WRS970 API Developer's Guide

October 8, 2009
Revision 3.2.3

Fleetwood Reply® WRS970 System
Application Programming Interface (API)
Software License Agreement
Copyright Fleetwood Group, Inc. © 2008

IMPORTANT!

PLEASE READ THIS LICENSE AGREEMENT CAREFULLY BEFORE USING THE REPLY WRS970 SYSTEM APPLICATION PROGRAMMING INTERFACE ("SOFTWARE").

FLEETWOOD GROUP, INC. ("FLEETWOOD") IS WILLING TO LICENSE THIS SOFTWARE TO YOU ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT.

BY ACCEPTING THIS SOFTWARE AND/OR ITS ASSOCIATED FILES AND DOCUMENTATION, YOU AS THE INDIVIDUAL, THE COMPANY, OR THE LEGAL ENTITY THAT WILL BE UTILIZING THE SOFTWARE (REFERENCED BELOW AS "YOU OR YOUR") BECOME BOUND BY THE TERMS OF THIS LICENSE.

THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU AND FLEETWOOD.

IF YOU DO NOT AGREE TO ABIDE BY ALL OF THE TERMS AND CONDITIONS OF THIS LICENSE, THEN YOU SHALL MAKE NO FURTHER USE OF THIS SOFTWARE, AND YOU SHALL IMMEDIATELY RETURN THE SOFTWARE (PLUS ALL ASSOCIATED FILES AND DOCUMENTATION) TO FLEETWOOD INTACT AND UNUSED, AND YOU SHALL RETAIN NO COPIES OF THIS SOFTWARE OR ITS DOCUMENTATION FOR ANY PURPOSE WHATSOEVER.

THIS SOFTWARE IS PROTECTED BY COPYRIGHT LAWS AND INTERNATIONAL TREATIES. ANY UNAUTHORIZED POSSESSION, REPRODUCTION, OR DISTRIBUTION OF THE SOFTWARE, ITS COMPONENTS, OR ITS DERIVATIVE APPLICATIONS MAY RESULT IN SERIOUS CIVIL AND/OR CRIMINAL PENALTIES.

1. License. The Software that accompanies this License is the sole property of Fleetwood. However, while Fleetwood continues to own the Software, You will have certain rights specified by this License to use the Software after Your acceptance of this License. This License applies to this Software as well as any prior and future releases, revisions, modifications, or enhancements to the Software that Fleetwood may furnish to You.

Subject to the terms below, You are hereby granted the right to use the Software and its associated materials for application development and distribute selected Software controls/elements ("OCX") with Your applications.

This Software is LICENSED, NOT SOLD, to You by Fleetwood for uses only according to the terms of this License Agreement. Fleetwood maintains exclusive and permanent ownership of the Software and reserves any and all rights not expressly granted to You.

This is NOT free Software, and You acquire NO interest in it or its subsets (i.e., computer code, Reply® WRS970 system commands, computer communications functions, etc.). You only own the media on which the Software is recorded or fixed, but Fleetwood retains sole ownership of the Software itself.

This License Agreement allows You, as the Licensee, to:

a. Use the Software on ONE (1) computer or local workstation. To "use" the Software means that the Software is either loaded in the memory (i.e., RAM) of a computer or installed on the permanent memory of a computer (i.e.,

hard disk, etc.). One registered copy of the Software may either be used by a single person who uses the Software personally on one computer, or installed on a single workstation used non-simultaneously by multiple people, but not both.

b. Distribute ONE copy of the OCX with each application You develop with this Software ROYALTY FREE.

c. Make ONE copy of the Software in machine readable form solely for backup purposes. As an express condition of this License, You must reproduce on each copy any copyright notice or other proprietary notice that is on the original copy supplied by Fleetwood.

d. Notwithstanding any other terms in this License Agreement, if the Software is licensed as an upgrade or update, then You may only use the Software to replace previously validly licensed versions of the same Software. You agree that the upgrade or update does not constitute the granting of a second license to Software (i.e., You may not use the upgrade or update in addition to the Software it is replacing, nor may You transfer the Software which is being replaced to another party).

2. Restrictions. This Software contains valuable trade secrets and, to protect them:

YOU MAY NOT MAKE ANY ATTEMPT TO DISCOVER THE SOURCE CODE OF THE SOFTWARE. YOU MAY NOT REVERSE ENGINEER, DECOMPILE, DISASSEMBLE OR OTHERWISE REDUCE THE SOFTWARE TO ANY HUMAN PERCEIVABLE FORM. YOU MAY NOT MODIFY, ADAPT, OR TRANSLATE THE SOFTWARE. YOU MAY NOT SUBLICENSE, RENT, SELL, LEASE, OR LOAN ALL OR ANY PORTION OF THE SOFTWARE. YOU MAY NOT CREATE DERIVATIVE WORKS BASED UPON THE SOFTWARE OR ANY PART THEREOF. BASED ON KNOWLEDGE OR USAGE OF THE SOFTWARE, YOU MAY NOT CREATE OTHER REPLY WRS970 SYSTEM CONTROL SOFTWARE THAT REPRESENTS COMPUTER OR MICROCONTROLLER CODE THAT CAN BE OR IS SOLD AS "PROGRAMMING TOOLS" OR “SYSTEM INTERFACES”.

While this license specifies certain rights to distribute components of the Software with applications You may develop from time to time for use with the Reply WRS970 system, YOU MAY NOT USE OR CONVEY THIS SOFTWARE IN ITS FULL, PARTIAL, DISASSEMBLED, OR CONCEPTUAL FORMS FOR ANY OTHER PURPOSE. ALSO, YOU MAY NOT CREATE, OFFER, OR DISTRIBUTE SOFTWARE AND/OR SYSTEM CONTROLS FOR THE REPLY WRS970 SYSTEM SIMILAR IN FUNCTIONALITY TO THE SOFTWARE CONVEYED BY THIS LICENSE.

3. Dual Media. Even if the Software product includes the Software on more than one medium (e.g., on a CD, on magnetic disks, or as a file sent by email or downloaded from the Internet), You are only licensed to use ONE copy of the Software as described in Section 1. You MAY NOT use the Software stored on the other medium on another computer or common storage device, NOR may You rent, lease, sell, loan or transfer the Software or its source information or its documentation to another user.

4. Export Law Assurances. Export of this Software is governed by the laws and regulations of the United States and import laws and regulations of certain other countries. You agree that neither the Software nor any direct product thereof is being or will be shipped, transferred or re-exported, directly or indirectly, into any country prohibited by the United States Export Administration Act and the regulations thereunder or will be used for any purpose prohibited by the Act. Export or re-export of Software to any entity on the Denied Parties List and other lists promulgated by various agencies of the United States Federal Government is strictly prohibited. (Note: Other Federal trade rules and regulations may apply to this product and its derivatives from time to time, and You agree to be exclusively responsible for compliance as well as exclusively liable for any noncompliance.)

5. Termination. This License is effective to all users of the Software until terminated by its owner, Fleetwood. This License will terminate immediately without notice from Fleetwood if You fail to comply with any provision of this License Agreement. This License may also terminate immediately by judicial resolution. You may also be subject to damages awarded by court or arbitration proceedings. Upon such termination You must destroy the Software and all copies thereof. Sections 6 through and including 15 of this License will survive any termination of this License.

6. Limited Warranty. The Software is licensed 'as is'. Fleetwood will offer to replace defective Software reported within 30 days of delivery. Technical support for the Software is not included in the License but may be available according to policies established by Fleetwood from time to time. When available, this support is limited to providing Software documentation and explaining component functionality. Support will NOT include advice for or assistance in designing an application that incorporates the Software. The availability and cost of support will vary according to policies that are subject to change without notice.

THIS LIMITED WARRANTY IS THE ONLY WARRANTY PROVIDED BY FLEETWOOD AND ITS LICENSERS. FLEETWOOD AND ITS LICENSERS EXPRESSLY DISCLAIM ALL OTHER WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE WITH REGARD TO THE SOFTWARE AND ACCOMPANYING WRITTEN MATERIALS.

BECAUSE SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

7. Limitation of Remedies and Damages. In no event will Fleetwood, its directors, officers, owners, employees or affiliates of any of the foregoing, past or present, be liable to You or Your application users for any consequential, incidental, indirect, special, or punitive damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information and the like), whether foreseeable or unforeseeable, arising out of the use of or inability to use the Software (or its accompanying materials and Reply hardware), regardless of the basis of the claim and even if Fleetwood or a representative of Fleetwood has been advised of the possibility of such damage. Fleetwood's liability for direct damages for any cause whatsoever, and regardless of the form of the action, will be limited to money damages not to exceed the total amount paid to Fleetwood for the Software License.

The disclaimers and limitations set forth above will apply regardless of whether You accept the Software.

BECAUSE SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

8. Applicable Law. This License Agreement shall be governed for all purposes by the laws of the State of Michigan.

9. Survivorship of Provisions. If any provision of this License Agreement shall be held by a court of competent jurisdiction to be contrary to law, that provision will be enforced to the maximum extent permissible, and the remaining provisions of this License Agreement will remain in full force and effect.

10. Disputes. You agree to resolve any and all disputes or controversies arising from or relating to this License Agreement by arbitration in lieu of legal and/or court action. Arbitration will be conducted in the City of Holland, Michigan, by the American Arbitration Association, by a panel of three or more arbitrators mutually acceptable to the parties, and in accordance with the procedural rules and regulations of the selected association. Fleetwood reserves the right to refuse or cease arbitration and pursue legal and/or court action should the dispute or controversy result from Your violation of this License's terms, particularly if such violation involves (i) unauthorized use or distribution of the Software or other Fleetwood proprietary information or (ii) Your failure to promptly and fully pay any prearranged License fees.

11. Nonassignability. This License and the rights and duties hereunder are personal to the Licensee upon whose skill, judgment, reputation, and form and method of doing business Fleetwood is relying. You MAY NOT transfer this License or any of its rights in whole or in part by assignment, sale, merger, or consolidation, whether by operation of law or otherwise, without the prior written consent of Fleetwood. You MAY NOT rent, lease, sell, loan or transfer the Software or its source information or its documentation to another user under any circumstance except as expressly provided in this license.

