the IME

	supports full shape mode. Otherwise, it does not.
IME_CMODE_ROMAN	Bit On indicates that the IME supports ROMAN input mode. Otherwise, it does not.
IME_CMODE_CHARCODE	Bit On indicates that the IME supports CODE input mode. Otherwise, it does not.
IME_CMODE_HANJACONVERT	Bit On indicates that the IME supports HANJA convert mode. Otherwise, it does not.
IME_CMODE_SOFTKBD	Bit On indicates that the IME supports soft keyboard mode. Otherwise, it does not.
IME_CMODE_NOCONVERSION	Bit On indicates that the IME supports No-conversion mode. Otherwise, it does not.
IME_CMODE_EUDC	Bit On indicates that the IME the IME supports EUDC mode. Otherwise, it does not.
IME_CMODE_SYMBOL	Bit On indicates that the IME supports SYMBOL mode. Otherwise, it does not.
IME_CMODE_CHARCODE	Set to 1 if the IME supports character code input mode, but zero if it does not.
IME_CMODE_FIXED	Set to 1 if the IME supports fixed conversion mode, but zero if not. This mode allows preconversion by the IME, but not full conversion. An example of this is Fixed Conversion Mode with DBCS HIRAGANA ROMAN. Under this mode, the IME can convert key input characters to DBCS HIRAGANA by Roman Conversion. However, it prevents conversion from DBCS HIRAGANA to Kanji characters.

fdwSentenceCaps

Same constant definition as the sentence mode. If the relative bit is off, the IME does not have the capability to handle the sentence mode no matter if the corresponding bit of sentence mode is on or off.

Conversion mode	Description
IME_SMODE_PLAURALCLAUSE	Bit On indicates that the IME supports plural clause sentence mode.
IME_SMODE_SINGLECONVERT	Bit On indicates that the IME supports single character sentence mode.
IME_SMODE_AUTOMETIC	Bit On indicates that the IME supports automatic sentence mode.
IME_SMODE_PHRASEPREDICT	Bit On indicates that the IME supports phrase predict sentence mode.
IME_SMODE_CONVERSATION	IME uses conversation mode. This is useful for chat applications. Chat applications can change the sentence mode of the IME to conversation style. This is a new mode for Windows 98 and Windows 2000.

fdwUICaps

The **fdwUICaps** bits specify the UI ability of the IME. The following bits are provided.

Bit	Description
UI_CAP_2700	UI supported when **LogFont** escape is zero or 2700.
UI_CAP_ROT90	UI supported when **LogFont** escape is zero, 900, 1800, or 2700.
UI_CAP_ROTANY	UI supported with any escape.
UI_CAP_SOFKBD	IME uses soft keyboard provided by the system.

fdwSCSCaps

The **fdwSCSCaps** bits specify the **SetCompositionString** capability that the IME has. The following bits are provided.

Bit	Description
SCS_CAP_COMPSTR	IME can generate the composition string by SCS_SETSTR.
SCS_CAP_MAKEREAD	When calling **ImmSetCompositionString** with SCS_SETSTR, the IME can create the

reading of the composition string without *lpRead*. Under the IME that has this capability, the application does not need to set *lpRead* for SCS_SETSTR.

fdwSelectCaps

The **fdwSelectCaps** capability is for the application. When a user changes the IME, the application can determine if the conversion mode will be inherited or not by checking this capability. If the newly selected IME does not have this capability, the application will not receive the new mode and will have to retrieve the mode again. The following bits are provided.

Bit	Description
SELECT_CAP_CONVMODE	IME has the capability of inheritance of conversion mode at **ImeSelect**.
SELECT_CAP_SENTENCE	IME has the capability of inheritance of sentence mode at **ImeSelect**.

Structures Used for IME Communication

The following topics describe the structures used for communication with IMEs.

CANDIDATELIST

The **CANDIDATELIST** structure contains information about a candidate list.

```
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;
```

Members

dwsize
Size, in bytes, of the structure, the offset array, and all candidate strings.

dwStyle
Candidate style values. It can be one or more of the following values.

Value	Meaning
IME_CAND_UNKNOWN	Candidates are in a style other than listed here.
IME_CAND_READ	Candidates have the same reading.
IME_CAND_CODE	Candidates are in one code range.
IME_CAND_MEANING	Candidates have the same meaning.
IME_CAND_RADICAL	Candidates are composed of same radical character.
IME_CAND_STROKES	Candidates are composed of same number of strokes.

