Windows 95 / Windows NT

Multilingual IME API Specification for IME

Version 1.10
Last Revised: May 27,1996

Windows 95 Multilingual IME

In Windows 95, IMEs are provided as Dynamic Link Library(DLL) in contrast to IMEs of Windows 3.1 Far East Edition, and each IME runs as one of Multilingual Keyboard Layouts. Comparing with 3.1 IME, Windows 95 Multilingual IME provides following advantage.

- Run as a component of Multilingual Environment.
- Multiple Input Context for each application task.
- One Active IME for one application thread.
- Gives any information to application through message loop (No message order broken).
- Strong support for both IME unaware and IME aware applications .

To take these advantages fully, an application needs to support Windows 95 IME application I/F.

This document mainly describes application I/F of Windows 95 IME architecture.

1 Overview

1.1 Structure of IME

The Windows 95 IME has to provide two compornent. One is the IME Conversion Interface and the other is the IME's User Interface. The IME Conversion Interface is provided as a set of functions that are exported from IME module. These functions are called by IMM. The IME User Interface is provided as windows. These windows recieve some messages and provide the IME’s user interface.

1.2 IME Aware Applications

We can see how an application could be involved with IME as follows.

- IME unaware applications : This kind of applications never intend to control the IME. However as long as it accepts DBCS characters, the end user can type any DBCS character to the application using IME.
- IME half-aware applications: This kind of applications typically control various context of IME, such as open / close, composition form and so on, but it doesn't display any user interface for IME.
- IME full-aware applications: This kind of applications typically want to be fully responsible to display any information given by IME.

In Windows 95, one IME unaware application will be supported with one *Default IME window* and one *Default Input Context.*

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

IME full-aware applications will handle the Input Context by itself. They will display any necessary information given by the Input Context not using IME window.

2 IME User Interface

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

2.1 Features

"IME" class is a predefined global class that carries out any user interface portion of the IME. The normal characteristics of "IME" class are same with other common control. Its window instance can be created by CreateWindowEx function. Like static controls, the IME class window doesn't respond to user input by itself; instead, it receives various type of control messages to realize entire user interface of the IME. An

applications can create its own IME window(s) by using this IME class or obtain the Default IME window by ImmGetDefaultIMEWnd, the application which wants to control IME with these window handles (IME-aware application) will obtain following benefits against Windows 3.1.

- Including candidates listing windows, each application can have its own window instance of UI so that the end user can stop in the middle of any kind of operations to change focus to another application, while Windows 3.1 Japanese Edition limits the user to abandon his / her operation when moving to another application.
- Since the IME user interface windows will be informed about application's window handle, it can provide lots of default behavior for the application, such as, move automatically according to the application moving, trace automatically the caret position of the window, mode indication for each application, and so on.

Even though System provides only one "IME" class, there are 2 kinds of IME window. One is created by System for DefWindowProc function especially for IME unaware program. DefWindowProc function's IME User Interface is shared by all IME unaware windows of a thread. In this document, this is called *Default IME window*. The others are created by IME aware applications. In this document, this is called *Application IME window*.

2.2 Default and Application IME window

System creates *Default IME window* at a thread initializing time. I.e., this *Default IME window* will be given to a thread automatically. This window will handle any IME user interface for IME unaware application. When the IME or IMM generates WM_IME_xxx messages, IME unaware application will pass them to DefWindowProc(). DefWindowProc() sends necessary messages to *Default IME window* then the window provide default behavior of IME UI for unaware application. An IME aware application also uses this window when it doesn't hook messages from IME. An application can use its own *Application IME window* only when it is necessary.

2.3 IME class

Windows 95-FE will provide "IME" class by system default. This class will be defined by USER.EXE just as "Edit" pre-defined class. System "IME" class handles entire UI of the IME and whole control messages from IME and Application including IMM function. Applications can create its own IME User Interface by using this class. System IME class itself won't be replaced by any IME. Windows 95-FE will keep this just as pre-defined class.

This class has a window procedure that will actually handle WM_IME_SELECT message. This message has the hKL of newly selected IME. System IME class retrieve name of class defined by each IME with this hKL. Using this name, the System IME class will create a UI window of the current active IME.

2.4 UI class from IME

In this design, every IME is expected to register its own UI class for system. UI class that will be provided by each IME should be responsible for IME unique functionality. The IME may register the classes that are used by IME itself when the IME is attached to the process. It is a time that the DllEntry is called with DLL_PROCESS_ATTACH. Then the IME has to set the UI class name into the lpszClassName that is the second parameter of ImeInquire(). The UI class should be registered with CS_IME specified at style field so that every application can use it through "IME" class. The UI class name (including null terminator) is up to 16 TCHAR and may be extended in future version.

And the cbWndExtra of UI class have to be 2 * sizeof(LONG). And the purpose of this WndExtra is defined by the system. (IMMGWL_HIMC and IMMGWL_PRIVATE)

The IME can register any class and create any window during it is working for applications.

```
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;
}
```

2.5 UI window

The IME windows of the “IME” class are created by applications or created by System. When the IME window is created, the UI window that is provided by IME itself is created and owned by the IME window. Each UI window has the current Input Context. This Input Context can be get by calling GetWindowLong() with IMMGWL_HIMC when the UI window get the WM_IME_xxx messages. The UI window can refer this Input Context and handle the messages. The Input Context from GetWindowLong with IMMGWL_HIMC ia available anytime in the UI window procedure except of during handling WM_CREATE message.

The cbWndExtra of the UI windows can not be enhanced by IME. When the IME want to use the extra byte of the window instance. The UI window uses SetWindowLong and GetWindowLong with

IMMGWL_PRIVATE. This IMMGWL_PRIVATE provide a LONG value extra of the window instance. When the UI window want to use more than one LONG value extra for private usage, the UI window can put a handle of memory block into the IMMGWL_PRIVATE area.

