Windows 95 / Windows NT

Multilingual IME Specification for Applications

Version 1.18

Last Revised: May 21, 1996

1 OVERVIEW

For Windows Far East versions, Input Method Editor (IME) had been introduced as an special process to help applications to generate double byte characters. IME aware applications had to call IME APIs and handle IME specific messages. However, there were several difficulties for programmers whose programs intended to interact with the IME through the interface.

- The design of APIs and messages were rather different from the other Windows APIs and messages. This caused difficulties to understand and use the interface.
- There was not enough information that an application was able to implement application specific User Interface (UI) rather than IME's User Interface. This caused UI inconsistency between IME and application.
- Each Windows Far East version had slight different IME APIs and messages. This caused difficulties to develop an application for Windows Far East versions simultaneously.

Windows 95 completely revised IME architecture to achieve better consistency to other Win32 APIs and messages, and integrated Japanese, Korean, Traditional Chinese and Simplified Chinese versions into single Win 32 interface.

As a result, an IME becomes a set of Win32 DLLs and Win32 applications can use the IME's functions through Win32 IME APIs, new IME messages and "IME" predefined global class.

- All necessary information between IME and Win32 application is maintained in at least one Input Context per application thread. An Win32 application is able to get or set necessary information in the Input Context through IME APIs.

  Each application thread may maintain different IME and Input Context.
- IME User Interface is defined as "IME" predefined global class. An Win32 application may use either IME's User Interface or application's UI through the new IME User Interface.

The following sections describe more detail information regarding Input Context and IME User Interface.

Starting at version 3.51, FarEast language version of Windows NT supports Win95 compatible Win32 IMM API function set. Adding to the ANSI version of IMM32 API functions supported by Win95, Unicode version of IMM32 API functions are also supported on Windows NT. Windows NT 4.0 IMM32 records appropriate error code when the function is failed. Windows NT 3.51 and Windows 95 IMM32 doesn't record error codes. Application can get the error code by calling GetLastError().

2 About Input Context

To handle the IME, Windows supplies the Input Context. When an application wants to do some actions on an IME, the application uses the Input Context and calls IMM functions. An Input Context is the context of the IME status. Using an Input Context prevents an application from interfering with changing the IME's status that is preferred by the application. This is multiple Input Context environment. An Input Context can be associated with each window. (A window can have its own input context.)

2.1 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.

2.2 Application 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.

2.3 Input Context Creation

An application uses IMM functions to create / maintain / utilize Input Context. An application can calls IMM to create its own/new Input Context and associate a window or more to this/new Input Context. (Before the window is destroyed, the previous/old associated Input Context need to be associated back to this window.)

2.3.1 Input Context Creation

To create new Input Context, an application may call **ImmCreateContext** function. The application can get the Input Context handle by calling **ImmCreateContext**. The Input Context handle is defined in type HIMC. When the application won't use the Input Context that was created by it any more, the application have to destroy the Input Context by calling **ImmDestroyContext** function.

2.3.2 Input Context Association

Before an Input Context works, the application has to associate with a window by calling **ImmAssociateContext** function. The **ImmAssociateContext** function establish a relation between the Input Context and the window. After this relation established, Windows can use the Input Context when the window gets focus and the keyboard input is effective for the window.

2.4 IMM functions

When a window of an application is being activated, system send a notification message WM_IME_SETCONTEXT to the application. For full IME aware applications, IMM provides a set of functions to retrieve Input Context. When an application receives related messages from IME, it can call these functions to get entire information of current IME status to display.

3 About IME User Interface

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

3.1 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 Far East Edition limits the user to abandon his / her operation when moving to another application.
- Since the IME User Interface 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 two kinds of IME window. One is created by System for DefWindowProc function especially for IME unaware program. This IME window is shared by all IME unaware windows of a thread, it is called the *Default IME window*. The other is created by IME aware applications, it is called the *Application IME window*.

3.2 IME class

Windows 95-FE will provide "IME" class by system default. System "IME" class will handle entire UI of IME. Applications can create its own Application IME window 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. The System or the application calls CreateWindowEx with “IME” class, and window style WS_POPUP|WS_DISABLE.

3.3 Default IME window

System creates *Default IME window* automatically for each thread. 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 it provide default behavior of IME User Interface 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. A Default IME window is shared by the windows that are in one thread.

3.4 Application IME window

Applications can create the *Application IME window by* using System "IME" class. An application that wants to control IME by itself basically should create *Application IME window.* The window handle of created *Application IME window* will be maintained by the application so that the application can pass all IME related messages to the window instance. Following example shows how an application will use *Application IME window.*

3.4.1 WM_IME_NOTIFY / WM_IME_CONTROL / WM_IME_COMPOSITION

Application uses given window handle of *Application IME window* to pass messages sent from IME / IMM. These messages notify an application whole information of conversion processing by IME. The application can have default behavior of IME User Interface by sending all these messages to it. If the application wants to change the default behavior, for instance, change the position of composition window, the application can also use WM_IME_CONTROL message to specify the status of the IME User Interface. WM_IME_COMPOSITION / WM_IME_NOTIFY message basically should be passed to the user interface window unless the application processes it.

