TWOx12, Nx12 & 2101 COM API v2.5

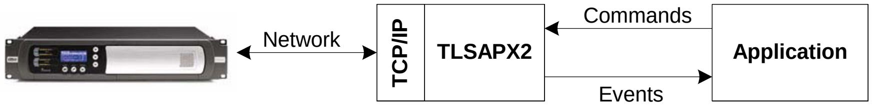
Author: *Ioan L. Rus*
Date: *2009.November.12*

Overview

The TLSAPX2 component is a COM/ActiveX control which provides a high level API for controlling and getting information from Telos telephone hybrid products.

Since component is usable from many programming languages, the specific steps involved in using this component will depend on your choice of language. Separate documents describe access methods for Microsoft Visual C++ and Borland C++ Builder. If you need assistance in using the component with another programming language please contact the author by email at ioan@telos-systems.com.

The application sends commands to the component by calling functions defined by the component. The component returns results back to the application through a success/error HRESULT[i] return value. The component will also send events back to the application as results are received from the device.



COM events are similar to callback functions; the application defines the functions that will be invoked when a particular event takes place. Please see the examples for more details on this topic.

What's New in v2.5

Version 2.5 implements the following changes:

1. A new async mode allows the client to have more control over the connection process. Previous versions would block until a connection to

---

[i] C++ returns a result code but other languages translate the error return codes to language exceptions.

the server was established. When using the async mode, the call to Connect() returns right away. Success or failure is then communicated through SendError() events from TLSAPX to the application. The application may cancel a Connect() request at any time by issuing a Disconnect(). It may also issue a new Connect(), perhaps to a different address, or with different credentials. Please see the "Connecting in Asynchronous Mode" section on page 7 for more details.

2. Added two new methods: PostAcousticAdapt() and PostMuteRinger(). You can find these in these on page 26 and page 34, respectively.

3. Added new message codes:

   ERROR_SOCKET_CONNECT_SUCCESS (98)
   ERROR_SOCKET_CONNECT_FAILED (98001)
   ERROR_SOCKET_READ_ERROR (98002)
   ERROR_SOCKET_WRITE_ERROR (98003)
   ERROR_UNABLE_TO_PARSE_MESSAGE (98004)
   ERROR_DATA_PARSE_ERROR (98005)

   Please note that since these messages do not relate to a session, a session number of 0 will be reported. These messages are sent to the application when socket events are detected.

4. The control will now display internal errors and unexpected conditions through a MessageBox message. This will bring the error condition to the user's attention rather than attempting to continue with the application exhibiting unexpected behavior.

5. In addition to logging messages to a log file, this version can also send log messages to a debugger window. Please see the "Debug Output" section on page 8 for more information.

Installing the Control

The component is contained in a single DLL file named TLSAPX2.DLL. You need to include this DLL in your product installer. You may install this file either in your application directory or in the Windows system directory. After installation you must register the component. Most install packages have built-in facilities to register components. You may also register the control manually using the following command:

```
c:\> RegSvr32 <path>\TLSAPX2.DLL
```

Operation Sequence

When using the control you will generally follow this sequence:

1. Call *OpenControl* once to initialize the control. You must call *CloseControl* when you are done using the control.
2. Call *Connect* to establish a connection to a compatible Telos telephony device. You must call *Disconnect* to close the connection when done.
3. Call *EnumerateShows* to get a list of shows in the system. You must create an event handler for *EnumerateShowsCallback* in order to receive the results. The *EnumerateShowsCallback* will be fired once for each show in the system (whether active or not).
4. Call *ConnectToShow* to establish a connection to an active show. If a password is required you must provide it when making the call.
5. Once connected to a show you may call *ChangeShow* to change the show running on the host to another show. Call *EnumerateShowsForHost* to determine which shows are allowed to run on this host. If the new show requires a password you must provided when calling *ChangeShow*.
6. Once we connected to a show you can call *EnumerateDirectors* to get information about the desktop directors attached to this show.
7. At this point you can either attach to an existing (and free) desktop director by calling *AttachToDirector* or you can attach stand-alone in talent mode by calling *AttachAsTalent*.
8. At this point you may call any of the *Post…* functions listed below. You can also receive events from the control (as long as you provide handlers for them).