The UI window procedure can use DefWindowProc function, but the UI window can not pass WM_IME_xxx messages to DefWindowProc(). Even if the message is not handled by the UI window procedure, the UI window don't pass it to DefWindowProc().

```
LRESULT UIWndProc (HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
        HIMC hIMC;
        HGLOBAL hMyExtra;

        switch(msg){
                case WM_CREATE:
                        // Allocate the memory bloack 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 do anything by referring the current Input Context that is selected. When a window of an application is activated, the UI window receives the message that gives the current Input Context. After this, the UI window will run on the Input Context that is now selected. So that the Input Context must have all of information that is needed by the UI window for showing the composition window, the status window and so on.
The UI window will refer the Input Context, but UI window don't needs to update it by itself. When UI window wants to update Input Context, it should call IMM functions. Because the Input Context is managed by IMM, and when the Input Context is changed, IMM and IME should get the notification.

For example, sometimes the UI window want to change the conversion mode of the Input Context at mouse clicking time. At this time, the UI window have to call ImmSetConversionMode function. ImmSetConversionMode function make 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 WM_IME_NOTIFY message.

2.6 Components of UI window

UI window can register and show the composition window and the status window by referring the current Input Context. The class style of components of UI window must include the CS_IME bit. A window instance of UI window will get information of composition string, font, position and so on from the current Input Context. When a window of the application is getting focus, System will be given the Input Context that is owned by this window and set the current Input Context to UI window. System will send WM_IME_SETCONTEXT with the handle of its Input Context to the application, and the application will pass this message to the UI window at last. If current Input Context is replaced to another one, the UI window should repaint the composition window. Anytime current context is changed, the UI window will show a correct composition window, and the status of IME can be assured.

The UI windows may create its child windows or popup windows to show its status, composition string or candidate lists. But these windows have to be owned windowof the UI window and cretated as the disabled window. Any windows that is created by IME should not get the focus.

3 Input Context

3.1 Default Input Context

System gives an Input Context to each thread by default. This context is shared by all IME unaware windows of the thread.

3.2 Application create Input Context

A window of an application can associate its window handle to an Input Context to maintain any status of IME including intermediate composition string. Once an application associates an Input Context to a window handle, system automatically select the context whenever the window gets activated. Using this feature, an application can be free from such complicated in-out focus processing as 3.1 application.

3.3 Using Input Context

When the application or System creates new Input Context, System prepares the new Input Context and this new input context already has IMCC (the conponents of IMC). They are 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.

3.3.1 Asscess HIMC

When an IME accesses the Input Context, the IME has to call ImmLockIMC to get the pointer of the Input Context. ImmLockIMC increment the imm lock count for IMC and ImmUnlockIMC decrement the imm lock count for IMC.

3.3.2 Asscess HIMCC

When an IME accesses one component of the Input Context, the IME has to call ImmLockIMCC to get the pointer of IMCC. ImmLockIMCC increment the imm lock count for IMCC and ImmUnlockIMC decrement the imm lock count for IMCC. ImmReSizeIMCC may resize IMCC to the size that is specified as parameter.
Sometime an IME want to create one component of Inpu Context by itself. Then IME can call ImmCreateIMCC and IME can get the handle of IMCC. This IMCC can be the member of INPUTCONTEXT (hCompStr, hCandInfo, hGuideLine, hPrivate or hMsgBuf).
ImmDestroyIMCC destroys one component of Input Context.

3.3.3 How to use

```
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);
}
```

4 Generating Messages

The IMEs need to generate IME messages. When the IME start conversion, the IME has to generate WM_IME_STARTCOMPOSITION. If the IME change the conposition string, the IME has to generate WM_IME_COMPOSITION. The events from IMEs are realized as generating message to the window that is associated with the Input Context. Basically the IMEs use the lpdwTransKey buffer that is provided by the parameter of ImeToAsciiEx() to generate the message. The IME put the messages into the lpdwTransKey buffer when ImeToAsciiEx() is called. Even if ImeToAsciiEx() is not called, the IME can generate the message to the window that is associated with the Input Context by using the message buffer of the Input Context. The Input Context has the message buffer as a handle of a memory block . The IME puts the messages into the memory block that is provided by the handle of the message buffer. Then the IME call ImmGenerateMessage(). ImmGenerateMessage function send the messages that are stored in the message buffer to the proper window.

4.1 Using the lpdwTransBuf in ImeToAsciiEx

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;
}
```

The buffer that is specified by lpdwTransBuf is provided by System. ImeToAsciiEx() can put the messages into this buffer at one time. The real number of the mesages that can be put in this buffer is given at the first double word of it. But if the ImeToAsciiEx() wants to generate more messages than the given number, ImeToAsciiEx() can put all of the messages into hMsgBuf that is in the input context, and ImeToAsciiEx returns the number of the messages. When the return value of ImeToAsciiEx() is bigger than the specified value in lpdwTransBuf, System does not pick up the messages fron lpdwTransBuf. Then System will look up hMsgBuf of the input context that is passed as paramater of ImeToAsciiEx().

4.2 Using the Message Buffer

```
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 bloack 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);
}
```

4.3 WM_IME_COMPOSITION

When the IME generate WM_IME_COMPOSITION, the IME specify the lParam as GCS bits. The GCS bits means the available member of the COMPOSITIONSTRING Structure. Even if the IME does not update and the member is available now, the IME can set the GCS bit.

4.3.1 Service Definition for IME

When IME generate the WM_IME_COMPOSITION, the IME may change the string, the attribute and the clause information at once. The IME can use the follow definitions.

GCS_COMP
GCS_COMPREAD
GCS_RESULT
GCS_RESULTREAD

5 About ImeSetCompositionString

5.1 The capability of ImeSetCompositionString

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

The IME that has SCS_CAP_COMPSTR capability.

ImeSetCompositionString will be called. The IME put new composition string that is come from application and generate WM_IME_COMPOSITION message.

The IME that has SCS_CAP_MAKEREAD capability.

The IME can make the reading string from the composition string.

5.2 About SCS_SETSTR

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

If it is neccesary, IME may update the condidate information and generate the candidate messages WM_IME_NOTIFY:: IMN_OPENCANDIDATE/ CHANGECANDIDATE or IMN_CLOSECANDIDATE.

If lpRead of parameter of ImeSetCompositonString is available...