3.4.2 ImmIsUIMessage()

ImmIsUIMessage() API will be provided for application developer to distinguish IME User Interface related messages in application window procedure, and if it is, process the message. When IME window handle is NULL, ImmIsUIMessage() API will return the result of checking without processing the message. The application will use this API to pick up the message and pass it to IME window. Below is a sample code fragment to use this API.

```
HWND hIMEWnd;  // The window handle of IME window.
long CALLBACK AppWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
               :

        If(ImmIsUIMessage(hIMEWnd, uMsg, wParam, lParam)==TRUE) {
        // The message was handled by IME window !
        // But you still can do what you want.
            switch(uMsg) {
            case WM_IME_COMPOSITION:
                if(lParam & GCS_RESULTSTR) {
                    hIMC=ImmGetContext(hWnd);
                    ImmGetCompositionString(hIMC, GCS_RESULTSTR, lpBufResult, dwBufLen);
                    ImmReleaseContext(hWnd, hIMC);
                 }
                 break;
             default   break;
            }
            return 0;
        }
         else {
         // The message was not handled by IME window.
             switch(uMsg) {
                 :
             }
         }
}
```

3.5 UI window

System IME Class handles whole control messages from IME and Application and pass some messages to the UI window. The UI class that will be provided by each IME should be responsible for IME unique User Interface functionality. The IME window 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.

3.6 Components of the UI window

The UI window of an IME can create different windows for its own User Interface, these window are called Components of the UI window. For example, it can includes the status window, composition window, and the candidate list.

4 Using INPUT CONTEXT and IME USER INTERFACE

4.1 Getting the result string

There are some ways to get the string that is determined by an IME. Basically applications wait some messages to get the result string.

4.1.1 Using WM_CHAR

When applications get **WM_CHAR** message, applications can get the character code that is one byte in wParam. When a DBCS character is determined by IME, **WM_CHAR** message will be sent twice. The wParam of the first **WM_CHAR** message is the lead byte of the DBCS character. And the wParam of the second **WM_CHAR** message is the second byte of the DBCS character. When a string is determined by IME, applications will get **WM_CHAR** messages for each byte of the string.

4.1.2 Using WM_IME_CHAR

When applications get **WM_IME_CHAR** message, applications can get the character code that is one or two bytes in wParam. When a DBCS character is determined by IME, **WM_IME_CHAR** message will be set once. The wParam of **WM_IME_CHAR** message is the code of the DBCS character.

```
long CALLBACK AppWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    WORD wChar;

    switch(uMsg) {
        case WM_IME_CHAR:
            wChar = LOWORD(wParam);
            if (IsDBCSLeadByte(HIBYTE(wChar) == TRUE) {
                // wChar is a DBCS character.
            }
            else {
                // wChar is a SBCS character.
            }
            break;
        default:
            return DefWindowProc(hWnd, uMsg, wParam, lParam);
    }
}
```

When applications handle **WM_IME_CHAR** message, the **WM_CHAR** message for the DBCS character that is specified by wParam of the **WM_IME_CHAR** message will not be sent to applications.

4.1.3 Using WM_IME_COMPOSITION

When applications get WM_IME_COMPOSITION message, applications may get the result string at one time. If lParam has GCS_RESULTSTR bit, the input context that is associated with a window that receive WM_IME_COMPOSITION has the result string.

```
long CALLBACK AppWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    HIMC hIMC:
    HGLOBAL hstr;
    LPSTR lpstr;
    DWORD dwSize;

    switch(uMsg) {
        case WM_IME_COMPOSITION:
            if (lParam & GCS_RESULTSTR){
                if (hIMC = ImmGetContext(hWnd)){
                    // Get the size of the result string.
                    dwSize = ImmGetCompositionString(hIMC,GCS_RESULTSTR,NULL,0);

                    // Here, you can allocate the memory for string buffer with dwSize.
                    dwSize++;     // Now dwSize is the size of buffer including the
terminating null character.

                    hstr = GlobalAlloc(GHND,dwSize);
```

```
                    if (hstr == NULL)
                        MyError(ERROR_GLOBALALLOC):

                    lpstr = GlobalLock(hstr);
                    if (lpstr == NULL)
                        MyError(ERROR_GLOBALLOCK):

                    // Get the result strings that is generated by IME into lpstr.
                    ImmGetCompositionString(hIMC,GCS_RESULTSTR,lpstr,dwSize);
                    ImmReleaseContext(hWnd,hIMC);

                    // Here, Do something with result string.

                    GlobalUnlock(hstr);
                    GlobalFree(hstr);
                }
            }
            break;
        default
            return DefWindowProc(hWnd, uMsg, wParam, lParam);
    }
}
```

When applications handle **WM_IME_COMPOSITION** message and lParam has the GCS_RESULTSTR bit, the **WM_IME_CHAR** message for the result string will not be sent to applications.

4.2 Changing the status of IME

4.2.1 Changing the open status

Using **ImmGetOpenStatus** function, applications can get the current open status of the input context.

Using **ImmSetOpenStatus** function, applications can set the current open status of the input context as applications like.

```
HIMC hIMC;
BOOL bOpen;
if (hIMC = ImmGetContext(hWnd)){
    // Get the status of the input context.
    bOpen = ImmGetOpenStatus(hIMC);
    // Set the invert status of the current status.
    ImmSetOpenStatus(hIMC, !bOpen);
    ImmReleaseContext(hWnd,hIMC);
}
```

4.2.2 Changing the conversion status

Applications can get the current conversion mode and sentence mode by calling ImmGetConversionStatus function. To set the new status, applications should call ImmGetConversionStatus to get the current status at first. To make the new conversion status, applications set some bits of the values that is the result of ImmGetConversionStatus function. ImmSetConversionStatus function sets the status that is specified by applications into input context.

```
HIMC hIMC;
```

```
DWORD dwConvMode;
DWORD dwSentence;

if (hIMC = ImmGetContext(hWnd)){
    // Get the status of the input context.
    ImmGetConversionStatus(hIMC, &fdwConversion, &fdwSentence);

    // Make the value for full shape input mode.
    fdwConversion |= IME_CMODE_FULLSHAPE;

    // Set new status of the current status.
    ImmSetOpenStatus(hIMC, fdwConversion, fdwSentence);
    ImmReleaseContext(hWnd,hIMC);
}
```

