

Programming the Microsoft Agent Server Interface

ActiveX™ Technology for Interactive Software Agents



Last updated: October 1997
(updated info in Accessing Services Using Java)
Microsoft Corporation

Note: This document is provided for informational purposes only and Microsoft makes no warranties, either expressed or implied, in this document. The entire risk of the use or the results of this document remains with the user.

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property rights. Microsoft, MS, MS-DOS, Windows, Windows NT, and the Windows logo are either registered trademarks or trademarks of Microsoft Corporation in the U.S. and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

Contents

Introduction

Adding Microsoft Agent Functionality to Your Application

Loading Character and Animation Data

Creating a Notification Sink

Accessing Services Using Java

Reference

Introduction

Microsoft Agent provides services that enable you to program animated characters from an application. These services are implemented as an OLE Automation server. OLE Automation enables an application to control another application's object programmatically. This document assumes an understanding of the Component Object Model (COM) and OLE. For an introduction of these services, see the Programming Interface Overview. Sample programs are available at the Microsoft Agent Web site at <http://www.microsoft.com/workshop/prog/agent/>.

Adding Microsoft Agent Functionality to Your Application

To access Microsoft Agent services, create an instance of the server and request a pointer to a specific interface that the server supports using the standard COM convention. In particular, the COM library provides an API function, **CoCreateInstance**, that creates an instance of the object and returns a pointer to the requested interface of the object. Request a pointer to the **IAgent** interface in your **CoCreateInstance** call or in a subsequent call to **QueryInterface**.

The following code illustrates this in C/C++:

```
hRes = CoCreateInstance(CLSID_AgentServer,
                        NULL,
                        CLSCTX_SERVER,
                        IID_IAgent,
                        (LPVOID *) &pAgent);
```

If the Microsoft Agent server is running, this function connects to the server; otherwise, it starts up the server.

Functions that take pointers to BSTRs allocate memory using **SysAllocString**. It is the caller's responsibility to free this memory using **SysFreeString**.

Loading Character and Animation Data

Once you have a pointer to the **IAgent** interface, you can use the **Load** method to load a character and retrieve its **IDispatch** interface:

```
// Create a variant to store the full path of the character to load
VariantInit(&vPath);

vPath.vt = VT_BSTR;
vPath.bstrVal = SysAllocString(kpwszCharacter);

// Load the character

hRes = pAgent->Load(vPath, &lCharID, &lRequestID);

// Get its IDispatch interface

hRes = pAgent->GetCharacter(lCharID, &pdCharacter);
```

You can then use this information to request a pointer to the **IAgentCharacter**:

```

    // Query for IAgentCharacter

    hRes = pdCharacter->QueryInterface(IID_IAgentCharacter, (LPVOID
*) &pCharacter);

    // Release the IDispatch

    pdCharacter->Release();

```

You can use this interface to access the character's methods:

```

    // Show the character. The first parameter tells Microsoft
    // Agent to show the character by playing an animation.

    hRes = pCharacter->Show(FALSE, &lRequestID);

    // Make the character speak

    bszSpeak = SysAllocString(L"Hello World!");

    hRes = pCharacter->Speak(bszSpeak, NULL, &lRequestID);

    SysFreeString(bszSpeak);

```

When you no longer need Microsoft Agent services, such as when your client application shuts down, release its interfaces. Note that releasing the character interface does not unload the character. Call the **Unload** method to do this before releasing the **IAgent** interface:

```

    // Clean up

    if (pCharacter) {

        // Release the character interface

        pCharacter->Release();

        // Unload the character. NOTE: releasing the character
        // interface does NOT make the character go away. You must
        // call Unload.

        pAgent->Unload(lCharID);
    }

    // Release the Agent

    pAgent->Release();

    VariantClear(&vPath);

```

Creating a Notification Sink

To be notified of events by Microsoft Agent, you must implement the **IAgentNotifySink** interface, and create and register an object of that type following COM conventions:

```

// Create a notification sink

pSink = new AgentNotifySink;

pSink->AddRef();

// And register it with Microsoft Agent

hRes = pAgent->Register((IUnknown *)pSink, &lNotifySinkID);

```

Remember to unregister your notification sink when your application shuts down and releases Microsoft Agent's interfaces.

Accessing Services Using Java

You can also access Microsoft Agent services from a Java™ applet. Many of the functions accessible through the Microsoft Agent interfaces return values through parameters passed by reference. In order to pass these parameters from Java, it is necessary to create single-element arrays in your code and pass them as parameters to the appropriate function. If you're using Microsoft Visual J++™ and have run the Java Type Library Wizard on the Microsoft Agent server, refer to the summary.txt file to review which functions require array arguments. The procedure is similar to that in C; you use the **IAgent** interface to create an instance of the server, then load the character:

```

private IAgent          m_Agent = null;
private IAgentCharacter m_Merlin[] = {null};
private int             m_MerlinID[] = {-1};
private int             m_RequestID[] = {0};
private final String    m_CharacterPath = "c:\\agentx\\agtchared\\merlin.acs";

public void start()
{
    // Start the Microsoft Agent Server

    m_Agent = (IAgent) new AgentServer();

    try
    {
        // The filespec parameter of the Load method is a
        // COM variant to accept alternate Agent data providers.
        // We want a standard provider so we can just specify
        // the filespec for our character.

        Variant characterPath = new Variant();
        characterPath.putString(m_CharacterPath);

        // Load the character

        m_Agent.Load(characterPath,
                     m_MerlinID,
                     m_RequestID);
    }
}

```

The procedure is slightly different when loading characters from a HTTP remote location such as a Web site. In this case the **Load** method is asynchronous and will raise a COM exception of E_PENDING (0x8000000a). You will need to catch this exception and handle it correctly as is done in the following functions:

```
// Constants used in asynchronous character loads

private final int E_PENDING = 0x8000000a;
private final int NOERROR = 0;

// This function loads a character from the specified path.
// It correctly handles the loading of characters from
// remote sites.

// This sample doesn't care about the request id returned
// from the Load call. Real production code might use the
// request id and the RequestComplete callback to check for
// a successful character load before proceeding.

public int LoadCharacter(Variant path, int[] id)
{
    int requestid[] = {-1};
    int hRes = 0;

    try
    {
        // Load the character

        m_Agent.Load(path, id, requestid);
    }
    catch (com.ms.com.ComException e)
    {
        // Get the HRESULT

        hRes = e.getHResult();

        // If the error code is E_PENDING, we return NOERROR

        if (hRes == E_PENDING)
            hRes = NOERROR;
    }

    return hRes;
}

public void start()
{
    if (LoadCharacter(characterPath, m_MerlinID) != NOERROR)
    {
        stop();
        return;
    }

    // Other initialization code here
}
```

```

        .
        .
        .
    }

```

Then get the **IAgentCharacter** interface that enables you to access its methods:

```

// Get the IAgentCharacter interface for the loaded
// character by passing its ID to the Agent server.

m_Agent.GetCharacter(m_MerlinID[0], m_Merlin);

// Show the character

m_Merlin[0].Show(FALSE, m_RequestID);

// And speak hello

m_Merlin[0].Speak("Hello World!", "", m_RequestID);

```

Similarly, to be notified of events, you must implement the **IAgentNotifySink** interface, creating and registering an object of that type:

```

...
// Declare an Agent Notify Sink so that we can get
// notification callbacks from the Agent server.

private AgentNotifySink m_Sink = null;
private int             m_SinkID[] = {-1};

public void start()
{
    ...
    // Create and register a notify sink

    m_Sink = new AgentNotifySink();

    m_Agent.Register(m_Sink, m_SinkID);
    ...
    // Give our notify sink access to the character

    m_Sink.SetCharacter(m_Merlin[0]);
    ...
}

```

In order to access Microsoft Agent from a Java applet, you must generate Java classes that you then install with the applet. You can use the Visual J++ Java Type Library Wizard, for example, to generate these files. If you plan to host the applet on a Web page, you build a signed Java CAB that includes the generated class files and that downloads with the page. The class files are necessary to access the Microsoft Agent Server because it is a COM object that executes outside of the Java sandbox. To learn more about Trust-Based Security for Java, see <http://www.microsoft.com/java/security/>.

Reference

This reference contains the following sections:

Interfaces

Functions

Events

Interfaces

Microsoft Agent defines interfaces that allow applications to access its services, enabling an application to control the animation of a character, support user input events, and specify output.

All the Microsoft Agent interfaces are defined in header (.h) files.

IAgent

IAgent defines an interface that allows applications to load characters, receive events, and check the current state of the Microsoft Agent Server.

Methods in Vtable Order

<u>IAgent Methods</u>	<u>Description</u>
Load	Loads a character's data file.
Unload	Unloads a character's data file.
Register	Registers a notification sink for the client.
Unregister	Unregisters a client's notification sink.
GetCharacter	Returns the IAgentCharacter interface for a loaded character.
GetSuspended	Returns whether the server is currently suspended.

IAgent::GetCharacter

```
HRESULT GetCharacter(  
    long dwCharID // character ID  
);
```

Retrieves the **IAgentCharacter** for a loaded character.

- Returns S_OK to indicate the operation was successful.

DwCharID

The character's ID.

IAgent::GetSuspended

```
HRESULT GetSuspended(  
    long * pbSuspended // address of variable for suspended flag  
);
```

Retrieves whether the Microsoft Agent server is currently suspended.

- Returns S_OK to indicate the operation was successful.

pbSuspended

Address of a variable that receives TRUE if the Microsoft Agent server is in the suspended state and FALSE if not.

Microsoft Agent loads in a suspended state when a client application attempts to start it up after the user has previously quit (by choosing the Exit command on the Microsoft Agent taskbar icon). In the suspended state Microsoft Agent handles connection requests, but returns failure on any animation methods. Therefore, a character cannot be displayed in this state. Client applications can advise users to restart the server (by choosing Restart on the taskbar pop-up menu), but cannot restart the server directly.

IAgent::Load

```
HRESULT Load(  
    VARIANT vLoadKey, // data provider  
    long * pdwCharID, // address of a variable for character ID  
    long * pdwReqID // address of a variable for request ID  
);
```

Loads a character into the **Characters** collection.

- Returns S_OK to indicate the operation was successful.

vLoadKey

A variant datatype that must be one of the following:

<i>filespec</i>	The local file location of the specified character's definition file.
<i>URL</i>	The HTTP address for the character's definition file.
<i>provider</i>	An alternate character definition provider.

pdwCharID

Address of a variable that receives the character's ID.

pdwReqID

Address of a variable that receives the **Load** request ID.

Microsoft Agent's data provider supports loading character data stored as a single structured file (.ACS) with character data and animation data together, or as separate character data (.ACF) and animation (.AAF) files. Generally, use the single structured .ACS file to load a character that is stored on a local disk drive or network and accessed using conventional file protocol (such as UNC pathnames). Use the separate .ACF and .AAF files when you want to load the animation files individually from a remote site where they are accessed using HTTP protocol.

For .ACS files, using the **Load** method provides access a character's animations. For .ACF files, you also use the **Prepare** method to load animation data. The **Load** method does not support downloading .ACS files from an HTTP site.

Loading a character does not automatically display the character. Use the **Show** method first to make the character visible.

The *vLoadKey* parameter also enables you specify your own data provider (that would be loaded separately) that can have its own methods for loading animation data. You need to create a data provider object only if you supply character data in special formats.

IAgent::Register

```
HRESULT Register(  
    IUnknown * punkNotifySink // IUnknown address for client notification sink  
    long * pdwSinkID           // address of the notification sink ID  
);
```

Registers a notification sink for the client application.

- Returns S_OK to indicate the operation was successful.

IUnknown

Address of **IUnknown** for your notification sink interface.

pdwSinkID

Address of notification sink ID (used to unregister the notification sink).

You need to register your notification sink (also known as a notify sink or event sink) to receive events from the Microsoft Agent server.

See also **IAgent::Unregister**

IAgent::UnLoad

```
HRESULT UnLoad(  
    long * dwCharID //character ID  
);
```

Unloads the character data for the specified character from the **Characters** collection.

- Returns S_OK to indicate the operation was successful.

dwCharID

The character's ID.

Use this method when you no longer need a character, to free up memory used to store information about the character. If you access the character again, use the **Load** method.

See also **IAgent::Load**

IAgent::Unregister

```
HRESULT Unregister(  
    long dwSinkID //notification sink ID  
);
```

Unloads the character data for the specified character from the **Characters** collection.

- Returns S_OK to indicate the operation was successful.

dwSinkID

The notification sink ID.

Use this method when you no longer need Microsoft Agent services, such as when your application shuts down.

See also **IAgent::Register**

IAgentCharacter

IAgentCharacter defines an interface that allows applications to query character properties and play animations.

Methods in Vtable Order

IAgentCharacter Methods	Description
GetVisible	Returns whether the character (frame) is currently visible.
SetPosition	Sets the position of the character frame.
GetPosition	Returns the position of the character frame.