Commands

Once connected to a show you may issue any of the commands listed below.

```
PostBusyAllButton(int nSession);
PostDelayDumpButton(int nSession, bool bDown);
PostDialKey(int nSession, int chDialKey);
PostDropButton(int nSession, int nColumn);
PostHoldButton(int nSession, int nColumn);
PostLineButton(int nSession, int nLine, int nColumn);
PostMuteButton(int nSession);
PostNextButton(int nSession);
PostRecordButton(int nSession);
PostSpeakerButton(int nSession);
PostText(int nSession, int nTextType,int nLine,const char *pszText);
PostTransferButton(int nSession);
PostUpdateAll(int nSession);
PostMuteRinger(int nSession,int bOn);
PostAcousticAdapt(int nSession,int bOn);
```

Events

The following events are used by the control to send information back to the application:

```
EnumerateShowsCallback(BSTR strShowName,
                       BOOL bIsActive,
                       BSTR strHostName,
                       BOOL bIsLast);
EnumerateHostsCallback(BSTR strHostName,BOOL bIsLast);
EnumerateDirectorsCallback(int nSession,
                           BSTR strDirectorName,
                           BOOL bIsTalent,
                           BOOL bIsFree,
                           BOOL bIsLast);
SendShowInfo(int nSession,int nMaxLines,int nHybrids);
SendMode(int nSession, BOOL bTalent);
SendShowChange(int nSession,BSTR strNewShowName,BSTR strOldShowName);
SendRecord(int nSession,BOOL bOn);
SendBusyAll(int nSession,BOOL bOn);
SendTransfer(int nSession,BOOL bOn);
SendMute(int nSession,BOOL bOn);
SendRingMute(int nSession,BOOL bOn);
SendDelayDump(int nSession,BOOL bOn);
SendHandset(int nSession,BOOL bOn);
SendSpeaker(int nSession,BOOL bOn);
SendLineState(int nSession,
              int nLine,
              int nColumn,
              int nLineState,
              long lElapsedTimeSec);
SendSeizeLine(int nSession,int nLine,int nColumn);
SendRingLine(int nSession,BSTR strCallerId,int nLine);
SendDropLine(int nSession,BSTR strCallerId,int nLine);
SendAnswerLine(int nSession,BSTR strCallerId,int nLine,int nColumn);
SendSelectLine(int nSession,BSTR strCallerId,int nLine,int nColumn);
SendDialKey(int nSession,int chKey);
SendError(int nSession,int nErrorCode,BSTR strErrorText);
SendText(int nSession,int nTextType,BSTR strText,int nLine);
```

AP Line States

The nLineState parameter in SendLineState can have one of the following values:

```
LINE_STATE_INVALID                 =0x0000,
LINE_STATE_IDLE                    =0x0100,
LINE_STATE_RESERVED                =0x0200,
LINE_STATE_RINGING                 =0x0301,
LINE_STATE_HOLD                    =0x0402,
LINE_STATE_READY_HOLD              =0x0504,
LINE_STATE_READY_NEXT              =0x0604,
LINE_STATE_TALENT_HOLD             =0x0702,
LINE_STATE_ON_AIR                  =0x0808,
LINE_STATE_ON_AIR_LOCKED           =0x0908,
LINE_STATE_HANDSET                 =0x0A10,
LINE_STATE_HANDSET_LOCKED          =0x0B10,
LINE_STATE_SPEAKER                 =0x0C10,
LINE_STATE_SPEAKER_LOCKED          =0x0D10,
LINE_STATE_USED_ELSEWHERE          =0x0E20,
LINE_STATE_BUSY_ALL                =0x0F20,
LINE_STATE_SHARED_UNAVAILABLE      =0x1020
```

SendError Event Codes

The following codes may be sent by the SendError event. Please note that the event name is not entirely consistent (for historical reasons); the event also sends success messages, not only error ones. Codes less than 1000 indicate success of an operation while codes 1000 and above indicate an error condition.