Basically, IME should make the composition string from the reading string that is in lpRead. Then the IME make the attribute and clause information for both new composition string and reading string of lpRead. The IME generate WM_IME_COMPOSITION with (GCS_COMP | GCS_COMPREAD). Sometime, IME want to make it finalize automatically. Then the IME may generate WM_IME_COMPOSITION with (GCS_RESULT | GCS_RESULTREAD) instead of GCS_COMPxxx.

If lpComp of parameter of ImeSetCompositonString is available...

Basically, IME should make the composition attribute and clause information from the composition string that is in lpComp. The IME generate WM_IME_COMPOSITION with GCS_COMP. If the IME has the capability of SCS_CAP_MAKEREAD, the IME should make new reading string also at same time. Then the IME will generate WM_IME_COMPOSITION with (GCS_COMP | GCS_COMPREAD). Sometime, IME want to make it finalize automatically. Then the IME may generate WM_IME_COMPOSITION with (GCS_RESULT | GCS_RESULTREAD) instead of GCS_COMPxxx.

If Both lpRead and lpComp is available.....

Basically, IME should make the composition string and the reading string. At this time, IME does not needs to follow lpComp and lpRead completely. If IME can not make the relation between lpRead and lpComp that are specified by the application. The IME should correct the composition string. Then the IME make the attribute and clause information for both new composition string and reading string of lpRead. The IME generate WM_IME_COMPOSITION with (GCS_COMP | GCS_COMPREAD). Sometime, IME want to make it finalize automatically. Then the IME may generate WM_IME_COMPOSITION with (GCS_RESULT | GCS_RESULTREAD) instead of GCS_COMPxxx.

5.3 About 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 and the clause information of reading of the composition string.

At first, the IME have to check the new attribute if acceptable or not. Secondly, the IME set the new attribute into the COMPOSITIONSTRING structure of hIMC . At last, the IME generate WM_IME_COMPOSITION message.

If it is neccesary, IME may update the condidate informations and generate the candidate messages WM_IME_NOTIFY:: IMN_OPENCANDIDATE/ CHANGECANDIDATE or IMN_CLOSECANDIDATE.

Here IME can not finalize the composition string.

If lpRead of parameter of ImeSetCompositonString is available...

The IME follow the new attribute in lpRead. And the IME should make new attribute of the composition string for the current composition string. At this time, the clause information will not be changed.

The composition string, the attribute, the clause information, the reading string, the reading attribute and the reading clause information have to be available. The IME generates WM_IME_COMPOSITION with (GCS_COMP | GCS_COMPREAD). If the IME can not accept the new attribute that is in lpRead, IME don’t need to generate any message and return FALSE.

If lpComp of parameter of ImeSetCompositonString is available...

The IME follow the new attribute in lpComp. At this time, the clause information will not change.

If the capability of the IME has SCS_CAP_MAKEREAD and the reading string is available, the IME should make new attribute of the reading of the composition string for the current reading of the composition string.

If Both lpRead and lpComp is available.....

If the IME can accept these new information, the IME will set the new information into the COMPOSITION Structure of hIMC and generate WM_IME_COMPOSITION with (GCS_COMP | GCS_COMPREAD).

5.4 About SCS_CHANGECLAUSE

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

If it is neccesary, IME may update the condidate informations and generate the candidate messages WM_IME_NOTIFY:: IMN_OPENCANDIDATE/ CHANGECANDIDATE or IMN_CLOSECANDIDATE.

Here IME can not finalize the composition string.

If lpRead of parameter of ImeSetCompositonString is available...

The IME follows the new reading clause information of lpRead. The IME have to correct the attribute of the reading of the composition string. Then the IME may update the composition string, the attribute and the clause information of the composition string. The IME generate WM_IME_COMPOSITION with (GCS_COMP | GCS_COMPREAD).

If lpComp of parameter of ImeSetCompositonString is available...

The IME follows the new clause information. The IME have to correct the composition string and the attribute of the composition string. Then the IME may update the reading attribute and the clause information of reading. The IME generates WM_IME_COMPOSITION with (GCS_COMPSTR | GCS_COMPREAD).

If Both lpRead and lpComp is available.....

If the IME can accept these new information, the IME will set the new information into the COMPOSITION Structure of hIMC and generate WM_IME_COMPOSITION with (GCS_COMP | GCS_COMPREAD).

6 Soft Keyboard

6.1 About Soft Keyboard

1. Some IMEs have special reading characters. For example, an IME may use bo po mo fo as its reading characters, another IME may use some radials as its reading characters. An IME can provide soft keyboard to show these special reading characters so end user don’t need to remember the reading character for each key.

2. IME wants to change the reading characters of keys according to different conversion state. Using soft keyboard can notify the end user the change of keys. In candidate selection time, an IME can only show those selection keys to end user.

6.2 Using Soft Keyboard

1. The IME may want to create better user interface for soft keyboard or it wants to make usage of system predefined soft keyboard. If IME wants to use the system predefined soft keyboard, it needs to specify UI_CAP_SOFTKBD in the fdwUICaps field of IMEINFO on ImeInquire is called.

2. The IME need to call into ImmCreateSoftKeyboard to create window of soft keyboard. And it can call ImmShowSoftKeyboard to show or hide it. The soft keyboard window is one component of the UI window, so the owner should be the UI window.

3. The IME may need to decide whether it want to destroy the window whenever the focus is gone. The soft keyboard may occupy some system resource.

4. There are some types of soft keyboard. One type may only be designed for a specific country or for a special purpose. The way to change reading character may be different for each type of soft keyboard. There are two ways to do this, one is IMC_SETSOFKBDSUBTYPE another is IMC_SETSOFKBDDATA. Different type of soft keyboard has different winodw procedure, it performs different user interface to end user.

7 Data Structure and File format

7.1 File Format of IME

The IMEs need to specify the following fields correctly.

The dwFileVersionMS need to be specified in root block of version information. The upper word is the major Windows version of the IME associate to, the lower word is the minor version.

The dwProductVersionMS need to be specified in root block of version information. The upper word is the major version of this IME, the lower word is the minor version.

The dwFileOS need to be VOS__WINDOWS32 for Windows 95 IME.

The dwFileType need to be specified in root block of version information. The value is VFT_DRV.