IAgentCharacter Methods	Description
SetSize	Sets the size of the character frame.
GetSize	Returns the size of the character frame.
GetName	Returns the name of the character.
GetDescription	Returns the description for the character.
GetTTSSpeed	Returns the current TTS output speed setting for the character.
GetTTS Pitch	Returns the current TTS pitch setting for the character.
Activate	Sets whether a client is active or a character is topmost.
SetIdleOn	Sets the server's idle processing.
GetIdleOn	Returns the setting of the server's idle processing.
Prepare	Retrieves animation data for the character.
Play	Plays a specified animation.
Stop	Stops an animation for a character.
StopAll	Stops all animations for a character.
Wait	Holds the character's animation queue.
Interrupt	Interrupts a character's animation.
Show	Displays the character and plays the character's Showing state animation.
Hide	Plays the character's Hiding state animation and hides the character's frame.
Speak	Plays spoken output for the character.
MoveTo	Moves the character frame to the specified location.
GestureAt	Plays a gesturing animation based on the specified location.
GetMoveCause	Retrieves the cause of the character's last move.
GetVisibilityCause	Retrieves the cause of the last change to the character's visibility state.
HasOtherClients	Retrieves whether the character has other current clients.
SetSoundEffectsOn	Determines whether a character animation's sound effects play.
GetSoundEffectsOn	Retrieves whether a character's sound effects setting is enabled.
SetName	Sets the character's name.
SetDescription	Sets the character's description.
GetExtraData	Retrieves additional data stored with the character.

IAgentCharacter::Activate

```
HRESULT Activate(
    short sState, // topmost character or client setting
);
```

Sets whether a client is active or a character is topmost.

- Returns S_OK to indicate the operation was successful.

- Returns `S_FALSE` to indicate the operation was not successful.

sState

You can specify the following values for this parameter:

- 0 Set as not the active client.
- 1 Set as the active client.
- 2 Make the topmost character.

When multiple characters are visible, only one of the characters receives speech input at a time. Similarly, when multiple client applications share the same character, only one of the clients receives mouse input (for example, Microsoft Agent control click or drag events) at a time. The character set to receive mouse and speech input is the topmost character and the client that receives input is the character's active client. (The topmost character's window also appears at the top of the character window's z-order.) Typically, the user determines which character is topmost by explicitly selecting it. However, topmost activation also changes when a character is shown or hidden (the character becomes or is no longer topmost, respectively.)

You can also use this method to explicitly manage when your client receives input directed to the character, such as when your application itself becomes active. For example, setting **State** to 2 makes the character topmost, and your client receives all mouse and speech input events generated from user interaction with the character. Therefore, it also makes your client the input-active client of the character. However, you can also set the active client for a character without making the character topmost, by setting **State** to 1. This enables your client to receive input directed to that character when the character becomes topmost. Similarly, you can set your client to not be the active client (to not receive input) when the character becomes topmost, by setting **State** to 0. You can determine if a character has other current clients using **IAgentCharacter::HasOtherClients**.

Avoid calling this method directly after a **Show** method. **Show** automatically sets the input-active client. When the character is hidden, the **Activate** call may fail if it gets processed before the **Show** method completes.

If you call this method to a function, it returns a Boolean value that indicates whether the method succeeded. Attempting to call this method with the **State** parameter set to 2 when the specified character is hidden will fail. Similarly, if you set **State** to 0 and your application is the only client, this call fails because a character must always have a topmost client.

See also **IAgentCharacter::HasOtherClients**

IAgentCharacter::GestureAt

```
HRESULT GestureAt(
    short x,           // x-coordinate of specified location
    short y,           // y-coordinate of specified location
    long * pdwReqID    // address of a request ID
);
```

Plays the associated **Gesturing** state animation based on the specified location.

- Returns S_OK to indicate the operation was successful. When the function returns, *pdwReqID* contains the ID of the request.

x

The x-coordinate of the specified location in pixels, relative to the screen origin (upper left).

y

The y-coordinate of the specified location in pixels, relative to the screen origin (upper left).

pdwReqID

Address of a variable that receives the **GestureAt** request ID.

The server automatically determines and plays the appropriate gesturing animation based on the character's current position and the specified location. When using the HTTP protocol to access character and animation data, use the **Prepare** method to ensure that the animations are available before calling this method.

IAgentCharacter::GetDescription

```
HRESULT GetDescription(
    BSTR * pbszDescription    // address of buffer for character description
);
```

Retrieves the description of the character.

- Returns S_OK to indicate the operation was successful.

pbszDescription

The address of a BSTR that receives the value of the description for the character. A character's description is defined when it is compiled with the Microsoft Agent Character Editor. The description setting is optional and may not be supplied for all characters.

IAgentCharacter::GetExtraData

```
HRESULT GetExtraData(
    BSTR * pbszExtraData    // address of buffer for additional character data
);
```

Retrieves additional data stored as part of the character.

- Returns S_OK to indicate the operation was successful.

pbszExtraData

The address of a BSTR that receives the value of the additional data for the character. A character's additional data is defined when it is compiled with the Microsoft Agent Character Editor. A character developer can supply this string by editing the .ACD file for a character. The setting is optional and may not be supplied for all characters, nor can the

data be defined or changed at run time. In addition, the meaning of the data supplied is defined by the character developer.

IAgentCharacter::GetIdleOn

```
HRESULT GetIdleOn(  
    long * pbOn // address of idle processing flag  
);
```

Indicates the automatic idle processing state for a character.

- Returns S_OK to indicate the operation was successful.

pbOn

Address of a variable that receives TRUE if the Microsoft Agent server automatically plays **Idling** state animations for a character and FALSE if not.

See also **IAgentCharacter::SetIdleOn**

IAgentCharacter::GetMoveCause

```
HRESULT GetMoveCause(  
    long * pdwCause // address of variable for cause of character move  
);
```

Retrieves the cause of the character's last move.

- Returns S_OK to indicate the operation was successful.

pdwCause

Address of a variable that receives the cause of the character's last move and will be one of the following:

const unsigned short NeverMoved = 0;	Character has not been moved.
const unsigned short UserMoved = 1;	User dragged the character.
const unsigned short ProgramMoved = 2;	Your application moved the character.
const unsigned short OtherProgramMoved = 3;	Another application moved the character.

See also **IAgentNotifySink::Move**

IAgentCharacter::GetName

```
HRESULT GetName(  
    BSTR * pbszName // address of buffer for character name
```

```
);
```

Retrieves the name of the character.

- Returns S_OK to indicate the operation was successful.

pbszName

The address of a BSTR that receives the value of the name for the character. A character's default name is defined when it is compiled with the Microsoft Agent Character Editor. The name setting is optional and may not be supported for all characters. You can also set the character's name using **IAgentCharacter::SetName**; however, this changes the name for all current clients of the character.

See also **IAgentCharacter::SetName**

IAgentCharacter::GetPosition

```
HRESULT GetPosition(  
    long * plLeft,    // address of variable for left edge of character  
    long * plTop      // address of variable for top edge of character  
);
```

Retrieves the character's animation frame position.

- Returns S_OK to indicate the operation was successful.

plLeft

Address of a variable that receives the screen coordinate of the character animation frame's left edge in pixels, relative to the screen origin (upper left).

plTop

Address of a variable that receives the screen coordinate of the character animation frame's top edge in pixels, relative to the screen origin (upper left).

Even though the character appears in an irregularly shaped region window, the location of the character is based on its rectangular animation frame.

See also **IAgentCharacter::SetPosition**, **IAgentCharacter::GetSize**

IAgentCharacter::GetSize

```
HRESULT GetSize(  
    long * plWidth,    // address of variable for character width  
    long * plHeight    // address of variable for character height  
);
```

Retrieves the size of the character's animation frame.

- Returns S_OK to indicate the operation was successful.

plWidth

Address of a variable that receives the width of the character animation frame in pixels, relative to the screen origin (upper left).

plHeight

Address of a variable that receives the height of the character animation frame in pixels, relative to the screen origin (upper left).

Even though the character appears in an irregularly shaped region window, the location of the character is based on its rectangular animation frame.

See also **IAgent::SetSize**

IAgentCharacter::GetSoundEffectsOn

```
HRESULT GetSoundEffectsOn(  
    long * pbOn // address of variable for sound effects setting  
);
```

Retrieves whether the character's sound effects setting is enabled.

- Returns S_OK to indicate the operation was successful.

pbOn

Address of a variable that receives TRUE if the character's sound effects setting is enabled, FALSE if disabled.

The character's sound effects setting determines whether sound effects compiled as a part of the character are played when you play an associated animation. The setting is subject to the user's global sound effects setting in

IAgentAudioOutputProperties::GetUsingSoundEffects.

See also **IAgentCharacter::SetSoundEffectsOn**,
IAgentAudioOutputProperties::GetUsingSoundEffects

IAgentCharacter::GetTTS Pitch

```
HRESULT GetTTPitch(  
    long * pdwPitch // address of variable for character TTS pitch  
);
```

Retrieves the character's TTS output pitch setting.

- Returns S_OK to indicate the operation was successful.

pdwPitch

Address of a variable that receives the character's current TTS pitch setting in Hertz.

Although your application cannot write this value, you can include pitch tags in your output text that will temporarily increase the pitch for a particular utterance. This method applies only to characters configured for TTS output. If the speech synthesis (TTS) engine is not enabled (or installed) or the character does not support TTS output, this method returns zero (0).

IAgentCharacter::GetTTSSpeed

```
HRESULT GetTTSSpeed(  
    long * pdwSpeed // address of variable for character TTS output speed  
);
```

Retrieves the character's TTS output speed setting.

- Returns S_OK to indicate the operation was successful.

pdwSpeed

Address of a variable that receives the output speed of the character in words per minute.

Although your application cannot write this value, you can include speed tags in your output text that will temporarily speed up the output for a particular utterance.

This property returns the current speaking output speed setting for the character. For characters using TTS output, the property returns the actual TTS output for the character. If TTS is not enabled or the character does not support TTS output, the setting reflects the user setting for output speed.

IAgentCharacter::GetVisibilityCause

```
HRESULT GetVisibilityCause(  
    long * pdwCause // address of variable for cause of character visible state  
);
```

Retrieves the cause of the character's visible state.

- Returns S_OK to indicate the operation was successful.

pdwCause

Address of a variable that receives the cause of the character's last visibility state change and will be one of the following:

const unsigned short NeverShown = 0;	Character has not been shown.
const unsigned short UserHid = 1;	User hid the character.
const unsigned short UserShowed = 2;	User showed the character.
const unsigned short ProgramHid = 3;	Your application hid the character.
const unsigned short ProgramShowed = 4;	Your application showed the character.

const unsigned short OtherProgramHid = 5;	Another application hid the character.
const unsigned short OtherProgramShowed = 6;	Another application showed the character.

See also **IAgentNotifySink::Hide**, **IAgentNotifySink::Show**

IAgentCharacter::GetVisible

```
HRESULT GetVisible(
    long * pbVisible // address of variable for character Visible setting
);
```

Determines whether the character's animation frame is currently visible.

- Returns S_OK to indicate the operation was successful.

pbVisible

Address of a variable that receives TRUE if the character's frame is visible and FALSE if hidden.

You can use this method to determine whether the character's frame is currently visible. To make a character visible, use the **Show** method. To hide a character, use the **Hide** method.

IAgentCharacter::HasOtherClients

```
HRESULT HasOtherClients(
    long * pbHasOtherClients // address of variable for whether character has
);                          // other clients
```

Retrieves whether a character has other clients.

- Returns S_OK to indicate the operation was successful.

pbHasOtherClients

Address of a variable that receives TRUE if the character has other clients and FALSE if not.

IAgentCharacter::Hide

```
HRESULT Hide(
    long bFast,          // play Hiding state animation flag
    long * pdwReqID // address of request ID
);
```

Hides the character.

- Returns S_OK to indicate the operation was successful. When the function returns, *pdwReqID* contains the ID of the request.

bFast

Hiding state animation flag. If this parameter is TRUE, the **Hiding** animation does not play before the character frame is hidden; if FALSE, the animation plays.

pdwReqID

Address of a variable that receives the **Hide** request ID.

The server queues the animation associated with the **Hide** method in the character's queue. This allows you to use it to hide the character after a sequence of other animations. You can play the action immediately by using the **Stop** method before calling the **Hide** method.

When using the HTTP protocol to access character and animation data, use the **Prepare** method to ensure the availability of the **Hiding** state animation before calling this method.

Hiding a character can also result in triggering the **ActivateInput** event of another visible character.

Hidden characters cannot access the audio channel. The server will pass back a failure status in the **RequestComplete** event if you generate an animation request and the character is hidden.

See also **IAgentCharacter::Show**

IAgentCharacter::Interrupt

```
HRESULT Interrupt(
    long dwReqID,      // request ID to interrupt
    long * pdwReqID    // address of request ID
);
```

Interrupts the specified animation (request) of another character.

- Returns S_OK to indicate the operation was successful. When the function returns, *pdwReqID* contains the ID of the request.

dwReqID

An ID of the request to interrupt.

pdwReqID

Address of a variable that receives the **Interrupt** request ID.

You can use this method to sync up animation between characters. For example, if another character is in a looping animation, this method will stop the looping animation and start the next animation in the character's queue.

Interrupt halts the existing animation, but does not flush the character's animation queue. It starts the next animation in the character's queue. To halt and flush a character's queue, use the **Stop** method.

You cannot use this method to have a character interrupt itself because the Microsoft Agent server queues the **Interrupt** method in the character's animation queue. Therefore, you can only use **Interrupt** to halt the animation of another character you have loaded.

IAgentCharacter::MoveTo

```
HRESULT MoveTo(  
    short x,           // x-coordinate of new location  
    short y,           // y-coordinate of new location  
    long lSpeed,        // speed to move the character  
    long * pdwReqID    // address of request ID  
);
```

Plays the associated **Moving** state animation and moves the character frame to the specified location.