4.3 Drawing the composition string by applications

When applications want to draw the composition string without IME User Interface, application should handle WM_IME_COMPOSITION, WM_IME_STARTCOMPOSITION and WM_IME_ENDCOMPOSITION messages by itself. If lParam has GCS_COMPSTR bit, the input context that is associated with a window that receive WM_IME_COMPOSITION has the composition string.

```
long CALLBACK AppWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    HIMC hIMC:
    LPSTR lpCompStr;
    LPBYTE lpbAttr;
    LPSTR lpResultStr;
    DWORD dwStrSize;
    DWORD dwAttrSize;

    switch(uMsg) {
        case WM_IME_COMPOSITION:
           if (lParam & GCS_RESULTSTR){
                if (hIMC = ImmGetContext(hWnd)){
                     // Get the size of the result string.
                     dwSize = ImmGetCompositionString(hIMC,GCS_RESULTSTR,NULL,0);

                     // Here, you can allocate the memory for string buffer with dwSize.

                     // Get the result strings that is generated by IME into lpstr.
                     ImmGetCompositionString(hIMC,GCS_RESULTSTR,lpResultStr,dwSize);

                     // Drawing the result string.

// Here, you can free the memory for string buffer.

                     // Release input context.
                     ImmReleaseContext(hWnd,hIMC);
                }
           }
            if (lParam & (GCS_COMPSTR | GCS_COMPATTR)){
                if (hIMC = ImmGetContext(hWnd)){
                     // Get the size of composition string and attribute information.
                     dwStrSize = ImmGetCompositionString(hIMC,GCS_COMPSTR,NULL,0);
                     dwAttrSize = ImmGetCompositionString(hIMC,GCS_COMPATTR,NULL,0);

                     // Here, you can allocate the memory for string buffer with dwSize.
```

```
                    // Get the composition strings that is generated by IME into
lpCompStr.
                    ImmGetCompositionString(hIMC,GCS_COMPSTR,lpCompStr,dwSize);
                    ImmGetCompositionString(hIMC,GCS_COMPATTR,lpAttrStr,dwSize);

                    // Drawing text with attribute information.

                    // Here, you can free the memory for string buffer.

                    // Release input context.
                    ImmReleaseContext(hWnd,hIMC);

                }
            }
             break;
       case WM_IME_SETCONTEXT:
            // Handle the setting context.
            // Need to remove the bits of lParam.
            // This window can draw the composition string.
            lParam &= ~(ISC_SHOWUICOMPOSITIONWINDOW);
            return DefWindowProc(hWnd, uMsg, wParam, lParam);

        default
             return DefWindowProc(hWnd, uMsg, wParam, lParam);
    }
}
```

4.4 Handling IME Window

4.4.1 Creating IME window and using ImmIsUIMessage

```
HWND hIMEWnd;  // The window handle of IME window.
long CALLBACK AppWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
               :

        If(ImmIsUIMessage(hIMEWnd, uMsg, wParam, lParam)==TRUE) {
        // The message was handled by IME window !
        // But you still can do what you want.
             switch(uMsg) {
             case WM_IME_COMPOSITION:
                 if(lParam & GCS_RESULTSTR) {
                     hIMC=ImmGetContext(hwnd);
                     ImmGetCompositionString(hIMC, GCS_RESULTSTR, lpBufResult, dwBufLen);
                     ImmReleaseContext(hwnd, hIMC);
                  }
                  break;
             default   break;
             }
             return 0;
        }
         else {
         // The message was not handled by IME window.
              switch(uMsg) {
                  case WM_CREATE:
                      // Creating IME window that is owned by this application.
                      hIMEWnd = CreateWindowEx("IME",                 // IME class.
                                               NULL,                  // No window title.
                                               WS_DISABLED | WS_POPUP,// Disabled window.
                                               0, 0, 0, 0,            // No need to set size.
                                               hWnd,         // Parent window.
                                               (int)NULL,
                                               (HINSTACE)GetWindowLong(hWnd,GWL_HINSTANCE),
                                               NULL);
                      break;

                  case WM_DESTORY:
                      DestroyWindow(hIMEWnd);
```

```
                    break;
                :
            }
        }
}
```

4.4.2 Controlling IME window

Using WM_IME_CONTROL message for IME window, applications can have the effects for IME User Interface. Applications can set the position of the composition string by sending the WM_IME_CONTROL message with IMC_SETCOMPOSITIONWINDOW. Application can set the font of the composition string by sending WM_IME_CONTROL message with IMC_SETCOMPOSITIONFONT and so on.

```
HWND hIMEWnd;  // The window handle of IME window.
long CALLBACK AppWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    COMPOSITIONFORM cf;
    CANDIDATEFORM cdCandForm;
    POINT ptMyPoint;

        If(ImmIsUIMessage(hIMEWnd, uMsg, wParam, lParam)==TRUE) {

            :
            :

        }
        else {
        // The message was not handled by IME window.
            switch(uMsg) {
                :

                    // Send WM_IME_CONTROL with IMC_GETCOMPOSITIONWINDOW to hIMEWnd.
                    SendMessage(hIMEWnd, WM_IME_CONTROL, IMC_GETCOMPOSITIONWINDOW, &cf);

                    // Send WM_IME_CONTROL with IMC_GETCANDIDATEPOS to hIMEWnd.
                    SendMessage(hIMEWnd, WM_IME_CONTROL, IMC_GETCANDIDATEPOS,
&cdCandForm);

                :
            }
        }
}
```