The dwFileSubtype need to be specified in root block of version information. The value is VFT2_DRV_INPUTMETHOD.

The FileDescription is specified in language specific block of version information. This should include the IME name and the version. This string is for display purpose. (32 TCHARS, maybe it will be extended in future version)

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

The code page (character set ID) and language ID are specified in variable information block of version information resource. If there are many code pages (character set ID) and language IDs are specified in the

block, the first code page is the IME use to show the text and the first language ID is language for this IME.

The file name of IME is 8.3 in Windows 95-FE.

A Windows 95 IME must be a 32 bit DLL.

7.2 Contents of IME Registry

Under HKEY_CURRENT_USER\Control Panel, we have a key "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 can reference it. If there are no these four value names there, an near operation IME can set a default value to it.

Value Name	Value Data
Perpendicular Distance	The distance is perpendicular to the text escapement. This is the perpendicular distance (pixels) from caret position to composition window without font height/width. The near caret operation IME will adjust composition window position according to this value and Parallel Distance. It is in REG_DWORD format.
Parallel Distance	The distance (pixels) is parallel to the text escapement. This is the parallel distance from caret position to composition window. It is in REG_DWORD format.
Perpendicular Tolerance	The tolerance (pixels) is perpendicular to the text escapement. This is the perpendicular distance from caret position to 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	The tolerance (pixels) is parallel to the text escapement. This is the parallel distance from caret position to composition window. It is in REG_DWORD format.

An IME can put the per user setting into.

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

An IME can put the per computer setting into.

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

7.3 Structures used by IMM and IME

These structures that are described in this section are used for only communication of IME and IMM. IME may access these structure directly. But the application can not access these structure directly.

7.3.1 INPUTCONTEXT Structure

```
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;
```

Member	Description
hWnd	The window handle that uses this Input Context. If this Input Context is shared some windows, this must be the handle of window that is activated and this will be reset by ImmSetActiveContext().
fOpen	the present status of opened or closed IME.
ptStatusWndPos	The position of the status window.
ptSoftKbdPos	The position of the soft keyboard.
fdwConversion	This is conversion mode that will be used by IME composition engine.
fdwSentence	This is sentence mode that will be used by IME composition engine.
lfFont	This is LogFont Structure that will be used by IME User Interface when it draws the composition string.
cfCompForm	This is the COMPOSITIONFORM structure that will be use by IME User Interface when it creates the composition window.

cfCandForm[4]	This is the CANDIDATEFORM structures that will be use by IME User Interface when it creates the candidate windows. This IMC supports 4 candidate forms.
hCompStr	This is memory handle that points to COMPOSITIONSTR structure. This handle will be available when there is the composition string.
hCandInfo	This parameter is the memory handle of Candidate. This memory block has the CANDIDATEINFO structure and some CANDIDATELIST structure. This handle will be available when there are the candidate strings.
hGuideLine	This parameter is the memory handle of GuideLine. This memory block has the GUIDELINE structure. This handle will be available when there are the guideline information.
hPrivate	This is the memory handle that will be used by IME for its private date area.
dwNumMsgBuf	The number of messages that are stored in the hMsgBuf.
hMsgBuf	This is the memory block that store the messages. The format of this memory bloack is [Message1] [wParam1] [lParam1] {[Message2] [wParam2] [lParam2]{...{...{...}}}}. And all values are double word.
fdwInit	The initialize flag. When IME initialize the member of INPUTCONTEXT Structure, IM have to se the bit of this field.

The bit of fdwInit	Description
INIT_STATUSWNDPOS	Initialized ptStatusWndPos
INIT_CONVERSION	Initialized fdwConversion
INIT_SENTENCE	Initialized fdwSentence
INIT_LOGFONT	Initialized lfFont
INIT_COMPFORM	Initialized cfCompForm
INIT_SOFTKBDPOS	initialized ptSoftKbdPos

dwReserve[3]	This is reserved for future usage.

During calling ImeToAsciiEx(), IME can generate the messages into lpdwTransKey buffer, but when IME wants to generate the messages to the Applications out of ImeToAsciiEx(), IME can store the messages into hMsgBuf and calls ImmGenerateMessage(). ImmGenerateMessage will send the messages in hMsgBuf to the application.