For the IME_CAND_CODE style, the candidate list has a special structure depending on the value of the **dwCount** member. If **dwCount** is 1, the **dwOffset** member contains a single DBCS character rather than an offset, and no candidate string is provided. If the **dwCount** member is greater than 1, the **dwOffset** member contains valid offsets, and the candidate strings are text representations of individual DBCS character values in hexadecimal notation.

dwCount
Number of candidate strings.

dwSelection
Index of the selected candidate string.

dwPageStart
Index of the first candidate string in the candidate window. This varies as the user presses the Page Up and Page Down keys.

dwPageSize
Number of candidate strings to be shown in one page in the candidate window. The user can move to the next page by pressing IME-defined

keys, such as the Page Up or Page Down key. If this number is zero, an application can define a proper value by itself.

dwOffset
Offset to the start of the first candidate string, relative to the start of this structure. The offsets for subsequent strings immediately follow this member, forming an array of 32-bit offsets.

Comments

The **CANDIDATELIST** structure is used for the return of **ImmGetCandidateList** . The candidate strings immediately follow the last offset in the **dwOffset** array.

COMPOSITIONFORM

The **COMPOSITIONFORM** structure is used for **IMC_SETCOMPOSITIONWINDOW** and **IMC_SETCANDIDATEPOS** messages.

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

Members

dwStyle
Position style. The following values are provided.

Value	Meaning
CFS_DEFAULT	Move the composition window to the default position. The IME window can display the composition window outside the client area, such as in a floating window.
CFS_FORCE_POSITION	Display the upper-left corner of the composition window at exactly the position given by **ptCurrentPos**. The coordinates are relative to the upper-left corner of the window containing the composition window and are *not* subject to adjustment by the IME.
CFS_POINT	Display the upper-left corner of the composition window at the position given by **ptCurrentPos**. The coordinates are relative to the upper-left corner of the window

	containing the composition window and are subject to adjustment by the IME.
CFS_RECT	Display the composition window at the position given by **rcArea**. The coordinates are relative to the upper-left of the window containing the composition window.

ptCurrentPos
Coordinates of the upper-left corner of the composition window.

rcArea
Coordinates of the upper-left and lower-right corners of the composition window.

Comments

When the style of the **COMPOSITIONFORM** structure is CFS_POINT or CFS_FORCE_POINT, the IME will draw the composition string from the position specified by **ptCurrentPos** of the **COMPOSITIONFORM** structure that is given by the application. If the style has CFS_RECT, the composition string will be inside the rectangle specified by **rcArea**. If not, **rcArea** will be the client rectangle of the application window.

When the application specifies the composition font, the composition window is rotated as the escapement of the composition font. The direction of the composition string follows the escapement of the font in a composition window. The IME then draws the composition string. Following is an example of this process using various values for the escapement of the composition font:

- Escapement of the composition font is zero

  Typically, the escapement of the composition font is zero. When this is the case, **ptCurrentPos** of the composition form structure points to the left and top of the string. All IMEs support this type.

- Escapement of the composition font is 2700

  This is in the case of a vertical writing. When the application provides the vertical writing, the application can set the 2700 escapement in the composition font set by **ImmCompositoinFont**. The composition string will then be drawn downward. IMEs that have UI_CAP_2700, UI_CAP_ROT90, or UI_CAP_ROTANY capability will support this type of composition window.

- Escapement of the composition font is 900 or 1800

  IMEs that have UI_CAP_ROT90 or UI_CAP_ROTANY capability will support this type of composition window.

- Escapement of the composition font is any value

IMEs that have UI_CAP_ROTANY capability will support this type of composition window.

Note
UI_CAP_ROT90 and UI_CAPS_ANY are the option for the enhancement of the IME. UI_CAP_2700 is recommended.

CANDIDATEFORM

The **CANDIDATEFORM** structure is used for IMC_GETCANDIDATEPOS and IMC_SETCANDIDATEPOS messages.

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

Members

dwIndex
Specifies the ID of the candidate list. Zero is the first candidate list, 1 is the second one, and so on up to 3.

dwStyle
Specifies CFS_CANDIDATEPOS or CFS_EXCLUDE. For a near-caret IME, the **dwStyle** also can be CFS_DEFAULT. A near-caret IME will adjust the candidate position according to other UI components, if the **dwStyle** is CFS_DEFAULT.

ptCurrentPos
Depends on **dwStyle**. When **dwStyle** is CFS_CANDIDATEPOS, **ptCurrentPos** specifies the recommended position where the candidate list window should be displayed. When **dwStyle** is CFS_EXCLUDE, **ptCurrentPos** specifies the current position of the point of interest (typically the caret position).

rcArea
Specifies a rectangle where no display is allowed for candidate windows in the case of CFS_EXCLUDE.

STYLEBUF

The **STYLEBUF** structure contains the identifier and name of a style.

```
typedef struct tagSTYLEBUF {
```

```
DWORD dwStyle;
TCHAR szDescription[32]
} STYLEBUF;
```

Members

dwStyle
Style of register word.

szDescription
Description string of this style.