12. Trademarks. You ARE NOT authorized to employ the trademarks, trade names, or commercial symbols of Fleetwood without the expressed and advance written approval of Fleetwood, except as specifically authorized by this License Agreement. This License authorizes these two specific uses of such information, PROVIDING You agree that You shall immediately terminate such uses upon notice by Fleetwood Group, Inc.: (1) You MAY use the names

"Fleetwood Group, Inc." and "Reply®" in text form to provide necessary reference to the manufacturer and its products in your application's documentation and promotion, providing that ownership of those names is properly credited to Fleetwood Group, Inc., and (2) You MAY hyperlink the web-enabled content of your application documentation and promotion to the www.replysystems.com website, so long as the information contained in Your documentation and promotion is recognized by Fleetwood Group, Inc. to accurately and positively represent itself and its products. You DO NOT acquire any proprietary interest in Fleetwood products, trademarks, trade names, or commercial symbols as the result of using the Software.

13. Status of Parties. You are acting pursuant to this License Agreement in the capacity of an independent contractor. The management of Your business, including the formulation and execution of plans, policies, and procedures, are Your sole prerogative and responsibility. You SHALL NOT use any Fleetwood trademark, trade name, or commercial symbol as part of the name under which you conduct business, and nothing contained in this License shall be interpreted or construed so as to characterize the relationship between You and Fleetwood as a joint venture, partnership, agency or franchise for any purpose whatsoever.

14. Reservations. Fleetwood reserves the right to appoint other Licensees of this Software within any geographic area, recommend applications, establish and adjust License fees, and modify or discontinue the distribution of the Software or any product it supports, all without incurring any obligation of any kind whatsoever to You or other Licensees. Additionally, Fleetwood reserves any and all rights not expressly granted to You.

15. Binding Effect. This License Agreement shall be binding upon the parties hereto and upon their respective executor, administrators, legal representatives, successors, and assigns.

<END OF LICENSE>

DOCUMENT CONTROL

File Name	WRS970 API Developer's Guide
Original Author(s)	Steve Davis
Current Revision Author(s)	Steve Davis/Tim Lambrix

Version	Date	Author(s)	Notes on Revisions
1.0.0	04/26/2007	SD	Preliminary Document
1.0.1	05/21/2007	SD	Added getbaseinfo(), pingTimer, corrected OnKeypadPing event, PowerKeyEnabled
1.0.2	07/20/2007	SD	Changed to WRS970 Added new functionality
1.0.3	08/06/07	TL/SD	In depth system operation descriptions. Added possible types for properties that require them.
1.0.4	08/13/07	TL	Minor Description Updates
1.0.5	08/24/07	SD	Fixed the Custom Message code example
	09/17/07	SD	Added VS 6.0 C++ documentation
1.0.6	10/16/07	SD	Added keypad messaging, removed auto from Wifi Avoidance, added "mt" suffix to menu access type
1.0.7	11/21/07	SD, TL	Added to the description of AddStaticKeypad() method and ClearKeypadList(); Updated Custom messaging description
1.0.8	11/30/07	SD, TL	Added to the description of Keypad list, DuplicateID() event, and the Alert Key functionality
3.0.0	04/18/08	SD	Added 2.0 / 3.0 functionality Add "link" and "all" to key lock / unlock SetSession() method CustomSoftKeys property Model property SMSEntry property MsgAckEnabled property OnKeypadMsgTimeout event Ftd2xx.dll Explanation
3.0.1	07/07/08	SD	Added KeypadPowerTimer
3.1.0	08/08/08	SD	Removed the keypad ID from the OnNotAuthorized event
3.1.1	08/21/08	SD	Clarified the need of Ftd2xx.dll
3.2.1	07/06/09	SD	Added ScrollFaster, HideBattery, HideAntenna, SMSVoteTimeout, AutomateQuestionNumber, onSMSVoteTimeout, GetTCPBases, SetSession
3.2.2	08/13/09	SD	Added OnKeypadData event Obsoleted OnKeypadDataReceived event
3.2.3	10/08/09	SD	Added Version3 property to handle legacy software

Table of Contents

LIST OF METHODS

LIST OF EVENTS

LIST OF PROPERTIES

1 INTRODUCTION

The Reply® WRS970 wireless response system is comprised of wireless handheld units, a base station receiver/transmitter, and a control computer. The control computer connects to the base station via a serial or Ethernet connection. Commands are issued from the control computer to the handsets via the base station. The ActiveX Toolkit will allow software developers to create customized applications for communicating with the Reply® WRS970 wireless response system. The ActiveX Toolkit encapsulates communication between the Reply® WRS970 base station and the control computer. This is intended to foster an open architecture concept for the Reply® WRS970 system while simultaneously simplifying implementation.

2 INSTALLATION AND SETUP

The Reply® WRS970 Toolkit is contained in WRS970.ocx. All runtime libraries are compiled into the ActiveX control. In order to use the toolkit in your application, it must be registered. This can be accomplished by using RegSvr32.exe in Windows. For example, the following command registers the Reply® WRS970 Toolkit under a typical Windows installation.

```
c:\winnt\system32\regsvr32.exe "c:\Program Files\Fleetwood Group\ReplyWRS970\WRS970.ocx"
```

The toolkit can be uninstalled by issuing the same command with a "–u" argument. The WRS970.ocx file can then be deleted.
To use the Reply® WRS970 Toolkit in your application, it must be referenced in the project in your development environment. For example, in Visual Basic 6, with an open project, select the Project -> Components… menu item. Select WRS970 Toolkit from the list of controls. The Toolbox (control palette) will contain the control WRS970X. This control can be used at design time by adding it to a form or container object from the toolbox or it can be instantiated within code.

2.1 Finding a Base Station (USB)

GetAvailableBases() requires an instance of AxWRS970X. The API communicates with the base station via native USB. The method will find all of the devices attached to the network matching the criteria as set forth by the design of the product. A comma delimited string returning the serial numbers of the base stations will be returned. A serial number is the parameter needed for the Connect() method. Please see GetAvailableBases() in the methods section of this document for details.

2.2 Finding a Base Station (TCP)

In Reply Plus 3.0 you may use the method GetTCPBases() to retrieve Ethernet base IP addresses and ports on your network. This method uses a UDP broadcast to detect bases. The bases detected may or may not be already connected to software. If you choose to set the WithDescription parameter true, the API will only return the bases that are available with information about that base. Please see GetTCPBases() in the methods section of this document for details. However, using WithDescription greatly increases the time needed to determine the connection status of a base that is already connected to other software. Windows takes a significant amount of time to return a communication error for a socket that is in use. You must allow up to sixty seconds for the proper number of bases to be returned.

If a base is disconnected improperly from a network, a port will remain open for some time before it times out and is released again for connection. In the event of a base power failure or a cable is pulled unexpectedly, you must give the connection time to release before you are able to connect again to the WRS970 base.

2.3 Connecting to a Base Station

To connect to the base station, begin by selecting the communication type. Choose TCP/IP or USB. Set the WRS970 instance CommType property to ctTCP or ctUSB appropriately. If ctTCP is selected, the property IPAddress must be specified in standard x.x.x.x notation and the property TCPPort supplied with an integer. The default TCPPort is 2101 for WRS970 devices. Otherwise, if ctUSB is selected, the base serial number must have been retrieved with GetAvailableBases() method. You can then call the Connect() method with a null parameter for TCP or the serial number for USB. If a connection failure occurs (base not found, base already connected, base not powered on, etc.) an exception will be raised for TCP that the base is busy or cannot be found. The USB will return "No device Found" in GetAvailableBases(). Please see the Connect() in the methods section of this document for details.

2.4 Configuring a Base Station

The WRS970 API provides control over the operation of the base station. The properties and methods responsible for configuring base operations are below:
BaseName – A property stored by the base to help describe the base station.
CommType – A property managing the type of communications, USB or TCP, of the system.
BaseChannel – A property managing the current base station channel. Used for multiple base stations in the same receive area.
AddressMode – A property managing the type, dynamic or static, of the WRS970 system. A static system allows the keypads to communicate with the address that is currently stored. A dynamic system requires the keypads to link to the system, which assigns the keypad an address. An address can be assigned by either the keypad list or sequentially by the system.
PowerLevel – A property managing the RF power of the WRS970 system. A higher power increases the RF range thereby increasing the WRS970 system area.
AvoidWifi – A property managing the RF frequency ranges used by the WRS970 system. This property can be used to increase system efficiency by avoiding a high noise level used by Wifi access points.
KeypadLoginEnabled – A property managing the security of a WRS970 system. When a link to the system is attempted, this property can require a login to be entered for the keypad to receive an address.
Authenticate – A property managing another level of security for access to the WRS970 system. If authenticate is set to true, only the keypads in the keypad list are authorized to use the system.
MaxKeypadsAllowed – A property that allows a maximum number of keypads to be added to the keypad list.
For details of the above properties and methods please see the properties and method section of this document.

2.5 Specifying Keypads

The WRS970 control maintains a list of keypads that are allowed on the system. To add keypads to the system manually, call the AddKeypad for dynamic addressing, or if you wish to specifically assign an address call AddStaticKeypad. Please see these methods in the methods section of this document. Please see scenarios below:

Authentication – false
KeypadList – empty
AddressMode – static or dynamic

Keypads are added to the KeypadList sequentially as they communicate with the system.

Authentication – false
KeypadList – populated
AddressMode – static or dynamic

Keypads are given the address equal to the index of where they exist in the keypad list.

Authentication – true
KeypadList – populated
AddressMode – static

The keypad's serial number is checked against the serial number and the address is checked against the index in the keypad list. If they do not match, the keypad is not authorized on the system.

Authentication – true
KeypadList – populated
AddressMode – dynamic

The keypad's serial number is checked against the serial numbers in the keypad list. If the serial number is found the keypad is given an address equal to the index of where it was found in the keypad list. If it's not found in the keypad list, the keypad is not authorized on the system.

2.6 Setting up Keypads

The following methods are used to setup keypads:

MenuAccess – Limits the accessibility of the user to the keypad menu.

BackLightOpMethod – Sets the operation of the keypad backlight.

KeypadBacklightOnTimer – Sets the number of seconds the keypad backlight stays on for the specified operation method.

KeypadBacklight – Turns the backlight on or off.

ReceiveArrowOnAnyBase – Changes the operation of the receive arrow blink on keypads.

AutoConnect – Automatically connects a keypad to a base when there is only one base found.

KeypadsOutOfRange – Keypads beep when taken outside of the system area for an extended period of time.

KeyLockFeedback – Keypads that have locked keys will report that those keys are locked to the LCD screen.

SecureLogin – Keypads will display an asterisk instead of the login data on the LCD screen.

KeypadDigiEcho – Keypad votes will be concealed on the LCD screen.

