Registry Key Assignments

Dated August 30, 2011
(JNIOR OS v4.1.830 and earlier)

INTEG process group, inc.
2919 E. Hardies Rd, First Floor
Gibsonia, PA 15044

www.integpg.com

JNIORsales@integpg.com

PH (724) 933-9350
FAX (724) 443-3553

Contents

1. About The Registry

The JNIOR Registry is a memory based structured system providing for the storage of configuration related information. The content is available to the JNIOR operating system and through protocols to external applications. The Registry may be used to store application specific user information as well as pre-defined operating system related content.

During boot JNIOR initializes (loads) the Registry from the /flash/jnior.ini file. During operation JNIOR maintains synchronization between the Registry and this file. If the file is updated remotely (changed through FTP for instance) the Registry will be reloaded from with new content. If the Registry is changed by the operating system, using the `registry` command or through one of the protocols, the INI file will be updated to reflect the changes. If the file is accidentally deleted, JNIOR will regenerate it from the current Registry. All of these actions are logged to the /jniorsys.log file providing an audit trail for configuration management.

2. Browsing and Editing

A Registry Editor is available for use in an interactive mode. The editor is started using the `registry` command entered at the command prompt. The command prompt is available for example through a Telnet (or serial0) connection. This `registry` command is available only to those logged in as administrator.

The Registry has a structure much like a file system. There are no “files” per say but there are *keys* at various levels each containing one or more *values*. Analogous to a file path, the “Registry Key” specifically defines a Registry entry and the ‘/’ separator breaks the Registry down into *sections* (like folders).

The Registry editor displays one section at a time and provides for options to navigate through the Registry. In a Telnet session (or through a direct serial connection) log in as an administrator. Use the command “registry” to enter the editor. Nothing else on the command line is required although the “help” option is available. User keystrokes are shown in **bold characters**.

```
JNIOR /> help registry
registry

Display and Modify JNIOR Registry settings.
Command is available only to administrators.

Command line syntax:  registry [keyname [= [newcontent]]]

Valid at prompt:
   ##  - number from menu displays item, <previous> moves up
  ".." - moves up, same as <previous>
  "."  - redisplays current section
  "*"  - selects all entries for removal
  "<section>/.." - opens that section
  Type any key to Add/Edit that key (relative to current section)
  Type key prefixed by '/' to specify it from the Root
  Immediate blank line Exits the editor
```

When you start the Registry Editor the very “top” of the Registry or the *root* is displayed. You will see that the root contains the valid System Parameters as well as paths to one or more *sections*.

```
JNIOR /> registry
JNIOR Registry Editor
Copyright (C) 2005 INTEG process group, inc.  All Rights Reserved.
See Help for more information.


Content of /..
   1  $BootTime = Mon Apr 11 13:28:53 GMT 2005
   2  $SerialNumber = 4904004
   3  $Version = 2.01.219
   4  Device/..
   5  Email/..
   6  Events/..
   7  IO/..
   8  IpConfig/..
   9  JniorServer/..
  10  ModbusServer/..
  11  <exit>

Key (or ## selection) to Add/Edit/Remove?
```

At this prompt you may either enter the Registry Key string or a selection by number from the displayed menu. One feature of the <u>very first</u> prompt after the display of the menu is that a blank line will exit the Registry Editor and return you to the command prompt. You can always get out of the Registry Editor by hitting repeated Enter keystrokes. At the <Root> menu you might also use selection number 11 (in this particular Registry) to exit. The various Registry sections are shown listed by there initial *node* followed by "/.." to indicate that there is more below. Enter a numeric selection to descend into any section.

```
Key (or ## selection) to Add/Edit/Remove? 6

Content of Events/..
   1  OnAlarm = enabled
   2  OnBoot = enabled
   3  <previous>

Key (or ## selection) to Add/Edit/Remove?
```

Here we have elected by entering #6 to view the `Events` section. In this particular JNIOR we see that the Power-On Reset and Alarm event features have been enabled (see *Configuration Examples*). We can see that there is the option shown presently as #3 to return to the previous menu and effectively moving you back up a level in the structure. Entering the number 3 at this point would in fact redisplay the Root menu as previous shown.