7.3.2 COMPOSITIONSTR 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;
```

This structure is the information of composition. During conversion, IME puts convert information into this structure.

Member	Description
dwSize	The memory block size of this structure.
dwCompReadAttrLen	The length of the attribute information of the reading string of the composition string.
dwCompReadAttrOffset	The offset from the start position of this structure. And this attribute information is stored at this point.
dwCompReadClsLen	The length of the clause information of the reading string of composition.
dwCompReadClsOffset	The offset from the start position of this structure. And this clause information is stored at this point.
dwCompReadStrLen	The length of the reading string of the composition string.
dwCompReadStrOffset	The offset from the start position of this structure. And the reading string of the composition string is stored at this point.
dwCompAttrLen	The length of the attribute information of the composition string.
dwCompAttrOffset	The offset from the start position of this structure. And this attribute information is stored at this point.
dwCompClsLen	The length of the clause information of the composition string.
dwCompClsOffset	The offset from the start position of this structure. And this clause information is stored at this point.
dwCompStrLen	The length of the composition string.
dwCompStrOffset	The offset from the start position of this structure. And the composition string is stored at this point.
dwCursorPos	The sursor 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	The length of the clause information of the reading string of the result string.
dwResultReadClsOffset	The offset from the start position of this structure. And this clause information is stored at this point.
dwResultReadStrLen	The length of the reading string of the result string.
dwResultReadStrOffset	The offset from the start position of this structure. And the reading string of the result string is stored at this point.
dwResultClsLen	The length of the clause information of the result string.
dwResultClsOffset	The offset from the start position of this structure. And this clause information is stored at this point.
dwResultStrLen	The length of the result string.
dwResultStrOffset	The offset from the start position of this structure. And the result string is stored at this point.
dwPrivateSize	The private area in this memory block.
dwPrivateOffset	The offset from the start position of this structure. And the private area is stored at this point.

Windows NT-Unicode: All dw*StrLen contain the size in unicode characters of the string in the corresponding buffer, other dw*Len contain the size in bytes of the corresponding buffer.

The format of the attribute information.

The attribute information is a single-byte array and specifies the attribute of string. The contents are as follows:

Value	Content
	Specifies the status of composition string.
ATTR_INPUT	Character currently being entered
ATTR_TERGET_CONVERTED	Character currently being converted (already converted)
ATTR_CONVERTED	Character given from conversion
ATTR_TERGET_NOTCONVERTED	Character currently being converted (yet to be converted)
ATTR_INPUT_ERROR	The charecter is error character and can not be converted by IME.
Other than above:	Reserved

Each content is as follows:

Character currently being entered:

The character the user is entering. In Japanese case, this character is a hiragana, katakana, or alphanumeric, which is yet to be converted by the IME.

Character currently being converted (already converted):

The character that has been selected for conversion by the user and converted by the IME.

Character given from conversion:

The character to which the IME has converted.

Character currently being converted (yet to be converted):

The character that has been selected for conversion by the user and not yet converted by the IME. In Japanese case, this character is a hiragana, katakana, or alphanumeric, which the user has entered.

Character is error character and can not be converted by IME:

The character is an error character, the IME can not convert this character. For example, some consonants can not put together.

The length of the attribute information is 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.
Windows NT-Unicode: The length of the attribute information is 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.

The clause information is a double word array and specifies the numbers that are the positions of the clause. The position of the clause is one of a position of composition string and this clause starts from this position. At least, this 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 0. This 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 0. The second double word specigies the start position of the secound clause. The third double word specifies the start position of the third clause. The last double word is the length of this string.
Windows NT-Unicode: the position of each clause and length of string is counted in unicode characters.

The rule of dwCursorPos

Specifies the cursor position. This value indicates at what character in the composition string the cursor is, in terms of the count of that character. The counting starts at 0. If the cursor is to be positioned immediately after the composition string, this value shall be equal to the length of the composition string. In case there is no cursor (if such a condition exists), a value -1 is specified here. If an composition string does not exist, this member is invalid.
Windows NT-Unicode: the cursor position is counted in unicode characters.

7.3.3 CANDIDATEINFO Structure

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

```
    DWORD       dwPrivateOffset;
} CANDIDATEINFO;
```

This structure is a header of whole candidate informations. This structure can have 32 candidate lists at most, and these candidate lists have to be in same memory block.

Member	Description
dwSize	The memory block size of this structure.
dwCount	The number of the candidate lists that are included in this memory block.
dwOffset[32]	Each contents are the offset from the start position of this structure. And each offset specifies the start position of each candidate list.
dwPrivateSize	The private area in this memory block.
dwPrivateOffset	The offset from the start position of this structure. And the private area is stored at this point.

7.3.4 GUIDELINE Structure

```
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;
```

Windows NT-Unicode: dwStrLen specifies the size in unicode characters of error string. Other size parameters such as dwSize and dwPrivateSize contains value counted in bytes.

dwLevel	Meaning
GL_LEVEL_NOGUIDELINE	There is no guideline. If old guideline is shown, UI should hide old guideline.
GL_LEVEL_FATAL	The fatal error occurs. Some data may be lost.
GL_LEVEL _ERROR	The error occurs. The handling may not be continued.
GL_LEVEL _WARNING	IME worns for user. The unexpected thing occurs, but IME can continue to handle.
GL_LEVEL _INFORMATION	The information for user.

dwIndex	Meaning

GL_ID_UNKNOWN	Unknown Error. The application should refer Error String.
GL_ID_NOMODULE	IME can not find the module that IME needs.
GL_ID_NODICTIONARY	IME can not find the dictionary or the dictionary is strange.
GL_ID_CANNOTSAVE	Dictionary or the statistic data can not be saved.
GL_ID_NOCONVERT	IME can not convert any more.
GL_ID_TYPINGERROR	Typing error. IME can not handle this typing.
GL_ID_TOOMANYSTROKE	There are two many strokes for one character or one clause.
GL_ID_READINGCONFLICT	For example, some vowels can not put together.
GL_ID_INPUTREADING	IME prompts the end user - now it is in inputting reading charcater state.
GL_ID_INPUTRADICAL	IME prompts the end user - now it is in inputting radical charcater state.
GL_ID_INPUTCODE	IME prompts the end user - now it is in inputting charcater code state.
GL_ID_CHOOSECANIDATE	IME prompts the end user - now it is in choosing candidate string state.
GL_ID_REVERSECONVERSION	IME prompts the user end - the information of reverse conversion. The information of reverse conversion can be got by ImmGetGuideLine(*hIMC*, GGL_PRIVATE. *lpBuf*, *dwBufLen*).The information filled in *lpBuf* is in CANDIDATELIST format.
GL_ID_PRIVATE_FIRST	The ID which is between GL_ID_PRIVATE_FIRST and GL_ID_PRIVATE_LAST is reserved for the IME, IME can freely use these ID for its own GUIDELINE.
GL_ID_PRIVATE_LAST	The ID which is between GL_ID_PRIVATE_FIRST and GL_ID_PRIVATE_LAST is reserved for the IME, IME can freely use these ID for its own GUIDELINE.

dwPrivateSize	The private area in this memory block.
dwPrivateOffset	The offset from the start position of this structure. And the private area is stored at this point.

7.4 Structures used to manage IMEs

7.4.1 IMEINFO Structure

This structure is used internally by IMM and IME interface.

