GO-GLOBAL

Session Manager API

Version 4.8.2

COPYRIGHT AND TRADEMARK NOTICE

CONTENTS

1. Session Manager Interface — page 5

SMI_AddPublishedApplication
SMI_ConnectClient
SMI_CreateSession
SMI_DestroySession
SMI_DisconnectClient
SMI_DisconnectServer
SMI_EditApplicationUserSettings
SMI_EditPublishedApplication
SMI_EnumerateApplications
SMI_EnumerateApplicationUserSettings
SMI_EnumerateClients
SMI_EnumeratePublishedApplications
SMI_EnumerateServers
SMI_EnumerateSessions
SMI_GetApplicationInfo
SMI_GetClientInfo
SMI_GetServerInfo
SMI_GetServerMetrics
SMI_GetSessionInfo
SMI_Initialize
SMI_LaunchApplication
SMI_RemovePublishedApplication
SMI_RestoreApplicationUserSetting
SMI_ResumeSession
SMI_SetConnectionOptions
SMI_SetEventClient
SMI_SetServerInfo
SMI_Shutdown
SMI_SuspendSession
SMI_TerminateApplication

2. Session Manager Callback Prototypes — page 31

OnApplicationConnect
OnApplicationDisconnect
OnClientConnect
OnClientDisconnect
OnServerConnect
OnServerDisconnect

3. Session Manager Callback Initialization Functions — page 35

SMI_SetApplicationConnectCB
SMI_SetApplicationDisconnectCB
SMI_SetClientConnectCB
SMI_SetClientDisconnectCB
SMI_SetServerConnectCB
SMI_SetServerDisconnectCB

4. GO-Global Authentication (GGA) Interface — page 37

GGA_AuthenticateUser

5. GO-Global ActiveX Client Interface — page 38

IEGCtrl Class Properties

get/put Compression
get/put Embedded
get/put HostName
get/put HostPort
get/put Launch
get/put LaunchArgs
get/put OverlayWindow
put Password
get/put Session
put SessionCredentialsAuthority
put SessionCredentialsString
get/put UserName

IEGCtrl Class Methods

Connect
Create

6. Session Manager Interface Structures — page 47

SMI_APPLICATION_INFO
SMI_APPLICATION_SETTING
SMI_CLIENT_INFO
SMI_PUBLISHED_APPLICATION
SMI_SERVER_INFO
SMI_SERVER_METRICS
SMI_SESSION_CREATON_INFO
SMI_SESSION_CREDENTIALS
SMI_SESSION_INFO

7. Session Manager Interface Constants — page 61

SMI_APPLICATION_SETTING_ERROR
SMI_CLIENT_OPTIONS
SMI_CONNOPT
SMI_METRICS
SMI_PASS_THROUGH_AUTHENTICATION
SMI_PERMISSION
SMI_PRINTER_DRIVER_SRC
SMI_SESSIONCREATION_FLAGS
SMI_SESSIONSTATE
SMI_STARTSTATE

Event Sequence Diagrams — page 69

Create Session Sequence
Connect Client Sequence
Destroy Session Sequence

OVERVIEW

The GO-Global Session Manager API provides programmatic access to the functionality of the GO-Global Session Manager. It allows clients to configure GO-Global Hosts and to create and manage GO-Global sessions.

1. SESSION MANAGER INTERFACE

The Session Manager Interface supports operations such as configuring published applications, creating sessions, adding users and applications to a session, and specifying which client has the mouse and keyboard control. With this interface, it is possible to have multiple processes (*e.g.,* an administration application and a session manager) running on different machines and communicating with the same GO-Global Session Manager. To accomplish this, the Session Manager Interface is implemented using the GraphOn protocol.

The Session Manager Interface functions are as follows:

SMI_AddPublishedApplication

Function **SMI_AddPublishedApplication** publishes an application on the specified server.

```
SM_BOOL SMI_AddPublishedApplication
          (
            SMI_SERVER_ID               serverID,
            SMI_PUBLISHED_APPLICATION*  application
          );
```

Parameters

serverID
[in] Server

application
[in] Application

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_ConnectClient

Function **SMI_ConnectClient** connects a client to the specified session.

```
SM_BOOL SMI_ConnectClient
         (
           SMI_SERVER_ID   sessionServerID,
           SMI_SESSION_ID  sessionID,
           SMI_SERVER_ID   clientServerID,
           SMI_CLIENT_ID   clientID,
           SM_BOOL         requestPermission
         );
```

Parameters

sessionServerID
[in] Server on which the session is running

sessionID
[in] Session

clientServerID
[in] Server to which the client is connected

clientID
[in] Client

requestPermission
[in] SM_TRUE if the session's owner should be prompted to authorize the connection. SM_FALSE, if the client should be connected without notifying the session's owner.

Return Values

SM_TRUE on success, SM_FALSE on error

SMI_CreateSession

Function **SMI_CreateSession** creates a session.

```
SMI_SESSION_ID SMI_CreateSession
          (
            SMI_SERVER_ID              serverID,
            SMI_SESSION_CREATION_INFO creationInfo,
            SM_WCHAR*                  applicationName,
            SM_WCHAR*                  applicationArguments,
            SM_ULONG                   bufferSize,
            void*                      buffer
          );
```

Parameters

serverID
[in] Server

creationInfo
[in] "Shape" of the new session (see SMI_SESSION_CREATION_INFO)

applicationName
[in] Application name; must be identical to field *applicationName* in the SMI_PUBLISHED_APPLICATION structure

applicationArguments
[in] Application arguments

bufferSize
[in] The size of buffer. If *bufferSize* is 0, then buffer will be ignored.

buffer
[in] User-defined data for custom authentication. Please refer to the GGA component for how to customize authentication.

Return Values

A positive value identifies the new session; -1 indicates failure.

SMI_DestroySession

Function **SMI_DestroySession** terminates the specified session.

```
SM_BOOL SMI_DestroySession
         (
           SMI_SERVER_ID   serverID,
           SMI_SESSION_ID  sessionID
         );
```

Parameters

serverID
[in] Server

sessionID
[in] The session to destroy

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_DisconnectClient

Function **SMI_DisconnectClient** disconnects a client from the specified session.

```
SM_BOOL SMI_DisconnectClient
        (
          SMI_SERVER_ID   serverID,
          SMI_CLIENT_ID   clientID,
          SM_UINT         sessionTimeout,
          SM_BOOL         notifyUser
        );
```

Parameters

serverID
[in] Server

clientID
[in] Client

timeout
[in] Minimum number of seconds the session should remain running after the client is disconnected. This value is only used if the session is configured to terminate immediately when there are no clients connected to it.

notifyUser
[in] SM_TRUE if the user is to be notified that he or she has been disconnected. SM_FALSE if the user is not be notified.

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_DisconnectServer

Function **SMI_DisconnectServer** disconnects a server from the relay server, dropping all client connections. All sessions on the server will be terminated.

```
SM_BOOL SMI_DisconnectServer
          (
            SMI_SERVER_ID  serverID
          );
```

Parameters

serverID
[in] Server to disconnect

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_EditApplicationUserSettings

Function **SMI_EditApplicationUserSettings** modifies an existing user/group setting for a particular application.

```
SM_BOOL SMI_EditApplicationUserSettings
          (
            SMI_SERVER_ID             serverID,
            SM_WCHAR*                 applicationName,
            SMI_APPLICATION_SETTING*  setting
          );
```

Parameters

serverID
[in] Server

applicationName
[in] name of application to modify

setting
[in] User/group setting

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_EditPublishedApplication

Function **SMI_EditPublishedApplication** edits an existing application on a particular server.

```
SM_BOOL SMI_EditPublishedApplication
        (
          SMI_SERVER_ID               serverID,
          SM_WCHAR*                   applicationName
          SMI_PUBLISHED_APPLICATION*  newSetting
        );
```

Parameters

serverID
[in] Server

applicationName
[in] current display name of the application to modify

newSetting
[in] new configuration of the application

Return Values

SM_TRUE if successful, SM_FALSE if not

Remarks

Because the application's current display name is passed as a separate parameter, this function can be used to change all attributes of a published application, including the display name itself. If the display name should not be changed, `newSetting->applicationName` must be set to the same string as parameter *applicationName*.