Say that we wish to disable the Power-On Reset (or Boot) Notification. We can simply select that Registry key and change its value. Here's how that would go:

```
Key (or ## selection) to Add/Edit/Remove? 2
  Events/OnBoot = enabled
[C]hange, [E]dit, [R]emove? c
New Value(s)? disabled
  Events/OnBoot = disabled
Accept [Y/N]? y

Content of Events/..
   1  OnAlarm = enabled
   2  OnBoot = disabled
   3  <previous>

Key (or ## selection) to Add/Edit/Remove?
```

Now in the background JNIOR is quietly updating the jnior.ini file to reflect this change. When JNIOR is rebooted (using the `reboot` command for instance) it will come up and load the Registry from the jnior.ini file. When JNIOR gets to the point where it must decide whether or not to send the Boot Notification it will see that it is not desired. No notification will be sent.

Note that the responses to the questions are case-insensitive. The lower case ‘c’ is as good as ‘C’ to elect to [C]hange a value. The [E]dit option differs only in that it displays the original content first allowing you to backspace and change it.

Since by default the Power-On Reset or Boot Notification is disabled, this Registry key being actually set to “disabled” is not really needed. We could actually have simply deleted it and achieved the goal of stopping the email notification. This removal is easily done:

```
Key (or ## selection) to Add/Edit/Remove? 2
  Events/OnBoot = disabled
[C]hange, [E]dit, [R]emove? r
Delete Key [Y/N]? y

Content of Events/..
   1  OnAlarm = enabled
   2  <previous>

Key (or ## selection) to Add/Edit/Remove?
```

The key has been removed from the Registry. Now if we really do want these notifications we can redefine the key. To do so we enter the desired key name “relative” to the current section. We only have to enter the key name at this point and not the entire string. Note that Registry keys are case-sensitive.

```
Key (or ## selection) to Add/Edit/Remove? OnBoot
  Events/OnBoot = <undefined>
Add Key [Y/N]? y
New Value(s)? enabled
  Events/OnBoot = enabled
Accept [Y/N]? y

Content of Events/..
   1  OnAlarm = enabled
   2  OnBoot = enabled
 3  <previous>

Key (or ## selection) to Add/Edit/Remove?
```

In this fashion we can also define new keys within new substructure. This example shows the creation of a key in a different section of the Registry below the currently displayed section.

```
Key (or ## selection) to Add/Edit/Remove? OnBoot/Email
  Events/OnBoot/Email = <undefined>
Add Key [Y/N]? y
New Value(s)? enabled, AdminEmail
  Events/OnBoot/Email = enabled, AdminEmail
Accept [Y/N]? y

Content of Events/OnBoot/..
   1  Email = enabled, AdminEmail
   2  <previous>

Key (or ## selection) to Add/Edit/Remove? ..

Content of Events/..
   1  OnAlarm = enabled
   2  OnBoot = enabled
   3  OnBoot/..
   4  <previous>

Key (or ## selection) to Add/Edit/Remove?
```

So in this case we had wanted to create the Registry key `Events/OnBoot/Email` which can be used to reference a unique email definition that we will arbitrarily call "AdminEmail". Since we were already displaying the `Events` section we merely need to enter the balance of the key to create it.

Once the new structure has been defined the Registry Editor will display the new section. We see here something new that we can do at the prompt. If you enter two periods ".." the Registry editor will move up to the previous level. In the example above we would have achieved the same result if we have selected the <previous> option. Similarly you can use a single period "." to redisplay the current section. These have been defined analogous to their use in file system directory navigation.

One other special character is available and it provides the ability to remove an entire Registry section including all of the subsections contained therein. The “*” asterisk command is valid in any section except the root. It selects all of the keys and subsections listed. It then confirms your intention to delete them. An affirmative response will remove the entire section of the Registry. Obviously this is very powerful and care should be exercised in its use. It is very helpful in cleaning house.

Look at the current content of the `Events` section. One thing to note is that a key name (`OnBoot` in this case) can specify both a value and further substructure.

One further example demonstrates the removal of a Registry section. To remove an entire section you must separately delete each key within the section. If there is a lot of this to do then perhaps external editing of the jnior.ini file would be more expeditious. The following removes the section just entered. Note that the Registry Editor removes the whole section and displays the first level above that with remaining content.

```
Key (or ## selection) to Add/Edit/Remove? 3

Content of Events/OnBoot/..
   1  Email = enabled, AdminEmail
   2  <previous>

Key (or ## selection) to Add/Edit/Remove? 1
  Events/OnBoot/Email = enabled, AdminEmail
[C]hange, [E]dit, [R]emove? r
Delete Key [Y/N]? y

Content of Events/..
   1  OnAlarm = enabled
   2  OnBoot = enabled
   3  <previous>

Key (or ## selection) to Add/Edit/Remove?
```