KeypadAlertEnabled – Enables or disables the keypad alert key on the BeginQuestion method call.

ShowReceived – Keypads will show "Received" on the acknowledgement from the base after keypad transmission.

ShowSending – Keypads will show "Sending" while it waits for the base to acknowledge its transmission.

2.7 Visual Studio C++ 6.0 Considerations

The WRS970 API contains hidden properties that require the addition of the following code to the header files to expose these properties. These properties were hidden as they require a connection to the base station before they can be changed.

Property	DISPID	Parms	Returns
KeypadList	0xDD		CString
BaseVersion	0xFB		CString
BaseSerial	OxFC		CString
BaseName	0xFE	BSTR	CString
CustomMessage	0x112	BSTR	CString
BeepKeypads	0x115	BOOL	BOOL
Connected	0xEE		BOOL
ForceMenu	0x114	BOOL	BOOL
KeypadListCount	0xEF		long
LowBatteryNotification	0xF2	BOOL	BOOL
KeypadsPowerDown	0xF9	BOOL	BOOL
ShowAnswer	0x113	BOOL	BOOL

2.8 Driver and API Considerations

The WRS970 systems use FTDI drivers. The interface to these drivers is the ftd2xx.dll located in the systems folder of your Windows machine. If for any reason you're not able to find your device on the machine with the drivers installed properly and the device is listed in the device manager, check the ftd2xx.dll to be sure the date is later than 06/2007. If this DLL is old, please download the latest drivers from http://www.ftdichip.com/Drivers/D2XX.htm. Unzip the drivers file, locate the new DLL in the i386 folder and replace your systems32 DLL with the new.

The WRS970 API is dependent upon the FTD2XX.DLL. You must have the DLL installed by either installing the drivers or by installing the file in your software's setup before you are able to register the OCX.

If you have problems please contact Fleetwood support at replyapi@fleetwoodgroup.com.

3 PROPERTIES AND METHODS

3.1 Base Communication and System Identification

Properties

CommType
BaseChannel
BaseName
BaseVersion
CommunicationTimeout
Connected
Model

Methods

GetAvailableBases
Connect
Disconnect
GetBaseInfo
GetTCPBases
SetSession

Events

none

3.1.1 Communication

CommType – A property managing the type of communications, USB or TCP, of the system.

Available Types: TCommType
ctUndefined
ctTCP
ctUSB

```
if TCP
   axWRS970X1.CommType = WRS970.TCommType.ctTCP;
else
   axWRS970X1.CommType = WRS970.TCommType.ctUSB;
```

GetAvailableBases– A method used to detect bases connected via USB. A Boolean parameter tells whether or not to include the name of the base along with the serial number. Results are comma “,” delimited and the name is separated from the serial number with a dash “-“. If no device is found the result is, “No device Found”.

```
private void mnuBasesConnectionUSB_Click(object sender, System.EventArgs e)
{
    string bases = axWRS970X1.GetAvailableBases( true );

    string[] sbases = bases.Split( ',' );

    cboBases.Items.Clear();

    for ( int i = 0; i < sbases.Length; i++)
    {
        cboBases.Items.Add( sbases[i] );
    }

    cboBases.SelectedIndex = 0;

    if (( cboBases.SelectedIndex > -1 ) && ( cboBases.Text.IndexOf( "No device") < 0))
        baseConnect( cboBases.Text );
    else
        MessageBox.Show( "No base chosen!" );
}
```

TCPPort – A property describing the TCP/IP port of the base to connect. The default property is 2101.

```
private void mnuBasesConnectionTCP_Click(object sender, System.EventArgs e)
{
    axWRS970X1.CommType = WRS970.TCommType.ctTCP;
    frmTCPConnection tcpForm = new frmTCPConnection();

    if ( tcpForm.ShowDialog() == DialogResult.OK )
    {
        if ((tcpForm.txtTCPAddress.Text.Length > 0) &&
                            (tcpForm.txtTCPPort.Text.Length > 0))
        {
            string data = null;
            data = tcpForm.txtTCPAddress.Text + "," + tcpForm.txtTCPPort.Text;
            baseConnect( data );
        }
        else
            MessageBox.Show( "Invalid parameters specified!", "Input Error",
                                    MessageBoxButtons.OK, MessageBoxIcon.Error );
    }
}
```

GetTCPBases - A method used to detect bases connected via TCP. A Boolean parameter tells whether or not to include the name of the base along with the serial number. The result is an array of string objects. Data in the string array is delimited by a "-" and contains the IPAddress, Subnet, HostName, and when the WithDescription parameter is set true, the BaseName. If no device is found the result is a single indexed array with, "No device Found". Please see Section 2.2 for further details.

```
private void mnuBasesConnectionTCP_Click(object sender, System.EventArgs e)
{
    axWRS970X1.CommType = WRS970.TCommType.ctTCP;

    object[] tempArray = (object[])axWRS970X1.GetTCPBases(true);

     foreach (object tempObject in tempArray)
     {
         string tcpBase = (string)tempObject;

         string[] fields = tcpBase.Split('-');
         tcpForm.TCPBases[baseIndex].IPAddress = fields[0];
         tcpForm.TCPBases[baseIndex].Subnet = fields[1];
         tcpForm.TCPBases[baseIndex].HostName = fields[2];
         tcpForm.TCPBases[baseIndex].BaseName = fields[3];
         tcpForm.TCPBases[baseIndex].SerialNumber = fields[4];
         baseIndex++;
     }
}
```

TCPAddress – A property describing the TCP/IP address of the base to connect. All products are shipped with a default property of 200.0.0.78. See TCPPort for example.

Connect – A method used to connect the PC to the base station via USB or TCP. CommType must be set to desired connection before calling this method. Method is called with null as a parameter for a TCP/IP connection and a serial number for a USB connection.

```
private void baseConnect( string info )
{
    if ( axWRS970X1.CommType == WRS970.TCommType.ctUSB )
    {
        string[] baseSplit = info.Split( '-' );
        axWRS970X1.Connect( baseSplit[0] );
    }
    else
    {
        string[] infoSplit = info.Split( ',' );
        axWRS970X1.TCPAddress = infoSplit[0];
        axWRS970X1.TCPPort = int.Parse( infoSplit[1] );
        axWRS970X1.Connect( null );
    }
}
```

Disconnect – A method used to disconnect the PC from the base station. Calling this method does not lose the keypad list which is described later. As long as the instance of the OCX exists, the keypad list will remain as is.

```
private void mnuSystemDisconnect_Click(object sender, System.EventArgs e)
{
    if ( axWRS970X1.Connected )
        axWRS970X1.Disconnect();

    debugMessage( "Connected: " + axWRS970X1.Connected.ToString() );
}
```

Connected – A read-only property indicating whether or not the API is connected to the base station.

```
private bool checkConnected()
{
    return axWRS970X1.Connected;
}
```

3.1.2 Settings

GetBaseInfo – A method used to populate the variables such as base channel, base name, and firmware. Call this method before reading these variables and after connection to the base.

```
private void btnBaseInfo_Click(object sender, System.EventArgs e)
{
    axWRS970X1.GetBaseInfo();
    showBaseInfo();
}
```

BaseChannel – A property managing the current base station channel. The term Base ID is also used interchangeably with Base Channel. Base channel is used to operationally separate Reply systems that share RF operating range. This value is used by keypads to distinguish which base unit to 'talk' with and therefore must be unique to each system.

```
private void btnChannelLogin_Click(object sender, System.EventArgs e)
{
    axWRS970X1.BaseChannel = int.Parse( txtChannel.Text );
}

private void getBaseInformation_Click( object sender, System.EventArgs e )
{
    axWRS970X1.GetBaseInfo();
    txtChannel.Text = axWRS970X1.BaseChannel;
}
```

BaseName – A property stored by the base to help describe the base station. The base must be connected for this property to be set. Naming options can be as simple as "Base 1" and "Base 2" or more descriptive such as "Conference Room A" or "Room 202". Any description is allowed up to 24 characters including spaces. The name is stored in the base when power is removed from the base station. The getAvailableBases method with a TRUE parameter will show the BaseName stored in the bases.

```
private void btnBaseName_Click(object sender, System.EventArgs e)
{
    AxWRS970X1.BaseName = txtBaseName.Text;
}
```

BaseVersion – A property showing the current firmware version of the base. The base must be connected and GetBaseInfo method will set this property.

```
lblBaseVersion.Text = axWRS970X1.BaseVersion;
```

CommunicationTimeout – A property in milliseconds that sets the amount of time the API will wait for the base station to respond before reporting an error condition. The default value is 3 seconds. This property should not have to be changed unless there is a problem with the PC being able to process incoming data due to the PC being bogged down. An example of this could be connecting 30 bases with 15,000 keypads to one computer. This property is not intended for end users to modify but could be placed in a configuration file or registry for ability to change later.

```
private void mnuSetCommTimeout_Click(object sender, System.EventArgs e)
{
    try
    {
        axWRS970X1.CommunicationTimeout = int.Parse( newTimeout.txtSerialNumber.Text );
    }
    catch ( Exception E )
    {
        MessageBox.Show( E.Message, "Input Error", MessageBoxButtons.OK,
                                    MessageBoxIcon.Error );
    }
}
```

Model – A property that reflects the model number of the device connected. The model number of the USB Stick hardware is WRS971. The original base hardware model is WRS970.

```
lblModel.Text = "Model: " + axWRS970X1.Model;
```

SetSession – A method used to set the session name and group bases on a system. This only applies to systems used in Dynamic mode. See section 3.2 for more information on the Addressing mode. Session name describes the polling event and displays that description on the keypad when searching for available sessions. A session name is different than the Base Name as it is not stored in the base. Base Name examples could be Room 224 and Room 115 where Session Names could be MATH 101 and ENG 450. Only Session Names will display on the keypads. If no session name is set, the keypad will display the Base ID number. Though session name is not required, it is highly recommended to help make things clear for the keypad users. It is limited to 8 characters.

Base Group allows a set of bases that maintain the same group number to be recognized by the keypads as the same system. A session name IS REQUIRED in order to group bases. A base group of 0 means the base is a single base and not part of any system. All single bases should be set to 0. Up to 7 systems are allowed with any number of bases per system. The Base ID of each base covering any area must always be unique. Therefore, if 15 bases were placed on Base Group 1, there would only be 16 available Base ID's to set up another system or single bases. No two bases covering the same area can be set to the same Base ID.