SMI_EnumerateApplications

Function **SMI_EnumerateApplications** enumerates the applications connected to the specified session.

```
SMI_APP_ID SMI_EnumerateApplications
          (
            SMI_SERVER_ID   serverID,
            SMI_SESSION_ID  sessionID,
            SMI_APP_ID      applicationID
          );
```

Parameters

serverID
[in] Server

sessionID
[in] Session

applicationID
[in] Application identifier

Return Values

A positive value for the next application identifier, 0 when *applicationID* is the last element, or –1 to indicate failure

SMI_EnumerateApplicationUserSettings

Function **SMI_EnumerateApplicationUserSettings** retrieves user/group settings for a particular application.

```
int SMI_EnumerateApplicationUserSettings
          (
            SMI_SERVER_ID             serverID,
            SM_WCHAR*                 applicationName,
            SM_WCHAR*                 currentSetting,
            SMI_APPLICATION_SETTING*  setting
          );
```

Parameters

serverID
[in] Server

applicationName
[in] name of application to modify

currentSetting
[in] Current user/group settings

setting
[out] Next user/group settings

Return Values

A positive value for the next user/group settings, 0 when *currentSetting* is the last element, or –1 to indicate failure

SMI_EnumerateClients

Function **SMI_EnumerateClients** enumerates clients connected to a particular session.

```
SMI_EnumerateClients
          (
            SMI_SERVER_ID   sessionServerID,
            SMI_SESSION_ID  sessionID,
            SMI_SERVER_ID   clientServerID,
            SMI_CLIENT_ID   clientID
          );
```

Parameters

sessionServerID
[in] Server on which the session is running

sessionID
[in] Session

clientServerID
[in] Server to which the client is connected

clientID
[in] The current client

Return Values

A positive value for the next client identifier, 0 when *clientID* is the last element, or –1 to indicate failure

SMI_EnumeratePublishedApplications

Function **SMI_EnumeratePublishedApplications** enumerates the published applications on a particular server.

```
int SMI_EnumeratePublishedApplications
          (
            SMI_SERVER_ID               serverID,
            SMI_PUBLISHED_APPLICATION*  currentApplication,
            SMI_PUBLISHED_APPLICATION*  application
          );
```

Parameters

serverID
[in] Server

currentApplication
[in] Current published application

application
[out] The next published application

Return Values

A positive value for the next application, 0 when *currentApplication* is the last element, or –1 to indicate failure

SMI_EnumerateServers

Function **SMI_EnumerateServers** retrieves the list of servers.

```
SMI_SERVER_ID SMI_EnumerateServers
          (
            SMI_SERVER_ID  currentID
          );
```

Parameters

currentID
[in] Server

Return Values

A positive value for the next server ID, 0 when *currentID* is the last element, or –1 to indicate failure

SMI_EnumerateSessions

Function **SMI_EnumerateSessions** retrieves the list of sessions for a particular server.

```
SMI_SESSION_ID SMI_EnumerateSessions
          (
            SMI_SERVER_ID   serverID,
            SMI_SESSION_ID  currentID
          );
```

Parameters

serverID
[in] Server

currentID
[in] Session

Return Values

A positive value for the next session ID, 0 when *currentID* is the last element, or –1 to indicate failure

SMI_GetApplicationInfo

Function **SMI_GetApplicationInfo** retrieves the application data for a particular application.

```
SM_BOOL SMI_GetApplicationInfo
         (
           SMI_SERVER_ID          serverID,
           SMI_SESSION_ID         sessionID,
           SMI_APP_ID             applicationID,
           SMI_APPLICATION_INFO*  info
         );
```

Parameters

serverID
[in] Server

sessionID
[in] Session

applicationID
[in] Application

info
[out] Application data

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_GetClientInfo

Function **SMI_GetClientInfo** retrieves information about a particular client.

```
SM_BOOL SMI_GetClientInfo
         (
           SMI_SERVER_ID     serverID,
           SMI_CLIENT_ID     clientID,
           SMI_CLIENT_INFO*  info
         );
```

Parameters

serverID
[in] Server

clientID
[in] Client

info
[out] Client information

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_GetServerInfo

Function **SMI_GetServerInfo** retrieves information for a particular server.

```
SM_BOOL SMI_GetServerInfo
          (
            SMI_SERVER_ID     serverID,
            SMI_SERVER_INFO*  info
          );
```

Parameters

serverID
[in] Server

info
[out] The server information (IP address, *etc.*)

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_GetServerMetrics

Function **SMI_GetServerMetrics** obtains server performance metrics, *e.g.,* CPU and memory usage. Prior to calling the function, set structure field SMI_SERVER_METRICS.flags to one or more values enumerated in SMI_METRICS.

```
SM_BOOL SMI_GetServerMetrics
         (
           SMI_SERVER_ID       serverID,
           SMI_SERVER_METRICS* metrics
         );
```

Parameters

serverID
[in] Server

metrics
[in/out] Server metrics information

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_GetSessionInfo

Function **SMI_GetSessionInfo** retrieves the information associated with a session.

```
SM_BOOL SMI_GetSessionInfo
         (
           SMI_SERVER_ID      serverID,
           SMI_SESSION_ID     sessionID,
           SMI_SESSION_INFO*  info
         );
```

Parameters

serverID
[in] Session

sessionID
[in] Session

info
[out] Session information

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_Initialize

Function **SMI_Initialize** initializes the module. It must be called after all callback functions have been initialized. For information about initializing a callback, refer to the Session Manager Callback Prototypes section of this document.

```
SM_BOOL SMI_Initialize
         (
           SM_WCHAR_CPtr   hostName,
           SM_USHORT        port
         );
```

Parameters

hostName
[in] Host name of the relay server to access.

port
[in] port number on which relay server is listening (0 for default)

Return Values

SM_TRUE if successful, SM_FALSE if not

See Also

SMI_Shutdown

SMI_LaunchApplication

Function **SMI_LaunchApplication** starts an application within a particular session.

```
SM_BOOL SMI_LaunchApplication
        (
          SMI_SERVER_ID   serverID,
          SMI_SESSION_ID  sessionID,
          SM_WCHAR*       applicationName,
          SM_WCHAR*       applicationArguments
        );
```

Parameters

serverID
[in] Server

sessionID
[in] Session

applicationName
[in] Application name; must be identical to field *applicationName* in the SMI_PUBLISHED_APPLICATION structure

arguments
[in] Application arguments. If NULL, the defaults are used. If an empty argument list is required, use an empty string (“”).

Return Values

SM_FALSE if an error was detected, SM_TRUE otherwise.

Remarks

An error may occur and not be detected. To determine whether a launch was successful, an application should wait for the corresponding OnApplicationConnect event.

Due to implementation requirements, a single SMI_LaunchApplication call may result in multiple occurrences of both OnApplicationConnect and OnApplicationDisconnect events.

SMI_RemovePublishedApplication

Function **SMI_RemovePublishedApplication** removes an application from a particular server.

```
SM_BOOL SMI_RemovePublishedApplication
         (
           SMI_SERVER_ID  serverID,
           SM_WCHAR*      applicationName
         );
```

Parameters

serverID
[in] Server

applicationName
[in] name of application to remove

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_RestoreApplicationUserSetting

Function **SMI_RestoreApplicationUserSetting** restores the default setting for an existing user/group.

```
SM_BOOL SMI_RestoreApplicationUserSetting
         (
           SMI_SERVER_ID  serverID,
           SMI_WCHAR*     applicationName,
           SMI_WCHAR*     settingName
         );
```

Parameters

serverID
[in] Server

applicationName
[in] Application

settingName
[in] User/group setting

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_ResumeSession

Function **SMI_ResumeSession** resumes a particular suspended session.

```
SM_BOOL SMI_ResumeSession
         (
           SMI_SERVER_ID   serverID,
           SMI_SESSION_ID  sessionID
         );
```

Parameters

serverID
[in] Server

sessionID
[in] Session

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_SetConnectionOptions

Function **SMI_SetConnectionOptions** sets different connection options, *e.g.,* encryption. See SMI_CONNOPT for available options. This function must be called before SMI_Initialize.

```
SM_BOOL SMI_SetConnectionOptions
          (
            SM_ULONG  connectionOptions
          );
```

Parameters

connectionOptions
[in] Connection options

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_SetEventClient

Function **SMI_SetEventClient** gives a particular client the input (keyboard and mouse) focus within the client's session.

```
SM_BOOL SMI_SetEventClient
          (
            SMI_SERVER_ID  serverID,
            SMI_CLIENT_ID  clientID
          );
```

Parameters

serverID
[in] Server

clientID
[in] Client

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_SetServerInfo

Function **SMI_SetServerInfo** modifies the settings for the specified server.

```
SM_BOOL SMI_SetServerInfo
          (
            SMI_SERVER_ID          serverID,
            SMI_SERVER_INFO_Ptr    serverInfo
          );
```

Parameters

serverID
[in] Server

serverInfo
[in] New settings for the server (client options, timeouts, etc.)

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_Shutdown

Function **SMI_Shutdown** releases whatever resources have been allocated to the module.

```
SM_BOOL SMI_Shutdown
          (void);
```

Return Values

SM_TRUE if successful, SM_FALSE if not

Remarks

This function is generally called either during shutdown of the Session Manager Client application to release the Session Manager Client DLL's resources or to disconnect from one server and then connect to the same or a different server using SMI_Initialize.