Finally it is worth noting that we can specify a new key relative to the Root no matter where we are in the Registry structure. This is done by prefixing the key with an additional ‘/’ character. While the leading ‘/’ is not normally used in referencing Registry keys it is appropriate here to insure that the entry is attached to the correct part of the structure.

```
Key (or ## selection) to Add/Edit/Remove? /Email/AdminEmail/ToAddress
  Email/AdminEmail/ToAddress = <undefined>
Add Key [Y/N]? y
New Value(s)? jdoe@integpg.com
  Email/AdminEmail/ToAddress = jdoe@integpg.com
Accept [Y/N]? n

Key (or ## selection) to Add/Edit/Remove? .

Content of Events/..
   1  OnAlarm = enabled
   2  OnBoot = enabled
   3  <previous>

Key (or ## selection) to Add/Edit/Remove?
```

And in this case we changed our mind. Had the key been created the Registry Editor would have displayed the related section. Instead we declined to “Accept” the new key and so we were prompted for additional action. The example used the single period to redisplay the original section.

Navigation about the Registry can be handled by moving down through the levels to different sections by selecting them from the content menu and moving back up using the <previous> selection or the “..” entry. You may also enter the desired section as if you were entering a key by using a trailing pair of periods. The Registry Editor will move to the section nearest the key or section that you have entered. For example:

```
Content of JniorServer/..
   1  Login = disabled
   2  Port = 9200
   3  <previous>

Key (or ## selection) to Add/Edit/Remove? /IO/Inputs/din1..

Content of IO/Inputs/din1/..
   1  Alarm1/..
   2  Count/..
   3  CountDisplay = enabled
   4  Desc = Generator
   5  <previous>

Key (or ## selection) to Add/Edit/Remove? Count/..

Content of IO/Inputs/din1/Count/..
   1  Alarm1 = enabled
   2  Alarm2 = enabled
   3  Limit1 = 5
   4  Limit2 = 10
   5  <previous>

Key (or ## selection) to Add/Edit/Remove? /Events/OnBoot/Email..

Content of Events/OnBoot/..
   1  Email = enabled, Boot
   2  <previous>

Key (or ## selection) to Add/Edit/Remove? /IO/Inputs/din12/WhoKnows..

Content of IO/Inputs/..
   1  din1/..
   2  din2/..
   3  <previous>

Key (or ## selection) to Add/Edit/Remove?
```

As can be seen from the last entry you do not have to specify a valid key or section. The Registry Editor will move to the nearest valid section.

The Registry Editor is best learned by doing. It is intended to provide access to the JNIOR's configuration through a minimal connection that does not support cursor movement, screen formatting, or even a mouse.

3. Command Line Usage

You may use the `registry` command at the command line prompt to view, change and remove individual Registry Keys. The general syntax is as follows where the `[]` indicate optional parameters.

```
registry [keyname [= [newcontent]]]
```

The `registry` command alone enters the interactive editing session as previously described. If the command is to be used on the command line to view, change or remove any Registry Key a valid `keyname` parameter must be specified. In order to change or remove a Registry Key the ‘`=`’ must be supplied.

3.1. Querying a Registry Key

The following command retrieves a registry key value if one is defined.

```
JNIOR /> registry Device/Test
Device/Test is not defined.

JNIOR /> registry JniorServer/Port
JniorServer/Port = 9200

JNIOR /> registry $Version
$Version = 2.12.121
```

3.2. Setting/Changing a Registry Key

A new value either defining a new Registry Key or changing the value of an existing Registry Key may be supplied at the command line. If the new value is to contain spaces you must enclose the value with double quotes. The following are some examples:

```
JNIOR /> registry Device/Test = TestingNewValue
JNIOR /> registry Device/Test
Device/Test = TestingNewValue

JNIOR /> registry Device/Test = "Testing New Value"
JNIOR /> registry Device/Test
Device/Test = Testing New Value

JNIOR /> registry Device/Test = "Parameter1, Parameter2"
JNIOR /> registry Device/Test
Device/Test = Parameter1, Parameter2
```

3.3. Removing a Registry Key

Finally you may remove a Registry Key by omitting any new value. The following demonstrates the usage:

```
JNIOR /> registry Device/Test =
JNIOR /> registry Device/Test
Device/Test is not defined.
```

Note that Registry Keys are case-sensitive and care must be used to correctly enter the key. A registry Key that may be correctly spelled but that does not contain the exact case for each character will not be recognized properly. These command line shortcuts are for those advanced users who are very familiar with the proper Registry Key formatting.