Code	**Message**
1	Connect: Success!
2	EnumerateShows: Success!
3	EnumerateHostsForShow: Success!
4	ChangeShow: Success!
5	ConnectToShow: Success!
6	EnumerateDirectors: Success!
7	AttachToDirector: Success!
8	AttachAsTalent: Success!
9	PostBusyAllButton: Success!
10	PostDelayDumpButton: Success!
11	PostDialKey: Success!
12	PostDropButton: Success!
13	PostHoldButton: Success!
14	PostLineButton: Success!
15	PostMuteButton: Success!
16	PostNextButton: Success!
17	PostRecordButton: Success!
18	PostSpeakerButton: Success!
19	PostText: Success!
20	PostTransferButton: Success!
21	PostUpdateAll: Success!
22	EnumerateShowsForHost: Success!
98	Socket connected successfully.
1001	Connect: No storage manager found!
1002	Connect: Unable to read user info from storage manager!
1003	Connect: Invalid password!
2001	EnumerateShows: No storage manager found!
3001	EnumerateHostsForShow: No storage manager found!
4001	ChangeShow: Not connected to a show!
4002	ChangeShow: Failed!
5001	ConnectToShow: Unable to find active show!
5002	ConnectToShow: Invalid show password!
6001	EnumerateDirectors: Not connected to a show!
7001	AttachToDirector: Not connected to a show!
7002	AttachToDirector: Attach failed!
8001	AttachAsTalent: Not connected to a show!
8002	AttachAsTalent: Attach failed!
9001	PostBusyAllButton: Not attached to show!
10001	PostDelayDumpButton: Not attached to show!
11001	PostDialKey: Not attached to show!
12001	PostDropButton: Not attached to show!
13001	PostHoldButton: Not attached to show!
14001	PostLineButton: Not attached to show!
15001	PostMuteButton: Not attached to show!
16001	PostNextButton: Not attached to show!

17001	PostRecordButton: Not attached to show!
18001	PostSpeakerButton: Not attached to show!
19001	PostText: Not attached to show!
20001	PostTransferButton: Not attached to show!
21001	PostUpdateAll: Not attached to show!
22001	EnumerateShowsForHost: No storage manager found!
98001	Socket connect failed.
98002	Socket read error.
98003	Socket write error.
98004	Data stream not understood.
98005	Message parse error.
99001	Bad session id!

Connecting in Asynchronous Mode

Normally, the AP component will block on connect until either a connection is established or the attempt fails. Due to the TCP/IP connect timeout, it may take up to 45 seconds or longer for the connection attempt to fail if the remote side does not respond. The asynchronous mode avoids this issue by allowing the client application to issue the Connect() command then regain control right away. The component will notify the application of success or failure through the SendError() event. While waiting for the success or failure message, the application has the opportunity to cancel the connect request by sending a Disconnect(). The application may also issue another Connect() either to a different address or with different login credentials.

You may select asynchronous connect mode by adding the ",async" string to the address of the host. For example, if you are connecting to host "telos.com:9998" you would call Connect("telos.com:9998,async",user,pwd).

When using the async mode the Connect() command will return S_OK immediately. After the socket is connected to the remote host, the control will notify the application by sending a code 98:

```
SendError(session=0,code=98,msg='Socket connect success')
```

If the socket connect fails, then a code 98001 is sent:

```
SendError(session=0,code=98001,msg='Socket connect failed')
```

If the connect is successful and the login is successful then the connect success message (code 1) will follow:

```
SendError(session=0,code=1,msg='Connect: Success!')
```

Otherwise, one of the 1XXX error codes will be sent.

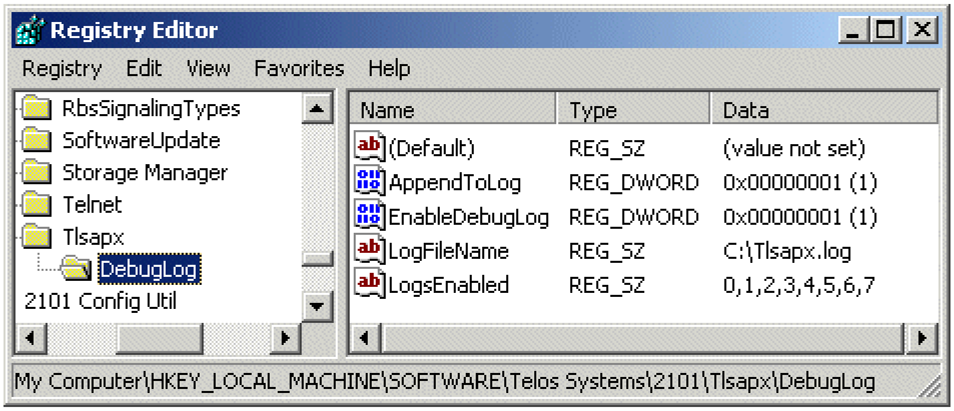
Multiple Session Support

The AP component supports multiple sessions through a single connection. This means that you may connect to multiple shows and/or desktop directors through a single connection. The session parameter works as a “channel” identifier allowing you to specify a particular show connection.