See Also

SMI_Initialize

SMI_SuspendSession

Function **SMI_SuspendSession** suspends a particular session.

```
SM_BOOL SMI_SuspendSession
         (
           SMI_SERVER_ID   serverID,
           SMI_SESSION_ID  sessionID
         );
```

Parameters

serverID
[in] Server

sessionID
[in] Session

Return Values

SM_TRUE if successful, SM_FALSE if not

SMI_TerminateApplication

Function **SMI_TerminateApplication** terminates a particular application.

```
SM_BOOL SMI_TerminateApplication
         (
           SMI_SERVER_ID   serverID,
           SMI_SESSION_ID  sessionID,
           SMI_APP_ID      applicationID
         );
```

Parameters

serverID
[in] Server

sessionID
[in] Session

applicationID
[in] Application

Return Values

SM_TRUE if successful, SM_FALSE if not

2 SESSION MANAGER CALLBACK PROTOTYPES

The Session Manager callback can be used to modify and extend the functionality of the GO-Global Session Manager.

OnClientConnect

The **OnClientConnect** callback function allows an application to decide which actions to perform when a client connects. This callback should be overridden because there is no user authentication.

```
typedef SM_BOOL (* OnClientConnect)
                                (
                                  SMI_SERVER_ID         clientServerID,
                                  SMI_CLIENT_ID         clientID,
                                  SMI_SERVER_ID         sessionServerID,
                                  SMI_SESSION_ID        sessionID,
                                  SM_WCHAR_CPtr         sessionUUID,
                                  SMI_SESSION_CREDENTIALS_Ptr credentials
                                );
```

Parameters

clientServerID
[in] The ID of the server to which the client is connected.

clientID
[in] The ID of the client. Use SMI_GetClientInfo to retrieve additional information.

sessionServerID
[in] The ID of the server on which the session is running.

sessionID
[in] The session to which the client should be attached. This value is retrieved from the client. If *sessionID* is 0, a new session will be created using the default client information.

sessionUUID
[in] The session UUID specified in SMI_CreateSession.

credentials
[in] The credentials used to authenticate the client's connection to the session. The default implementation does not use this parameter.

Return Values

SM_TRUE if the client is accepted, SM_FALSE if it is rejected.

Remarks

Typical actions performed by this callback are client authentication, session creation, and, using SMI_ConnectClient, attachment of the client to a session. The return value is SM_TRUE if no callback is installed.

If a client connection is rejected by at least one session manager, the client connection is closed and *clientID* becomes invalid. If all session managers accept the connection but none uses SMI_ConnectClient, the client is connected to the session identified by parameter *sessionID* if nonzero; otherwise, the client is connected to a new session.

OnClientDisconnect

The **OnClientDisconnect** callback function is called when a client has disconnected. There is no default behavior to override because disconnect has already occurred.

```
typedef void (* OnClientDisconnect)
                        (
                              SMI_SERVER_ID         clientServerID,
                              SMI_CLIENT_ID         clientID,
                              SMI_SERVER_ID         sessionServerID,
                              SMI_SESSION_ID        sessionID,
                              SM_WCHAR_CPtr         sessionUUID
                        );
```

Parameters

clientServerID
[in] The ID of the server to which the client is connected.

clientID
[in] The ID of the client. Use SMI_GetClientInfo to retrieve additional information.

sessionServerID
[in] The ID of the server on which the session is running.

sessionID
[in] The session to which the client should be attached. This value is retrieved from the client. If *sessionID* is 0, a new session will be created using the default client information.

sessionUUID
[in] The session UUID specified in SMI_CreateSession.

Return Values

This function does not return a value.

Remarks

Parameter *clientID* will not be valid after this function returns.

OnServerConnect

The **OnServerConnect** callback function allows an application to decide which actions to take when a server connects to the Session Manager. In particular, the callback must decide whether the new connection is accepted.

```
typedef SM_BOOL (* OnServerConnect)
                              (
                                 SMI_SERVER_ID  serverID,
                                 SMI_WCHAR_CPtr hardwareID,
                                 SMI_WCHAR_CPtr credentials,
                                 SMI_WCHAR_CPtr address,
                                 SM_ULONG       port,
                                 SM_ULONG       connOpt,
                                 SM_ULONG       role
                              );
```

Parameters

serverID
[in] Server. Use SMI_GetServerInfo to retrieve additional information on the server.

hardwareID
[in] Globally unique server identifier. Generally the server's MAC address.

credentials
[in] Server credentials. Required to connect to a GO-Global Portal.

address
[in] Network address of the server

port
[in] Port on which the server accepts connections

connOpt
[in] Connection options

role
[in] The server's role

Return Values

SM_TRUE if the connection is accepted, SM_FALSE if it is rejected.

Remarks

If no callback is installed, all connections are accepted automatically.

If SM_FALSE is returned, the server connection is closed and *serverID* becomes invalid.

OnServerDisconnect

The **OnServerDisconnect** callback function notifies the implementer that a server has disconnected. There is no default behavior to override.

```
typedef void (* OnServerDisconnect)
                        (
                          SMI_SERVER_ID serverID
                        );
```

Parameters

serverID
[in] The ID of the server that has disconnected. Use SMI_GetServerInfo to retrieve additional information.

Return Values

This function does not return a value.

3 SESSION MANAGER CALLBACK INITIALIZATION FUNCTIONS

The following list contains the functions needed to initialize the callback functions. Each function in this section returns the previously installed callback (or NULL if no callback is installed). The purpose of each callback is noted in the callback prototype descriptions. Setting the callback function to NULL will unregister the installed callback (if any). These functions may be called at any time, both before and after SMI_Initialize is called.

SMI_SetClientConnectCB

Function **SMI_SetClientConnectCB** sets the client connect callback function that is called when a client connects.

```
OnClientConnect SMI_SetClientConnectCB
          (
            OnClientConnect  callback
          );
```

Parameters

callback
[in] The callback function, or NULL to reset

Return Values

The previously installed callback

SMI_SetClientDisconnectCB

Function **SMI_SetClientDisconnectCB** sets the client disconnect callback function that is called when a client disconnects from the Session Manager.

```
OnClientDisconnect SMI_SetClientDisconnectCB
          (
            OnClientDisconnect  callback
          );
```

Parameters

callback
[in] The callback function, or NULL to reset

Return Values

The previously installed callback

SMI_SetServerConnectCB

Function **SMI_SetServerConnectCB** sets the server connection callback function that is called when a server connects to the Session Manager.

```
OnServerConnect SMI_SetServerConnectCB
          (
            OnServerConnect  callback
          );
```

Parameters

callback
[in] The callback function, or NULL to reset

Return Values

The previously installed callback

SMI_SetServerDisconnectCB

Function **SMI_SetServerDisconnectCB** sets the server disconnect callback function that is called when a server disconnects from the Session Manager.

```
OnServerDisconnect SMI_SetServerDisconnectCB
          (
            OnServerDisconnect  callback
          );
```

Parameters

callback
[in] The callback function, or NULL to reset

Return Values

The previously installed callback

4 GO-GLOBAL AUTHENTICATION (GGA) INTERFACE

This interface allows a service provider to implement custom authentication on a GO-Global Host. To customize the authentication, a dynamic-link library (DLL) must export the entry point below.

GGA_AuthenticateUser

The **GGA_AuthenticateUser** callback function allows service providers to override the default authentication method used by the GO-Global Host. By default the authentication is done by calling the Win32 API LogonUser (). GO-Global still requires a user token in order to load the appropriate user profile and to execute applications in the correct security context.

```
HANDLE GGA_AuthenticateUser
         (
           DWORD  bufferSize,
           void*  buffer
         );
```

Parameters

bufferSize
[in] The size of buffer. If *bufferSize* is 0, then *buffer* will be ignored.

buffer
[in] User-defined data for performing custom authentication. These data, which originate from a call to SMI_CreateSession, typically consist of user information/password. The parameter's size must not exceed *bufferSize* bytes.

Return Values

This interface must return a valid user token (*e.g.,* LogonUser), else –1 on failure.

5 GO-GLOBAL ACTIVEX CLIENT INTERFACE

The ActiveX Client Interface allows the functionality of the GO-Global client to be embedded within other Windows applications. The basic steps to start a client connection include creating the ActiveX window, setting connection properties, initializing the connection, and creating/joining a session.

Perform the following steps to include the ActiveX Client in a project created in the Microsoft Development Environment (MSDEV):

1. Register the control from a DOS command prompt: **regsvr32** ieg4.dll
2. Generate source code for an interface class to the control. The particulars of this step depend on which version of MSDEV you use.

   Version 6.0: From the **Project** entry on the main menu bar, select **Add To Project** and then **Components and Controls…** A dialog box entitled **Components and Controls Gallery** will appear. Open its **Registered ActiveX Controls** folder and double-click on the **IEGCtrl Class**. In the **Confirm Classes** dialog that pops up you will be given an opportunity to name the interface class and the source (.h and .cpp) files to which the class will be written.

   Version 7.0: From the **Project** entry on the main menu bar, select **Add Class…** and then open the **MFC Class From ActiveX Control** icon. A dialog box entitled **Add Class From ActiveX Control Wizard** will appear. Drop down the dialog's list control with the caption **Available ActiveX controls** and select its **IEGCtrl Class** entry. Highlight the class name after it is added to the dialog's **Interfaces** list view and then press the **>** button to enable specification of the class you want generated. Use the dialog's **Class**, **.h file**, and **.cpp file** edit boxes to name your class and its source files whatever you wish. Click the dialog's **Finish** button.