- Returns S_OK to indicate the operation was successful. When the function returns, this variable contains the ID of the request.

x

The x-coordinate of the new position in pixels, relative to the screen origin (upper left). The location of a character is based on the upper left corner of its animation frame.

y

The y-coordinate of the new position in pixels, relative to the screen origin (upper left). The location of a character is based on the upper left corner of its animation frame.

lSpeed

A parameter specifying in milliseconds how quickly the character's frame moves. The recommended value is 1000. Specifying zero (0) moves the frame without playing an animation.

pdwReqID

Address of a variable that receives the **MoveTo** request ID.

When using the HTTP protocol to access character and animation data, use the **Prepare** method to ensure the availability of the **Moving** state animations before calling this method. Even if the animation is not loaded, the server still moves the frame.

See also **IAgentCharacter::SetPosition**

IAgentCharacter::Play

```

HRESULT Play(
    BSTR bszAnimation,    // name of an animation
    long * pdwReqID       // address of request ID
);

```

Plays the specified animation.

- Returns S_OK to indicate the operation was successful. When the function returns, *pdwReqID* contains the ID of the request.

bszAnimation

The name of an animation.

pdwReqID

Address of a variable that receives the **Play** request ID.

An animation's name is defined when the character is compiled with the Microsoft Agent Character Editor. Before playing the specified animation, the server attempts to play the **Return** animation for the previous animation (if one has been assigned).

When a character's animation data is stored on the user's local machine, you can use the **Play** method and specify the name of the animation. When using the HTTP protocol to access animation data, use the **Prepare** method to ensure the availability of the animation before calling this method.

See also **IAgentCharacter::Prepare**

IAgentCharacter::Prepare

```

HRESULT Prepare(
    long dwType,          // type of animation data to load
    BSTR bszName,         // name of the animation
    long bQueue,          // queue the request
    long * pdwReqID       // address of request ID
);

```

Retrieves animation data for a character.

- Returns S_OK to indicate the operation was successful. When the function returns, *pdwReqID* contains the ID of the request.

dwType

A value that indicates the animation data type to load that must be one of the following:

const unsigned short PREPARE_ANIMATION = 0;	A character's animation data.
const unsigned short PREPARE_STATE = 1;	A character's state data.
const unsigned short PREPARE_WAVE = 2	A character's sound file (.WAV or .LWV) for spoken output.

bszName

The name of the animation or state.

The animation name is based on that defined for the character when it was saved using the Microsoft Agent Character Editor.

For states, the value can be one of the following:

“Gesturing”	To retrieve all Gesturing state animations.
“GesturingDown”	To retrieve GesturingDown animations.
“GesturingLeft”	To retrieve GesturingLeft animations.
“GesturingRight”	To retrieve GesturingRight animations.
“GesturingUp”	To retrieve GesturingUp animations.
“Hiding”	To retrieve the Hiding state animations.
“Hearing”	To retrieve the Hearing state animations.
“Idling”	To retrieve all Idling state animations.
“IdlingLevel1”	To retrieve all IdlingLevel1 animations.
“IdlingLevel2”	To retrieve all IdlingLevel2 animations.
“IdlingLevel3”	To retrieve all IdlingLevel3 animations.
“Listening”	To retrieve the Listening state animations.
“Moving”	To retrieve all Moving state animations.
“MovingDown”	To retrieve all Moving animations.
“MovingLeft”	To retrieve all MovingLeft animations.
“MovingRight”	To retrieve all MovingRight animations.
“MovingUp”	To retrieve all MovingUp animations.
“Showing”	To retrieve the Showing state animations.
“Speaking”	To retrieve the Speaking state animations.

For .WAV files, set *bszName* to the URL or file specification for the .WAV file. If the specification is not complete, it is interpreted as being relative to the specification used in the **Load** method.

bQueue

A Boolean specifying whether the server queues the **Prepare** request. TRUE queues the request and causes any animation request that follows it to wait until the animation data it specifies is loaded. FALSE retrieves the animation data asynchronously.

pdwReqID

Address of a variable that receives the **Prepare** request ID.

You can specify multiple animations and states by separating them with commas. However, you cannot mix types in the same **Prepare** statement.

IAgentCharacter::SetDescription

```
HRESULT SetDescription(  
    BSTR bszDescription    // character description  
);
```

Sets the description of the character.

- Returns S_OK to indicate the operation was successful.

bszDescription

A BSTR that sets the description for the character. A character's default description is defined when it is compiled with the Microsoft Agent Character Editor. The description setting is optional and may not be supplied for all characters. You can change the character's description using **IAgentCharacter::SetDescription**; however, this value is not persistent (stored permanently). The character's description reverts to its default setting whenever the character is first loaded by a client.

See also **IAgentCharacter::GetDescription**

IAgentCharacter::SetIdleOn

```
HRESULT SetIdleOn(  
    long bOn    // idle processing flag  
);
```

Sets automatic idle processing for a character.

- Returns S_OK to indicate the operation was successful.

bOn

Idle processing flag. If this parameter is TRUE, the Microsoft Agent automatically plays **Idling** state animations.

The server automatically sets a time out after the last animation played for a character. When this timer's interval is complete, the server begins the **Idling** states for a character, playing its associated **Idling** animations at regular intervals. If you want to manage the **Idling** state animations yourself, set the property to FALSE.

See also **IAgentCharacter::GetIdleOn**

IAgentCharacter::SetName

```
HRESULT SetName(
    BSTR bszName    // character name
);
```

Sets the name of the character.

- Returns S_OK to indicate the operation was successful.

bszName

A BSTR that sets the character's name. A character's default name is defined when it is compiled with the Microsoft Agent Character Editor. You can change it using **IAgentCharacter::SetName**; however, this changes the character name for all current clients of the character. This property is not persistent (stored permanently). The character's name reverts to its default name whenever the character is first loaded by a client.

The server uses the character's name setting in parts of the Microsoft Agent's interface, such as the Commands Window title when the character is input-active and in the Microsoft Agent taskbar pop-up menu.

See also **IAgentCharacter::GetName**

IAgentCharacter::SetPosition

```
HRESULT SetPosition(
    long lLeft,    // screen coordinate of the left edge of character
    long lTop     // screen coordinate of the top edge of character
);
```

Sets the position of the character's animation frame.

- Returns S_OK to indicate the operation was successful.

lLeft

Screen coordinate of the character animation frame's left edge in pixels, relative to the screen origin (upper left).

lTop

Screen coordinate of the character animation frame's top edge in pixels, relative to the screen origin (upper left).

Even though the character appears in an irregularly shaped region window, the location of the character is based on its rectangular animation frame.

Note Unlike the **MoveTo** method, this function is not queued.

See also **IAgent::GetPosition**

IAgentCharacter::SetSize

```
HRESULT SetSize(  
    long * lWidth,    // width of the character frame  
    long * lHeight    // height of the character frame  
);
```

Sets the size of the character's animation frame.

- Returns S_OK to indicate the operation was successful.

lWidth

The width of the character's animation frame in pixels.

lHeight

The height of the character's animation frame in pixels.

Changing the character's frame size scales the character to the size set with this method.

Even though the character appears in an irregularly shaped region window, the location of the character is based on its rectangular animation frame.

See also **IAgentCharacter::GetSize**

IAgentCharacter::SetSoundEffectsOn

```
HRESULT SetSoundEffectsOn(  
    long bOn    // character sound effects setting  
);
```

Determines whether the character's sound effects are played.

- Returns S_OK to indicate the operation was successful.

bOn

Sound effects setting. If this parameter is TRUE, the sound effects for animations are played when the animation plays; if FALSE, sound effects are not played.

This setting determines whether sound effects compiled as a part of the character are played when you play an associated animation. The setting is subject to the user's global sound effects setting in **IAgentAudioOutputProperties::GetUsingSoundEffects**.

See also **IAgentCharacter::GetSoundEffectsOn**,
IAgentAudioOutputProperties::GetUsingSoundEffects

IAgentCharacter::Show

```
HRESULT Show(  
    long bFast,        // play Showing state animation flag
```

```

    long * pdwReqID // address of request ID
);

```

Displays a character.

- Returns S_OK to indicate the operation was successful. When the function returns, *pdwReqID* contains the ID of the request.

bFast

Showing state animation flag. If this parameter is TRUE, the **Showing** state animation plays after making the character visible; if FALSE, the animation does not play.

pdwReqID

Address of a variable that receives the **Show** request ID.

Avoid setting the *bFast* parameter to TRUE without playing an animation beforehand, otherwise, the character frame may be displayed, but have no image to display. In particular, note that if you call **MoveTo** when the character is not visible, it does not play any animation. Therefore, if you call the **Show** method with *bFast* set to TRUE, no image will be displayed. Similarly, if you call **Hide** then **Show** with *bFast* set to TRUE, there will be no visible image.

When using the HTTP protocol to access character and animation data, use the **Prepare** method to ensure the availability of the **Showing** state animation before calling this method.

See also **IAgentCharacter::Hide**

IAgentCharacter::Speak

```

HRESULT Speak(
    BSTR bszText,      // text to speak
    BSTR bszURL,       // URL of a file to speak
    long * pdwReqID    // address of a request ID
);

```

Speaks the

- Returns S_OK to indicate the operation was successful.

bszText

The text the character is to speak.

bszURL

The URL (or file specification) of a sound file to use for spoken output. This can be a standard sound file (.WAV) or linguistically enhanced sound file (.LWV).

pdwReqID

Address of a variable that receives the **Speak** request ID.

To use this method with a character configured to speak using a text-to-speech (TTS) engine; simply provide the *bszText* parameter. You can include vertical bar characters (|) in the *bszText* parameter to designate alternative strings, so that each time the server processes the method, it randomly choose a different string. Support of TTS output is defined when the character is compiled using the Microsoft Agent Character Editor.

If you want to use sound file output for the character, specify the location for the file in the *bszURL* parameter. When using the HTTP protocol to download a sound file, use the **Prepare** method to ensure the availability of the file before using this method. You can use the *bszText* parameter to specify the words that appear in the character's word balloon. If you specify a linguistically enhanced sound file (.LWV) for the *bszURL* parameter and do not specify text, the *bszText* parameter uses the text stored in the file.

The **Speak** method uses the last animation played to determine which speaking animation to play. For example, if you precede the **Speak** command with a **Play "GestureRight"**, the server will play **GestureRight** and then the **GestureRight** speaking animation.

If you call **Speak** and the audio channel is busy, the character's audio output will not be heard, but the text will display in the word balloon. The word balloon's **Enabled** property must also be TRUE for the text to display.

See also **IAgentCharacter::Play**, **IAgentBalloon::Enabled**, **IAgentCharacter::Prepare**

IAgentCharacter::Stop

```
HRESULT Stop(  
    long dwReqID // request ID  
);
```

Stops the specified animation (request) and removes it from the character's animation queue.

- Returns S_OK to indicate the operation was successful.

dwReqID

The ID of the request to stop.

Stop can also be used to halt any queued **Prepare** calls.

See also **IAgentCharacter::Prepare**, **IAgentCharacter::StopAll**

IAgentCharacter::StopAll

```
HRESULT StopAll()  
    long lType, // request type
```

Stops all animations (requests) and removes them from the character's animation queue.

lType

A bit field that indicates the types of requests to stop (and remove from the character's queue), comprised from the following:

const unsigned long STOP_TYPE_ALL = 0xFFFFFFFF;	Stops all animation requests, including non-queued Prepare requests.
const unsigned long STOP_TYPE_PLAY = 0x00000001;	Stops all Play requests.
const unsigned long STOP_TYPE_MOVE = 0x00000002;	Stops all Move requests.
const unsigned long STOP_TYPE_SPEAK = 0x00000004;	Stops all Speak requests.
const unsigned long STOP_TYPE_PREPARE = 0x00000008;	Stops all queued Prepare requests.
const unsigned long STOP_TYPE_NONQUEUEDPREPARE = 0x00000010;	Stops all non-queued Prepare requests.
const unsigned long STOP_TYPE_VISIBLE = 0x00000020;	Stops all Hide or Show requests.

See also **IAgentCharacter::Stop**

IAgentCharacter::Wait

```
HRESULT Wait(
    long dwReqID,      // request ID
    long * pdwReqID    // address of request ID
);
```

Holds the character's animation queue at the specified animation (request) until another request for another character completes.

- Returns S_OK to indicate the operation was successful.

dwReqID

The ID of the request to wait for.

pdwReqID

Address of a variable that receives the **Wait** request ID.

Use this method only when you support multiple (simultaneous) characters and want to sequence their interaction (as a single client). (For a single character, each animation request is played sequentially--after the previous request completes.) If you have two characters and want one character's animation request to wait until the other character's animation completes, set the **Wait** method to the other character's animation request ID.

IAgentCommands

The Microsoft Agent server maintains a list of commands that are currently available to the user. This list includes commands that the server defines for general interaction, such as Hide and Microsoft Agent Properties, the list of available (but non-input-active) clients, and the commands defined by the current active client. The first two sets of commands are global commands; that is, they are available at any time, regardless of the input-active client. Client-defined commands are available only when that client is input-active.

Retrieve an **IAgentCommands** interface by querying the **IAgentCharacter** interface for **IAgentCommands**. Each Microsoft Agent client application can define a collection of commands called a **Commands** collection. To add a **Command** to the collection, use the **Add** or **Insert** method. Although you can specify a **Command's** properties using **IAgentCommand** methods, for optimum code performance, specify all of a **Command's** properties in the **IAgentCommands::Add** or **IAgentCommands::Insert** methods when initially setting the properties for a new **Command**. You can use the **IAgentCommand** methods to query or change the property settings.