For example, if three bases are needed to cover an event up to 1500 keypads, those three bases can be set to the same Base Group 3 and the keypads will only find one system. Furthermore, if auto connect enabled and only those three bases are used at the event, those keypads will connect to any one of the three base stations automatically. In firmware version 3.0 keypads and later, the keypads have the ability to detect a base at capacity and try to connect to another base within the same system. Only if all the bases are full will the message "System at Capacity" be displayed on the keypad LCD. In keypad firmware 2.x, the keypads will display the Session name in the list but will NOT auto try another base in the system. Use version 3.0 and later to take full advantage of this feature.

Login can still be used with multiple base systems. It is also important that each base be set up with the same operational settings so all keypads perform the same. This includes session name, login options, auto connect, and question settings..

```
private void mnuSetSession_Click(object sender, System.EventArgs e)
{
    frmSetSession myNewSession = new frmSetSession();

    if ( myNewSession.ShowDialog() == DialogResult.Cancel )
        return;

    axWRS970X1.SetSession( myNewSession.txtSessionName.Text, int.Parse(
    myNewSession.txtSessionGroup.Text ) );
}
```

3.2 Addressing Mode

Properties

AddressMode

Methods

none

Events

none

3.2.1 Addressing

AddressMode – A property managing the type, dynamic or static, of the WRS970 base station and therefore, the entire system. A static system is one in which the keypads maintain a stored unique address (up to the maximum of 500) and the Base ID of the base station they communicate with. A dynamic system requires the keypads to connect to the selected base station and receive an address from the system. In a dynamic system, the address can be assigned sequentially by the system (API) or from a keypad list. The keypad list is described in detail in Section 3.4. It is important to note that static and dynamic base stations on the same base ID are seen as 2 separate systems. This means a keypad configured to static addressing on Base ID 1 will not 'talk' to a dynamic base station on Base ID 1. Set this property to configure the base to the desired addressing mode. The keypads must be set locally through the keypads' menu system (see Hardware User Manual for instructions). The API must be connected to the base in order for keypads to receive an ID in dynamic addressing mode. Consider this when developing software. The addressing process takes longer the more keypads that try to connect simultaneously. It is better to allow random connections (say, as users enter the room or sit in their seats) versus asking everyone (500 people) to connect at once.

Available Types: TAddressMode
addrDynamic addrStatic

```
private void setAddressing()
{
    if ( mnuBasesAddrModeStatic.Checked )
        axWRS970X1.AddressMode = WRS970.TAddressMode.addrStatic;
    else
        axWRS970X1.AddressMode = WRS970.TAddressMode.addrDynamic;
}
```

3.3 System Performance

Properties

PowerLevel
ReceiveArrowOnAnyBase
AvoidWifi

3.3.1 RF Power Level

PowerLevel – A read/write property managing the RF power of the WRS970 system. A higher power increases the RF range thereby increasing the WRS970 system area. The system is certified for European use but may not be operated at the same power level as allowed in the US and Canada. For this reason, the power levels are named according to their limits. It is recommended to set the power level to an appropriate level to cover the operational area only. This is especially practical for Reply systems in close proximity to each other. Due to other RF equipment in the area, this level may need to be set higher at times but will always default to the pwrEuroMax setting upon connecting power.

Available Types: TBasePowerLevel
pwrLow
pwrMid
pwrHigh
pwrEuroMax
pwrUSMax

```
private void setPower()
{
    if ( mnuPowerLevelLow.Checked )
        axWRS970X1.PowerLevel = WRS970.TBasePowerLevel.pwrLow;
    if ( mnuPowerLevelMid.Checked )
        axWRS970X1.PowerLevel = WRS970.TBasePowerLevel.pwrMid;
    if ( mnuPowerLevelHigh.Checked )
        axWRS970X1.PowerLevel = WRS970.TBasePowerLevel.pwrHigh;
    if ( mnuPowerLevelEuroMax.Checked )
        axWRS970X1.PowerLevel = WRS970.TBasePowerLevel.pwrEuroMax;
    if ( mnuPowerLevelUSMax.Checked )
        axWRS970X1.PowerLevel = WRS970.TBasePowerLevel.pwrUSMax;
}
```

ReceiveArrowOnAnyBase – A property to change the down arrow icon on the keypad LCD display. This property is available for system flexibility. It does not have to be implemented with the option of turning on or off for the user. It can be beneficial as a debug tool when used according to the keypad list. When this property is cleared, the receive (down) arrow on the keypad LCD will blink when the keypad is specifically polled by the base station. This means, the more keypads that are polled (more groups) the slower the icon will flash. If the display is not flashing but the RF indicator on the keypad display is showing strong signal, the keypad's group is not being polled (verify the keypad list is correct). If the property is set to true, the keypad icon flashes with each base polling packet regardless of group. This is beneficial for troubleshooting RF performance in the room.

```
private void mnuKeypadsRecvGroup_Click(object sender, System.EventArgs e)
{
    axWRS970X1.ReceiveArrowOnAnyBase = mnuKeypadsRecvGroup.Checked;
}
```

3.3.2 WiFi Avoidance

AvoidWifi – A property managing the RF frequency ranges used by the WRS970 system. This property can be used to increase system efficiency by avoiding a high noise level caused by Wifi access points. Typically, access points are configured to one of three popular channels: 1, 6, and 11. For example, if an access point is using channel 1, set the Reply system to avoid the low band (`WRS970.TBand.bndLow`). Determining which channel an access point is located on will require help from the event locations' IT department.

Available Types: TBand
bndNone
bndLowMid
bndLowHigh
bndMidHigh
bndLow
bndMid
bndHigh

```
if ( mnuWifiNone.Checked )
    axWRS970X1.AvoidWifi = WRS970.TBand.bndNone;
if ( mnuWifiLowMid.Checked )
    axWRS970X1.AvoidWifi = WRS970.TBand.bndLowMid;
if ( mnuWifiLowHigh.Checked )
    axWRS970X1.AvoidWifi = WRS970.TBand.bndLowHigh;
if ( mnuWifiMidHigh.Checked )
    axWRS970X1.AvoidWifi = WRS970.TBand.bndMidHigh;
if ( mnuWifiLow.Checked )
    axWRS970X1.AvoidWifi = WRS970.TBand.bndLow;
if ( mnuWifiMid.Checked )
    axWRS970X1.AvoidWifi = WRS970.TBand.bndMid;
if ( mnuWifiHigh.Checked )
    axWRS970X1.AvoidWifi = WRS970.TBand.bndHigh;
```

3.4 Keypad List

Properties

MaxKeypadsAllowed
KeypadList
KeypadListCount
AutoConnect
Authenticate
KeypadLoginEnabled
SecureLogin

Methods

IsKeypadInList
AddKeypad
AddStaticKeypad
RemoveKeypad
RemoveStaticKeypad
ClearKeypadList
KeypadLoginResult
LogoutAllKeypads
LogoutKeypad

Events

OnKeypadLogin
OnDuplicateID
OnKeypadNotAuthorized

Summary

The API maintains a list of keypad serial numbers and their associated addresses for the system. This list is called Keypad List. The Keypad list is used for both dynamically and statically addressed keypads. Both the API and the software have access to maintain or modify the list. This section explains how to add, subtract, clear, modify and view the list. The list is available as long as the instance of the OCX has not been destroyed. Whether a base is connected or not the list will remain intact.

3.4.1 List Maintenance

MaxKeypadsAllowed – A property that sets the maximum number of keypads to be added to the keypad list. The maximum value allowed is 500 (system capacity).

```
private void btnMaxKeypads_Click(object sender, System.EventArgs e)
{
    try
    {
        axWRS970X1.MaxKeypadsAllowed = int.Parse( txtMaxKeypads.Text );
    }
    catch ( Exception exx )
    {
        MessageBox.Show( exx.Message );
        txtMaxKeypads.Focus();
    }
}
```

KeypadList – The list of keypads that are used for authentication, address assignment, and limiting the number of keypads to a system. This is a read-only property. Use the following methods in this section to add, remove, verify, and clear the Keypad List. All keypads whether static or dynamically addressed, will get placed in the list when a keypad is in use. For instance, a static keypad addressed to 5 will be put into the list automatically when it votes via the API. A manual add by the software puts the keypad in the list regardless of a vote. If another static keypad is also addressed to 5, the OnDuplicateID (See below) event will occur. For dynamic addressing mode, an 'Authorizing' keypad is in the process of being added to the list and the API adds the keypad automatically. In return, the API sends the assigned address (KeypadID) back to the keypad (See Section 3.4.3 and 3.4.4 for more list control options). The keypad then reports 'Connected' briefly (see Hardware User Manual for keypad addressing process definition). The keypad list will report zeroes added via software to signify the groups of keypads being polled. For additional options with the keypad list, see the remainder of this section.

```
private void updateDebug()
{
    debugMessage( "List: " + axWRS970X1.KeypadList );
}
```

KeypadListCount – A property of the number of keypads currently in the keypad list. This is a read-only property.

```
private void updateDebug()
{
    debugMessage( "Count: " + axWRS970X1.KeypadListCount.ToString() );
}
```

IsKeypadInList – A method used to determine if a keypad serial number is currently in the keypad list.

```
private void btnIsInList_Click(object sender, System.EventArgs e)
{
    if ( axWRS970X1.IsKeypadInList( txtKeypad.Text ) )
        debugMessage( txtKeypad.Text + " is in the list!" );
    else
        debugMessage( txtKeypad.Text + " is NOT in the list!" );
}
```

AddKeypad – A method used to add a keypad dynamically to the system. The KeypadID assigned to the keypad is determined by the system. Even though the API automatically assigns an address to the keypad, it first searches the list for the keypads' serial number and verifies it is not already in the list. If it is in the list, the keypad receives the ID already assigned. This means that a keypad powered down unexpectedly, receives the same ID it had previously. If the keypad is not in the list, it receives the next available address.

```
private void btnKeypadAdd_Click(object sender, System.EventArgs e)
{
    axWRS970X1.AddKeypad( txtKeypad.Text );
}
```

AddStaticKeypad – A method used to add a keypad statically to the system. The Keypad ID addressed to the system is passed as a parameter to the method. The second parameter is a Boolean value to determine if the keypad assigned is to overwrite any keypads already at that address. It is possible to add a dynamic keypad to a specific address with this method.