IEGCtrl Class Properties

The entries in this subsection are not intended as function prototypes, but rather as ActiveX property interfaces. The actual prototypes will differ depending on how the targeted application is implemented, *e.g.,* through VB or MFC (and even on which version of MFC is used).

[get/put] _Compression

The **[get/put] _Compression** functions retrieve/change the flag that determines if communication data should be compressed. If this property is not set explicitly, it retains its default setting of FALSE.

```
[get/put] _Compression
          <
            BOOL compression
          >
```

Parameters

compression
TRUE if this property is wanted, FALSE if not

[get/put] _Embedded

The **[get/put] _Embedded** functions retrieve/change the flag that determines if GO-Global applications will run within the ActiveX window (TRUE), or as top-level windows (FALSE). If this property is not set explicitly, it retains its default setting of FALSE.

```
[get/put] _Embedded
          <
            BOOL embedding
          >
```

Parameters

embedding
TRUE if this property is wanted, FALSE if not

[get/put] _HostName

The **[get/put] _HostName** functions retrieve/change the name of a server running the GO-Global Session Manager. If this property is not set explicitly, it retains its default setting of NULL.

```
[get/put] _HostName
          <
            BSTR hostName
          >
```

Parameters

hostName
Server address

[get/put] _HostPort

The **[get/put] _HostPort** functions retrieve/change the port that the client will use to connect to the GO-Global Session Manager. If this property is not set explicitly, it retains its default setting of the standard GO-Global host port.

```
[get/put] _HostPort
          <
            long port
          >
```

Parameters

port
Port number

[get/put] _Launch

The **[get/put] _Launch** functions retrieve/change the name of an application that GO-Global will run at startup in place of the GraphOn Program Window. If this property is not set explicitly, it retains its default setting of NULL, causing the Program Window to be launched when GO-Global starts.

```
[get/put] _Launch
          <
            BSTR applicationName
          >
```

Parameters

applicationName
startup application name

[get/put] _LaunchArgs

The **[get/put] _LaunchArgs** functions retrieve/change the command-line arguments that will be sent to the application that GO-Global will run at startup in place of the GraphOn Program Window. If this property is not set explicitly, it retains its default setting of NULL.

```
[get/put] _LaunchArgs
          <
            BSTR commandLine
          >
```

Parameters

commandLine
command-line arguments to send to the startup application

[get/put] _OverlayWindow

Functions **[get/put] _OverlayWindow** retrieve/change the overlay window used with the ActiveX client. An overlay window is a transparent window that covers the ActiveX client window. When specified, the overlay window will be notified when painting occurs within the ActiveX client window and when the
GO-Global client is disconnected from the application server.

Painting notifications take the form of standard WM_PAINT messages. Disconnection notification is done through a private message of value (WM_USER+1), with a *wParam* value of 1 and a *lParam* value of 0. Future versions of the control may support other notifications by using the private message (WM_USER+1) with different *wParam* and *lParam* values.

```
[get/put] _OverlayWindow
          <
            HWND overlayWindow
          >
```

Parameters

overlayWindow
Either a valid handle to an overlay window or else NULL to indicate that no such window exists

put_Password

Function **put_Password** changes the password of the user who is to be authenticated on the server. The setting of this property and of the user name property will override the default authentication. This property will be ignored if shadowing or custom authentication is performed.

```
put_Password
          <
            BSTR password
          >
```

Parameters

password
User's password

[get/put] _Session

Functions **[get/put] _Session** retrieve/change the session to which the client will connect. If this property is not set explicitly, it retains its default setting of zero to indicate that a new session is to be created.

```
[get/put] _Session
          <
            unsigned long sessionID
          >
```

Parameters

sessionID
Session; this identifier will be passed to OnClientConnect callback for custom processing.

[put] _SessionCredentialsAuthority

Function **[put] _SessionCredentialsAuthority** sets the *authority* member of the SMI_SESSION_CREDENTIALS structure. If this property is not set explicitly, it retains its default setting of NULL.

```
[put] _SessionCredentialsAuthority
        <
          BSTR authorityString
        >
```

Parameters

authorityString
string that defines the session's origin, passed on to the OnClientConnect callback for custom processing.

[put] _SessionCredentialsString

Function **[put] _SessionCredentialsString** sets the *credentials* member of the SMI_SESSION_CREDENTIALS structure. If this property is not set explicitly, it retains its default setting of NULL.

```
[put] _SessionCredentialsString
        <
          BSTR credentialsString
        >
```

Parameters

credentialsString
string that is passed through to an OEM's application, uninspected and unmodified, for whatever purpose(s) the OEM desires

[get/put] _UserName

Functions **[get/put] _UserName** retrieve/change the name of the user who is to be authenticated on the server. The setting of this property and the password property will override the default authentication. This property will be ignored if shadowing or custom authentication is performed.

```
[get/put] _UserName
          <
            BSTR userName
          >
```

Parameters

userName
User's name

IEGCtrl Class Methods

Connect

Function **Connect** connects to the GO-Global Session Manager.

```
long Connect
          (
            VARIANT*  dataObject
          );
```

Parameters

dataObject

[in] This parameter is no longer used.

Return Values

If a connection is established to the GO-Global Session Manager, the return value is nonzero (TRUE). Otherwise, on failure, the return value is zero (FALSE). OEMs are warned that, although the implementation of Connect in GraphOn's ieg.dll will not raise an exception, the Microsoft generated wrapper for the function can. Our *ActiveX Control Demonstration* program illustrates one way to handle such an exception, should it occur.

CreateS

Function **Create** creates the ActiveX Client window.

```
BOOL Create
          (
            LPCTSTR      windowName,
            DWORD        style,
            const RECT&  rect,
            CWnd*        parentWnd,
            UINT         ID
          );
```

Parameters

For information about the parameters of this function (which is generated by Visual Studio), please refer to the Microsoft Developer Network (MSDN) documentation for ActiveX controls.

Return Values

TRUE if successful, FALSE if not

6 SESSION MANAGER INTERFACE STRUCTURES

This section describes the various structures used throughout the API specification.

SMI_APPLICATION_INFO

Structure **SMI_APPLICATION_INFO** is used to describe an application running within a session.

Fields

applicationID
The application identifier

pid
The operating system process identifier for the application

sessionID
The GO-Global numeric identifier of the session in which the application was launched

applicationName
Human-readable name describing the application

pathName
Actual file name of the application

arguments
Actual arguments used to launch application

startTime
Time at which application was launched, in seconds since January 1, 1970

userName
Name of the user account used to launch this application

SMI_APPLICATION_SETTING

Structure **SMI_APPLICATION_SETTING** is used to customize published application settings per user/group.

Fields

isUser
When SM_TRUE, *name* is a user name; otherwise, *name* is a group name.

name
The user/group name used

arguments
The command line arguments used when launching an application

startDirectory
The directory in which the application will run

startState
The initial state of the application (see SMI_STARTSTATE)

permissions
The access permission for the given user/group (see SMI_PERMISSION)

errorValue
Flag indicating any errors associated with the publication of the application (see SMI_APPLICATION_SETTING_ERROR)

SMI_CLIENT_INFO

Structure **SMI_CLIENT_INFO** is used to describe a particular client.

Fields

clientID
The identifier of the client

sessionID
The session to which the client is connected

hostName
The host name of the client machine

ipAddress
The IP address of the client

port
The port number of the client

connectTime
Time at which client connected, in seconds since January 1, 1970

majorVersion
The major version number of the client

minorVersion
The minor version number of the client

revisionVersion
The revision number of the client

buildVersion
The build number of the client

bytesRead
The number of bytes that have been received from the client

bytesWritten
The number of bytes that have been sent to the client

SMI_PUBLISHED_APPLICATION

Structure **SMI_PUBLISHED_APPLICATION** is used to describe an application published on an APS server.

Fields

applicationName
Human-readable name describing the application

pathName
Actual file name of the application

arguments
Default arguments used to launch the application

startDirectory
Default directory in which the application will run

startState
The initial state of the application (see SMI_STARTSTATE)

iconFile
The file containing the icon to use to represent this application

iconIndex
The index of the icon to use within *iconFile*

errorValue
Flag indicating any errors associated with the publication of the application (see SMI_APPLICATION_SETTING_ERROR)

SMI_SERVER_INFO

Structure **SMI_SERVER_INFO** is used to describe an APS server.