Note
The style of the register string includes IME_REGWORD_STYLE_EUDC. The string is in EUDC range:

IME_REGWORD_STYLE_USER_FIRST and IME_REGWORD_STYLE_USER_LAST.

The constants range from IME_REGWORD_STYLE_USER_FIRST to IME_REGWORD_STYLE_USER_LAST and are for private IME ISV styles. The IME ISV can freely define its own style.

SOFTKBDDATA

The **SOFTKBDDATA** defines the DBCS codes for each virtual key.

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

Members

uCount
Number of the 256-word virtual key mapping to the internal code array.

wCode[][256]
256-word virtual key mapping to the internal code array. There may be more than one 256-word arrays.

Note
It is possible for one type of soft keyboard to use two 256-word arrays. One is for the nonshift state and the other is for the shift state. The soft keyboard can use two internal codes for displaying one virtual key.

RECONVERTSTRING

The **RECONVERTSTRING** structure defines the strings for IME reconversion. It is the first item in a memory block that contains the strings for reconversion.

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

Members

dwSize
Memory block size of this structure.

dwVersion
Reserved by the system. This must be zero.

dwStrlen
Length of the string that contains the composition string.

dwStrOffset
Offset from the start position of this structure. The string containing the reconverted words is stored at this point.

dwCompStrLen
Length of the string that will be the composition string.

dwCompStrOffset
Offset of the string that will be the composition string.

dwTargetStrLen
Length of the string that is related to the target clause in the composition string.

dwTargetStrOffset
Offset of the string that is related to the target clause in the composition string.

Note

The **RECONVERTSTRING** structure is a new structure for Windows 98 and Windows 2000. The **dwCompStrOffset** and **dwTargetOffset** members are the relative position of **dwStrOffset**. For Windows NT/Windows 2000 Unicode, **dwStrLen**, **dwCompStrLen**, and **dwTargetStrLen** are the TCHAR count,

and **dwStrOffset**, **dwCompStrOffset**, and **dwTargetStrOffset** are the byte offset.

Comments

If an application starts the reconversion process by calling **ImmSetCompositionString** with SCS_SETRECONVERTSTRING and SCS_QUERYRECONVERTSTRING, the application is then responsible for allocating the necessary memory for this structure as well as the composition string buffer. The IME should not use the memory later. If the IME starts the process, it should allocate the necessary memory for the structure and the composition string buffer.

IMEMENUITEMINFO

The **IMEMENUITEMINFO** structure contains information about IME menu items.

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

Members

cbSize
Size of the structure in bytes

fType
Menu item type. This member can be one or more of the following values.

Value	Meaning
IMFT_RADIOCHECK	Displays checked menu items using a radio-button mark instead of a check mark if the **hbmpChecked** member is NULL.
IMFT_SEPARATOR	Specifies that the menu item is a separator. A menu item separator appears as a horizontal dividing line. The **hbmpItem** and

szString members are ignored.

IMFT_SUBMENU	Specifies that the menu item is a submenu.

fState

Menu item state. This member can be one or more of the following values.

Value	Meaning
IMFS_CHECKED	Checks the menu item. For more information about checked menu items. See the **hbmpChecked** member.
IMFS_DEFAULT	Specifies that the menu item is the default. A menu can contain only one default menu item, which is displayed in bold.
IMFS_DISABLED	Disables the menu item so it cannot be selected, but does not gray it out.
IMFS_ENABLED	Enables the menu item so it can be selected. This is the default state.
IMFS_GRAYED	Disables the menu item and grays it out so it cannot be selected.
IMFS_HILITE	Highlights the menu item.
IMFS_UNCHECKED	Unchecks the menu item. For more information about unchecked menu items, see the **hbmpUnchecked** member.
IMFS_UNHILITE	Removes the highlight from the menu item. This is the default state.

wID

Application-defined 16-bit value that identifies the menu item.

hbmpChecked

Handle to the bitmap to display next to the item if it is checked. If this member is NULL, a default bitmap is used. If the IMFT_RADIOCHECK type value is specified, the default bitmap is a bullet. Otherwise, it is a check mark.

hbmpUnchecked

Handle to the bitmap to display next to the item if it is not checked. If this member is NULL, no bitmap is used.

dwItemData

Application-defined value associated with the menu item.

szString

Content of the menu item. This member is a null-terminated string.

hbmpItem

Bitmap handle to display.

Note

The **IMEMENUITEMINFO** structure is a new structure for Windows 98 and Windows 2000. The Unicode version of this structure has the **szString** member as the WCHAR.