```
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;
```

The HIWORD of fdwProperty has the follow bits. These properties are used by the applications.

Properties	**Description**
IME_PROP_AT_CARET	This bit on indicates IME conversion window is at caret position. This bit off indicates a near caret position operation IME.
IME_PROP_SPECIAL_UI	This bit on indicates IME having a special UI. IME should set this bit only in case if it has unordinal UI which an application will never be able to display. Normaly an IME will not set this flag.
IME_PROP_CANDLIST_START_FROM_1	This bit on indicates the UI of candidate list strings start from 0 or 1. Application can draw the candidate list string by adding the “1”, “2”, “3”, or etc in front of the candidate string.
IME_PROP_UNICODE	This bit on indicates the string contents of the Input Context is in UNICODE or not.

The LOWORD of fdwProperty has the follow bits. These properties are used by the system.

Properties	**Description**
IME_PROP_END_UNLOAD	This bit on indicates IME unload when there is no one using it.
IME_PROP_KBD_CHAR_FIRST	Before IME translating the DBCS character, the system translates character by keyboard first. This character is passed to IME as an aid information. No aid information when this bit is off.
IME_PORP_NEED_ALTKEY	This IME need the ALT key pass into ImeProcessKey.
IME_PROP_IGNORE_UPKEYS	This IME don’t need up keys pass into ImeProcessKey.

The fdwConversionCaps share the same definition for the conversion mode. If the relative bit is off, this IME does not have the capability to handle either the coresponding bit of conversion mode is on or off.

conversion mode	**Description**
IME_CMODE_KATAKANA	This bit on indicates IME is in KATAKANA mode, else the IME is in HIRAGANA mode.
IME_CMODE_NATIVE	This bit on indicates IME is in NATIVE mode, else the IME is in ALPHANUMERIC mode.
IME_CMODE_FULLSHAPE	This bit on indicates IME is in full shape mode, else the IME is in SBCS mode.
IME_CMODE_ROMAN	This bit on indicates IME is in ROMAN input mode, else the IME is in non ROMAN input mode.

IME_CMODE_CHARCODE	This bit on indicates IME is in CODE input mode, else the IME is in non CODE input mode.
IME_CMODE_HANJACONVERT	This bit on indicates IME is in HANJA convert mode, else the IME is in non HANJA convert mode.
IME_CMODE_SOFTKBD	This bit on indicates IME is in soft keyboard mode, else the IME is in non soft keyboard mode.
IME_CMODE_NOCONVERSION	This bit on indicates IME is in no conversion mode, else the IME is in conversion mode.
IME_CMODE_EUDC	This bit on indicates IME is in EUDC mode, else the IME is not in EUDC mode. Under this mode, if GCS_RESULTSTR is generated, it means the current reading string is a valid one and can be converted. In some IMEs, if the vowel is not included, it is not a valid reading string. The EUDC editor should not provide such a reading string. Anyway the result string may contain nothing under GCS_RESULTSTR generated, because the string is the EUDC string that EUDC editor is going to register.
IME_CMODE_SYMBOL	This bit on indicates IME is in SYMBOL mode, else the IME is not in SYMBOL mode.

The fdwSentenceCaps share the same constant definition for the sentence mode. If the relative bit is off, this IME does not have the capability to handle either the coresponding bit of sentence mode is on or off.

conversion mode	**Description**
IME_SMODE_PLAURALCLAUSE	This bit on indicates IME support plaural clause sentence mode.
IME_SMODE_SINGLECONVERT	This bit on indicates IME support single character sentence mode.
IME_SMODE_AUTOMETIC	This bit on indicates IME support automatic sentence mode.
IME_SMODE_PHRASEPREDICT	This bit on indicates IME support phrase predict sentence mode.

The fdwUICaps has the follow bits.

Properties	**Description**
UI_CAP_2700	The UI support when escape of LogFont is 0 or 2700.
UI_CAP_ROT90	The UI support when escape of LogFont is 0, 900, 1800 or 2700.
UI_CAP_ROTANY	The UI support any escape.
UI_CAP_SOFKBD	The IME uses soft keyboard provided by the system.

The fdwSCSCaps has the follow bits.

Properties	**Description**
SCS_CAP_COMPSTR	This 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 composition string without lpRead. Under IME that has this capability, the application does not need to set lpRead for SCS_SETSTR.

The fdwSelectCaps has the follow bits.

Properties	Description
SELECT_CAP_CONVMODE	The IME has the capability of inheritance of conversion mode at ImeSelect()
SELECT_CAP_SENTENCE	The IME has the capability of inheritance of sentence mode at ImeSelect()

This capability is for the application. When the end user change the IME, the application can know the conversion mode will be inherited or not by seeing this capability. If the new selected IME does not have this caps, the application can not expect the new mode and it have to get the mode again

7.5 Structures used for communication with IME

7.5.1 CANDIDATELIST Structure