4.5 Making IME Full Aware Applications

The IME full aware applications are the applications that can draw any IME User Interface by themselves. They don’t use any IME window and don’t create IME window. The IME full aware must handle most of IME messages and do not pass them to DefWindowProc function.

```
HIMC hOldIMC;
long CALLBACK AppWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    COMPOSITIONFORM cf;
    POINT ptMyPoint;
    LOGFONT lf;

    // The message was not handled by IME window.
    switch(uMsg) {
        case WM_CREATE:
            hIMC = ImmCreateContext();
```

```
            hOldIMC = ImmAssociateContext(hWnd ,hIMC);
            break;

        case WM_IME_STARTCOMPOSITION:
           // Prepare to receive WM_IME_COMPOSITION message.
           break;

        case WM_IME_ENDCOMPOSITION:
           // Finish handling compositoin string.
           break;

        case WM_IME_COMPOSITION:
           // Get the composition string, the result string and the information to
           // display these strings.
           break;

        case WM_IME_SETCONTEXT:
           // Handle the setting context.
           // Need to remove the bits of lParam.
           // This window can draw the composition string and index 0 candidate list.
           lParam &= ~(ISC_SHOWUICOMPOSITIONWINDOW & ISC_SHOWUICANDIDATEWINDOW);
           return DefWindowProc(hWnd, uMsg, wParam, lParam);

        case WM_IME_NOTIFY:
           // Handle each IMN_ sub message.
           // In this message, the status of IME and candidate list should be displayed,
           // changed, and erased.
           switch (wParam)
           {
               case IMN_OPENCANDIDATE:
               case IMN_CHANGECANDIDATE:
               case IMN_CLOSECANDIDATE:
                   This application can draw only one candidate list which index is 0.
                   if (lParam == 0x01)    // Bit 0 is On, it is index 0.
                   {
                        // Draw the candidate list....
                   }
                   else
                       return DefWindowProc(hWnd, uMsg, wParam, lParam);
                   break;

               default:
                   // Make a notification to IME window.
                   return DefWindowProc(hWnd, uMsg, wParam, lParam);
           }

           break;

        case WM_IME_COMPOSITIONFULL:
           // Make sure the size for drawing the composition string.
           // Application should draw the composition string correctly.

           break;


            :
            :

        case WM_DESTROY:
            ImmAssociateContext(hWnd, hOldIMC);
            ImmDestroyContext(hIMC);
            break;


    }
}
```

4.6 About WM_IME_SETCONTEXT

The application's windows that draw the composition string by itself. This kind of windows hooks WM_IME_COMPOSITION (WM_IME_STARTCOMPOSITION, WM_IME_ENDCOMPOSITION) and does not call DefWindowProc() nor ImmIsUIMessage() with WM_IME_COMPOSITION.

```
long CALLBACK AppWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
        switch(uMsg) {

                case WM_IME_COMPOSITION:
                        // Handle All of WM_IME_COMPOSITION...
                        break;

                case WM_IME_STARTCOMPOSITION:
                        // Handle All of WM_IME_STARTCOMPOSITION...
                        break;

                case WM_IME_ENDCOMPOSITION:
                        // Handle All of WM_IME_ENDCOMPOSITION...
                        break;

                // Need to handle WM_IME_SETCONTEXT..
                case WM_IME_SETCONTEXT:
                        lParam &= ~ISC_SHOWUICOMPOSITIONWINDOW;
                        return DefWindowProc(hWnd, uMsg, wParam, lParam);

}
```

4.6.1 The applications that draw the composition string

This applications does not call DefWindowProc or ImmIsUIMessage for following messages.

- WM_IME_STARTCOMPOSITION
- WM_IME_ENDCOMPOSTION
- WM_IME_COMPOSITION

This application has to handle WM_IME_SETCONTEXT and need to clear ISC_SHOWUICOMPOSITIONWINDOW bit before calling DefWindowProc or ImmIsUIMessage.

4.6.2 The applications that draw the candidate lists.

This applications does not call DefWindowProc or ImmIsUIMessage for following messages.

- WM_IME_NOTIFY/IMN_OPENCANDIDATEWINDOW
- WM_IME_NOTIFY/IMN_CLOSECANDIDATEWINDOW
- WM_IME_NOTIFY/IMN_CHANGECANDIDATEWINDOW

If this application can draw all of the candidate lists, it has to handle WM_IME_SETCONTEXT and need to clear all of ISC_SHOWUICANDIDATEWINDOW bits before calling DefWindowProc or ImmIsUIMessage. It may use ISC_SHOWUIALLCANDIDATEWINDOW mask as follows

lParam &= ~ISC_SHOWUIALLCANDIDATEWINDOW.

If this application can draw only one candidate list which index is 2, it needs to clear (ISC_SHOWUICANDIDATEWINDOW << 2) bit of lParam before calling DefWindowProc or ImmIsUIMessage.

4.6.3 The applications that draw the guideline string.

This applications does not call DefWindowProc or ImmIsUIMessage for following messages.

- WM_IME_NOTIFY/IMN_GUIDELINE

This application has to handle WM_IME_SETCONTEXT and needs to clear ISC_SHOWUIGUIDEWINDOW before calling DefWindowProc or ImmIsUIMessage.

4.6.4 The applications that want to hide the soft keyboard.

This application has to handle WM_IME_SETCONTEXT and needs to clear ISC_SHOWUISOFTKBD before calling DefWindowProc or ImmIsUIMessage.

4.6.5 The application that use IME's UI.

Don't need to handle WM_IME_SETCONTEXT. This should call DefWindowProc or ImmIsUIMessage without modifying lParam.

5 About ImmSetCompositionString

5.1 Groceries

5.1.1 Clause

The clause consists of one or more characters. And the clause is a unit for conversion. Basically IME convert one clause at one time. All chars of one clause must have same attribute.