The base set in static mode will power up polling all groups. However, once a keypad votes and is added to the keypad list, the OCX optimizes the list to poll the smallest number of keypads; a group of 100. This means that the first keypad to vote, say static keypad 234, will be added to the list and the polling groups will change based on the single keypad in the list to poll keypads 201-300. If it is desired to poll static keypads 1-300, this method can be used to manually add each keypad to the list. If the number of keypads is known but not what each address is, the following paragraph describes how to only add three 'ghost' keypads to ensure all 300 are polled.

A serial number of zero passed into this function allows polling control of groups of keypads. For instance, if keypads 101-200 are not being polled, calling this method with a keypad ID of 101 to 200 and a serial number of zero will force polling on that group. The keypad list will reflect the zero added serial number at the keypad ID position. When a valid keypad is added at the position of any zero added serial number, the valid serial number will replace the zero serial number. A zero serial number will not be allowed to overwrite an existing valid serial number at any keypad ID in the keypad list.

```
private void btnAddStatic_Click(object sender, System.EventArgs e)
{
    try
    {
        axWRS970X1.AddStaticKeypad( txtKeypad.Text, int.Parse( txtKeypadID.Text ),
            chkOverwrite.Checked );
    }
    catch ( Exception E )
    {
           MessageBox.Show( E.Message, "Input Error" );
    }
}
```

RemoveKeypad – A method used to remove a keypad from the keypad list by serial number. The address of the serial number removed then becomes available to the next keypad requesting or added through the AddKeypad method.

```
private void btnRemove_Click(object sender, System.EventArgs e)
{
    axWRS970X1.RemoveKeypad( txtKeypad.Text );
}
```

RemoveStaticKeypad – A method used to remove a keypad from the keypad list by Keypad ID or address. It does not matter which addressing mode is active. For example, 10 dynamic keypads are in use and this method is called to remove Keypad ID 3. A new keypad comes in requesting an address from the system. The API looks for the first available ID and gives the new keypad ID 3.

```
private void btnRemoveStatic_Click(object sender, System.EventArgs e)
{
    axWRS970X1.RemoveStaticKeypad( int.Parse( txtKeypadID.Text ) );
}
```

ClearKeypadList – A method used to clear the entire keypad list. This method should be used with caution! Clearing the list while some keypads are still connected with an ID already could cause a duplicate ID to be assigned. The event OnDuplicateID then occurs (see below). A keypad that votes after the list is cleared, is added to the list at the ID it currently has. That ID is then used and will not be assigned to another keypad. Keypads that do have an ID and doesn't vote after the list is cleared, risks (in dynamic mode) that ID being assigned to a new keypad. Consider powering down the connected keypads before clearing the list when using the system in dynamic addressing mode. Clearing a keypad list causes all groups of keypads to be polled until the addition of a valid keypad. See AddStaticKeypad method for more description.

```
private void btnClearList_Click(object sender, System.EventArgs e)
{
    axWRS970X1.ClearKeypadList();
}
```

OnDuplicateID – An event triggered when a keypad votes at the address or Keypad ID of another keypad. This event occurs no matter which addressing mode is in use. This event will not trigger when a keypad serial number of zeroes is being added over a valid serial number or vice versa.

```
    private void axWRS970X1_OnDuplicateID(object sender,
AxWRS970.IWRS970XEvents_OnDuplicateIDEvent e)
    {
        debugMessage( "Duplicate ID: " + e.keypadID + " from " + e.serialNumber );
    }
```

3.4.2 AutoConnect

AutoConnect – Automatically connects a keypad to a base when there is only one system (single base or grouped bases) found. This property simplifies connecting for systems configured dynamically. Though the method does not effect the keypad list, the keypad list is used to determine what ID the keypad receives.

```
private void mnuKeypadsAutoConnect_Click(object sender, System.EventArgs e)
{
    axWRS970X1.AutoConnect = mnuKeypadsAutoConnect.Checked;
}
```

3.4.3 Login

KeypadLoginEnabled – A property managing the security of a WRS970 system. When a keypad link to the system is attempted, this property can require a login to be entered for the keypad to receive a Keypad ID. The login data is passed to the software via the event OnKeypadLogin. The API does not handle or compare the validity of the data. This allows more flexibility in the system configuration. Login enabled does not separate systems; the Base ID or channel identifies a system.

```
private void mnuBasesLoginEnabled_Click(object sender, System.EventArgs e)
{
    axWRS970X1.KeypadLoginEnabled = mnuBasesLoginEnabled.Checked;
}
```

SecureLogin – Keypads will display an asterisk instead of the login data on the LCD screen to the keypad user.

```
private void mnuKeypadsSecureLogin_Click(object sender, System.EventArgs e)
{
    axWRS970X1.SecureLogin = mnuKeypadsSecureLogin.Checked;
}
```

KeypadLoginResult – A method used to verify the login data sent by a keypad. This method is called after the OnKeypadLogin event.

```
private void axWRS970X1_OnKeypadLogin(object sender,
                                      AxWRS970.IWRS970XEvents_OnKeypadLoginEvent e)
{
    bool passed = false;

    try
    {
        if ( string.Compare( e.data, 0, txtLogin1.Text, 0, e.data.Length ) == 0 )
        {
            debugMessage( "Login matched!" );
            passed = true;
        }
    }
    catch( Exception tex )
    {
        debugMessage( "Ex: " + tex.Message );
    }
    axWRS970X1.KeypadLoginResult( e.serialNumber, passed );
}
```

LogoutAllKeypads – A method used to logout all keypads. A keypad must search for bases and select a base with which to connect. This method does not clear the keypad list.

```
private void mnuLogoutAll_Click(object sender, System.EventArgs e)
{
    axWRS970X1.LogoutAllKeypads();
}
```

LogoutKeypad – A method used to logout a specific keypad by its address. The address can be obtained by looking up the serial number in the keypad list. The keypad must be on a system with Login enabled. Calling this method does not remove the keypad from the Keypad List. Call the RemoveKeypad or RemoveStaticKeypad for this.

```
private void mnuLogoutSpecific_Click(object sender, System.EventArgs e)
{
    if ( txtSerialNumber.Text.Length > 0 )
    {
        try
        {
            axWRS970X1.LogoutKeypad( int.Parse( txtSerialNumber.Text ) );
        }
        catch ( Exception E )
        {
            MessageBox.Show( E.Message, "Input Error" );
        }
        else
            MessageBox.Show( "No Keypad ID given!", "Input Error" );
        }
    }
}
```

OnKeypadLogin – An event triggered when a keypad has requested to login to the system. Upon determining if the keypad data is valid, the KeypadLoginResult method should be called within this event. First, login can be used globally such that each keypad connecting to the system must enter the same login value. An alternate option is to compare the serial number of the login data against a table of specific login values per keypad serial number.

```
private void axWRS970X1_OnKeypadLogin(object sender,
                        AxWRS970.IWRS970XEvents_OnKeypadLoginEvent e)
{
    debugMessage( "Login Request: " + e.serialNumber + " - " + e.data );

    bool passed = false;

    try
    {
        if ( string.Compare( e.data, 0, txtLogin1.Text, 0, e.data.Length ) == 0 )
        {
            debugMessage( "Login 1 matched!" );
            passed = true;
        }
    }
    catch( Exception tex )
    {
        debugMessage( "Ex: " + tex.Message );
    }

    axWRS970X1.KeypadLoginResult( e.serialNumber, passed );
}
```

3.4.4 Authenticate

Authenticate – A property managing another level of security for access to the WRS970 system. If authenticate is set to true, only the keypads in the Keypad List are authorized to use the system. Keypads that vote and are not authorized will be reported under the KeypadNotAuthorized event. The event is most likely to occur when the system is configured to static addressing mode. In dynamic addressing mode, an unauthorized keypad will not be assigned an address and therefore cannot cast a vote.

```
private void mnuBasesAuth_Click(object sender, System.EventArgs e)
{
    axWRS970X1.Authentication = mnuBasesAuth.Checked;
}
```

OnKeypadNotAuthorized – An event triggered when the system is a static system, authentication is true, and a keypad outside of the keypad list attempts to vote.

```
private void axWRS970X1_OnKeypadNotAuthorized(object sender,
                                    AxWRS970.IWRS970XEvents_OnKeypadNotAuthorizedEvent e)
{
    debugMessage( "Keypad serial: " + e.serialNumber + " not authorized!" );
}
```

3.5 Keypad System Settings

Properties

MenuAccess
ForceMenu
BackLightOpMethod
KeypadBacklightOnTimer
KeypadBacklight
PingTimer
BeepKeypads
KeypadsOutOfRange
PowerDown
KeypadPowerTimer
HideBatteryIcon
HideAntennaIcon

Methods

none

Events

OnKeypadPing

Summary

This section describes options that will usually be set once prior to beginning a presentation. Menu access is included in this section though most of the time will not be needed.

3.5.1 Menu

MenuAccess – Limits the accessibility of the user to the keypad menu. If menu is disabled, the user will cannot access the keypad menu. If the menu is set to any of the other options, the keypad user can access all screens up to the limit.

```
private void setMenuAccess()
{
    if ( ShowResults )
        return;

    if ( mnuAccessDisabled.Checked )
        axWRS970X1.MenuAccess = WRS970.TMenuAccess.mtDisabled;

    if ( mnuAccessContrast.Checked )
        axWRS970X1.MenuAccess = WRS970.TMenuAccess.mtContrast;

    if ( mnuAccessBaseAddress.Checked )
        axWRS970X1.MenuAccess = WRS970.TMenuAccess.mtBaseAddress;

    if ( mnuAccessBaseID.Checked )
        axWRS970X1.MenuAccess = WRS970.TMenuAccess.mtBaseID;

    if ( mnuAccessKeyBeep.Checked )
        axWRS970X1.MenuAccess = WRS970.TMenuAccess.mtKeyBeep;

    if ( mnuAccessOpMode.Checked )
        axWRS970X1.MenuAccess = WRS970.TMenuAccess.mtOpMode;
}
```

ForceMenu – Forces the keypad into the menu option. The menu may be restricted by MenuAccess property. Setting this property while the menu access for the user is disabled, displays only the serial number and firmware revision screens of the menu. Clear this property to allow the user to exit the menu. If the MenuAccess property is set to anything but disabled, setting this property shows the current maximum menu limit on the keypad. See the Hardware User Manual for keypad menu operation.

```
private void mnuKeypadsForceMenu_Click(object sender, System.EventArgs e)
{
    mnuKeypadsForceMenu.Checked = !mnuKeypadsForceMenu.Checked;

    if ( ShowResults )
        return;

    axWRS970X1.ForceMenu = mnuKeypadsForceMenu.Checked;
}
```

3.5.2 Backlight

BackLightOpMethod – Sets the operation of the keypad backlight. opNormal keeps the backlight on as long as the KeypadBacklight property is set. The other modes rely on the KeypadBacklightOnTimer method below. See Hardware User Manual documentation for description of backlight operation.