```
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
                                        // 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;
```

dwCandidateStyle	Meaning
IME_CAND_UNKNOWN	Candidates are in the other style than listed above.
IME_CAND_READ	Candidates are in same reading.
IME_CAND_CODE	Candidates are in a code range.
IME_CAND_MEANING	Candidates are in same meaning.
IME_CAND_RADICAL	Candidates use same radical character.
IME_CAND_STROKE	Candidates are in same number of strokes.

This structure is used for a return of ImmGetCandidateList();

When the dwStyle is IME_CAND_CODE, this candidate list structure has the special case. There are two case. One is that dwCount is just 1, and another is that dwCount is larger than 1.

i) When the dwCount equals 1.

DWORD	dwSize;	→ valid
DWORD	dwStyle;	→ IME_CAND_CODE
DWORD	dwCount;	→ 1
DWORD	dwSelection;	→ 0
DWORD	dwPageStart;	→ 0
DWORD	dwPageSize;	→ valid
DWORD	dwOffset[0];	→ Offset of ‘A140’ // ‘A140’ is an example code of // DBCS char not a string.

At this time, the UI will show the candidate list as it likes, but selected one is "A140". For example, UI may show the candidate list from ‘A140’ to ‘A14F’ and the scroll bar at one page.

ii) When the dwCount ia larger than 1.

DWORD	dwSize;	→ valid
DWORD	dwStyle;	→ IME_CAND_CODE
DWORD	dwCount;	→ valid
DWORD	dwSelection;	→ just the index of dwOffset that is selected.
DWORD	dwPageStart;	→ just the fisrt index of dwOffset that is displayed.
DWORD	dwPageSize;	→ valid
DWORD	dwOffset[0];	→ Offset of the first string.
DWORD	dwOffset[1];	→ Offset of the second string.
DWORD	dwOffset[2];	→ Offset of the third string.
TCHAR	szCandStr[0]	→ "A140" string
TCHAR	szCandStr[1]	→ "A141" string
TCHAR	szCandStr[2]	→ "A142" string

At this time, the candidate list will be provided by IME conversion engine. Using this, the IME can support the input like "A1?3". When the end user input "A1?3", the IME provide the candidate list as follows.

DWORD	dwCount;	→ 0x10
DWORD	dwSelection;	→ just the index of dwOffset that is selected.
DWORD	dwPageStart;	→ just the first index of dwOffset that is displayed.
DWORD	dwPageSize;	→ valid
TCHAR	szCandStr[0x0]	→ "A103" string
TCHAR	szCandStr[0x1]	→ "A113" string
TCHAR	szCandStr[0x2]	→ "A123" string
	.....	
TCHAR	szCandStr[0x0F]	→ "A1F3" string

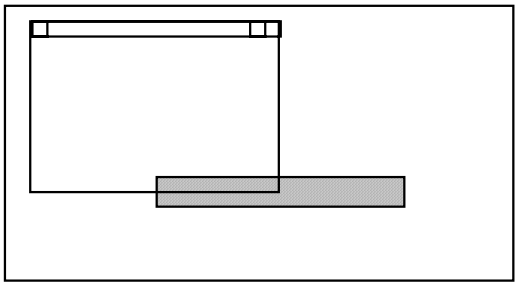
7.5.2 COMPOSITIONFORM Structure

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

The dwStyle member is specified by CFS_xxxx, and the POINT parameters will specify for the size and position of the bounding rectangle and the start position of the composition window.

CFS_DEFAULT

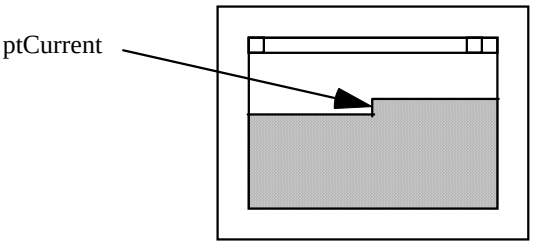
Moves the composition window out to the default positionIME can display the composition window out side client area. It might be on a floating window.



CFS_DEFAULT

CFS_POINT

Instructs to display the composition window at the upper left designated by ptCurrentPos, in the window. ptCurrentPos indicates the coordinates relative to the upper left of the latter window, with the low-order word representing the X coordinate and the high-order word the Y coordinate.

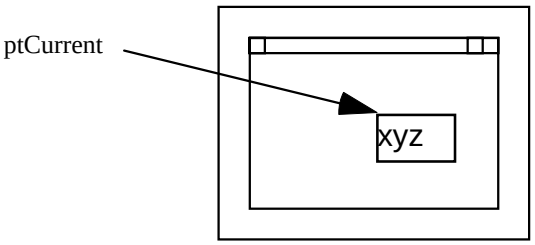


CFS_POINT

CFS_FORCE_POSITION

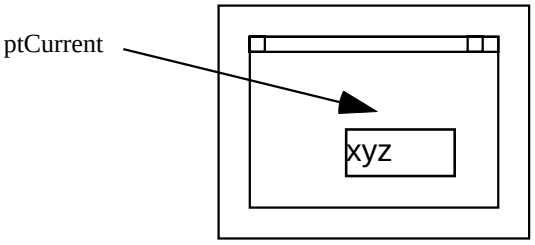
Enforces to display the composition window at the upper left designated by ptCurrentPos, in the window. ptCurrentPos indicates the coordinates relative to the upper left of the latter window, with the low-order word representing the X coordinate and the high-order word the Y coordinate.

Some near caret operation IMEs show its conversion window by adjust the position spcified by the system or the application. The CFS_FORCE_POSITION informs IMEs to stop this adjustment.



CFS__FORCE_POSITION

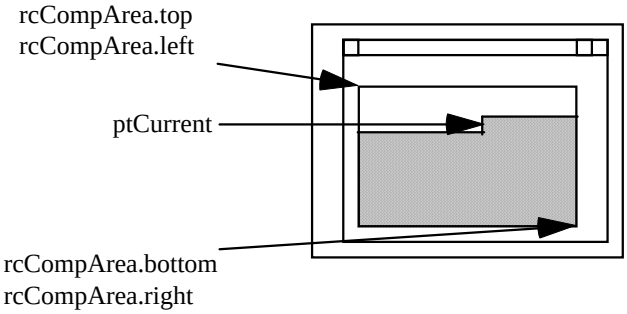
If this bit is off, some IME will adjust its position according to the position specifying in ptCurrentPos with CFS_POINT style.



CFS__FORCE_POSITION is off

CFS_RECT

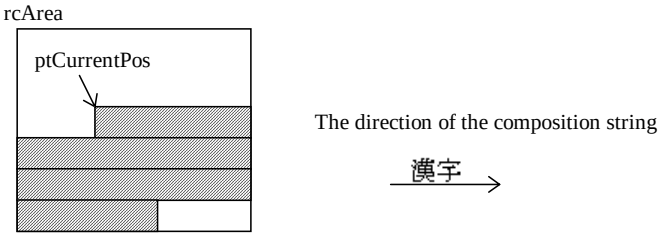
Same as CFS_POINT except that the bounding rectangle is specified by rcCompArea. The coordinates are relative to the upper left of the window.



CFS_RECT

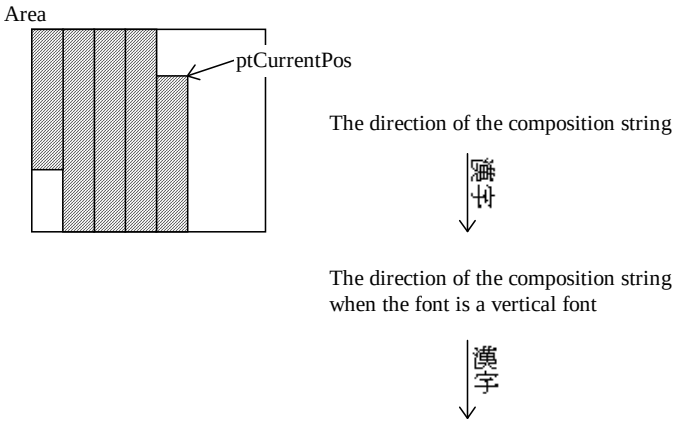
When the style of the COMPOSITIONFORM structure is CFS_POINT or CFS_FORCE_POINT, the IME will start to draw the composition string from the position that is specified by ptCurrentPos of the COMPOSITIONFORM structure that is given by the application. If the style has CFS_RECT, the composition string will be inside of the rectangle that is 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. And IME start to draw the composition string as follows.

The escapement of the composition font is 0



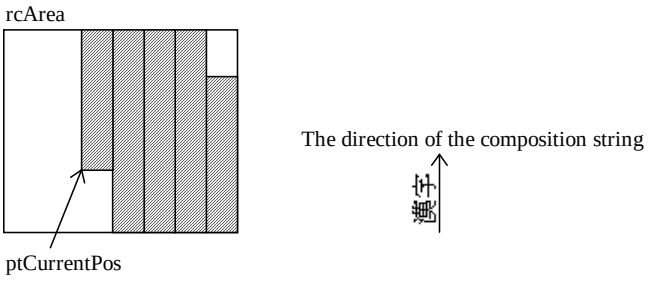
This is normal case. ptCurrentPos of the composition form structure points left and top of the string. All IMEs support this type.