5.1.2 Target Clause.

The target clause is one of clause. And it is the clause that the end user is converting. The attribute must be 0x01 or 0x11.

5.1.3 CompositionString.

The composition string is the string that is not be determined. And this may be updated by IME conversion. The composition string must have one or more clause. The composition string may have one target clause. But at most there is only one target clause in the composition string at one time.

5.1.4 Attribute

The attribute have to be same length with the composition string.

5.2 About ImmSetCompositionString

Using ImmSetCompositionString, the application can set the composition string into IME that has the capability of SCS_CAP_COMPSTR. If the IME has the capability of SCS_CAP_MAKEREAD, the application can receive the reading of the composition string also. Please refer ImmGetProperty() API description for these capability bits.

When the application call ImmSetCompositionString, the application will receive WM_IME_COMPOSITION message and this message is the real result of ImmSetCompositionString.

5.3 SCS Flags of ImmSetCompositionString

5.3.1 SCS_SETSTR

If the lpComp is not NULL, lpComp is the pointer of the composition string that the application request.

The old composition string and reading of the composition string will be thrown away.

In some conversion mode, the IME may not accept this string. For example, if the IME is closed, this has no meaning.

IME may cut off the composition string if it is too long for IME.

IME will make attribute and clause information for the composition string.

If lpRead is NULL and IME has the capability of SCS_CAP_MAKEREAD, IME will make the reading of the composition string. Then if IME make the reading of the composition string, IME have to make the attribute and the clause information for the reading composition string.

The application will receive WM_IME_COMPOSITION message, if the IME accept the composition string. Then lParam of WM_IME_COMPOSITION will be (GCS_COMPSTR | GCS_COMPATTR | GCS_COMPCLAUSE) or (GCS_COMPSTR | GCS_COMPATTR | GCS_COMPCLAUSE | GCS_COMPREADSTR | GCS_COMPREADATTR | GCS_COMPREADCLAUSE).

If the lpRead is not NULL, lpRead is the pointer of the reading of the composition string that the application request.

If the application set the reading of the composition string, the application will receive WM_IME_COMPOSITION with (GCS_COMPSTR | GCS_COMPATTR | GCS_COMPCLAUSE | GCS_COMPREADSTR | GCS_COMPREADATTR | GCS_COMPREADCLAUSE).

The old composition string and reading of the composition string will be thrown away.

IME will make the new composition string from the reading of the composition string.

In Japanese case, the specified the reading have to be SBCS chars (Alphanumeric, Number or SBCS Katakana)

In Korean case, the specified reading have to be Hangeul DBCS characters.

In Traditional and Simplified Chinese cases, each IME may limit its reading characters into some range.

If the application set both lpComp and lpRead, IME can refer both information to set the composition string and generate WM_IME_COMPOSITION.

5.3.2 SCS_CHANGEATTR

Setting the new attribute.

If lpComp is not NULL, lpComp is the pointer of the composition attribute that the application request. This can be used for changing the target clause. The new attribute have to follow the Basic Rule.

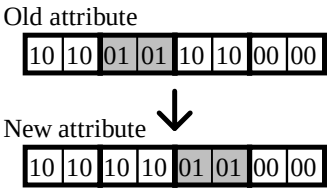
If lpRead is not NULL, lpRead is the pointer of the reading of the composition attribute that the application request. IME will change the attribute of the reading of the composition string of the composition string structure. IME have to sync the attribute of the composition string with this and generate WM_IME_COMPOSITION.

IME will synchronize the comp attribute and the comp read attribute with this new attribute.

If the application set both lpComp and lpRead, IME can refer both attribute information to set the composition string and generate WM_IME_COMPOSITION.

The application can change the attribute of the composition string. But the chars in one clause have to have same attribute. The target clause has the attribute ATTR_TARGET_CONVERTED or ATTR_TARGET_NOTCONVERTED.

As follows, the target clause can be changed by using SCS_CHANGEATTR.



The application can change from ATTR_TARGET_CONVERTED to ATTR_CONVERTED, ATTR_CONVERTED to ATTR_TARGET_CONVERTED, ATTR_TARGET_NOTCONVERTED to ATTR_INPUT or ATTR_INPUT to ATTR_TARGET_NOTCONVERTED.
The combination of the attribute that the application can change.

From\to	ATTR_INPUT	ATTR_TARGET_ COVERTED	ATTR_ CONVERTED	ATTR_TARGET_ NOTCONVERTED
ATTR_INPUT	O	X	X	O
ATTR_TARGET_ CONVERTED	X	O	O	X
ATTR_CONVERTED	X	O	O	X
ATTR_TARGET_ NOTCONVERTED	O	X	X	O
ATTR_INPUT_ERRO R	X	X	X	X

This means the application can not decide the clause is converted by IME or not. The ATTR_INPUT_ERROR only can be changed by end user input the character again or the application set a new character for it.