Fields

schema
The structure's schema number

serverID
The identifier of the server

hostName
The host name of the server

ipAddress
The IP address of the server

port
The number of the port that the server is listening to. Affiliated with the Transmission Control Protocol (TCP) transport mechanism, this value and that of the sibling portSSL field are mutually exclusive.

maxSessions
The maximum number of sessions allowed on the server for all users. Not to be confused with this structure's limitSessionsPerUserMax field.

sessionTimeout
Number of seconds that sessions will remain running on the server after they have been disconnected. Toggling on the high-order bit of this field causes disconnected sessions to continue running indefinitely.

clientOptions
Flag indicating which client options (SMI_CLIENT_OPTIONS) are enabled on the server.

connectionOptions
Flag indicating which connection options (SMI_CONNOPT) are enabled on the server.

cacheClientPrinters
Obsolete

printerDriverSources
Flag indicating which sources (SMI_PRINTER_DRIVER_SRC) GO-Global should use in trying to obtain printer driver files when the driver for a client printer is unavailable on a server; operative only if client printing is enabled.

printerDriversFolder
Fully qualified path of the Windows folder that contains printer drivers distributed with Windows; operative only if client printing is enabled and bit SMI_PRINTER_DRIVER_SRC_WINDOWS_FOLDER has been turned on in the *printerDriverSources* field of this structure.

printerDriverServer
Obsolete

relayServerName
The host name of the relay server to which the server is connected

sharedAccountNames
Names of accounts that multiple users will share. The delimiter character for individual names is localizable; in US-English it is the semi-colon (;).

broadcastInterval
When a Cluster Manager is running on the network, the APS broadcasts information about running sessions, *etc*. at the interval specified by this field, in seconds.

groupPolicy
When this field is set to SM_TRUE, Group Policy is applied during session startup.

limitSessionsPerUserMax
The maximum number of sessions that an individual user is permitted to run concurrently. Toggling on the high-order bit of this field causes the APS to ignore the settings of the remaining bits, allowing users to run an unlimited number of sessions concurrently. Not to be confused with this structure's maxSessions field.

limitMemoryPhysicalMin
The minimum amount of physical memory, measured in megabytes, which must be freely available on the server before a session is allowed to start.

limitMemoryVirtualMin
The minimum amount of virtual memory, measured in megabytes, which must be freely available on the server before a session is allowed to start.

limitSystemPageTablesMin
The minimum number of system page table entries that must be freely available on the server before a session is allowed to start.

logFolder
Fully qualified path of the Windows folder to which log files will be written and in which there are subfolders where backed-up logs will be stored and other subfolders from which templates and code pages can be read.

logMaintenanceAction
For inactive log files that are at least as many days old as the value of this structure's logMaintenanceCountdown field, this variable is the numeric index of the cleanup action to be taken (with zero signifying *Back Up* and one signifying *Delete*).

logMaintenanceCountdown
The number of days after which inactive log files undergo the cleanup action specified for them in this structure's logMaintenanceAction field.

logonScriptGlobal
The fully qualified path of an executable file to be run at session startup for all users who log on to the server; operative only if bit 0x0002 has been turned on in the logonScriptOptions field of this structure.

logonScriptOptions
Flag indicating which logon scripting options are enabled on the server.

The API currently lacks an enumeration to specify the individual bits of this field. The possibilities are as follows (in hexadecimal notation):

0x0001: Enable execution of user-specific logon script;
0x0002: Enable execution of global logon script;
0x0004: Apply non-null window station desktop name in call to CreateProcessAsUser () for executing user-specific logon script; and
0x0008: Apply non-null window station desktop name in call to CreateProcessAsUser () for executing global logon script.

logOutputLevel
A numeric index that designates the level of information written to the log file, with numbers one through six capturing ever-greater detail, and with zero prohibiting such capture.

progressMessageOptions
Flag indicating which progress message options are enabled on the server for a user's session startup.

The API currently lacks an enumeration to specify the individual bits of this field. The possibilities are as follows (in hexadecimal notation):

0x00001: Reserved; must be toggled on;
0x00080: Toggle on to cause progress messages to be shown at session startup, off to prevent such display;
0x10000: Toggle on to specify that the dialog box in which progress messages are shown should be placed above all nontopmost windows that GO-Global creates on the client and should stay above them even when the dialog box is deactivated. Toggle off to enable a user to bring a window other than the dialog box to the front of the Z-order on the client, *e.g.*, a custom dialog with editable fields created by a logon script.

All other bits in this mask are reserved and should be turned off. The GO-Global factory setting for this field is 0x10081.

clientDrivesHide

A mask that designates the alphabetic specifiers of client drives to hide, ranging from the low-order 0x00000001 for the letter *A* to the high-order 0x02000000 for *Z*. Bits in this mask may be combined. Thus, as just two examples, 0x02000001 denotes *A* and *Z*, while 0x03FFFFFF denotes *A* through *Z*.

This field is operative only if the client drives option has been enabled by turning on bit SMI_CLIENT_OPTIONS_DRIVES in this structure's clientOptions field.

Structure member clientDrivesHide should not to be confused with its serverDrivesHide sibling.

clientDrivesIncrementedBy

A numeric value by which to increment alphabetic specifiers when remapping client drives. Permissible settings range from zero to 25.

This field and its sibling clientDriveStartAt are mutually exclusive, *i.e.*, one or the other (but not both) of these SM_USHORT variables must be designated non-operational. Specify the unwanted field by turning on its high-order (0x8000) bit. This rule applies to clientDrivesIncrementedBy even when its fundamental value is zero.

This field is operative only if its high-order bit is unset and the client drives option has been enabled by turning on bit SMI_CLIENT_OPTIONS_DRIVES in this structure's clientOptions field.

clientDriveStartAt

A numeric index that corresponds to the alphabetic specifier at which to begin the sequential remapping of client drives. A setting of one denotes the letter *A*, two *B*, and so on, up through 26 for *Z*.

This field and its sibling clientDrivesIncrementedBy are mutually exclusive, *i.e.*, one or the other (but not both) of these SM_USHORT variables must be designated non-operational. Specify the unwanted field by turning on its high-order (0x8000) bit. This field is operative only if its high-order bit is unset and the client drives option has been enabled by turning on bit SMI_CLIENT_OPTIONS_DRIVES in this structure's clientOptions field.

connectionSSL_Certificate
The fully qualified path of a Secure Sockets Layer (SSL) certificate file. A non-blank setting must be assigned to this variable if sibling field connectionOptions is configured for the SSL transport mechanism.

limitPagedPoolBytesMin
The minimum number of bytes of paged pool memory that must be freely available on the server before a session is allowed to start. Assigning a value from 1 (one) to 176,160,768 turns the feature on; assigning the negative of a value in that range turns the feature off (the GO-Global default).

portSSL
The number of the port that the server is listening to. Affiliated with the Secure Sockets Layer (SSL) transport mechanism, this variable and that of its port sibling are mutually exclusive.

serverDrivesHide
A mask that designates the alphabetic specifiers of server drives to hide, ranging from the low-order 0x00000001 for the letter *A* to the high-order 0x02000000 for *Z*. Bits in this mask may be combined. Thus, as just two examples, 0x02000001 denotes *A* and *Z*, while 0x03FFFFFF denotes *A* through *Z*.

Structure member serverDrivesHide should not to be confused with its clientDrivesHide sibling.

serverRole
The role of the server.

sessionShutdownGracePeriod
The number of minutes that applications launched through a GO-Global session will have *after* the server begins shutting the session down to close themselves in an orderly manner or to interact with users to do so, *e.g*., by saving open files. Not to be confused with this structure's sessionTimeoutWarningPeriod field.

sessionTimeoutIdle
The number of minutes a session may be idle, *i.e*., how long it may go without a keyboard or mouse input event, before it becomes subject to the action specified by this structure's sessionTimeoutOptions field.

sessionTimeoutLoggedOn
The number of minutes a session may run on a server before it is automatically logged off.

sessionTimeoutOptions
A numeric index that designates what the server is to do automatically to a session whose idle time limit is reached, with a setting of one (0x00000001 hexadecimal) denoting *Disconnect* and two (0x00000002) denoting *Log off*.

sessionTimeoutWarningPeriod
The number of minutes *before* a session time limit or idle time limit is reached when users are warned that they are about to be disconnected or logged off automatically. Not to be confused with this structure's sessionShutdownGracePeriod field.

authenticationOptions
Flag indicating which authentication options (SMI_AUTH_OPTS) are enabled on the server.

logFileByteCountLimit
The maximum size of server log files.

useClientTimezone
When this field is set to SM_TRUE, the server will use the time zone of the client device.

m_OS_VersionMajor
The major version number of the Windows operating system. For Windows NT version 4.0, for example, the major version number is 4, and for Windows 2000 and 2003 it is 5. Set by the APS at startup.

m_OS_VersionMinor
The minor version number of the Windows operating system. For Windows 2000, for example, the minor version number is 0, and for Windows 2003 it is 2. Set by the APS at startup.

m_OS_VersionBuild
The build number of the Windows operating system. Set by the APS at startup.

m_OS_VersionSvcPackMajor
The major version number of the latest Service Pack installed on the system. For Windows 2000 Service Pack 3, for example, the major version number is three. The value is zero if no Service Pack has been installed or if the operating system is Windows NT. Set by the APS at startup.

m_OS_VersionSvcPackMinor
The minor version number of the latest Service Pack installed on the system. For Windows 2000 Service Pack 3, for example, the minor version number is zero. The value is also zero if no Service Pack has been installed or if the operating system is Windows NT. Set by the APS at startup.

m_mayUseStrongEncryption
The GO-Global Host sets this field to SM_TRUE if it detects licensing for strong forms of encryption for the Secure Sockets Layer (SSL) transport mechanism.