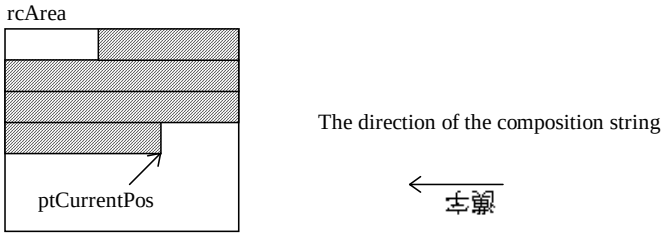
The escapement of the composition font is 2700.



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

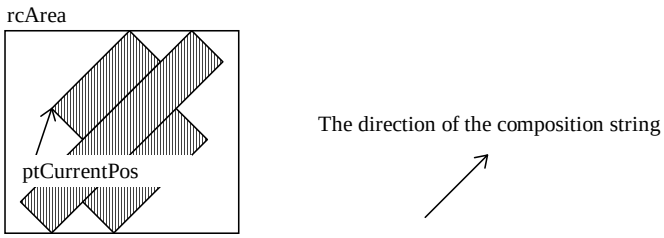
The escapement of the composition font is 900 or 1800





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

The escapement of the composition font is any value.



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

* UI_CAP_ROT90 and UI_CAPS_ANY are the option for the enhancement of the IME. UI_CAP_2700 is recomended.

This structure is used for IMC_SETCOMPOSITIONWINDOW / IMC_SETCANDIDATEPOS message.

7.5.3 CANDIDATEFORM Structure

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

dwIndex	Specifies the ID of candidate list. 0 is the first candidate list, 1 is the second one. It is up to 31.
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 componets, if the dwStyle is CFS_DEFAULT.
ptCurrentPos	Depends on dwStyle. When dwStyle = CFS_CANDIDATEPOS

		ptCurrentPos specifies the recommended position where the candidate list window should be displayed. When dwStyle = CFS_EXCLUDE ptCurrentPos specifies current position of the point of interest (typically the caret position).
	rcArea	Specifies a rectangle where no display is allowed for candidate windows in case of CFS_EXCLUDE.

This structure is used for IMC_GETCANDIDATEPOS / IMC_SETCANDIDATEPOS message.

7.5.4 STYLEBUF Structure

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

Member	Description
dwStyle	The style of register word.
szDescription	The description string of this style.

The style of the register string. It includes - IME_REGWORD_STYLE_EUDC : The string is in EUDC range.
IME_REGWORD_STYLE_USER_FIRST, IME_REGWORD_STYLE_USER_LAST : The constants range from IME_REGWORD_STYLE_USER_FIRST to IME_REGWORD_STYLE_USER_LAST are for private styles of the IME ISV. IME ISV can define its own style freely

7.5.5 SOFTKBDDATA Structure

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

Member	Description
uCount	The number of 256-word virtual key mapping to internal code array.
wCode[][256]	The 256-word virtual key mapping to internal code array. There may be more than one 256-word array.

It is possible for one type of soft keyboard using two 256-word array. One is for non shift state the other is for shift state. The soft keyboard can use two internal codes for displaying one virtual key.

8 Issues on Windows NT

8.1 Unicode interface

Adding to ANSI version of SYSTEM-IME interface originally supported by Windows 95, Windows NT supports Unicode interface for IME. Those IMEs that would like to communicate with the system by Unicode interface set the IME_PROP_UNICODE bit of fdwProperty field of the IMEINFO that is the first parameter of ImeInquire(). Although ImeInquire() is called to initialize IME for every thread of application process, IME is expected to return the same IMEINFO on a single system.

Unicode IME is not supported on Windows 95.

8.2 Security concern

8.2.1 named objects

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 on the system, the default security attribute that is 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, the first client process of IME may be WinLogon process that lets user log-on. Since WinLogon process belongs to SYSTEM account when doing the log-on session and will be alive until the system shutdown, named objects created by IME with the default security attribute during the log-on session can not be accessed from other processes that belong to a logged-on user.

Microsoft Windows NT-FE development team provides the sample source code that creates the security attribute appropriated for named object created by IME on Windows NT. By using the sample code, IME writers can create various named objects that can be accessed from all client processes of IME on the local system. The security attribute allocated by the sample code is per process. IME that frequently creates named objects may want to initialize the security attribute at the process attach time and free the security attribute at the process detach time. IME that doesn't create named object often may want to initialize the security attribute just before the named object creation and free the security attribute just after the object creation.

8.2.2 Winlogon

We recommend IME writers not to allow users configure IME during the log-on session. IME can check if the client process is WinLogon process that executes log-on session by seeing the IME_SYSTEMINFO_WINLOGON bit of the third parameter of ImeInquire() call. Since a user in the log-on session has not been granted yet, various information provided by the IME configure dialogs can be the security hole of the system. The system administrator may configure the system so that such IME can not be activated on the log-on session. Well-behaved IME will not allow user to open configure dialogs if the client process is WinLogon process.