For example, suppose you want to control two shows in talent mode. After calling *OpenControl* and *Connect* you call *ConnectToShow* and specify zero as session identifier for the first show. You then call *ConnectToShow* again and specify one as the session identifier for the second show. Subsequently, any commands you issue using zero as the session identifier will only affect the first show while a session identifier of one will affect the second show. All events received from the first show will be marked with a session identifier set to zero and events received from the second show will be marked with a session identifier set to one.

Debug Output

The TLSAPX component can send debug output to a log file. Since v2.5 it can send debug messages to a debugger output window as well. This may be useful during development or debugging. Debug output is enabled by adding registry entries under the “*HKLM\Software\Telos Systems\2101\Tlsapx\DebugLog*” key:



Name	Type	Data
(Default)	REG_SZ	(value not set)
AppendToLog	REG_DWORD	0x00000001 (1)
EnableDebugLog	REG_DWORD	0x00000001 (1)
LogFileName	REG_SZ	C:\Tlsapx.log
LogsEnabled	REG_SZ	0,1,2,3,4,5,6,7

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Telos Systems\2101\Tlsapx\DebugLog

The *EnableDebugLog* REG_DWORD parameter determines if the debug output is written to a log file. Set this to 1 (or other non-zero value) to send the output to a file or to zero to disable file output. The *LogFileName* REG_SZ parameter specifies the name of the log file where the debug output is written while the *AppendToLog* REG_DWORD parameter controls if the file is overwritten or appended to each time the control is opened. If the *LogFileName* option is not specified, then the output is written to “C:\TLSAPX.LOG”.

The *EnableDebugOut* DWORD parameter determines if debug messages are sent to the currently active debugger. If set to 1 (or other non-zero value) then the debug output is sent to the debugger, if zero then it is disabled.

Set the *DisableMessageBox* DWORD parameter to 1 to disable MessageBox error display. MessageBox error display is enabled by default to display critical internal errors. We recommend to turn this off only if absolutely necessary.

The *LogsEnabled* parameter (used in v2) is obsolete and no longer used.

AttachAsTalent

Attach to a show in Talent mode.

Declaration

```
AttachAsTalent(int nSession)
```

Parameters

nSession	Session identifier.

Return Value

S_OK on success, E_FAIL on error.

Remarks

This method attaches to a show in Talent mode. You must already be connected to a show before using *AttachAsTalent*.

AttachToDirector

Attach to a director.

Declaration

```
AttachToDirector(int nSession, BSTR strDirectorName)
```

Parameters

nSession	Session identifier.
strDirectorName	Name of director as returned by *EnumerateDirectors*().

Return Value

S_OK on success, E_FAIL on error.

Remarks

This method attaches to a desktop director. You can then control the desktop director from software. User actions on the director are reflected back as notification events.

CloseControl

Close control.

Declaration

```
CloseControl()
```

Return Value

S_OK on success, E_FAIL on error.

Remarks

Call this method to close the control when no longer needed.

Connect

Connect to 2101 server.

Declaration

```
Connect(int  nSession,
        BSTR strHostPort,
        BSTR strUser,
        BSTR strPassword
        )
```

Parameters

nSession	Session identifier.
strHostPort	Host and port in this format: “host:port”. If a port is not specified a default value of 9998 is assumed.
strUser	User name registered on the specified server.
strPassword	Password.

Return Value

S_OK on success, E_FAIL on error.

Remarks

This method opens a connection to a 2101 server using the specified user name and password. The user name and password must already be registered on the 2101 server.

ConnectToShow

Connect to an existing show.

Declaration

```
ConnectToShow(int  nSession,
              BSTR strShowName,
              BSTR strShowPassword,
              BSTR strHostName
              )
```

Parameters

nSession	User specified session identifier.
strShowName	Show name
strShowPassword	Show password
strHostName	Host name

Return Value

S_OK on success, E_FAIL on error.