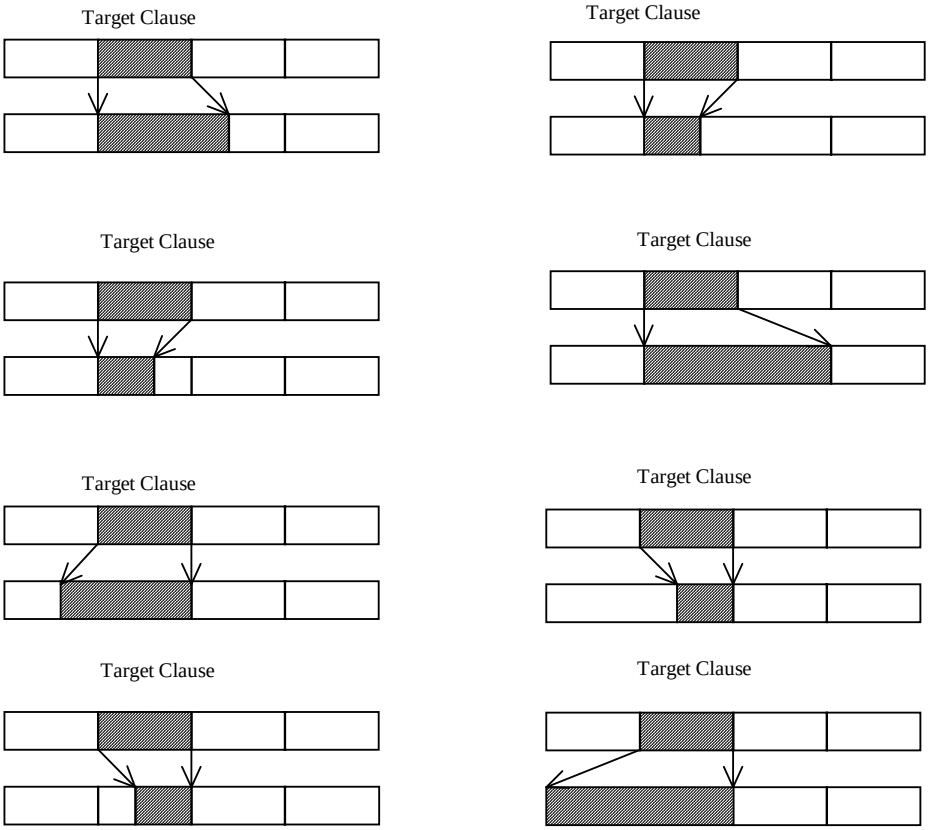
5.3.3 SCS_CHANGECLAUSE

The application can set the new clause information.

At one time, only one clause can be changed. And the changed clause have to be the target clause. Otherwise this function will cause error.

If the only lpComp is given, IME must synchronize the comp read clause information with this new clause information. The clause that is changed have to be target clause. If only lpRead is given, IME must synchronize it with the comp clause information. If both lpComp and lpRead are given, IME can refer both information. If necessary, IME may synchronize these information.

Acceptable type of the changing clause.



Only one boundary of the clause can be changed at one time.

Irregular type of changing clause

The two or more clauses are updated.
Non target clause is updated.

After calling this function, IME will generate WM_IME_COMPOSITION message. IME may update whole of composition string, attribute and clause information. At this time, the clause and attribute information except of the target clause may be changed. Then the application have to reset the attribute and clause information again. Because the IME will modify the clause of whole composition string by modifying one clause.

6 About ImmGetConversionList

An application or an IME may want to retrieve a conversion result without generates any IME related messages. It can call ImmGetConversionList to get the information.

6.1 GCL_CONVERSION

Normally, end user will type the reading string and get a candidate list of result strings by the user inferface. The application can get the same result strings without IME related messages by calling ImmGetConversionList with GCL_CONVERSION flag. This time no user interface will show up because the IME will not generate the IME related messages. The IME will treat the pass in source string as a reading string and convert it into a candidate list of result strings.

6.2 GCL_REVERSECONVERSION

If the end user want to know the reading string of another IME in the current active IME, the IME can get reading string of another IME by calling ImmGetConversionList with GCL_REVERSECONVERSION. The current active IME will pass the result string as a source string and expect another IME return the reading string. The end user could know how to type this result string in another IME by this reading string.

One possible example is one pronouciation IME user want to know how to type the same result string in a radical IME. In future, maybe one Far East country user may want to know how to get a result string from another Far East country IME.

It is possible that a result string mapping into multiple reading strings.

6.3 GCL_REVERSE_LENGTH

Before calling into ImmGetConversionList with GCL_REVERSECONVERSION, the application may want to confirm the length of this string can be accepted by an IME. For example, some IME can not convert sentence period to a reading string so the application may need to pass a string without the sentence period. Maybe some IME is limited by a maximum string length.

One example, the application find a wrong sentence while spelling check time and the application highlight it. The application can call ImmGetConversionList with GCL_REVERSE_LENGTH to get a length that IME can handle. The application will pass in the string by this length on calling into ImmGetConversionList with the GCL_REVERSECONVERSION flag. If the return value of GCL_REVERSECONVERSION is not 0, it means the IME could convert this result string to an reading string. The application may adjust the hightlight string by the length of GCL_REVERSE_LENGTH and call into ImmSetCompoistionString with the reading string of GCL_REVERSECONVERSION. Then the end user can reconvert this reading string or modify the reading string.

7 Data Structure

7.1 Structures used for communication with IME

7.1.1 CANDIDATELIST Structure

```
typedef struct _tagCANDIDATELIST {
    DWORD       dwSize;         // the size of this data structure including whole
                                dwOffset array and all candidate strings.

    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 appended after this field. IME can do
                                        // this by reallocating the hCandInfo memory handle.
                                        // So IME can access dwCandStrOffset[2] (3rd
                                        // candidate string) or dwCandStrOffset[5] (6th
                                        // candidate string).
    // CHAR             szCandStr[];    // the array of the candidate strings.
} CANDIDATELIST;
```

dwStyle	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 ImeGetCandidateList();

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 the other 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 ‘A140’+dwPageSize-1 and the scroll bar at one page.

ii) When the dwCount is larger than 1.

```
DWORD   dwSize;         → valid
DWORD   dwStyle;        → IME_CAND_CODE
DWORD   dwCount;        → 3
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 strings.
DWORD   dwOffset[1];    → Offset of the second strings.
DWORD   dwOffset[2];    → Offset of the third strings.
```

char	szCandStr[0]	→ "A140" string
char	szCandStr[1]	→ "A141" string
char	szCandStr[2]	→ "A142" string