Available Types: TOpMode
opNormal
opOnKeypress
opOnAcknowledged

```
private void setOpMethod()
{
    if ( mnuOpMethodNormal.Checked )
        axWRS970X1.KeypadBacklightOpMethod = WRS970.TOpMode.opNormal;
    if ( mnuOpMethodKeypress.Checked )
        axWRS970X1.KeypadBacklightOpMethod = WRS970.TOpMode.opOnKeypress;
    if ( mnuOpMethodAcknowledge.Checked )
        axWRS970X1.KeypadBacklightOpMethod = WRS970.TOpMode.opOnAcknowledged;

}
```

KeypadBacklightOnTimer – Sets the number of seconds the keypad backlight stays on for the specified operation method.

Available Types: TOnTimer
onTwoSeconds
onThreeSeconds
onFourSeconds
onFiveSeconds

```
private void setTimer()
{
    if ( mnuTimerTwoSecs.Checked )
        axWRS970X1.KeypadBacklightOnTimer = WRS970.TOnTimer.onTwoSeconds;
    if ( mnuTimerThreeSecs.Checked )
        axWRS970X1.KeypadBacklightOnTimer = WRS970.TOnTimer.onThreeSeconds;
    if ( mnuTimerFourSecs.Checked )
        axWRS970X1.KeypadBacklightOnTimer = WRS970.TOnTimer.onFourSeconds;
    if ( mnuTimerFiveSecs.Checked )
        axWRS970X1.KeypadBacklightOnTimer = WRS970.TOnTimer.onFiveSeconds;
}
```

KeypadBacklight – Turns the backlight operation on or off according to the above property settings.

```
private void mnuKeypadsBacklightOn_Click(object sender, System.EventArgs e)
{
    axWRS970X1.KeypadBacklight = mnuKeypadsBacklightOn.Checked;
}
```

3.5.3 Keypad Ping

PingTimer – The idle time in which a keypad will transmit a ping packet thereby triggers the OnKeypadPing event. Pings packets come in after the timeout period occurs with no voting activity. Event directors can know whether the keypad is not functioning or the user has decided not to participate. An alternate function of this feature is to keep the keypads on past the timeout period of 1 hour. If this property is set, the 1 hour starts over for each ping packet sent. The four available types are shown in the example below.

Available Types: TPingTimer
pngOff pngFive pngTen pngFifteen

```
private void setPingTimer()
{
    if ( ShowResults )
        return;
    if ( mnuPingTimerOff.Checked )
        axWRS970X1.PingTimer = WRS970.TPingTimer.pngOff;
    if ( mnuPingTimerFive.Checked )
        axWRS970X1.PingTimer = WRS970.TPingTimer.pngFive;
    if ( mnuPingTimerTen.Checked )
        axWRS970X1.PingTimer = WRS970.TPingTimer.PngTen;
    if ( mnuPingTimerFifteen.Checked )
        axWRS970X1.PingTimer = WRS970.TPingTimer.PngFifteen;
}
```

OnKeypadPing – An event triggered upon the response of keypads where the pingTimer time has expired.

```
private void axWRS970X1_OnKeypadPing(object sender,
                            AxWRS970.IWRS970XEvents_OnKeypadPingEvent e)
{
    debugMessage( "Ping! Serial: " + e.serialNumber + "  " + "ID: " +
                                e.keypadID.ToString() );
}
```

3.5.4 Audible Alert

BeepKeypads – When set, the keypads will continually beep until this property is set to false. This feature works independent of the one below.

```
private void mnuKeypadsBeepKeypads_Click(object sender, System.EventArgs e)
{
    axWRS970X1.BeepKeypads = mnuKeypadsBeepKeypads.Checked;
}
```

KeypadsOutOfRange – Keypad Return reminder that keypads beep when taken outside of the system RF area for an extended period of time (approximately 15 seconds of no RF). Receiving one packet resets the time and turns off the beep.

```
private void mnuKeypadsBeepKeypadsRanged_Click(object sender, System.EventArgs e)
{
    axWRS970X1.KeypadsOutOfRange = mnuKeypadsBeepKeypadsRanged.Checked;
}
```

3.5.5 Global Power Down

KeypadsPowerDown – Keypads will power down when this property is set to true. Most keypads will power down in less than a second so this feature doesn't need to remain set. If left set, a keypad connecting to the base unit powers down as soon as it is connected. For static keypads, the keypad will turn on and then right back off. Set the property to false to allow the keypads to connect to the base and stay powered up.

```
private void mnuKeypadsPowerDown_Click(object sender, System.EventArgs e)
{
    axWRS970X1.KeypadsPowerDown = mnuKeypadsPowerDown.Checked;
}
```

3.5.6 Global Power Timer

KeypadPowerTimer – Set this property to the desired time the keypads keep their settings when they are outside of the RF range of the base. This property can be set to determine how long a keypad stays "alive" without receiving RF. It's useful for keypads that leave the room, but need to keep their current settings upon return. The default setting is pwrTen. However the base does keep the prior setting and this property is populated on connection**.**

Available Types: TPowerTimer
pwrTen
pwrTwenty
pwrThirty
pwrForty

3.5.7 LCD Display Settings

HideBatteryIcon – Set this property to remove the battery icon from Plus 3.0 and greater keypads. In Mini+ keypads, this will prevent the low battery warning from blinking on the keypads. The battery level is still sent with the vote in both products.

```
private void mnuKeypadsHideBatteryIcon_Click(object sender, EventArgs e)
{
   axWRS970X1.HideBatteryIcon = mnuKeypadsHideBatteryIcon.Checked;
}
```

HideAntennaIcon – Set this property to remove the antenna icon from Plus 3.0 and greater keypads.

```
private void mnuHideAntennaIcon_Click(object sender, EventArgs e)
{
   axWRS970X1.HideAntennaIcon = mnuHideAntennaIcon.Checked;
}
```

3.6 Keypad Polling

Properties

AutomateQuestionNumber
KeyLockFeedback
KeypadAlertEnabled
PowerKeyDisabled
KeypadHideVote
ShowReceived
ShowSending
CustomMessage
ShowAnswer
CustomSoftKeys
MsgAckEnabled
SMSEntry
ScrollFaster
SMSVoteTimeout

Methods

BeginQuestion
BeginQuestionPrompt
BeginQuestionAnswer
LockKeys
UnlockKeys
SendMsgBySerial
SendMsgByID

Events

OnKeypadDataReceived (obsolete)
OnSoftKeyDataReceived
OnKeypadTimestamp
OnMomentToMoment
OnKeypadAlert
OnMessageAck
OnSMSVoteTimeout
OnKeypadData

3.6.1 Begin Question/Slide

BeginQuestion – A method used to format a question by just answer type. Using the properties described in this section, the system can be set up to obtain the desired information from the keypad users (lockout unused keys). Set the following properties in this section and then call this method to poll the users. It can also be used to prevent the user from entering data in cases where information is not being polled (lock out all keys). Some properties, though not listed here, could be configured on a per question/slide basis. One example is during a particular slide the lights are out and the users will be voting in the dark. For that slide only, the backlight could be enabled and then disabled at some time in the future. Or, the KeypadsOutOfRange property may be turned on only at the last couple slides of the presentation. Prompts, custom or built-in, and correct answer are a couple other options available. Question number can not exceed 255. The answer types are listed in the table below.

Available Types: TAnswerType
atSingleAlpha
atSingleDigit
atYesNo
atYesAbstainNo
atAgreeDisagree
atLowMedHigh
atTrueFalse
atMomentToMoment
atMultiAlphaNumeric
atNegativePlus
atCustomSoftKeys (see CustomSoftKeys property)

```
private void btnBeginQuestion_Click(object sender, System.EventArgs e)
{
    if ( cboLineOne.SelectedIndex > -1 )
        axWRS970X1.BeginQuestionPrompt( prompt, int.Parse( txtQuestionNumber.Text ),
                                        answerType, txtAnswer.Text );
    else
    {
        if ( cboAnswerType.SelectedIndex > -1 )
            axWRS970X1.BeginQuestionAnswer( int.Parse( txtQuestionNumber.Text ),
                                        answerType, txtAnswer.Text );
        else
            axWRS970X1.BeginQuestion( int.Parse( txtQuestionNumber.Text ), answerType );
    }
}
```

BeginQuestionPrompt – A method used to format a question by prompt, answer type, question number, and answer. See BeginQuestion example. The available built-in prompts are listed in the table below. These prompts do not effect the speed of the system where a custom prompts slows the system response very slightly.

Available Types: TLineOnePrompt
lopBlank
lopVoteNow
lopSystemIdle
lopThankYou
lopWelcome
lopChooseAgain
lopVoteEnded
lopBreakTime
lopQuestions
lopHello
lopPleaseVote
lopTimeIsUp
lopManyThanks
lopQuestionNumber
lopCustomPrompt
lopRespondNow
lopReturnKeypad

BeginQuestionAnswer – A method used to format a question by answer type, question number, and answer. See BeginQuestion example.

AutomateQuestionNumber – A property that will automatically increment the question number on each BeginQuestion. This is very convenient for SMS votes that may finish after the next BeginQuestion method is called.

```
private void chkAutomateQuestionNumber_CheckedChanged(object sender, EventArgs e)
{
   axWRS970X1.AutomateQuestionNumber = chkAutomateQuestionNumber.Checked;
}
```

3.6.2 Keypad Data Received

OnKeypadDataReceived – (Obsolete. Replaced by OnKeypadData event below). An event triggered by the incoming data of any authorized keypad on the WRS970 system. Please see example below for parameter details.

In Reply Plus 3.0 firmware and later SMS voting, question number is very important as you may get the previous questions data after a new BeginQuestion() method has been called. A long response from a keypad may take up to 40 seconds to reach the software, dependent upon the number of keypads active in the system; the fewer keypads active, the quicker the response of long keypad votes.

If you wish to only receive a SMS response to a question within the question time period, please allow plenty of time for the user to type their response and approximately 40 seconds after for the system to process it before calling another BeginQuestion() method. Once the data is received, please be sure the questionNumber field value matches that of the question number used in the beginQuestion() method call appropriately. The questionNumber field is only populated with Plus 3.0 and later keypads.