Remarks

This method connects to a show. The session identifier specified will be used by all subsequent calls to refer to this particular session.

ChangeShow

Change the show.

Declaration

```
ChangeShow(int  nSession,
           BSTR strNewShowName,
           BSTR strNewShowPassword
           )
```

Parameters

nSession	User specified session identifier.
strNewShowName	New show name.
strNewShowPassword	New show password.

Return Value

S_OK on success, E_FAIL on error.

Remarks

This method creates a show on the specified host. Use EnumHostsForShow() to determine all hosts where a particular show may be created. The session identifier specified will be used by all subsequent calls to refer to this particular session.

Disconnect

Close connection to 2101 server.

Declaration

```
Disconnect()
```

Return Value

S_OK on success, E_FAIL on error.

Remarks

This method closes the connection to the 2101 server.

EnumerateDirectors

Enumerate directors.

Declaration

```
EnumerateDirectors(int nSession)
```

Parameters

nSession	Session identifier as specified in the call to ConnectToShow().

Return Value

S_OK on success, E_FAIL on error.

Remarks

This method enumerates the directors attached to the show we're connected to. This method can only be called after a *CreateShow*() or *ConnectToShow*() returned success.

EnumerateDirectorsCallback

Director event.

Declaration

```
void EnumerateDirectorsCallback(int nSession,
                                const char *pszDirectorName,
                                bool bIsTalent,
                                bool bIsFree,
                                bool bIsLast
                                )
```

Parameters

nSession	Session identifier as specified in the call to ConnectToShow().
pszDirectorName	Name of director.
bIsTalent	true if in talent mode.
bIsFree	true if not controlled by attached software.
bIsLast	true if this is the last director.

Return Value

S_OK on success, E_FAIL on error.

Remarks

This event is fired in response to a call to *EnumerateDirectors*().

EnumerateHostsForShow

Request a host list for a show.

Declaration

```
EnumerateHostsForShow(BSTR strShowName)
```

Parameters

strShowName	Show name

Return Value

S_OK on success, E_FAIL on error.

Remarks

This method requests list of hosts where the specified show is allowed to run. If the show is not defined in the system this will return a list of all active hosts. The host callback event will be invoked for each host.

EnumerateHostsCallback

Host event.

Declaration

```
void EnumerateHostsCallback(char *pszHostName,
                            bool bIsLast
                            )
```

Parameters

pszHostName	Name of the host.
bIsLast	true if this is the last host.

Return Value

S_OK on success, E_FAIL on error.

Remarks

This event is fired in response to a call to EnumerateHostsForShow().

EnumerateShows

Request a show list.

Declaration

```
EnumerateShows()
```

Return Value

S_OK on success, E_FAIL on error.

Remarks

This method requests a show list known to 2101. The show callback event will be invoked for each show.

EnumerateShowsCallback

Show event.

Declaration

```
void EnumerateShowsCallback(const char *pszShowName,
                            bool bRequiresPass,
                            bool bIsActive,
                            const char *pszHostName,
                            bool bIsLast
                            )
```

Parameters

pszShowName	Name of the show.
bRequiresPass	true if show requires a password.
bIsActive	true if show is active.
pszHostName	Name of the host where show is running.
bIsLast	true if this is the last show.

Return Value

S_OK on success, E_FAIL on error.

Remarks

This event is fired in response to a call to *EnumerateShows*().

EnumerateShowsForHost

Request a list of show that may run on the specified host.

Declaration

```
EnumerateShowsForHost(BSTR strHostName)
```

Parameters

strHostName	Host name

Return Value

S_OK on success, E_FAIL on error.

Remarks

This method requests list of shows allowed to run on the specified host. If the host name is not defined in the system this will return a list of all shows. The host callback event will be invoked for each host.

EnumerateShowsForHostCallback

Host event.

Declaration

```
void EnumerateShowsForHostCallback(char *pszShowName,
                                   bool bIsLast
                                   )
```

Parameters

pszShowName	Name of the show.
bIsLast	true if this is the last show.

Return Value

S_OK on success, E_FAIL on error.

Remarks

This event is fired in response to a call to EnumerateShowsForHost().

OpenControl

Open control.

Declaration

```
OpenControl()
```

Return Value

S_OK on success, E_FAIL on error.

Remarks

Call this method first before using any other methods.