At this time, in case of some special IME, 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
char	szCandStr[0x0]	→ "A103" string
char	szCandStr[0x1]	→ "A113" string
char	szCandStr[0x2]	→ "A123" string
	.....	
char	szCandStr[0x0F]	→ "A1F3" string

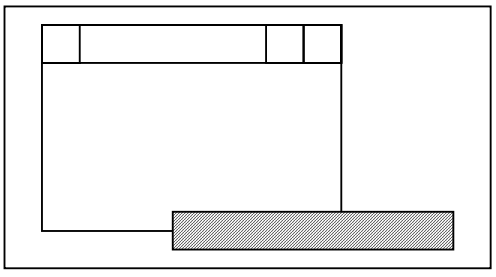
7.1.2 COMPOSITIONFORM Structure

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

The dwStyle member is specified by CFS_xxxx, and the ptCurrentPos and rcArea parameters will specify for the size and position of the bounding rectangle and the start position of the composition window. The composition window can be placed at outside of client area. This structure uses the client coordinates of a window's client area.

CFS_DEFAULT

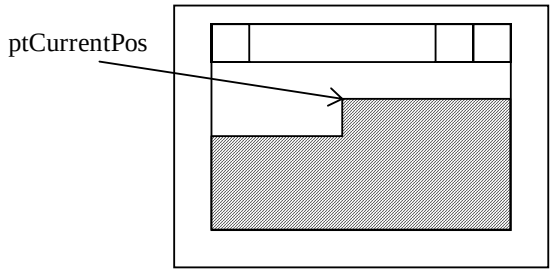
Moves the composition window to the default position.



CFS_DEFAULT

CFS_POINT

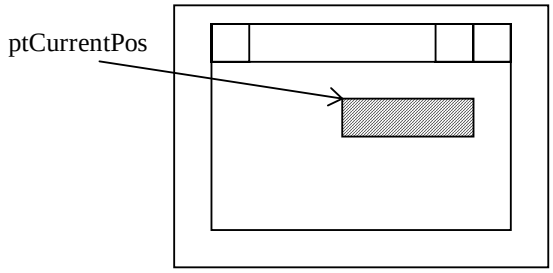
Instructs to display the composition window at the upper left designated by ptCurrentPos, in the window. ptCurrentPos indicates the coordinates relative to the upper left of the latter window.



CFS_POINT

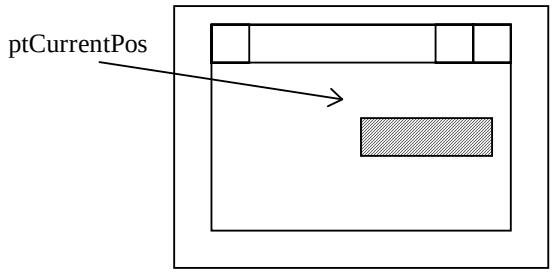
CFS_FORCE_POSITION

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



CFS__FORCE_POSITION

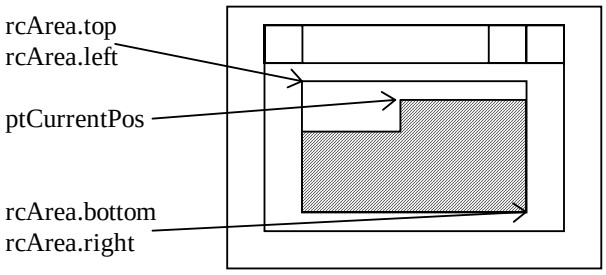
If this bit is off, some IME will adjust its position according to the position specifying in ptCurrentPos with CFS_POINT style.



CFS__FORCE_POSITION is off

CFS_RECT

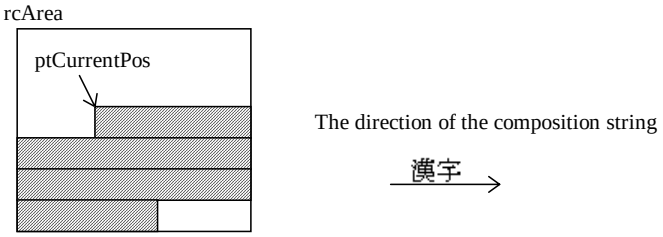
Same as CFS_POINT except that the bounding rectangle is specified by rcArea. The coordinates are relative to the upper left of the window.



CFS_RECT

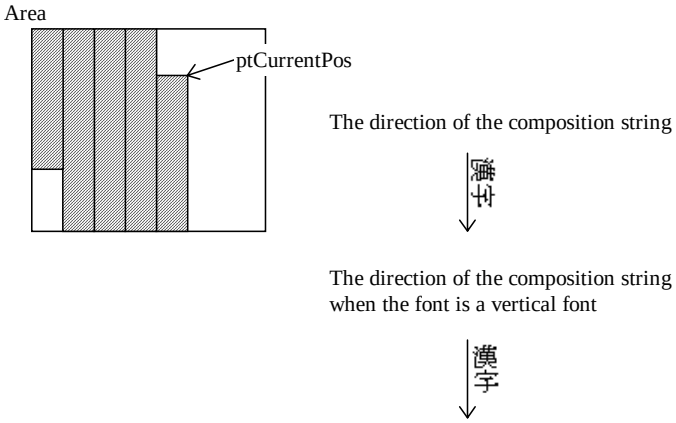
When the style of the COMPOSITIONFORM structure is CFS_POINT or CFS_FORCE_POINT, the IME will start to draw the composition string from the position that is specified by ptCurrentPos of the COMPOSITIONFORM structure that is given by the application. If the style has CFS_RECT, the composition string will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