The setting of this variable is relevant only when its connectionOptions sibling is configured for SSL transport.

SMI_SERVER_METRICS

Structure **SMI_SERVER_METRICS** contains server metrics.

Fields

flags
This flag indicates which members of the structure should contain valid information upon completion of the call to SMI_GetServerMetrics (see SMI_METRICS).

cpuUsage
Current CPU usage represented as a percentage

memoryUsage
Current memory usage represented as a percentage

sessionCount
Number of current sessions on the server

processCount
Number of current processes running on the server

serverUptime
The time the APS server has been running (in seconds)

machineUptime
The time that the host machine has been running (in seconds)

totalProcessCount
Total number of processes that have been started on the APS server during its lifetime

totalSessionCount
Total number of sessions that have been started on the APS server during its lifetime

SMI_SESSION_INFO

Structure **SMI_SESSION_INFO** is used to describe a particular session on a given APS server.

Fields

serverID
The identifier of the server on which the session is running

sessionID
The identifier of the session being described

sessionUUID
The session UUID specified in SMI_CreateSession

options
Options used to create the session

state
The current state of the session (see SMI_SESSIONSTATE)

serverID
The server on which the session is running

name
The human-readable name of the given session

userName
The user account used to create the session

clientCount
Number of clients connected to the session

applicationCount
Number of applications running in the session

connectionCount
Number of network connections

startTime
Time at which the session was created (in seconds since January 1, 1970)

inputID
The client ID of the user who has input control (mouse/keyboard)

creationInfo
Characteristics used to create the session (see SMI_CreateSession or SMI_SESSION_CREATION_INFO)

SMI_SESSION_CREATION_INFO

Structure **SMI_SESSION_CREATION_INFO** is used to define the characteristics of a new session. Used with SMI_CreateSession.

Fields

sessionUUID
An optional identifier for the session that is specified by the calling process. This identifier is not used by the GO-Global Host.

resolutionHorizontal
The width of the session's display. In loose windows mode, this should be largest display width used by clients. In embedded mode, this should be the width of the embedded GO-Global window.

resolutionVertical
The height of the session's display. See above for details.

workAreaLeft
The position of the work area's left edge (see below)

workAreaTop
The position of the work area's top edge (see below)

workAreaRight
The position of the work area's right edge (see below)

workAreaBottom
The position of the work area's bottom edge (see below)

flags
Options (SMI_SESSIONCREATION_FLAGS) defining how the session will be used.

The "work area" can be used to define a subset of the client display used for drawing. For example, it can be used to exclude the Task Bar from a client's full resolution in loose windows mode. In embedded mode, this structure will typically be filled as follows:

```
SMI_SESSION_CREATION_INFO info;
info.resolutionHorizontal = x;
info.resolutionVertical   = y;
info.workAreaLeft         = 0;
info.workAreaTop          = 0;
info.workAreaRight        = info.resolutionHorizontal;
info.workAreaBottom       = info.resolutionVertical;
info.flags                = SMI_SCF_LOOSEWINDOWS;
```

SMI_SESSION_CREDENTIALS

Structure **SMI_SESSION_CREDENTIALS** is used to authenticate a client's connection to a session. Used with the OnClientConnect callback.

Fields

authority

String specifying the session's origin. While GraphOn has not set rules on the text used for an authority name, we recommend that it consist of at least the long version of the OEM's corporate name. It may be further qualified with suite and product names as in the following example:

"GraphOn Corporation – GO-Global – Cluster Manager\0"

An OEM must implement its OnClientConnect callback to ignore credentials submitted by any authority besides itself, *i.e.,* **the OEM's implementation of the callback must accept the connection when given some other company's credentials**.

credentials

A string passed through to an OEM's application, uninspected and unmodified, for whatever purpose(s) the OEM desires

7 SESSION MANAGER INTERFACE CONSTANTS

This section describes the various manifest constants available for use in this API.

SMI_APPLICATION_SETTING_ERROR

The enumerated type **SMI_APPLICATION_SETTING_ERROR** contains the constants used to describe any errors associated with a published application or its users. It is used in the structure field SMI_APPLICATION_SETTING.errorValue and SMI_PUBLISHED_APPLICATION.errorValue. Each constant is a flag, which can be combined with other flags with the bitwise-OR operator (|).

Values

SMI_APPLICATION_SETTING_ERROR_NONE
Stands for "no errors." Defined as 0, OR-ing it to other values has no effect. May be used instead of an unadorned zero.

SMI_APPLICATION_SETTING_ERROR_APPLICATION_NAME
The display name is either invalid or in use.

SMI_APPLICATION_SETTING_ERROR_CONNOPT
Invalid connection option (see SMI_CONNOPT)

SMI_APPLICATION_SETTING_ERROR_PATH_NAME
The specified path to the application is incorrect.

SMI_APPLICATION_SETTING_ERROR_START_DIRECTORY
The specified start directory does not exist.

SMI_APPLICATION_SETTING_ERROR_START_STATE
Invalid start state (see SMI_STARTSTATE)

SMI_APPLICATION_SETTING_ERROR_32_BIT_APPLICATION
The application is not a 32-bit application.

SMI_APPLICATION_SETTING_ERROR_RELAY_SERVER
This bit-field is a modifier for others in this series. When any of them (other than the "no errors" entry) has been set, the value associated with this bit will indicate whether the problem occurred on a relay server or not. When on, this bit denotes a relay server; when off, either an independent-application or relayed server.

SMI_APPLICATION_SETTING_ERROR_UNKNOWN
A nondescript error

SMI_CLIENT_OPTIONS

The enumerated type **SMI_CLIENT_OPTIONS** contains the constants used to determine which client options are enabled on a server. It is used in the structure field SMI_SERVER_INFO.clientOptions. Each constant is a flag, which can be combined with other flags with the bitwise-OR operator (|).

Values

SMI_CLIENT_OPTIONS_NONE
Stands for "no options." Defined as 0, OR-ing it to other values has no effect. May be used instead of an unadorned zero.

SMI_CLIENT_OPTIONS_CLIPBOARD
The client clipboard component is enabled on the server.

SMI_CLIENT_OPTIONS_DRIVES
The client drives component is enabled on the server.

SMI_CLIENT_OPTIONS_PRINTING
The client printing component is enabled on the server.

SMI_CLIENT_OPTIONS_SOUND
The client sound component is enabled on the server.

SMI_CLIENT_OPTIONS_TAPI

SMI_CONNOPT

The enumerated type **SMI_CONNOPT** contains constants used by function SMI_SetConnectionOptions and in the SMI_SERVER_INFO.connectionOptions structure field. Each constant corresponds to one of three relationships that exist between clients and servers: *1)* encryption of transferred data, *2)* a transport mechanism, or *3)* user notification when connections are secure. A single primitive (one-bit) member of each group may be combined with a single primitive member of the other groups with the bitwise-OR operator (|). It is illegal to combine constants within a group.

The data encryption and user notification settings are true options, *i.e.*, all of their representational bits may be set to the off state simultaneously. Specification of a transport mechanism is, however, a requirement.

Values

SMI_CONNOPT_NONE

Stands for “no options.” Defined as 0, OR-ing it to other values has no effect. May be used instead of an unadorned zero as a notational convenience for variable initialization, but due to the need for a nonzero transport mechanism specifier, it is not a legitimate value for passing to function SMI_SetConnectionOptions.

SMI_CONNOPT_ENCRYPT_ group

Members of this group specify an encryption scheme for data transferred between clients and servers. They may not be combined with one another, but they may all be turned off if encryption is unwanted. Member suffixes and descriptions are as follows, with those above 56-bit DES considered “strong”:

056_BIT_DES

Use the Data Encryption Standard (DES) algorithm to transform 64-bit data blocks under a 56-bit secret key.

168_BIT_3DES

Use a variant of the DES algorithm that employs up to three 56-bit keys and makes three encryption/decryption passes over each data block.

168_BIT_RC4

Use the RC4 stream cipher with 168-bit encryption.

256_BIT_AES

Use the Advanced Encryption Standard (AES) algorithm to transform 128-bit data blocks under a 256-bit key.

_ALL

This constant is a sub-mask derived by turning on all the preceding one-bit constants in an SM_ULONG quantity. Apply it with a bitwise-AND (&) to SMI_SERVER_INFO.connectionOptions to determine which type of encryption (if any) is in effect.

SMI_CONNOPT_XPORT_MECHANISM_ group

Members of this group specify a transport mechanism for data transferred between clients and servers. They may not be combined with one another, and one must be turned on in the SM_ULONG argument passed to SMI_SetConnectionOptions. Member suffixes and descriptions are as follows:

SSL

Use the Secure Sockets Layer (SSL) protocol as the transport mechanism.

TCP

Use the Transmission Control Protocol (TCP) as the transport mechanism.

_ALL

This constant is a sub-mask derived by turning on all the preceding one-bit constants in an SM_ULONG quantity. Apply it with a bitwise-AND (&) to SMI_SERVER_INFO.connectionOptions to determine which type of transport mechanism is in effect.