For each **Command** in the **Commands** collection, you can determine whether the command appears on the character's pop-up menu, in the Commands Window, in both, or in neither. For example, if you want a command to appear on the pop-up menu for the character, set the command's **Caption** and **Visible** properties. To display the command in the **Commands** Window, set the command's **Caption** and **Voice** properties.

A user can access the individual commands in your Commands collection only when your client application is input-active. Therefore, you will typically want to set the **Caption** and **Voice** properties for the **Commands** collection object as well as for the commands in the collection, because this places an entry for your **Commands** collection on a character's pop-up menu and in the Commands Window. When the user switches to your client by choosing its entry, the server automatically makes your client input-active and makes the **Commands** in its collection available. This enables the server to present and accept only the **Commands** that apply to the current input-active client's context. It also serves to avoid **Command**-name collisions between clients.

When a character's pop-up menu is displayed, changes to the properties of a **Commands** collection or the commands in its collection do not appear until the user redisplay the menu. However, when open, the Commands Window does display changes as they happen.

IAgentCommands defines an interface that allows applications to add, remove, set, and query properties for a **Commands** collection. A **Commands** collection can appear as a command in both the pop-up menu and the Commands Window for a character. To make the **Commands** collection appear, you must set its **Caption** property. The following table summarizes how the properties of a **Commands** collection affect its presentation.

<u>Caption Property</u>	<u>Voice Property</u>	<u>Visible Property</u>	<u>Appears in Character's Pop-up Menu</u>	<u>Appears in Commands Window</u>
Yes	Yes	True	Yes	Yes
Yes	Yes	False	No	Yes
Yes	No	True	Yes	No
Yes	No	False	No	No

<u>Caption Property</u>	<u>Voice Property</u>	<u>Visible Property</u>	<u>Appears in Character's Pop-up Menu</u>	<u>Appears in Commands Window</u>
No	Yes	True	No	No*
No	Yes	False	No	No*
No	No	True	No	No
No	No	False	No	No

*The command is still voice-accessible. If the client is input-active and has **Commands** in its collection, "(command undefined)" appears in the Commands Window.

Methods in Vtable Order

IAgentCommands Methods	Description
GetCommand	Retrieves a Command object from the Commands collection.
GetCount	Returns the value of the number of Commands in a Commands collection.
SetCaption	Sets the value of the Caption property for a Commands collection.
GetCaption	Returns the value of the Caption property of a Commands collection.
SetVoice	Sets the value of the Voice property for a Commands collection.
GetVoice	Returns the value of the Voice property of a Commands collection.
SetVisible	Sets the value of the Visible property for a Commands collection.
GetVisible	Returns the value of the Visible property of a Commands collection.
Add	Adds a Command object to a Commands collection.
Insert	Inserts a Command object in a Commands collection.
Remove	Removes a Command object in a Commands collection.
RemoveAll	Removes all Command objects from a Commands collection.

IAgentCommands::Add

```
HRESULT Add(
    BSTR bszCaption, // Caption setting for Command
    BSTR bszVoice,   // Voice setting for Command
    long bEnabled,   // Enabled setting for Command
    long bVisible,   // Visible setting for Command
    long * pdwID     // address for variable for ID
);
```

Adds a **Command** to a **Commands** collection.

- Returns S_OK to indicate the operation was successful.

bszCaption

A BSTR that specifies the value of the **Caption** text displayed for a **Command** in a **Commands** collection.

bszVoice

A BSTR that specifies the value of the **Voice** text setting for a **Command** in a **Commands** collection.

bEnabled

A Boolean expression that specifies the **Enabled** setting for a **Command** in a **Commands** collection. If the parameter is TRUE, the **Command** is enabled and can be selected; if FALSE, the **Command** is disabled.

bVisible

A Boolean expression that specifies the **Visible** setting for a **Command** in a **Commands** collection. If the parameter is TRUE, the **Command** will be visible in the character's pop-up menu (if the **Caption** property is also set).

pdwID

Address of a variable that receives the ID for the added **Command**.

See also **IAgentCommand::SetCaption**, **IAgentCommand::SetEnabled**, **IAgentCommand::SetVisible**, **IAgentCommand::SetVoice**, **IAgentCommands::Insert**, **IAgentCommands::Remove**, **IAgentCommands::RemoveAll**

IAgentCommands::GetCaption

```
HRESULT GetCaption(  
    BSTR * pbszCaption // address of Caption text for Commands collection  
);
```

Retrieves the **Caption** for a **Commands** collection.

- Returns S_OK to indicate the operation was successful.

pbszCaption

The address of a BSTR that receives the value of the **Caption** text setting displayed for a **Commands** collection.

See also **IAgentCommands::SetCaption**, **IAgentCommands::GetVisible**, **IAgentCommands::GetVoice**

IAgentCommands::GetCommand

```
HRESULT GetCommand(  
    long dwCommandID, // Command ID  
    IUnknown ** ppunkCommand // address of IUnknown interface
```

);

Retrieves a **Command** object from the **Commands** collection.

- Returns S_OK to indicate the operation was successful.

dwCommandID

The ID of a Command object in the **Commands** collection.

IUnknown

The address of the **IUnknown** interface for the **Command** object.

See also **IAgentCommand**

IAgentCommands::GetCount

```
HRESULT GetCount(  
    long * pdwCount // address of count of commands  
);
```

Retrieves the number of **Command** objects in a **Commands** collection.

- Returns S_OK to indicate the operation was successful.

pdwCount

Address of a variable that receives the number of **Commands** in a **Commands** collection.

pdwCount includes only the number of **Commands** you define in your **Commands** collection. Server or other client entries are not included.

IAgentCommands::GetVisible

```
HRESULT GetVisible(  
    long * pbVisible // address of Visible setting for Commands collection  
);
```

Retrieves the value of the **Visible** property for a **Commands** collection.

- Returns S_OK to indicate the operation was successful.

pbVisible

The address of a variable that receives the value of the **Visible** property for a **Commands** collection.

See also **IAgentCommands::SetVisible**, **IAgentCommands::SetCaption**

IAgentCommands::GetVoice

```
HRESULT GetVoice(  
    BSTR * pbszVoice    // address of Voice setting for Commands collection  
);
```

Retrieves the value of the **Voice** property for a **Commands** collection.

- Returns S_OK to indicate the operation was successful.

pbszVoice

The address of a BSTR that receives the value of the **Voice** text setting for a **Commands** collection.

See also **IAgentCommands::SetVoice**, **IAgentCommands::GetCaption**, **IAgentCommands::GetVisible**

IAgentCommands::Insert

```
HRESULT Insert(  
    BSTR bszCaption,    // Caption setting for Command  
    BSTR bszVoice,      // Voice setting for Command  
    long bEnabled,      // Enabled setting for Command  
    long bVisible,      // Visible setting for Command  
    long dwRefID,       // reference Command for insertion  
    long dBefore,       // insertion position flag  
    long * pdwID        // address for variable for Command ID  
);
```

Inserts a **Command** object in a **Commands** collection.

- Returns S_OK to indicate the operation was successful.

bszCaption

A BSTR that specifies the value of the **Caption** text displayed for the **Command**.

bszVoice

A BSTR that specifies the value of the **Voice** text setting for a **Command**.

bEnabled

A Boolean expression that specifies the **Enabled** setting for a **Command**. If the parameter is TRUE, the **Command** is enabled and can be selected; if FALSE, the **Command** is disabled.

bVisible

A Boolean expression that specifies the **Visible** setting for a **Command**. If the parameter is TRUE, the **Command** will be visible in the character's pop-up menu (if the **Caption** property is also set).

dwRefID

The ID of a **Command** used as a reference for the relative insertion of the new **Command**.

dBefore

A Boolean expression that specifies where to place the **Command**. If this parameter is TRUE, the new **Command** is inserted before the referenced **Command**; if FALSE, the new **Command** is placed after the referenced **Command**.

pdwID

Address of a variable that receives the ID for the inserted **Command**.

See also **IAgentCommand::Add**, **IAgentCommands::Remove**, **IAgentCommands::RemoveAll**

IAgentCommands::Remove

```
HRESULT Remove(  
    long dwID // Command ID  
);
```

Removes the specified **Command** from a **Commands** collection.

- Returns S_OK to indicate the operation was successful.

dwID

The ID of a **Command** to remove from the **Commands** collection.

Removing a **Command** from a **Commands** collection also removes it from the pop-up menu and the Commands Window when your application is input-active.

See also **IAgentCommands::Add**, **IAgentCommands::Insert**, **IAgentCommands::RemoveAll**

IAgentCommands::RemoveAll

```
HRESULT Remove();
```

Removes all **Commands** from a **Commands** collection.

- Returns S_OK to indicate the operation was successful.

Removing all **Commands** from a **Commands** collection also removes them from the pop-up menu and the Commands Window when your application is input-active. **RemoveAll** does not remove server or other client's entries.

See also **IAgentCommands::Add**, **IAgentCommands::Insert**, **IAgentCommands::Remove**

IAgentCommands::SetCaption

```
HRESULT SetCaption(  
    BSTR bszCaption    // Caption setting for Commands collection  
);
```

Sets the **Caption** text displayed for a **Commands** collection.

- Returns S_OK to indicate the operation was successful.

bszCaption

A BSTR that specifies the value for the **Caption** property for a **Commands** collection.

A **Commands** collection with its **Caption** property set and its **Visible** property set to TRUE appears in the character's pop-up menu. If its **Voice** property is also set, it appears in the Commands Window. If you define commands for a **Commands** collection that have their **Caption**, **Enabled**, and **Voice** properties set, you typically also define **Caption** and **Voice** settings for the associated **Commands** collection. If the **Commands** collection has no **Voice** or no **Caption** setting and is currently input-active, but the **Commands** in its collection have **Caption** and **Voice** settings, the **Commands** appear in the **Commands** Window tree view under "(undefined command)" when your client application becomes input-active.

See also **IAgentCommands::GetCaption**, **IAgentCommands::SetVisible**, **IAgentCommands::SetVoice**

IAgentCommands::SetVisible

```
HRESULT SetVisible(  
    long bVisible    // the Visible setting for Commands collection  
);
```

Sets the value of the **Visible** property for a **Commands** collection.

- Returns S_OK to indicate the operation was successful.

bVisible

A Boolean value that determines the **Visible** property of a **Commands** collection. TRUE sets the **Commands** collection's **Caption** to be visible when the character's pop-up menu is displayed; FALSE does not display it.

A **Commands** collection must have its **Caption** property set and its **Visible** property set to TRUE to appear on the character's pop-up menu. The **Visible** property must also be set to TRUE for commands in the collection to appear when your client application is input-active.

See also **IAgentCommands::GetVisible**, **IAgent::SetCaption**

IAgentCommands::SetVoice

```
HRESULT SetVoice(
    BSTR bszVoice // the Voice setting for Command collection
);
```

Sets the **Voice** text property for a **Command**.

- Returns S_OK to indicate the operation was successful.

bszVoice

A BSTR that specifies the value for the **Voice** text property of a **Commands** collection.

A **Commands** collection must have its **Voice** text property set to be voice-accessible. It also must have its **Caption** property set to appear in the Commands Window and its **Visible** property set to TRUE to appear on the character's pop-up menu.

The BSTR expression you supply can include square bracket characters ([]) to indicate optional words and vertical bar characters (|) to indicate alternative strings. Alternates must be enclosed in parentheses. For example, "(hello [there] | hi)" tells the speech engine to accept "hello," "hello there," or "hi" for the command. Remember to include appropriate spaces between words you include in brackets or parentheses as well as other text. Remember to include appropriate spaces between the text that's in brackets or parentheses and the text that's not in brackets or parentheses.

You can also use an ellipsis (...) to support *word spotting*, that is, telling the speech recognition engine to ignore words spoken in this position in the phrase (sometimes called *garbage* words). When you use ellipses, the speech engine recognizes only specific words in the string regardless of when spoken with adjacent words or phrases. For example, if you set this property to "...check mail..." the speech recognition engine will match phrases like "please check mail" or "check mail please" to this command. Ellipses can be used anywhere within a string. However, be careful using this technique as voice settings with ellipses may increase the potential of unwanted matches.

When defining the words and grammar for your command, always make sure that you include at least one word that is required; that is, avoid supplying only optional words. In addition, make sure that the word includes only pronounceable words and letters. For numbers, it is better to spell out the word rather than using the numeric representation. Also, omit any punctuation or symbols. For example, instead of "the #1 \$10 pizza!", use "the number one ten dollar pizza". Including non-pronounceable characters or symbols for one command may cause the speech engine to fail to compile the grammar for all your commands. Finally, make your voice parameter as distinct as reasonably possible from other voice commands you define. The greater the similarity between the voice grammar for commands, the more likely the speech engine will make a recognition error. You can also use the confidence scores to better distinguish between two commands that may have similar or similar-sounding voice grammar.

The operation of this property depends on the state of Microsoft Agent server's speech recognition state. For example, if speech recognition is disabled or not installed, this function has no immediate effect. If speech recognition is enabled during a session, however, the command will become accessible when its client application is input-active.

See also **IAgentCommands::GetVoice**, **IAgentCommands::SetCaption**, **IAgentCommands::SetVisible**

IAgentCommand

A **Command** object is an item in a **Commands** collection. The server provides the user access to your commands your client application becomes input active. To retrieve a **Command**, call **IAgentCommands::GetCommand**.