PostAcousticAdapt

Send command to adapt hybrid to line conditions.

Declaration

```
PostAcousticAdapt(int nSession, int bOn)
```

Return Value

S_OK on success, E_FAIL on error.

Remarks

Send command to adapt hybrid to line conditions.

PostBusyAllButton

Post desktop director key press.

Declaration

```
PostBusyAllButton(int nSession)
```

Parameters

nSession	Session identifier.

Return Value

S_OK on success, E_FAIL on error.

Remarks

Calling this method has the same effect as pressing the corresponding key on the desktop director.

PostDelayDumpButton

Post desktop director key press.

Declaration

```
PostDelayDumpButton(int nSession, BOOL bDown)
```

Return Value

S_OK on success, E_FAIL on error.

Remarks

Calling this method has the same effect as pressing the corresponding key on the desktop director.

PostDialKey

Post desktop director key press.

Declaration

```
PostDialKey(int nSession, int chKey)
```

Return Value

S_OK on success, E_FAIL on error.

Remarks

Calling this method has the same effect as pressing the corresponding key on the desktop director.

PostDropButton

Post desktop director key press.

Declaration

```
PostDropButton(int nSession, int nColumn)
```

Return Value

S_OK on success, E_FAIL on error.

Remarks

Calling this method has the same effect as pressing the corresponding key on the desktop director.

PostHoldButton

Post desktop director key press.

Declaration

```
PostHoldButton(int nSession, int nColumn)
```

Return Value

S_OK on success, E_FAIL on error.

Remarks

Calling this method has the same effect as pressing the corresponding key on the desktop director.

PostLineButton

Post desktop director key press.

Declaration

```
PostLineButton(int nSession, int nLine, int nColumn)
```

Return Value

S_OK on success, E_FAIL on error.

Remarks

Calling this method has the same effect as pressing the corresponding key on the desktop director.

PostMuteButton

Post desktop director key press.

Declaration

```
PostMuteButton(int nSession)
```

Parameters

nSession	Session identifier.

Return Value

S_OK on success, E_FAIL on error.

Remarks

Calling this method has the same effect as pressing the corresponding key on the desktop director.

PostMuteRinger

Send a command to mute the ringer.

Declaration

```
PostMuteRinger(int nSession, int bOn)
```

Return Value

S_OK on success, E_FAIL on error.

Remarks

Mute ringer.

PostNextButton

Post desktop director key press.

Declaration

```
PostNextButton(int nSession)
```

Parameters

nSession	Session identifier.

Return Value

S_OK on success, E_FAIL on error.

Remarks

Calling this method has the same effect as pressing the corresponding key on the desktop director.

PostRecordButton

Post desktop director key press.

Declaration

```
PostRecordButton(int nSession)
```

Parameters

nSession	Session identifier.

Return Value

S_OK on success, E_FAIL on error.

Remarks

Calling this method has the same effect as pressing the corresponding key on the desktop director.

PostSpeakerButton

Post desktop director key press.

Declaration

```
PostSpeakerButton(int nSession)
```

Parameters

nSession	Session identifier.

Return Value

S_OK on success, E_FAIL on error.

Remarks

Calling this method has the same effect as pressing the corresponding key on the desktop director.

PostText

Send a text string.

Declaration

```
PostText(int nSession,
         int nTextType,
         int nLine,
         BSTR strText
         )
```

Parameters

nTextType	Application-defined text identifier.
nLine	Line number associated with the text.
strText	Text string.

Return Value

S_OK on success, E_FAIL on error.

Remarks

This method sends a text string to the 2101 server. The server then sends the string to all devices attached to the same show as us. The string is ignored by desktop directors but it is used by software programs to pass specific information about caller name, address, etc..

PostTransferButton

Post desktop director key press.

Declaration

```
PostTransferButton(int nSession)
```

Parameters

nSession	Session identifier.

Return Value

S_OK on success, E_FAIL on error.

Remarks

Calling this method has the same effect as pressing the corresponding key on the desktop director.

PostUpdateAll

Request a status update.

Declaration

```
PostUpdateAll(int nSession)
```

Parameters

nSession	Session identifier.

Return Value

S_OK on success, E_FAIL on error.

Remarks

Invoke this method to request a status update from the server. All status information will be sent back including line states, button states, and etc. through the appropriate callback functions.