If you wish to proceed to the next question without giving time for the system to processs SMS responses, the responses that were sent will still come through, so you must be sure the response given in this event match that of the polling question that was asked. You will still receive the proper responses for the current question, however be aware that you may receive a response from the prior question too in SMS responses.

The keypadType field will differentiate between 3.0 and later keypads and others. A one returned for a keypad type signifies a 3.0 and later Plus keypad. A zero value signifies Plus 2.0 and Mini Plus keypad votes.

```
private void axWRS970X1_OnKeypadDataReceived(object sender,
                                axWRS970.IWRS970XEvents_OnKeypadDataReceivedEvent e)
{

    debugMessage( "Battery: " + e.batteryLevel.ToString(), false );
    debugMessage( "ID: " + e.keypadID.ToString(), false );
    debugMessage( "Serial: " + e.serialNumber, false );
    debugMessage( "TimeStamp: " + e.timeStamp.ToString(), false );
    debugMessage( "Value: " + e.value, false );
    debugMessage( "Version: " + e.version, false );
    //OnKeypadData event only
    debugMessage( "Keypad Type: " + e.keypadType.ToString(), false);
    debugMessage( "Question Number: " + e.questionNumber.ToString(), false);


    int row = 0;
    int col = 0;

    col = e.keypadID / 10;
    row = e.keypadID % 10;

    try
    {
        gridKeypadVotes.SetData( row, col, int.Parse(e.value) );
    }
    catch
    {
        gridKeypadVotes.SetData( row, col, e.value );
    }
}
```

OnKeypadData – See OnKeypadDataReceived. OnKeypadDataReceived does not contain parameters for Question Number and KeypadType; this event does.

OnSoftKeyDataReceived – An event triggered by any "Soft Key" data of any authorized keypad on the WRS970 system.

```
private void axWRS970X1_OnSoftKeyDataReceived(object sender,
                                    AxWRS970.IWRS970XEvents_OnSoftKeyDataReceivedEvent e)
{
    debugMessage( "SoftKey received: " + e.serialNumber + "  ID:" + e.keypadID.ToString()
        + "  Value: " + e.value );
}
```

OnKeypadTimestamp – An event triggered relating timestamp information. Calling the BeginQuestion method resets the timer for time stamping votes. The system supports up to 56 minutes of available timestamps before the counter rolls over. Users that vote prior to the new question slide will have a zero value for the timestamp value.

```
private void axWRS970X1_OnKeypadTimestamp(object sender,
                            AxWRS970.IWRS970XEvents_OnKeypadTimestampEvent e)
{
    debugMessage( "KeypadTimeStamp received: " + e.serialNumber );
    debugMessage( "                       ID: " + e.keypadID );
    debugMessage( "                     time: " + e.timeStamp );
}
```

SMSVoteTimeout – Property set to limit the time the keypad is allowed to continue to transmit up to 140 characters of a vote before triggering the OnSMSVoteTimeout event. The default is 40000 or 40 seconds.

```
private void mnuSetSMSVoteTimeout_Click(object sender, EventArgs e)
{
   axWRS970X1.SMSVoteTimeout = int.Parse(newTimeout.txtSerialNumber.Text);
}
```

OnSMSVoteTimeout – An event triggered when a Plus 3.0 keypad sending more than 12 characters does not respond in the SMSVoteTimeout allotted time. Parameters given on timeout are serial number and partial vote data.

```
private void axWRS970X1_OnSMSVoteTimeout(object sender,
AxWRS970.IWRS970XEvents_OnSMSVoteTimeoutEvent e)
{
   lsvDebug.Items.Add(e.serialNumber + "- timed out!  Question Number: " +
   e.questionNumber.ToString() + "   Vote Data: " + e.vote);
}
```

OnMomentToMoment – An event triggered when a keypad sends its moment to moment data.

```
private void axWRS970X1_OnMomentToMoment(object sender,
                                        AxWRS970.IWRS970XEvents_OnMomentToMomentEvent e)
{
    if ( MomentChart.Stop )
    {
        cboAnswerType.SelectedIndex = 0;
        btnBeginQuestion_Click( sender, null );
    }

    string[] groupSplit = e.data.Split( ';' );
    int timeStamp = 0;
    int value = 0;
    decimal dtime = 0;

    if ( Serials[e.keypadID] != e.serialNumber )
    {
        Serials[e.keypadID] = e.serialNumber;
        DS[e.keypadID] =
               MomentChart.c1Chart1.ChartGroups[0].ChartData.SeriesList.AddNewSeries();

        DS[e.keypadID].SymbolStyle.Shape = C1.Win.C1Chart.SymbolShapeEnum.None;
        DS[e.keypadID].LineStyle.Thickness = 2;
    }

    for (int i = 0; i < groupSplit.Length; i++)
    {
        string[] timeSplit = groupSplit[i].Split();
        timeStamp = int.Parse( timeSplit[0] );
        value = int.Parse( timeSplit[1] );
        dtime = (decimal)timeStamp / 1000;

        if ((dtime > 40)&&(( dtime + 15 ) >
                      (decimal)MomentChart.c1Chart1.ChartArea.AxisX.Max))
        {
            MomentChart.c1Chart1.ChartArea.AxisX.Max += 15;
            MomentChart.c1Chart1.ChartArea.AxisX.Min += 15;
        }

        DS[e.keypadID].Label = e.serialNumber;
        DS[e.keypadID].X.Add( dtime );
        DS[e.keypadID].Y.Add( value );
    }
}
```

3.6.3 Lockout Keys

KeyLockFeedback (Boolean) – Keypads that have locked keys will report that those keys are locked to the LCD screen.

```
private void mnuKeypadsLockedFeedback_Click(object sender, System.EventArgs e)
{
    axWRS970X1.KeyLockFeedback = mnuKeypadsLockedFeedback.Checked;
}
```

KeypadAlertEnabled – A property to enable or disable the keypad alert key on the BeginQuestion method call. This property will override the LockKeys() method call for the “Alert” key.

```
private void mnuKeypadsAlertEnabled_Click(object sender, System.EventArgs e)
{
    axWRS970X1.KeypadAlertEnabled = mnuKeypadsAlertEnabled.Checked;
}
```

OnKeypadAlert – An event triggered when the alert key is pressed on the keypad. This event is only triggered if the KeypadAlertEnabled property is set to true.

```
private void axWRS970X1_OnKeypadAlert(object sender,
                                      AxWRS970.IWRS970XEvents_OnKeypadAlertEvent e)
{
    debugMessage( "Alert pressed from " + e.serialNumger );
}
```

PowerKeyDisabled – Disables the power key, not allowing a power down.

```
private void mnuKeypadPowerKey_Click(object sender, System.EventArgs e)
{
    axWRS970X1.PowerKeyDisabled = mnuKeypadPowerKey.Checked;
}
```

LockKeys – A method used to lock out keys from the keypad. Parameter is a comma delimited string of the keys that are to be locked. The "soft keys" are denoted by "Left", "Middle", and "Right" as depicted in the example. "SYM" is the symbol key and "Scan" and "Link" are for the "Search for Available Bases" key. The "All" parameter locks all keys. This method is overridden with the KeypadAlertEnabled flag for the alert key. However, it may still be locked when KeypadAlertEnabled is false.

```
private void btnLock_Click(object sender, System.EventArgs e)
{
    string lockKeys = null;

    lockKeys = "1,3,5,0,Left,Middle,Right,Sym,Scan";
    axWRS970X1.LockKeys( lockKeys );
}
```

UnlockKeys – A method used to unlock keys that were locked out via the LockKeys method. See LockKeys method for parameter explanation.

```
private void btnUnlock_Click(object sender, System.EventArgs e)
{
    string unlockKeys = null;

    unlockKeys = "1,3,5,0,Left,Middle,Right,Sym,Scan";
    axWRS970X1.UnlockKeys( unlockKeys );
}
```

3.6.4 Key Display

KeypadHideVote – Keypad votes will be concealed on the LCD screen.

```
private void mnuKeypadsHideVote_Click(object sender, System.EventArgs e)
{
    axWRS970X1.KeypadHideVote = mnuKeypadsHideVote.Checked;
}
```

ShowReceived – Keypads will show "Received" on the acknowledgement from the base after keypad transmission.

```
private void mnuKeypadsShowReceived_Click(object sender, System.EventArgs e)
{
    axWRS970X1.ShowReceived = mnuKeypadsShowReceived.Checked;
}
```

ShowSending – Keypads will show "Sending" while it waits for the base to acknowledge its transmission.

```
private void mnuKeypadsShowSending_Click(object sender, System.EventArgs e)
{
    axWRS970X1.ShowSending = mnuKeypadsShowSending.Checked;
}
```

3.6.5 Messaging

CustomMessage – Set or clear to display or not display a user defined message on the keypad when a BeginQuestionPrompt method is called. A table of available character that the WRS7200 LCD supports is located in Appendix A. Only a very small subset of the ANSI ACSII table corresponds to the custom table. Since many of the characters are there, a conversion lookup table could be created in software to change the character behind the scenes for the keypad to display the intended message properly. The @ symbol can be sent as @ and will display on keypads 3.0 and later.