IAgentCommand defines an interface that allows applications to set and query properties for **Command** objects that can appear in a character's pop-up menu and in the Commands Window. A **Command** object is an item in a **Commands** collection. The server provides the user access to your commands when your client application becomes input active.

A **Command** may appear in either or both the character's pop-up menu and the Commands Window. To appear in the pop-up menu, it must have a **Caption** and have the **Visible** property set to TRUE. The **Visible** property for its **Commands** collection object must also be set to TRUE for the command to appear in the pop-up menu when your client application is input-active. To appear in the Commands Window, a **Command** must have its **Caption** and **Voice** properties set.

A character's pop-up menu entries do not change while the menu is displayed. If you add or remove Commands or change their properties while the character's popup menu is displayed, the menu displays those changes when redisplayed. However, the Commands Window does display changes as you make them.

The following table summarizes how the properties of a command affect its presentation.

<u>Caption Property</u>	<u>Voice Property</u>	<u>Visible Property</u>	<u>Enabled Property</u>	<u>Appears in Character's Pop-up Menu</u>	<u>Appears in Commands Window</u>
Yes	Yes	True	True	Normal	Yes
Yes	Yes	True	False	Disabled	No
Yes	Yes	False	True	Does not appear	Yes
Yes	Yes	False	False	Does not appear	No
Yes	No	True	True	Normal	No
Yes	No	True	False	Disabled	No
Yes	No	False	True	Does not appear	No
Yes	No	False	False	Does not appear	No
No	Yes	True	True	Does not appear	No*
No	Yes	True	False	Does not appear	No
No	Yes	False	True	Does not appear	No*
No	Yes	False	False	Does not appear	No
No	No	True	True	Does not appear	No
No	No	True	False	Does not appear	No
No	No	False	True	Does not appear	No
No	No	False	False	Does not appear	No

*The command is still voice-accessible.

Generally, if you define a **Command** with a **Voice** setting, you also define **Caption** and **Voice** settings for its associated **Commands** collection. If the **Commands** collection for a set of commands has no **Voice** or no **Caption** setting and is currently input-active, but the **Commands** have **Caption** and **Voice** settings, the **Commands** appear in the Commands Window tree view under “(undefined command)” when your client application becomes input-active.

When the server receives input that matches one of the **Command** objects you defined for your **Commands** collection, it sends a **IAgentNotifySink::Command** event, and passes back the ID of the command as an attribute of the **IAgentUserInput** object. You can then use conditional statements to match and process the command.

Methods in Vtable Order

<u>IAgentCommand Methods</u>	<u>Description</u>
SetCaption	Sets the value for the Caption for a Command object.
GetCaption	Returns the value of the Caption property of a Command object.
SetVoice	Sets the value for the Voice text for a Command object.
GetVoice	Returns the value of the Caption property of a Command object.
SetEnabled	Sets the value of the Enabled property for a Command object.
GetEnabled	Returns the value of the Enabled property of a Command object.
SetVisible	Sets the value of the Visible property for a Command object.
GetVisible	Returns the value of the Visible property of a Command object.
SetConfidenceThreshold	Sets the value of the Confidence property for a Command object.
GetConfidenceThreshold	Returns the value of the Confidence property of a Command object.
SetConfidenceText	Sets the value of the ConfidenceText property for a Command object.
GetConfidenceText	Returns the value of the ConfidenceText property of a Command object.
GetID	Returns the ID of a Command object.

IAgentCommand::GetCaption

```
HRESULT GetCaption(
    BSTR * pbszCaption // address of Caption for Command
);
```

Retrieves the **Caption** for a **Command**.

- Returns S_OK to indicate the operation was successful.

pbszCaption

The address of a BSTR that receives the value of the **Caption** text displayed for a **Command**.

See also **IAgentCommand::SetCaption**, **IAgentCommand::SetEnabled**, **IAgentCommand::SetVisible**, **IAgentCommand::SetVoice**, **IAgentCommands::Add**, **IAgentCommands::Insert**

IAgentCommand::GetConfidenceText

```
HRESULT GetConfidenceText(  
    BSTR * pbszTipText // address of ConfidenceText setting for Command  
);
```

Retrieves the Listening Tip text previously set for a **Command**.

- Returns S_OK to indicate the operation was successful.

pbszTipText

The address of a BSTR that receives the value of the Listening Tip text for a **Command**.

See also **IAgentCommand::SetConfidenceThreshold**,
IAgentCommand::GetConfidenceThreshold, **IAgentCommand::SetConfidenceText**,
IAgentUserInput::GetItemConfidence

IAgentCommand::GetConfidenceThreshold

```
HRESULT GetConfidenceThreshold(  
    long * plConfidenceThreshold // address of ConfidenceThreshold  
);                               // setting for Command
```

Retrieves the value of the **ConfidenceThreshold** property for a **Command**.

- Returns S_OK to indicate the operation was successful.

plConfidenceThreshold

The address of a variable that receives the value of the **ConfidenceThreshold** property for a **Command**.

See also **IAgentCommand::SetConfidenceThreshold**,
IAgentCommand::SetConfidenceText, **IAgentUserInput::GetItemConfidence**

IAgentCommand::GetEnabled

```
HRESULT GetEnabled(  
    long * pbEnabled // address of Enabled setting for Command  
);
```

Retrieves the value of the **Enabled** property for a **Command**.

- Returns S_OK to indicate the operation was successful.

pbEnabled

The address of a variable that receives TRUE if the **Command** is enabled, or FALSE if it is disabled. A disabled **Command** cannot be selected.

See also **IAgentCommand::SetCaption**, **IAgent::SetVisible**, **IAgentCommand::SetVoice**, **IAgentCommands::Add**, **IAgentCommands::Insert**

IAgentCommand::GetID

```
HRESULT GetID(  
    long * pdwID // address of ID for Command  
);
```

Retrieves the ID for a **Command**.

- Returns S_OK to indicate the operation was successful.

pdwID

The address of a variable that receives the ID of a **Command**.

See also **IAgentCommands::Add**, **IAgentCommands::Insert**, **IAgentCommands::Remove**

IAgentCommand::GetVisible

```
HRESULT GetVisible(  
    long * pbVisible // address of Visible setting for Command  
);
```

Retrieves the value of the **Visible** property for a **Command**.

- Returns S_OK to indicate the operation was successful.

pbVisible

The address of a variable that receives the **Visible** property for a **Command**.

See also **IAgentCommand::SetVisible**, **IAgent::SetCaption**, **IAgentCommands::Add**, **IAgentCommands::Insert**

IAgentCommand::GetVoice

```
HRESULT GetVoice(  
    BSTR * pbszVoice // address of Voice setting for Command  
);
```

Retrieves the value of the **Voice** text property for a **Command**.

- Returns S_OK to indicate the operation was successful.

pbszVoice

The address of a BSTR that receives the **Voice** text property for a **Command**.

A **Command** with its **Voice** property set and its **Enabled** property set to TRUE will be voice-accessible. If its **Caption** property is also set it appears in the Commands Window. If its **Visible** property is set to TRUE, it appears in the character's pop-up menu.

See also **IAgentCommand::SetVoice**, **IAgentCommands::Add**, **IAgentCommands::Insert**

IAgentCommand::SetCaption

```
HRESULT SetCaption(  
    BSTR bszCaption    // Caption setting for Command  
);
```

Sets the **Caption** text displayed for a **Command**.

- Returns S_OK to indicate the operation was successful.

bszCaption

A BSTR that specifies the text for the **Caption** property for a **Command**.

A **Command** with its **Caption** property set and its **Visible** property set to TRUE appears in the character's pop-up menu. If its **Voice** property is also set, it appears in the Commands Window. To make it accessible, you must also set its **Enabled** property to TRUE.

See also **IAgentCommand::GetCaption**, **IAgentCommand::SetEnabled**, **IAgentCommand::SetVisible**, **IAgentCommand::SetVoice**, **IAgentCommands::Add**, **IAgentCommands::Insert**

IAgentCommand::SetConfidenceThreshold

```
HRESULT SetConfidenceThreshold(  
    long lConfidence    // Confidence setting for Command  
);
```

Sets the value of the **Confidence** property for a **Command**.

- Returns S_OK to indicate the operation was successful.

lConfidence

The value for the **Confidence** property of a **Command**.

If the confidence value returned of the best match returned in the **Command** event does not exceed the value set for the **ConfidenceThreshold** property, the text supplied in **SetConfidenceText** is displayed in the Listening Tip.

See also **IAgentCommand::GetConfidenceThreshold**, **IAgentCommand::SetConfidenceText**, **IAgentUserInput::GetItemConfidence**

IAgentCommand::SetConfidenceText

```
HRESULT SetConfidenceText (
    BSTR bszTipText // ConfidenceText setting for Command
);
```

Sets the value of the Listening Tip text for a **Command**.

- Returns S_OK to indicate the operation was successful.

bszTipText

A BSTR that specifies the text for the **ConfidenceText** property of a **Command**.

If the confidence value returned of the best match returned in the **Command** event does not exceed the value set for the **ConfidenceThreshold** property, the text supplied in *bszTipText* is displayed in the Listening Tip.

See also **IAgentCommand::SetConfidenceThreshold**, **IAgentCommand::GetConfidenceThreshold**, **IAgentCommand::GetConfidenceText**, **IAgentUserInput::GetItemConfidence**

IAgentCommand::SetEnabled

```
HRESULT SetEnabled(
    long bEnabled // Enabled setting for Command
);
```

Sets the **Enabled** property for a **Command**.

- Returns S_OK to indicate the operation was successful.

bEnabled

A Boolean value that sets the value of the **Enabled** setting of a **Command**. TRUE enables the **Command**; FALSE disables it. A disabled **Command** cannot be selected.

A **Command** must have its **Enabled** property set to TRUE to be selectable. It also must have its **Caption** property set and its **Visible** property set to TRUE to appear in the character's pop-up menu. To make the **Command** appear in the **Commands** Window, you must set its **Voice** property.

See also **IAgentCommand::GetCaption**, **IAgentCommand::SetVoice**, **IAgentCommands::Add**, **IAgentCommands::Insert**

IAgentCommand::SetVisible

```
HRESULT SetVisible(
    long bVisible // Visible setting for Command
);
```

Sets the value of the **Visible** property for a **Command**.

- Returns S_OK to indicate the operation was successful.

bVisible

A Boolean value that determines the **Visible** property of a **Command**. TRUE shows the **Command**; FALSE hides it.

A **Command** must have its **Visible** property set to TRUE and its **Caption** property set to appear in the character's pop-up menu.

See also **IAgentCommand::GetVisible**, **IAgent::SetCaption**, **IAgentCommands::Add**, **IAgentCommands::Insert**

IAgentCommand::SetVoice

```
HRESULT SetVoice(  
    BSTR bszVoice // voice text setting for Command  
);
```

Sets the **Voice** property for a **Command**.

- Returns S_OK to indicate the operation was successful.

bszVoice

A BSTR that specifies the text for the **Voice** property of a **Command**.

A **Command** must have its **Voice** property and **Enabled** property set to be voice-accessible. It also must have its **Caption** property set to appear in the Commands Window.

The BSTR expression you supply can include square bracket characters ([]) to indicate optional words and vertical bar characters (|) to indicate alternative strings. Alternates must be enclosed in parentheses. For example, "(hello [there] | hi)" tells the speech engine to accept "hello," "hello there," or "hi" for the command. Remember to include appropriate spaces between the text that's in brackets or parentheses and the text that's not in brackets or parentheses.

You can also use an ellipsis (...) to support *word spotting*, that is, telling the speech recognition engine to ignore words spoken in this position in the phrase (sometimes called *garbage* words). Therefore, the speech engine recognizes only specific words in the string regardless of when spoken with adjacent words or phrases. For example, if you set this property to "...check mail..." the speech recognition engine will match phrases like "please check mail" or "check mail please" to this command. Ellipses can be used anywhere within a string. However, be careful using this technique, because voice settings with ellipses may increase the potential of unwanted matches.

When defining the words and grammar for your command, always make sure that you include at least one word that is required; that is, avoid supplying only optional words. In addition, make sure that the word includes only pronounceable words and letters. For numbers, it is better to spell out the word rather than using the numeric representation. Also, omit any punctuation or symbols. For example, instead of "the #1 \$10 pizza!", use "the number one ten dollar pizza". Including non-pronounceable characters or symbols for one command may cause the speech engine to fail to compile the grammar for all your commands. Finally, make your voice parameter as distinct as reasonably possible from other voice commands you define. The greater the similarity between the voice grammar for commands, the more likely

the speech engine will make a recognition error. You can also use the confidence scores to better distinguish between two commands that may have similar or similar-sounding voice grammar.

The operation of this property depends on the state of Microsoft Agent server's speech recognition state. For example, if speech recognition is disabled or not installed, this function has no immediate effect. If speech recognition is enabled during a session, however, the command will become accessible when its client application is input-active.

See also **IAgentCommand::GetVoice**, **IAgentCommand::SetCaption**, **IAgentCommand::SetEnabled**, **IAgentCommands::Add**, **IAgentCommands::Insert**

IAgentUserInput

When a **Command** event occurs, the Microsoft Agent server returns information through the **UserInput** object. **IAgentUserInput** defines an interface that allows applications to query these values.

Methods in Vtable Order

<u>IAgentUserInput Methods</u>	<u>Description</u>
GetCount	Returns the number of command alternatives returned in a Command event.
GetItemId	Returns the ID for a specific Command alternative.
GetItemConfidence	Returns the value of the Confidence property for a specific Command alternative.
GetItemText	Returns the value of Voice text for a specific Command alternative.
GetAllItemData	Returns the data for all Command alternatives.