SMI_CONNOPT_NOTIFY_WHEN_SECURE

Enables server to notify user when connections are secure.

SMI_METRICS

The enumerated type **SMI_METRICS** contains the constants used to describe the information available in the SMI_SERVER_METRICS structure. It is used in the structure field SMI_SERVER_METRICS.flags. Each constant is a flag, which can be combined with other flags with the bitwise-OR operator (|).

Values

SMI_METRICS_NONE
Stands for “no fields”. Defined as 0, OR-ing it to other values has no effect. May be used instead of an unadorned zero.

SMI_METRICS_CPU
Refers to the field SMI_SERVER_METRICS.cpuUsage

SMI_METRICS_MEMORY
Refers to the field SMI_SERVER_METRICS.memoryUsage

SMI_METRICS_SESSIONS
Refers to the field SMI_SERVER_METRICS.sessionCount

SMI_METRICS_PROCESSES
Refers to the field SMI_SERVER_METRICS.processCount

SMI_METRICS_UPTIME_SERVER
Refers to the field SMI_SERVER_METRICS.serverUptime

SMI_METRICS_UPTIME_MACHINE
Refers to the field SMI_SERVER_METRICS.machineUptime

SMI_METRICS_TOTAL_SESSIONS
Refers to the field SMI_SERVER_METRICS.totalSessionCount

SMI_METRICS_TOTAL_PROCESSES
Refers to the field SMI_SERVER_METRICS.totalProcessCount

SMI_METRICS_ALL
Stands for “all fields”, represents the combination of all other flags

SMI_AUTH_OPT

The enumerated type **SMI_AUTH_OPT** contains mutually exclusive constants that control the enablement state of authentication options.

Values

SMI_AUTH_OPTS_STD
Users may be prompted for a user name and password.

SMI_AUTH_OPTS_CACHE_PASWDS
Passwords may be encrypted and stored on the client device.

SMI_AUTH_OPTS_IW
Integrated Windows Authentication may be used to authenticate users who connect from computers that are members of the same domain as the server.

SMI_AUTH_OPTS_IW_CATCH_PASWDS
Users' passwords may be encrypted and stored in their user profiles.

SMI_PERMISSION

The enumerated type **SMI_PERMISSION** contains constants that describe access rights of user and group accounts on the server. It is used in the structure field SMI_APLICATION_SETTING.permissions.

Values

SMI_PERMISSION_FULL
Account has unrestricted access rights.

SMI_PERMISSION_READ
Account has read access only.

SMI_PERMISSION_MODIFY
Account has read, write, and execute access.

SMI_PERMISSION_SPECIAL
Account has special access rights.

SMI_PERMISSION_DENY
Account has no access rights.

SMI_PERMISSION_NONE
Access rights are unset, *i.e.*, neither allowed nor denied.

SMI_SESSIONCREATION_FLAGS

The enumerated type **SMI_SESSIONCREATION_FLAGS** contains the constants used to describe the options used to create a new session. It is used in the structure field SMI_SESSION_CREATION_INFO.flags.

Values

SMI_SCF_DEFAULT
Defined as 0, this means "no special options"

SMI_SCF_LOOSE_WINDOWS
GO-Global clients running in "loose windows" mode will use this session. If this flag is not set, the session will expect clients to run in embedded mode.

SMI_PRINTER_DRIVER_SRC

The enumerated type **SMI_PRINTER_DRIVER_SRC** contains the constants used to describe the sources that GO-Global may optionally use in trying to obtain printer driver files when the driver for a client printer is unavailable on a server. It is used in the structure field SMI_SERVER_INFO.printerDriverSources. Each constant is a flag, which can be combined with other flags with the bitwise-OR operator (|).

Values

SMI_PRINTER_DRIVER_SRC__NONE
Stands for "no fields". Defined as 0, OR-ing it to other values has no effect. May be used instead of an unadorned zero.

SMI_PRINTER_DRIVER_SRC_WINDOWS_FOLDER
Allows specification of the Windows folder containing printer drivers that are distributed with Windows (see SMI_SERVER_INFO.printerDriversFolder)

SMI_SESSIONSTATE

The enumerated type **SMI_SESSIONSTATE** contains the constants used to describe the state of a session. It is used in the structure field SMI_SESSION_INFO.state.

Values

SMI_SESSIONSTATE_UNKNOWN
Session's state is unknown (probably because session ID was unknown).

SMI_SESSIONSTATE_RUNNING
Session is currently active.

SMI_SESSIONSTATE_SUSPENDED
Session exists but is currently suspended (through SMI_SuspendSession).

SMI_STARTSTATE

The enumerated type **SMI_STARTSTATE** contains the constants used to describe the startup state of published applications. It is used in structure fields SMI_APLICATION_SETTING.startState and SMI_PUBLISHED_APPLICATION.startState.

Values

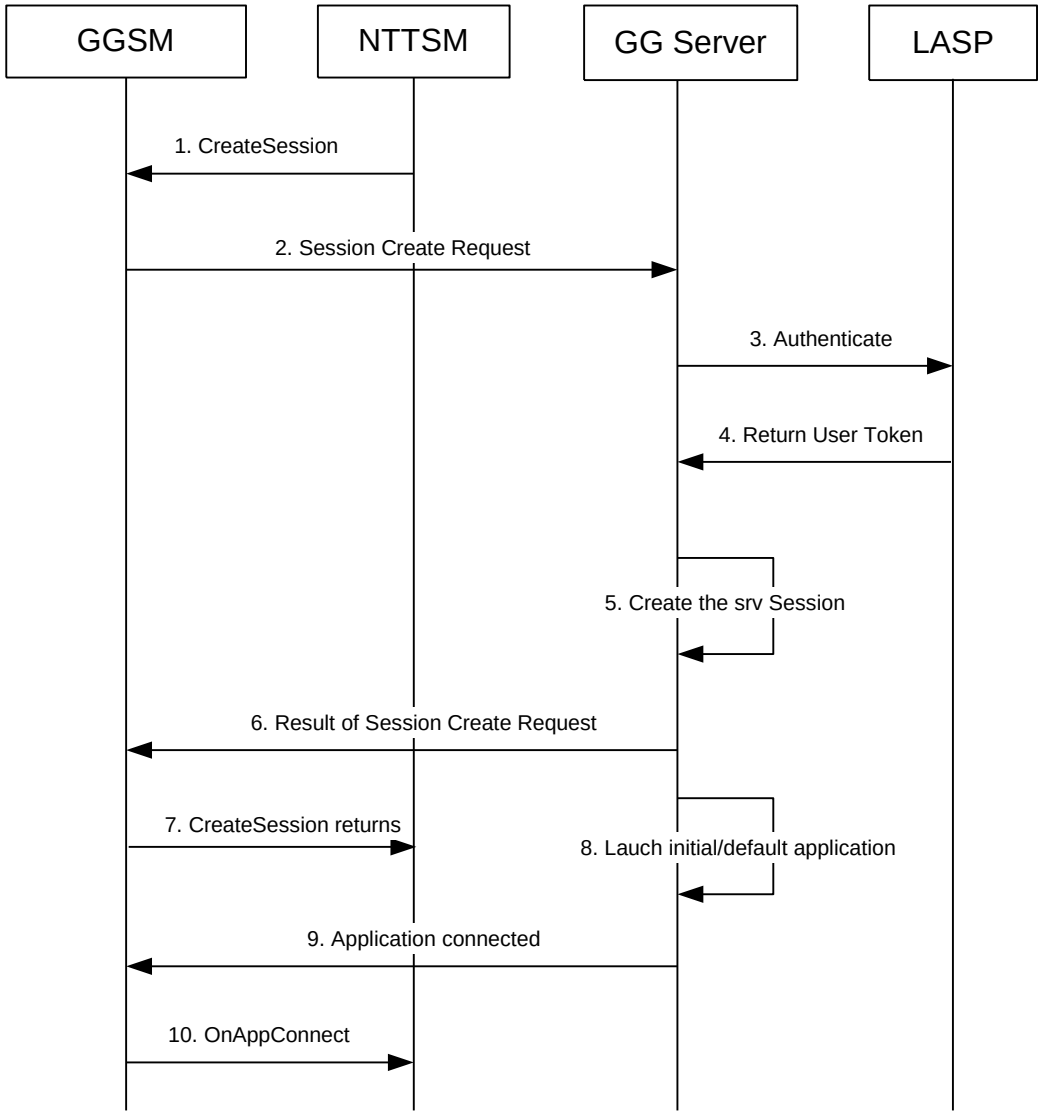
SMI_STARTSTATE_NORMAL
Application will start in a normal window.

SMI_ STARTSTATE_MINIMIZED
Application will start minimized.

SMI_ STARTSTATE_MAXIMIZED
Application will start maximized.