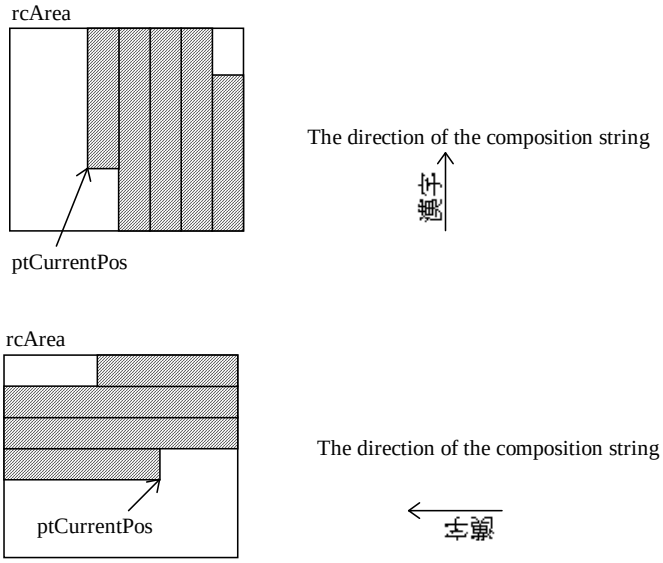
This is normal case. ptCurrentPos of the composition form structure points left and top of the string. All IMEs support this type.

The escapement of the composition font is 2700.



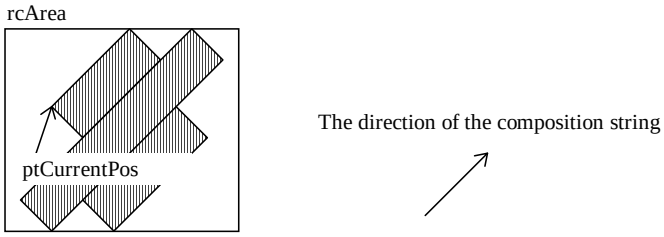
This is a case of a vertical writing. When the application provide the vertical writing, the application may set the 2700 escapement in the composition font that is set by ImmCompositionFont(). Then the composition string will be drawn downward. The IMEs that have UI_CAP_2700, UI_CAP_ROT90 or UI_CAP_ROTANY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

This structure is used for IMC_GETCOMPOSITIONWINDOW / IMC_SETCOMPOSITIONWINDOW message.

7.1.3 CANDIDATEFORM Structure

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

The dwStyle member is specified by CFS_xxxx, and the ptCurrentPos and rcArea parameters will specify for the size and position of the bounding rectangle and the start position of the candidate window. The candidate window can be placed at outside of client area. This structure uses the client coordinates of a window's client area.

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

7.1.4 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.

This structure is used for ImmGetRegisterWordStyle API.

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

7.1.5 REGISTERWORD Structure

```
typedef struct tagREGISTERWORD {
        LPTSTR      lpReading;
        LPTSTR      lpWord;
}REGISTERWORD;
```

Member	Description
lpReading	Reading info for the word to register.
szDescription	A word to register..

This structure is used for ImmConfigureIME API. When an application set a valid pointer to reading information or a word to register, they will appear in the configuration dialogbox of IME as initial values. An application can set NULL either of the member of this structure when it's not needed.

8 Support Old IMEs.

8.1 IME programs written for Windows 3.1

IME programs created for Windows 2.x/3.0/3.0A (Win3.1 IME) will run under Windows 95.
Win3.1 IME will be assigned to one keyboard layout as Win95 IME. System provide different hKL for each Win3.1 IME.
IMM APIs support to run Windows3.1 IME under Windows 95. The application does not need to use 3.1 old IME APIs for running 3.1 IME. The application can work with both Win95 IME and Win3.1 IME by using IMM APIs.

Korean Win3.1 IMEs are not supported by Korean Windows 95.

Japanese Windows NT 3.51 supports 32bit IME process type programs created for Windows NT 3.1/3.5. !6bit IME programs are not supported. Other language version of Windows NT doesn't support process type IME programs.
All version of Windows NT 4.0 or later version don't support process type IME programs.

8.2 IME programs written for Windows 2.x/3.0/3.0A

IME programs created for Windows 2.x/3.0/3.0A can not run under Windows 95 because of the following reasons.

There is no guarante that IME programs created for Window 2.x run in the protect mode.
IME programs created for Windows 3.0/3.0A (Win3.0 IME) does not use Win3.1 WINNLS APIs and they comunicate with the application directly by sending message. But Win3.0 IME can not generate the messages that Windows 95 applications expect. Win3.0 IMEs treat the handle and int types as 16-bit wide. Since Windows 95 applications treat them as 32-bit wide, Win3.0 IME can not comunicate with the applications that are created for Windows 95.

IME programs created for Windows 2.x/3.0/3.0A/3.1 can not run under Windows NT.

8.3 Getting the version of IME

The application can call ImmGetProperty with IGP_GETIMEVERSION to know the version of IME.

8.4 Functions that is not supported by Win3.1 IME.

There are some functions that are not supported when Win3.1 IMEs is selected in the application. Because 3.1 IME does not have these capabilities and did not need to support these functions for previous version of Windows.

8.4.1 The list of APIs and Messages that will fail when Win3.1 IME is selected.

ImmGetCandidateListCount()
ImmGetCandidateList()
ImmGetGuideLine()
ImmGetConvesionList()
IMC_OPENSTATUSWINDOW
IMC_CLOSESTATUSWINDOW
IMC_GETSTATUSWINDOWPOS
IMC_SETSTATUSWINDOWPOS

8.4.2 Other restriction.

The application can not get the clause information of the result string.
The application can not get the reading information of the composition string.