IAgentUserInput::GetAllItemData

```
HRESULT GetAllItemData(
    VARIANT * pdwItemIndices, // address of variable for alternative IDs
    VARIANT * plConfidences,   // address of variable for confidence scores
    VARIANT * pbszText        // address of variable for voice text
);
```

Retrieves the data for all **Command** alternatives passed to an **IAgentNotifySink::Command** callback.

- Returns S_OK to indicate the operation was successful.

pdwItemIndices

Address of a variable that receives the IDs of **Commands** passed to the **IAgentNotifySink::Command** callback.

plConfidences

Address of a variable that receives the confidence scores for **Command** alternatives passed to the **IAgentNotifySink::Command** callback.

pbszText

Address of a variable that receives the voice text for **Command** alternatives passed to the **IAgentNotifySink::Command** callback.

If voice input was not the source for the **Command**, for example, if the user selected the command from the character's pop-up menu, the Microsoft Agent server returns the ID of the **Command** selected, with a confidence score of 100 and voice text as NULL. The other alternatives return as NULL with confidence scores of zero (0) and voice text as NULL.

See also **IAgentUserInput::GetItemConfidence**, **IAgentUserInput::GetItemText**, **IAgentUserInput::GetItemID**

IAgentUserInput::GetCount

```
HRESULT GetCount(  
    long * pdwCount // address of a variable for number of alternatives  
);
```

Retrieves the number of **Command** alternatives passed to an **IAgentNotifySink::Command** callback.

- Returns S_OK to indicate the operation was successful.

pdwCount

Address of a variable that receives the count of **Commands** alternatives identified by the server.

If voice input was not the source for the command, for example, if the user selected the command from the character's pop-up menu, **GetCount** returns 1. If **GetCount** returns zero (0), the speech recognition engine detected spoken input but determined that there was no matching command.

IAgentUserInput::GetItemConfidence

```
HRESULT GetItemConfidence(  
    long dwItemIndex, // index of Command alternative  
    long * plConfidence // address of confidence value for Command  
);
```

Retrieves the confidence value for a **Command** passed to an **IAgentNotifySink::Command** callback.

- Returns S_OK to indicate the operation was successful.

dwItemIndex

The index of a **Command** alternative passed to the **IAgentNotifySink::Command** callback.

plConfidence

Address of a variable that receives the confidence score for a **Command** alternative passed to the **IAgentNotifySink::Command** callback.

If voice input was not the source for the command, for example, if the user selected the command from the character's pop-up menu, the Microsoft Agent server returns the confidence value of the best match as 100 and the confidence values for all other alternatives as zero (0).

See also **IAgentUserInput::GetItemID**, **IAgentUserInput::GetAllItemData**, **IAgentUserInput::GetItemText**

IAgentUserInput::GetItemID

```
HRESULT GetItemID(  
    long dwItemIndex,    // index of Command alternative  
    long * pdwCommandID // address of a variable for number of alternatives  
);
```

Retrieves the identifier of a **Command** alternative passed to an **IAgentNotifySink::Command** callback.

- Returns S_OK to indicate the operation was successful.

dwItemIndex

The index of the **Command** alternative passed to the **IAgentNotifySink::Command** callback.

pdwCommandID

Address of a variable that receives the ID of a **Command**.

If voice input triggers the **IAgentNotifySink::Command** callback, the server returns the IDs for any matching **Commands** defined by your application.

See also **IAgentUserInput::GetItemConfidence**, **IAgentUserInput::GetItemText**, **IAgentUserInput::GetAllItemData**

IAgentUserInput::GetItemText

```
HRESULT GetItemText(  
    Long dwItemIndex,    // index of Command alternative  
    BSTR * pbszText      // address of voice text for Command  
);
```

Retrieves the voice text for a **Command** alternative passed to the **IAgentNotifySink::Command** callback.

- Returns S_OK to indicate the operation was successful.

dwItemIndex

The index of a **Command** alternative passed to the **IAgentNotifySink::Command** callback.

pbszText

Address of a BSTR that receives the value of the voice text for the **Command**.

If voice input was not the source for the command, for example, if the user selected the command from the character's pop-up menu, the server returns NULL for the **Command's** voice text.

See also **IAgentUserInput::GetItemConfidence**, **IAgentUserInput::GetItemID**, **IAgentUserInput::GetAllItemData**

IAgentCommandWindow

IAgentCommandWindow defines an interface that allows applications to set and query the properties of the Commands Window. The Commands Window is a shared resource primarily designed for allowing users to view voice-enabled commands. If speech recognition is disabled or not installed, the Commands Window is not accessible. Attempting to set or query its properties will result in an error.

Methods in Vtable Order

IAgentCommandWindow Methods	Description
SetVisible	Sets the value of the Visible property of the Commands Window.
GetVisible	Returns the value of the Visible property of the Commands Window.
GetPosition	Returns the position of the Commands Window.
GetSize	Returns the size of the Commands Window.

IAgentCommandWindow::GetPosition

```
HRESULT GetPosition(  
    long * pLeft,    // address of variable for left-edge of Commands Window  
    long * pTop      // address of variable for top-edge of Commands Window  
);
```

Retrieves the Commands Windows' position.

- Returns S_OK to indicate the operation was successful.

pLeft

Address of a variable that receives the screen coordinate of the left edge of the Commands Window in pixels, relative to the screen origin (upper left).

plTop

Address of a variable that receives the screen coordinate of the top edge of the Commands Window in pixels, relative to the screen origin (upper left).

See also **IAgentCommandWindow::GetSize**

IAgentCommandWindow::GetSize

```
HRESULT GetSize(  
    long * plWidth,    // address of variable for Commands Window width  
    long * plHeight    // address of variable for Commands Window height  
);
```

Retrieves the current size of the Commands Window.

- Returns S_OK to indicate the operation was successful.

plWidth

Address of a variable that receives the width of the Commands Window in pixels, relative to the screen origin (upper left).

plHeight

Address of a variable that receives the height of the Commands Window in pixels, relative to the screen origin (upper left).

See also **IAgentCommandWindow::GetPosition**

IAgentCommandWindow::GetVisible

```
HRESULT GetVisible(  
    long * pbVisible    // address of variable for Visible setting for  
);                      // Commands Window
```

Determines whether the Commands Window is visible or hidden.

- Returns S_OK to indicate the operation was successful.

pbVisible

Address of a variable that receives TRUE if the Commands Window is visible, or FALSE if hidden.

See also **IAgentCommandWindow::SetVisible**

IAgentCommandWindow::SetVisible

```
HRESULT SetVisible(  
    long bVisible // Commands Window Visible setting  
);
```

Set the **Visible** property for the Commands Window.

- Returns S_OK to indicate the operation was successful.

bVisible

Visible property setting. A value of TRUE displays the Commands Window; FALSE hides it.

The user can override this property.

See also **IAgentCommandWindow::GetVisible**

IAgentSpeechInputProperties

IAgentSpeechInputProperties provides access to the speech recognition properties maintained by the server. Most of the properties are read-only for client applications, but the user can change them in the Microsoft Agent property sheet. The Microsoft Agent server only returns values if a compatible speech engine has been installed and is enabled. Querying these properties attempts to start the speech engine.

Methods in Vtable Order

IAgentSpeechInputProperties Methods	Description
GetInstalled	Returns whether a compatible speech recognition engine has been installed.
GetEnabled	Returns whether the speech recognition engine is enabled.
GetHotKey	Returns the current key assignment of the listening hot key.
GetLCID	Returns the locale (language) ID of the selected speech recognition engine.
GetEngine	Returns the ID of the selected speech recognition engine.
SetEngine	Sets the ID for the selected speech recognition engine.
GetListeningTip	Returns whether the Listening Tip is enabled.

IAgentSpeechInputProperties::GetEnabled

```
HRESULT GetEnabled(  
    long * pbEnabled // address of variable for speech recognition engine  
); // Enabled setting
```

Retrieves a value indicating whether the installed speech recognition engine is enabled.

- Returns S_OK to indicate the operation was successful.

pbEnabled

Address of a variable that receives TRUE if the speech engine is currently enabled and FALSE if disabled.

If **GetInstalled** returns FALSE, querying this setting returns an error.

See also **IAgentSpeechInput::GetInstalled**

IAgentSpeechInputProperties::GetEngine

```
HRESULT GetEngine(
    BSTR * pbszEngine // address of variable for speech engine mode ID
);
```

Retrieves the mode ID for the current selected speech recognition engine.

- Returns S_OK to indicate the operation was successful.

pbszEngine

Address of a BSTR that receives a string representation of the CLSID for the selected speech recognition engine.

If **GetInstalled** and **GetEnabled** return FALSE, querying this setting returns an error.

See also **IAgentSpeechInput::SetEngine**

IAgentSpeechInputProperties::GetHotKey

```
HRESULT GetHotKey(
    BSTR * pbszHotCharKey // address of variable for listening hotkey
);
```

Retrieves the current keyboard assignment for the speech input listening hot key.

- Returns S_OK to indicate the operation was successful.

pbszHotCharKey

Address of a BSTR that receives the current hot key setting used to open the audio channel for speech input.

If **GetInstalled** and **GetEnabled** return FALSE, querying this setting raises an error.

See also **IAgentSpeechInput::GetEnabled**, **IAgentSpeechInput::GetInstalled**

IAgentSpeechInputProperties::GetInstalled

```
HRESULT GetInstalled(
    long * pbInstalled // address of variable for speech recognition engine
);                      // installation flag
```

Retrieves a value indicating whether a speech recognition engine has been installed.

- Returns S_OK to indicate the operation was successful.

pbInstalled

Address of a variable that receives TRUE if a compatible speech recognition engine has been installed and FALSE if no engine is installed.

If **GetInstalled** and **GetEnabled** return FALSE, querying any other speech input properties returns an error.

See also **IAgentSpeechInput::GetEnabled**

IAgentSpeechInputProperties::GetLCID

```
HRESULT GetLCID(
    LCID * plcidCurrent // address of variable for locale ID
);
```

Retrieves the current setting for the locale ID.

- Returns S_OK to indicate the operation was successful.

plcidCurrent

Address of LCID that receives the current locale setting. The locale setting determines the language of the speech recognition engine.

If **GetInstalled** and **GetEnabled** return FALSE, querying this setting returns an error.

See also **IAgentSpeechInput::GetEnabled**, **IAgentSpeechInput::GetInstalled**

IAgentSpeechInputProperties::GetListeningTip

```
HRESULT GetListeningTip(
    long * pbListeningTip // address of variable for listening tip flag
);
```

Retrieves a value indicating whether the Listening Tip is enabled for display.

- Returns S_OK to indicate the operation was successful.

pbInstalled

Address of a variable that receives TRUE if the Listening Tip is enabled for display, or FALSE if the Listening Tip is disabled.

If **GetInstalled** and **GetEnabled** return FALSE, querying any other speech input properties returns an error.

See also **IAgentSpeechInput::GetEnabled**, **IAgentSpeechInput::GetInstalled**

IAgentSpeechInputProperties::SetEngine

```
HRESULT SetEngine(  
    BSTR bszEngine // speech engine mode ID  
);
```

Sets the selected speech recognition engine.

- Returns S_OK to indicate the operation was successful.

bszEngine

A BSTR that contains a string representation of the CLSID for the desired speech recognition mode (engine).

If **GetInstalled** and **GetEnabled** return FALSE, setting this property returns an error.

See also **IAgentSpeechInput::GetEngine**

IAgentAudioOutputProperties

IAgentAudioOutputProperties provides access to audio output properties maintained by the Microsoft Agent server. The properties are read-only, but the user can change them in the Microsoft Agent property sheet.

Methods in Vtable Order

IAgentAudioOutputProperties Methods	Description
GetEnabled	Returns whether audio output is enabled.
GetUsingSoundEffects	Returns whether sound-effect output is enabled.

IAgentAudioOutputProperties::GetEnabled

```
HRESULT GetEnabled(  
    long * pbEnabled // address of variable for audio output Enabled setting  
);
```

Retrieves a value indicating whether character speech output is enabled.

- Returns S_OK to indicate the operation was successful.

pbEnabled

Address of a variable that receives TRUE if the speech output is currently enabled and FALSE if disabled.

Because this setting affects spoken output (TTS and sound file) for all characters, only the user can change this property in the Microsoft Agent property sheet.

IAgentAudioOutputProperties::GetUsingSoundEffects

```
HRESULT GetUsingSoundEffects(  
    long * pbUsingSoundEffects // address of variable sound effects output  
);                               // setting
```

Retrieves a value indicating whether sound effects output is enabled.

- Returns S_OK to indicate the operation was successful.

pbUsingSoundEffects

Address of a variable that receives TRUE if the sound effects output is currently enabled and FALSE if disabled.

Sound effects for a character's animation are assigned in the Microsoft Agent Character Editor. Because this setting affects sound effects output for all characters, only the user can change this property in the Microsoft Agent property sheet.

IAgentPropertySheet

IAgentPropertySheet defines an interface that allows applications to set and query properties for the Microsoft Agent property sheet (window).

Methods in Vtable Order

IAgentPropertySheet Methods	Description
GetVisible	Returns whether the Microsoft Agent property sheet is visible.
SetVisible	Sets the Visible property of the Microsoft Agent property sheet.
GetPosition	Returns the position of the Microsoft Agent property sheet.
GetSize	Returns the size of the Microsoft Agent property sheet.
GetPage	Returns the current page for the Microsoft Agent property sheet.
SetPage	Sets the current page for the Microsoft Agent property sheet.