EVENT SEQUENCE DIAGRAMS

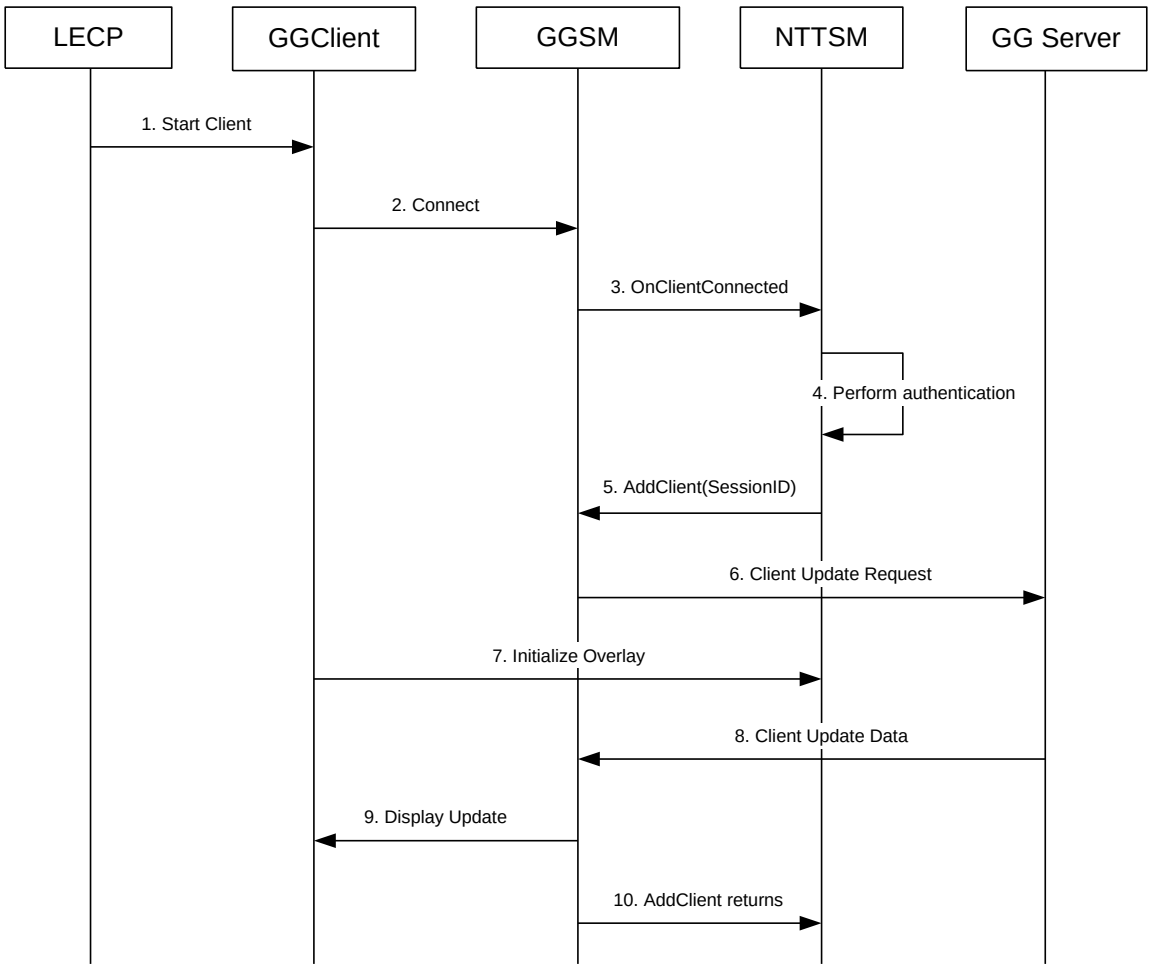
Create Session Sequence



1. The NTT Session Manager calls the CreateSession API, specifying the server, initial application and user authentication data.
2. Creation request is forwarded to the specified GO-Global Host.
3. The GO-Global Host invokes a callback with the user authentication data to allow custom authentication. In this case the callback is implemented by the LASP module.
4. The callback in the LASP performs the authentication and returns a Win32 user token.
5. With the user token, the GO-Global Host will then create the session and load the user's profile.
6. The result of the session creation operation is returned to the GO-Global Session Manager.

7. The GO-Global Session Manager returns the result of the CreateSession call (that was initiated in step 1) to the NTT Session Manager.
8. The initial/default application will now be launched in the security context of the user. NOTE: This step can occur in parallel with steps 6-7.
9. The GO-Global Host establishes an application connection to the GO-Global Session Manager.
10. The GO-Global Session Manager invokes the OnApplicationConnect callback.

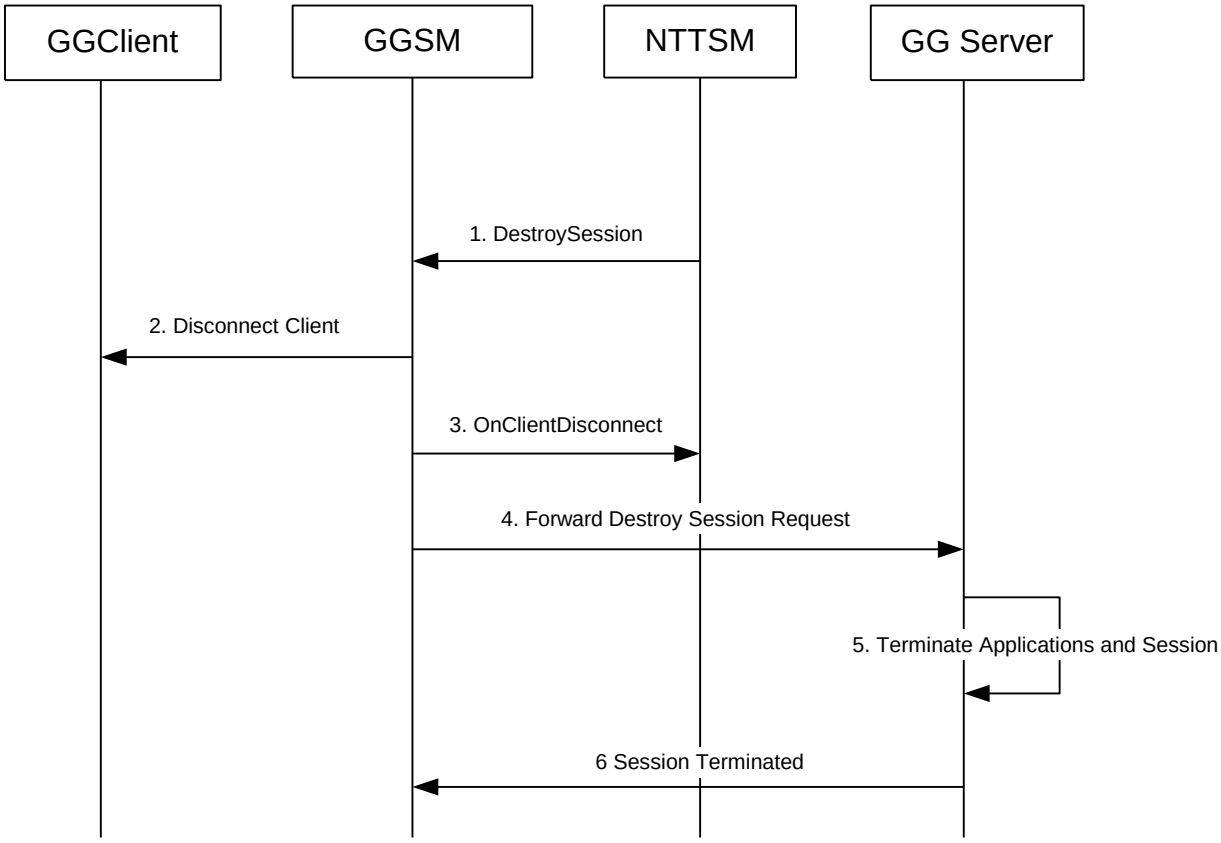
Connect Client Sequence



1. The LECP calls Connect in the ActiveX Control Client.
2. The GO-Global Client connects to the GO-Global Session Manager with the custom user credentials.
3. The GO-Global Session Manager calls the OnClientConnect callback in the NTT Session Manager.
4. The NTT Session Manager performs the user authentication using the supplied credentials.
5. If authentication is successful, the NTT Session Manager calls the AddClient API. This function adds the specified client to an existing session.

6. AddClient will request a client update to the GO-Global Host.
7. Once the connection is established the Overlay component will start the initialization process. This step will be done in parallel with the client display update.
8. The GO-Global Host sends the updated display data to the GO-Session Manager.
9. The GO-Global Session Manager forwards this data to the clients.
10. AddClient returns, specifying whether the operation was successful.

7 DESTROY SESSION SEQUENCE



1. NTT Session Manager invokes the DestroySession API.
2. The GO-Global Session Manager disconnects all clients.
3. The GO-Global Session Manager calls the OnClientDisconnect callback function in the NTT Session Manager for every client.
4. Once the callbacks return, the GO-Global Session Manager forwards a destroy session request to the GO-Global Host.
5. The GO-Global Host terminates all applications associated with the session. Once all applications are terminated, the session will be destroyed.
6. The GO-Global Host reports that the session is terminated.