Since this property is has a small impact on the polling speed, setting the property to 0 will cause the base unit to no longer broadcast the custom message packets.

```
private void btnCustomMessage_Click(object sender, System.EventArgs e)
{
    axWRS970X1.CustomMessage = txtCustomMessage.Text;
    axWRS970X1.BeginQuestionPrompt( lopCustomPrompt, int.Parse( txtQuestionNumber.Text ),
        answerType, txtAnswer.Text );
}
```

SendMsgBySerial – A method used to send a specific keypad, by serial number, a message. This method may be called multiple times to populate a group of keypads for individual messaging. Each keypad loaded into the API will default to a specific timeout period and if it's timed out will trigger the OnKeypadMsgTimeout event. When a keypad has timed out, it has not received the message. If a timeout occurs you may send the message again using this method. If MsgAckEnabled property is set to true and the keypad responds within the timeout period, the OnMessageAck event will trigger indicating the keypad's success. In a 500 keypad system with 500 different messages, every keypad should have responded within 10 seconds for messages 12 characters or less. Starting with version 3.0 and later, keypads can accept messages up to 140 characters long. These longer messages will take significantly longer to transmit and therefore require a longer timeout. Please use caution when sending multiple messages. The messages are queued in the API and are sent when the system is ready. If a subsequent call to this method is sent before the system has sent the message, the original message may not reach the keypad. It is recommended that you use the keypad's acknowledgment feature if you intend to display two different messages within a fifteen second period.

```
axWRS970X1.SendMsgBySerial( myMessage.txtSerialID.Text, myMessage.txtMessage.Text );
```

SendMsgByID – A method used to send a specific keypad, by keypad ID, a message. Note: See SendMsgBySerial.

```
axWRS970X1.SendMsgByID( int.Parse( myMessage.txtSerialID.Text ),
    myMessage.txtMessage.Text );
```

OnMessageAck – An event triggered when a keypad has shown its keypad specific message on the LCD. This event is triggered only on the call of the SendMsgByID or SendMsgBySerial methods and returns serial number and keypad ID of a responding keypad. The MessageAckEnabled property must be set to true to receive this event.

```
private void axWRS970X1_OnMessageAck(object sender,
AxWRS970.IWRS970XEvents_OnMessageAckEvent e)
{
    debugMessage( "Msg Ack! Serial: " + e.serialNumber + "  " + "ID: " +
    e.keypadID.ToString(), true );
}
```

OnKeypadMsgTimeout – An event triggered when a keypad has not responded from SendMsgByID or SendMsgBySerial method calls within the timeout period.

```
private void axWRS970X1_OnKeypadMsgTimeout(object sender,
AxWRS970.IWRS970XEvents_OnKeypadMsgTimeoutEvent e)
{
    lsvDebug.Items.Add( "Keypad: " + e.serialNumber + " ID: " + e.keypadID.ToString() + "
    individual message timed out!" );
}
```

MsgAckEnabled – Set to enable the OnMessageAck event. Use this property to turn keypad messaging on and off. See SendMsgBySerial and OnMessageAck.

```
private void chkMsgAckEnabled_CheckedChanged(object sender, System.EventArgs e)
{
    axWRS970X1.MsgAckEnabled = chkMsgAckEnabled.Checked;
}
```

ScrollFaster – Controls the scrolling speed of individual and global messages on Reply Plus 3.0 and greater keypads.

```
private void mnuKeypadsScrollFaster_Click(object sender, EventArgs e)
{
        axWRS970X1.ScrollFaster = mnuKeypadsScrollFaster.Checked;
}
```

3.6.6 Answers

ShowAnswer – Set to display the correct answer on the first line of the display, the answer given on the second line of the display and an "X" for incorrect or a check mark for correct. This feature does not work with Moment to moment question types.

```
private void btnShowAnswer_Click(object sender, System.EventArgs e)
{
    axWRS970X1.ShowAnswer = true;}
```

3.6.7 Keys

CustomSoftKeys – Set to display custom text above the top three keys, or soft keys, of keypads. You must set or clear this property before calling the method BeginQuestion with an answer type of atCustomSoftKeys. Be certain to lockout softkeys that you do not set text to.

```
frmSoftKey newSoftKey = new frmSoftKey();

if ( newSoftKey.ShowDialog() == DialogResult.OK )
{
    axWRS970X1.CustomSoftKeys = newSoftKey.txtSoftKeyText.Text;
}
```

SMSEntry – Set true to allow SMS entry from the keypad. You must have the answer type set to MultiAlphaNumeric to set this property. This property is implemented in keypads and bases with firmware 3.0 or greater.

```
axWRS970X1.SMSEntry = true;
```

3.7 API Properties

Properties

ControlVersion
DebugEnabled
ErrorsEnabled
ExceptionsEnabled
LowBatteryNotification
Version3

Methods

ClearDataBuffer

Events

OnDebug
OnError
OnLowBatteryWarning

ControlVersion – Reports the current version of the API being used. This is a read only property.

```
private void WinForm_Load(object sender, System.EventArgs e)
{
    txtVersion.Text = "ControlVersion: " + axWRS970X1.ControlVersion;
}
```

3.7.1 API Debug

DebugEnabled – A property that will allow the trigger of the OnDebug event. Use this property when the system appears to be in error only. There are many debug messages that are displayed as a result of setting this property to true.

```
private void mnuSystemDebugEnabled_Click(object sender, System.EventArgs e)
{
    axWRS970X1.DebugEnabled = mnuSystemDebugEnabled.Checked;
}
```

OnDebug – An event triggered to provide helpful debug messages describing the details of processing of the API. The event is only triggered if the DebugEventsEnabled property is set to true.

```
private void axWRS970X1_OnDebug(object sender, AxWRS970.IWRS970XEvents_OnDebugEvent e)
{
    if ( chkMessageBox.Checked )
        MessageBox.Show( e.data );
    else
        debugMessage( e.data );
}
```

3.7.2 API Errors

ErrorsEnabled – A property that will allow the trigger of the OnError event. The property should in most cases be set to true. If any errors happen in the API, this event will trigger.

```
private void mnuSystemErrorsEnabled_Click(object sender, System.EventArgs e)
{
    axWRS970X1.ErrorsEnabled = mnuSystemErrorEnabled.Checked;
}
```

ExceptionsEnabled – A property that will create an exception on any Error in the API. This can be used to control the errors from the application vs. having the API control error processing.

```
private void mnuSystemExceptionsEnabled_Click(object sender, System.EventArgs e)
{
    axWRS970X1. ExceptionsEnabled = mnuSystemExceptionsEnabled.Checked;
}
```

OnError – An event triggered to provide errors that may occur in the API. The event is only triggered if the ErrorEventsEnabled property is set to true.

```
private void axWRS970X1_OnError(object sender, AxWRS970.IWRS970XEvents_OnErrorEvent e)
{
    lsvDebug.Items.Add( "Error: " + e.data );
}
```

3.7.3 Miscellaneous

KeypadLowBatteryNotification – A property that will enable or disable notification on a keypad low battery event.

```
private void mnuLowBatteryNofication_Click(object sender, System.EventArgs e)
{
    axWRS970X1. KeypadLowBatteryNotification = mnuLowBatteryNofication.Checked;
}
```

Version3 – A property for new developers to set. If this property is set false, which is the default, the API uses the OnKeypadDataReceived event. If set true, which should be done for all new development, the OnKeypadData event is fired for keypad votes.

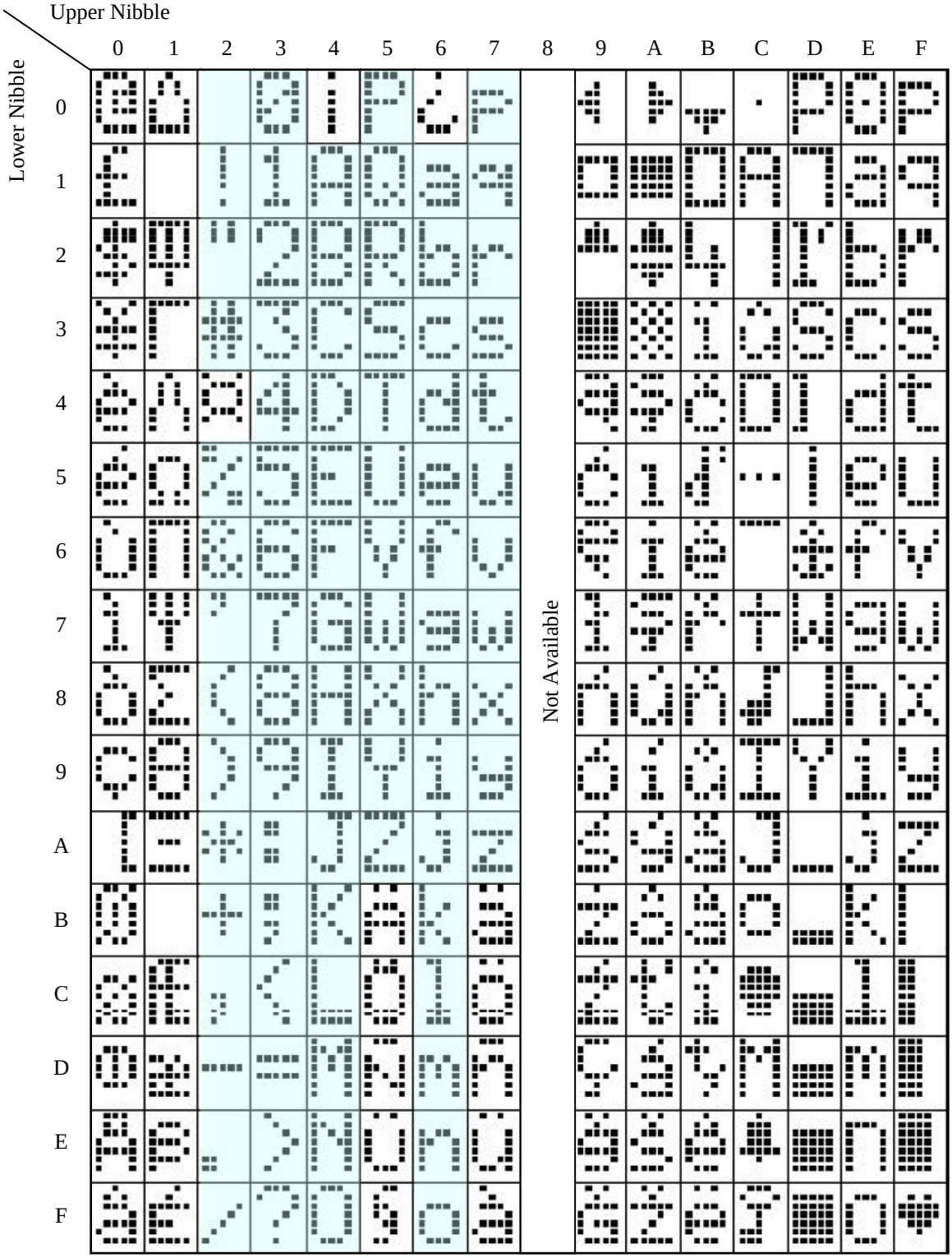
OnLowBatteryWarning – An event triggered by a keypad with a low battery.

```
private void axWRS970X1_OnLowBatteryWarning(object sender,
                            AxWRS970.IWRS970XEvents_OnLowBatteryWarningEvent e)
{
    debugMessage( "Low battery warning for " + e.serialNumber );
}
```

ClearDataBuffer – A method used to clear the messages from the API data buffer.

```
private void btnClearDataBuffer_Click(object sender, System.EventArgs e)
{
    axWRS970X1.ClearDataBuffer();
}
```

APPENDIX A – LCD Character Table



The table presents each nibble of the hex value. For example, to display a 2, an upper nibble of 3 and lower nibble of 2 = 32**h** = 50**d**.

Highlighted Matches ASCII table.