IAgentPropertySheet::GetPage

```
HRESULT GetPage(  
    BSTR * pbszPage // address of variable for current property page  
);
```

Retrieves the current page of the Microsoft Agent property sheet.

- Returns S_OK to indicate the operation was successful.

pbszPage

Address of a variable that receives the current page of the property sheet (last viewed page if the window is not open). The parameter can be one of the following:

“Speech”	The Speech Recognition page.
“Output”	The Output page.
“Copyright”	The Copyright page.

See also **IAgentPropertySheet::SetPage**

IAgentPropertySheet::GetPosition

```
HRESULT GetPosition(  
    long * plLeft,    // address of variable for left edge of property sheet  
    long * plTop      // address of variable for top edge of property sheet  
);
```

Retrieves the Microsoft Agent's property sheet window position.

- Returns S_OK to indicate the operation was successful.

plLeft

Address of a variable that receives the screen coordinate of the left edge of the property sheet in pixels, relative to the screen origin (upper left).

plTop

Address of a variable that receives the screen coordinate of the top edge of the property sheet in pixels, relative to the screen origin (upper left).

See also **IAgentPropertySheet::GetSize**

IAgentPropertySheet::GetSize

```
HRESULT GetSize(  
    long * plWidth,    // address of variable for property sheet width  
    long * plHeight    // address of variable for property sheet height  
);
```

Retrieves the size of the Microsoft Agent property sheet window.

- Returns S_OK to indicate the operation was successful.

plWidth

Address of a variable that receives the width of the property sheet in pixels, relative to the screen origin (upper left).

plHeight

Address of a variable that receives the height of the property sheet in pixels, relative to the screen origin (upper left).

See also **IAgentPropertySheet::GetPosition**

IAgentPropertySheet::GetVisible

```
HRESULT GetVisible(  
    long * pbVisible // address of variable for property sheet  
); // Visible setting
```

Determines whether the Microsoft Agent property sheet is visible or hidden.

- Returns S_OK to indicate the operation was successful.

pbVisible

Address of a variable that receives TRUE if the property sheet is visible and FALSE if hidden.

See also **IAgentPropertySheet::SetVisible**

IAgentPropertySheet::SetPage

```
HRESULT SetPage(  
    BSTR bszPage // current property page  
);
```

Sets the current page of the Microsoft Agent property sheet.

- Returns S_OK to indicate the operation was successful.

bszPage

A BSTR that sets the current page of the property. The parameter can be one of the following.

"Speech"	The Speech Recognition page.
"Output"	The Output page.

See also **IAgentPropertySheet::GetPage**

IAgentPropertySheet::SetVisible

```
HRESULT SetVisible(
    long bVisible // property sheet Visible setting
);
```

Sets the **Visible** property for the Microsoft Agent property sheet.

- Returns S_OK to indicate the operation was successful.

bVisible

Visible property setting. A value of TRUE displays the property sheet; a value of FALSE hides it.

See also **IAgentPropertySheet::GetVisible**

IAgentBalloon

IAgentBalloon defines an interface that allows applications to query properties for the Microsoft Agent word balloon.

Initial defaults for a character's word balloon are set in the Microsoft Agent Character Editor, but once the application is running, the user may override the Enabled and font properties. If a user changes the balloon's properties, the change affects all characters.

Methods in Vtable Order

IAgentBalloon Methods	Description
GetEnabled	Returns whether the word balloon is enabled.
GetNumLines	Returns the number of lines displayed in the word balloon.
GetNumCharsPerLine	Returns the average number of characters per line displayed in the word balloon.
GetFontName	Returns the name of the font displayed in the word balloon.
GetFontSize	Returns the size of the font displayed in the word balloon.
GetFontBold	Returns whether the font displayed in the word balloon is bold.
GetFontItalic	Returns whether the font displayed in the word balloon is italic.
GetFontStrkethru	Returns whether the font displayed in the word balloon is displayed as strikethrough.
GetFontUnderline	Returns whether the font displayed in the word balloon is underlined.

IAgentBalloon Methods	Description
GetForeColor	Returns the foreground color displayed in the word balloon.
GetBackColor	Returns the background color displayed in the word balloon.
GetBorderColor	Returns the border color displayed in the word balloon.
SetVisible	Sets the word balloon to be visible.
GetVisible	Returns the visibility setting for the word balloon.
SetFontName	Sets the font used in the word balloon.
SetFontSize	Sets the font size used in the word balloon.
SetFontCharSet	Sets the character set used in the word balloon.
GetFontCharSet	Returns the character set used in the word balloon.

IAgentBalloon::GetBackColor

```
HRESULT GetBackColor(
    long * plBGCOLOR // address of variable for background color displayed
);                  // in word balloon
```

Retrieves the value for the background color displayed in a word balloon.

- Returns S_OK to indicate the operation was successful.

plBGCOLOR

The address of a variable that receives the color setting for the balloon background.

The background color used in a character word balloon is defined in the Microsoft Agent Character Editor. It cannot be changed by an application. However, the user can change the background color of the word balloons for all characters through the Microsoft Agent property sheet.

See also **IAgentBalloon::GetForeColor**

IAgentBalloon::GetBorderColor

```
HRESULT GetBorderColor (
    long * plBorderColor// address of variable for border color displayed
);                  // for word balloon
```

Retrieves the value for the border color displayed for a word balloon.

- Returns S_OK to indicate the operation was successful.

plBorderColor

The address of a variable that receives the color setting for the balloon border.

The border color for a character word balloon is defined in the Microsoft Agent Character Editor. It cannot be changed by an application. However, the user can change the border color of the word balloons for all characters through the Microsoft Agent property sheet.

See also **IAgentBalloon::GetBackColor**, **IAgentBalloon::GetForeColor**

IAgentBalloon::GetEnabled

```
HRESULT GetEnabled(  
    long * pbEnabled // address of variable for Enabled setting  
); // for word balloon
```

Retrieves the value of the **Enabled** property for a word balloon.

- Returns S_OK to indicate the operation was successful.

pbEnabled

The address of a variable that receives TRUE when the word balloon is enabled and FALSE when it is disabled.

The Microsoft Agent server automatically displays the word balloon for spoken output, unless it is disabled. The word balloon can be disabled for a character in the Microsoft Agent Character Editor, or for all characters by the user, in the Microsoft Agent property sheet. If the user disables the word balloon, the client cannot restore it.

IAgentBalloon::GetFontBold

```
HRESULT GetFontBold(  
    long * pbFontBold // address of variable for bold setting for  
); // font displayed in word balloon
```

Indicates whether the font used in a word balloon is bold.

- Returns S_OK to indicate the operation was successful.

pbFontBold

The address of a value that receives TRUE if the font is bold and FALSE if not bold.

The font style used in a character word balloon is defined in the Microsoft Agent Character Editor. It cannot be changed by an application. However, the user can override the font settings for all characters through the Microsoft Agent property sheet.

IAgentBalloon::GetFontCharSet

```
HRESULT GetFontCharSet(  
    short * psFontCharSet // character set displayed in word balloon  
);
```

Indicates the character set of the font displayed in a word balloon.

- Returns S_OK to indicate the operation was successful.

psFontCharSet

The address of a value that receives the font's character set. The following are some common settings for value:

0	Standard Windows® characters (ANSI).
1	Default character set.
2	The symbol character set.
128	Double-byte character set (DBCS) unique to the Japanese version of Windows.
129	Double-byte character set (DBCS) unique to the Korean version of Windows.
134	Double-byte character set (DBCS) unique to the Simplified Chinese version of Windows.
136	Double-byte character set (DBCS) unique to the Traditional Chinese version of Windows.
255	Extended characters normally displayed by DOS applications.

For other character set values, consult the Microsoft Win32® documentation.

The default character set used in a character's word balloon is defined in the Microsoft Agent Character Editor. You can change it using **IAgentBalloon::SetFontCharSet**. However, the user can override the character set setting for all characters using the Microsoft Agent property sheet.

See also **IAgentBalloon::SetFontCharSet**

IAgentBalloon::GetFontItalic

```
HRESULT GetFontItalic(  
    long * pbFontItalic    // address of variable for italic setting for  
);                          // font displayed in word balloon
```

Indicates whether the font used in a word balloon is italic.

- Returns S_OK to indicate the operation was successful.

pbFontItalic

The address of a value that receives TRUE if the font is italic and FALSE if not italic.

The font style used in a character's word balloon is defined in the Microsoft Agent Character Editor. It cannot be changed by an application. However, the user can override the font settings for all characters through the Microsoft Agent property sheet.

IAgentBalloon::GetFontName

```

HRESULT GetFontName(
    BSTR * pbszFontName // address of variable for font displayed
);                      // in word balloon

```

Retrieves the value for the font displayed in a word balloon.

- Returns S_OK to indicate the operation was successful.

pbszFontName

The address of a BSTR that receives the font name displayed in a word balloon.

The default font used in a character word balloon is defined in the Microsoft Agent Character Editor. You can change it with **IAgentBalloon::SetFontName**. The user can override the font setting for all characters using the Microsoft Agent property sheet.

IAgentBalloon::GetFontSize

```

HRESULT GetFontSize(
    long * plFontSize // address of variable for font size
);                  // for font displayed in word balloon

```

Retrieves the value for the size of the font displayed in a word balloon.

- Returns S_OK to indicate the operation was successful.

plFontSize

The address of a value that receives the size of the font.

The default font size used in a character word balloon is defined in the Microsoft Agent Character Editor. You can change it with **IAgentBalloon::SetFontSize**. However, the user can override also the font size settings for all characters using the Microsoft Agent property sheet.

IAgentBalloon::GetFontStrikethru

```

HRESULT GetFontStrikethru(
    long * pbFontStrikethru // address of variable for strikethrough setting
);                          // for font displayed in word balloon

```

Indicates whether the font used in a word balloon has the strikethrough style set.

- Returns S_OK to indicate the operation was successful.

pbFontStrikethru

The address of a value that receives TRUE if the font strikethrough style is set and FALSE if not.

The font style used in a character word balloon is defined in the Microsoft Agent Character Editor. It cannot be changed by an application. However, the user can override the font settings for all characters using the Microsoft Agent property sheet.

IAgentBalloon::GetFontUnderline

```
HRESULT GetFontUnderline(  
    long * pbFontUnderline // address of variable for underline setting  
);                          // for font displayed in word balloon
```

Indicates whether the font used in a word balloon has the underline style set.

- Returns S_OK to indicate the operation was successful.

pbFontUnderline

The address of a value that receives TRUE if the font underline style is set and FALSE if not.

The font style used in a character word balloon is defined in the Microsoft Agent Character Editor. It cannot be changed by an application. However, the user can override the font settings for all characters using the Microsoft Agent property sheet.

IAgentBalloon::GetForeColor

```
HRESULT GetForeColor(  
    long * plFGColor // address of variable for foreground color displayed  
);                  // in word balloon
```

Retrieves the value for the foreground color displayed in a word balloon.

- Returns S_OK to indicate the operation was successful.

plFGColor

The address of a variable that receives the color setting for the balloon foreground.

The foreground color used in a character word balloon is defined in the Microsoft Agent Character Editor. It cannot be changed by an application. However, the user can override the foreground color of the word balloons for all characters through the Microsoft Agent property sheet.

See also **IAgentBalloon::GetBackColor**

IAgentBalloon::GetNumCharsPerLine

```
HRESULT GetNumCharsPerLine(  
    long * plCharsPerLine // address of variable for characters per line  
);                          // displayed in word balloon
```

Retrieves the value for the average number of characters per line displayed in a word balloon.

- Returns S_OK to indicate the operation was successful.

pbCharsPerLine

The address of a variable that receives the number of characters per line.

The Microsoft Agent server automatically scrolls the lines displayed for spoken output in the word balloon. The average number of characters per line for a character's word balloon is defined in the Microsoft Agent Character Editor. It cannot be changed by an application.

See also **IAgentBalloon::GetNumLines**

IAgentBalloon::GetNumLines

```
HRESULT GetNumLines(  
    long * pbcLines // address of variable for number of lines  
);                // displayed in word balloon
```

Retrieves the value of the number of lines displayed in a word balloon.

- Returns S_OK to indicate the operation was successful.

pbcLines

The address of a variable that receives the number of lines displayed.

The Microsoft Agent server automatically scrolls the lines displayed for spoken output in the word balloon. The number of lines for a character word balloon is defined in the Microsoft Agent Character Editor. It cannot be changed by an application.

See also **IAgentBalloon::GetNumCharsPerLine**

IAgentBalloon::GetVisible

```
HRESULT GetVisible(  
    long * pbVisible // address of variable for word balloon  
);                // Visible setting
```

Determines whether the word balloon is visible or hidden.

- Returns S_OK to indicate the operation was successful.

pbVisible

Address of a variable that receives TRUE if the word balloon is visible and FALSE if hidden.

See also **IAgentBalloon::SetVisible**

IAgentBalloon::SetFontCharSet

```
HRESULT SetFontCharSet(  
    short sFontCharSet // character set displayed in word balloon  
);
```

Sets the character set of the font displayed in the word balloon.

- Returns S_OK to indicate the operation was successful.

sFontCharSet

The character set of the font. The following are some common settings for value:

0	Standard Windows characters (ANSI).
1	Default character set.
2	The symbol character set.
128	Double-byte character set (DBCS) unique to the Japanese version of Windows.
129	Double-byte character set (DBCS) unique to the Korean version of Windows.
134	Double-byte character set (DBCS) unique to the Simplified Chinese version of Windows.
136	Double-byte character set (DBCS) unique to the Traditional Chinese version of Windows.
255	Extended characters normally displayed by DOS applications.

For other character set values, consult the Microsoft Win32 documentation.

The default character set used in a character's word balloon is defined in the Microsoft Agent Character Editor. You can change it with **IAgentBalloon::SetFontCharSet**. However, the user can override the character set setting for all characters using the Microsoft Agent property sheet.

See also **IAgentBalloon::GetFontCharSet**

IAgentBalloon::SetFontName

```
HRESULT SetFontName(  
    BSTR bszFontName // font displayed in word balloon  
);
```

Sets the font displayed in the word balloon.

- Returns S_OK to indicate the operation was successful.

bszFontName

A BSTR that sets the font displayed in the word balloon.

The default font used in a character's word balloon is defined in the Microsoft Agent Character Editor. You can change it with **IAgentBalloon::SetFontName**. However, the user can override the font setting for all characters using the Microsoft Agent property sheet.

See also **IAgentBalloon::GetVisible**

IAgentBalloon::SetFontSize

```
HRESULT SetFontSize(  
    long lFontSize // font size displayed in word balloon  
);
```

Sets the size of the font displayed in the word balloon.

- Returns S_OK to indicate the operation was successful.

lFontSize

The size of the font.

The default font size used in a character's word balloon is defined in the Microsoft Agent Character Editor. You can change it with **IAgentBalloon::SetFontSize**. However, the user can override the font size setting for all characters using the Microsoft Agent property sheet.

See also **IAgentBalloon::GetFontSize**

IAgentBalloon::SetVisible

```
HRESULT SetVisible(  
    long bVisible // word balloon Visible setting  
);
```

Sets the **Visible** property for the word balloon.

- Returns S_OK to indicate the operation was successful.

bVisible

Visible property setting. A value of TRUE displays the word balloon; a value of FALSE hides it.

See also **IAgentBalloon::GetVisibleEvents**

Microsoft Agent provides several events for tracking user interaction and server states. This section describes the event methods exposed by the **IAgentNotifySink** interface.

Events

Methods in Vtable Order

IAgentNotifySink	Description
Command	Occurs when the server processes a client-defined command.
ActivateInputState	Occurs when a character becomes or ceases to be input-active.
Restart	Occurs when the server restarts.
Shutdown	Occurs when the user exits the server.
VisibleState	Occurs when the character's Visible state changes.
Click	Occurs when a character is clicked.
DbClick	Occurs when a character is double-clicked.
DragStart	Occurs when a user starts dragging a character.
DragComplete	Occurs when a user stops dragging a character.
RequestStart	Occurs when the server begins processing a Request object.
RequestComplete	Occurs when the server completes processing a Request object.
Bookmark	Occurs when the server processes a bookmark.
Idle	Occurs when the server starts or ends idle processing.
Move	Occurs when a character has been moved.
Size	Occurs when a character has been resized.
BalloonVisibleState	Occurs when the visibility state of a character's word balloon changes.

IAgentNotifySink::ActivateInputState

```
HRESULT ActivateInputState(  
    long dwCharID,    // character ID  
    long bActivated    // input activation flag  
);
```

Notifies a client application that a character's input active state changed.

- No return value.

dwCharID

Identifier of the character whose input activation state changed.

bActivated

Input active flag. This Boolean value is TRUE if the character referred to by *dwCharID* became input active; and FALSE if the character lost its input active state.

See also **IAgentCharacter::SetInputActive**, **IAgentCharacter::GetInputActive**

IAgentNotifySink::BalloonVisibleState

```
HRESULT BalloonVisibleState(  
    long dwCharID,    // character ID  
    long bVisible     // visibility flag  
);
```

Notifies a client application when the visibility state of the character's word balloon changes.

- No return value.

dwCharID

Identifier of the character whose word balloon's visibility state has changed.

bVisible

Visibility flag. This Boolean value is TRUE when character's word balloon becomes visible; and FALSE when it becomes hidden.

This event is sent to all clients of the character.

IAgentNotifySink::Bookmark

```
HRESULT Bookmark(  
    long dwBookMarkID // bookmark ID  
);
```

Notifies a client application when its bookmark completes.

- No return value.

dwBookMarkID

Identifier of the bookmark that resulted in triggering the event.

When you include bookmark tags in a **Speak** method, you can track when they occur with this event.

See also **IAgentCharacter::Speak**, Speech Output Tags

IAgentNotifySink::Click

```
HRESULT Click(  
    long dwCharID,    // character ID  
    short fwKeys,     // mouse button and modifier key state  
    long x,           // x coordinate of mouse pointer  
    long y             // y coordinate of mouse pointer  
);
```

Notifies a client application when the user clicks a character.

- No return value.

dwCharID

Identifier of the clicked character.

fwKeys

A parameter that indicates the mouse button and modifier key state. The parameter can return any combination of the following:

0x0001 Left Button

0x0010 Middle Button

0x0002 Right Button

0x0004 Shift Key Down

0x0008 Control Key Down

0x0020 Alt Key Down

x

The x-coordinate of the mouse pointer in pixels, relative to the screen origin (upper left).

y

The y-coordinate of the mouse pointer in pixels, relative to the screen origin (upper left).

IAgentNotifySink::Command

```
HRESULT Command(
    long dwCommandID,           // Command ID of the best match
    IUnknown * punkUserInput    // address of IAgentUserInput object
);
```

Notifies a client application that a Command was selected by the user.

- No return value.

dwCommandID

Identifier of the best match command alternative.

punkUserInput

Address of the IUnknown interface for the **IAgentUserInput** object.

Use QueryInterface to retrieve the **IAgentUserInput** interface.

See also **IAgentUserInput**

IAgentNotifySink::DbClick

```
HRESULT DbClick(  
    long dwCharID,    // character ID  
    short fwKeys,     // mouse button and modifier key state  
    long x,           // x coordinate of mouse pointer  
    long y            // y coordinate of mouse pointer  
);
```

Notifies a client application when the user double-clicks a character.

- No return value.

dwCharID

Identifier of the double-clicked character.

fwKeys

A parameter that indicates the mouse button and modifier key state. The parameter can return any combination of the following:

0x0001	Left Button
0x0010	Middle Button
0x0002	Right Button
0x0004	Shift Key Down
0x0008	Control Key Down
0x0020	Alt Key Down

x

The x-coordinate of the mouse pointer in pixels, relative to the screen origin (upper left).

y

The y-coordinate of the mouse pointer in pixels, relative to the screen origin (upper left).

IAgentNotifySink::DragComplete

```
HRESULT DragComplete(  
    long dwCharID,    // character ID  
    short fwKeys,     // mouse button and modifier key state  
    long x,           // x-coordinate of mouse pointer  
    long y            // y-coordinate of mouse pointer  
);
```

Notifies a client application when the user stops dragging a character.

- No return value.

dwCharID

Identifier of the dragged character.

fwKeys

A parameter that indicates the mouse button and modifier key state. The parameter can return any combination of the following:

0x0001	Left Button
0x0010	Middle Button
0x0002	Right Button
0x0004	Shift Key Down
0x0008	Control Key Down
0x0020	Alt Key Down

x

The x-coordinate of the mouse pointer in pixels, relative to the screen origin (upper left).

y

The y-coordinate of the mouse pointer in pixels, relative to the screen origin (upper left).

IAgentNotifySink::DragStart

```
HRESULT DragStart(  
    long dwCharID,    // character ID  
    short fwKeys,     // mouse button and modifier key state  
    long x,           // x-coordinate of mouse pointer  
    long y            // y-coordinate of mouse pointer  
);
```

Notifies a client application when the user starts dragging a character.

- No return value.

dwCharID

Identifier of the dragged character.

fwKeys

A parameter that indicates the mouse button and modifier key state. The parameter can return any combination of the following:

0x0001	Left Button
0x0010	Middle Button
0x0002	Right Button
0x0004	Shift Key Down
0x0008	Control Key Down
0x0020	Alt Key Down

x

The x-coordinate of the mouse pointer in pixels, relative to the screen origin (upper left).

y

The y-coordinate of the mouse pointer in pixels, relative to the screen origin (upper left).

IAgentNotifySink::Idle

```
HRESULT Idle(
    long dwCharID,    // character ID
    long bStart       // start flag
);
```

Notifies a client application when a character's **Idling** state has changed.

- No return value.

dwCharID

Identifier of the request that started.

bStart

Start flag. This Boolean value is TRUE when the character begins idling and FALSE when it stops idling.

This event enables you to track when the Microsoft Agent server starts or stops idle processing for a character.

See also **IAgentCharacter::GetIdleOn**, **IAgentCharacter::SetIdleOn**

IAgentNotifySink::Move

```
HRESULT Move(
    long dwCharID,    // character ID
    long x,           // x-coordinate of new location
    long y,           // y-coordinate of new location
);
```

```

        long dwCause      // cause of move state
    );

```

Notifies a client application when the character has been moved.

- No return value.

dwCharID

Identifier of the character that has been moved.

x

The x-coordinate of the new position in pixels, relative to the screen origin (upper left). The location of a character is based on the upper left corner of its animation frame.

y

The y-coordinate of the new position in pixels, relative to the screen origin (upper left). The location of a character is based on the upper left corner of its animation frame.

dwCause

The cause of the character move. The parameter may be one of the following:

const unsigned short NeverMoved = 0;	Character has not been moved.
const unsigned short UserMoved = 1;	User dragged the character.
const unsigned short ProgramMoved = 2;	Your application moved the character.
const unsigned short OtherProgramMoved = 3;	Another application moved the character.

This event is sent to all clients of the character.

See also **IAgentCharacter::GetMoveCause**, **IAgentCharacter::MoveTo**

IAgentNotifySink::RequestComplete

```

HRESULT RequestComplete(
    long dwRequestID,    // request ID
    long hrStatus        // status code
);

```

Notifies a client application when a request completes.

- No return value.

dwRequestID

Identifier of the request that started.

hrStatus

Status code. This parameters returns the status code for the request.

This event enables you to track when a queued method completes.

See also **IAgentNotifySink::RequestStart**, **IAgent::Load**, **IAgentCharacter::GestureAt**, **IAgentCharacter::Hide**, **IAgentCharacter::Interrupt**, **IAgentCharacter::MoveTo**, **IAgentCharacter::Prepare**, **IAgentCharacter::Play**, **IAgentCharacter::Show**, **IAgentCharacter::Speak**, **IAgentCharacter::Wait**

IAgentNotifySink::RequestStart

```
HRESULT RequestStart(  
    long dwRequestID // request ID  
);
```

Notifies a client application when a request begins.

- No return value.

dwRequestID

Identifier of the request that started.

This event enables you to track when a queued request begins.

See also **IAgentNotifySink::RequestComplete**, **IAgent::Load**, **IAgentCharacter::GestureAt**, **IAgentCharacter::Hide**, **IAgentCharacter::Interrupt**, **IAgentCharacter::MoveTo**, **IAgentCharacter::Prepare**, **IAgentCharacter::Play**, **IAgentCharacter::Show**, **IAgentCharacter::Speak**, **IAgentCharacter::Wait**

IAgentNotifySink::Restart

```
HRESULT Restart();
```

Notifies a client application that the Microsoft Agent server restarted.

- No return value.

See also **IAgentNotifySink::Shutdown**

IAgentNotifySink::Shutdown

```
HRESULT Shutdown();
```

Notifies a client application that the Microsoft Agent server shut down.

- No return value.

This event fires only when the user explicitly chooses the Exit command on the pop-up menu of the Microsoft Agent taskbar icon. Requests sent after the server shuts down will fail.

See also **IAgentNotifySink::Restart**

IAgentNotifySink::Size

```
HRESULT Size(  
    long dwCharID,    // character ID  
    long lWidth,      // new width  
    long lHeight,     // new height  
);
```

Notifies a client application when the character has been resized.

- No return value.

dwCharID

Identifier of the character that has been resized.

lWidth

The width of the character's animation frame in pixels.

lHeight

The height of the character's animation frame in pixels.

This event is sent to all clients of the character.

See also **IAgentCharacter::GetSize**, **IAgentCharacter::SetSize**

IAgentNotifySink::VisibleState

```
HRESULT VisibleState(  
    long dwCharID,    // character ID  
    long bVisible,    // visibility flag  
    long dwCause,     // cause of visible state  
);
```

Notifies a client application when the visibility state of the character changes.

- No return value.

dwCharID

Identifier of the character whose visibility state is changed.

bVisible

Visibility flag. This Boolean value is TRUE when character becomes visible and FALSE when the character becomes hidden.

dwCause

Cause of last change to the character's visibility state. The parameter may be one of the following:

const unsigned short NeverShown = 0;	Character has not been shown.
const unsigned short UserHid = 1;	User hid the character.
const unsigned short UserShowed = 2;	User showed the character.
const unsigned short ProgramHid = 3;	Your application hid the character.
const unsigned short ProgramShowed = 4;	Your application showed the character.
const unsigned short OtherProgramHid = 5;	Another application hid the character.
const unsigned short OtherProgramShowed = 6;	Another application showed the character.

See also **IAgentCharacter::GetVisible**, **IAgentCharacter::SetVisible**, **IAgentCharacter::GetVisibilityCause**