4. INI File Format

The Registry is synchronized with the /flash/jnior.ini file. If JNIOR finds the /jnior.ini file in the main file system (under the root instead of in /flash) it will use that copy be default. Otherwise JNIOR expects to find the INI file in the flash file area.

The INI file format is standard. It uses “section” headings enclosed by a pair of ‘[‘ and ‘]’ brackets. Each section can contain any number of keys assigned values following the ‘=’ sign. A key may have any number of values each being listed after the ‘=’ and separated by a ‘,’ comma.

Comments may appear in the INI file. Any text on any line following the ‘;’ semicolon character is considered to be commentary (including the semicolon). In general, leading and trailing white space (one or more spaces or tabs) is ignored.

Any value that is intended to contain a ‘,’ comma or ‘;’ may be enclosed in “quotes”. Thus the key defined here

```
Desc = “Relay, Main”
```

provides a description having just one value of *Relay, Main*. A value that is intended to contain a ‘”’ double-quote character may be “escaped” by preceding it with a ‘\’ back-slash.

4.1. Boolean Keys

Many keys are used to enable or disable some function. These have a logical on/off kind of purpose. JNIOR writes these values as “enabled” meaning on or “disabled” meaning off. If the INI file is edited externally the following content will all result in an enabled key:

```
enable
enabled
True
1
on
yes
```

Basically any content starting with the characters “enable” will qualify. All other content will be considered “Off” or disabled. These Boolean keys are case-insensitive.

5. Registry Key Syntax

A Registry *key* is a string specifying specific content much like a file specification. The key details a path to the configuration *value* within a desired Registry *section*. This is best illustrated by example. The following is a valid Registry Key:

```
IO/Inputs/din1/Desc
```

Again in a very analogous fashion to a file specification the part of the above key `IO/Inputs/din1` specifies a specific Registry section. In this case it is the section that configures Digital Input number 1. Similar to a file name `Desc` specifies a particular configuration setting or value defined in that section. In this case it is the description to be used with the particular input.

In the registry editor this key might appear as follows:

```
IO/Inputs/din1/Desc = Generator
```

This clearly defines the content for this key as the word "Generator". Perhaps the JNIOR with such a configuration setting provides the status of a nearby generator through digital input 1. All Registry Values are themselves character strings. These strings are also used to represent Boolean conditions and numeric values at times. In those situations words like "enabled" and "disabled" could mean Boolean conditions True and False respectively or the string "0.0" might indicate the numeric value zero as might be appropriate for a configuration setting when a fractional value can be specified.

The corresponding entry in the INI file for the above Registry Key would appear as follows:

```
IO/Inputs/din1/Desc = Generator
```

In the Registry individual values are not always a single string as shown above but are in actuality arrays of strings. Multiple strings may be specified separated by commas. For example:

```
Email/ToAddress = jdoe@integpg.com, admin@integpg.com
```

This is a typical Registry key specifying a short list of email address to be used during Event Notification. There are actually two separate strings provided. The first being "jodoe@integpg.com" and the second "admin@integpg.com". JNIOR in this case knows how to handle a list of address and will process the notification Emails appropriately sending one to each or the listed recipients.

An individual string in a Registry value may be optionally enclosed in double quotes. In particular this is important if the string is intended to include a comma character (or perhaps a semicolon) where it is important that JNIOR not be confused into thinking that separate strings are being listed (or an INI comment included). For instance a valid and somewhat formal output description might appear as:

```
IO/Outputs/rout1/Desc = "Relay, Main"
```

And this would appear in the jnior.ini file like this:

```
IO/Outputs/rout1/Desc = "Relay, Main"
```

One can see that there is a wide variety of ways in which to include detailed configuration information in this type of Registry system. This document describes those Registry Keys that are specifically used by JNIOR's operating system and supplied applets. The Registry can be easily used to store unique and custom application specific configuration as well. Those details should be available to you in the application related documentation.

In the INI file every Registry value is located within a section (specified surrounded by the square brackets). Therefore it would appear that in every case when a Registry key is viewed through the registry editor (or specified through one of the protocols) it would have to include at least one '/' separator as follows in this generic definition:

```
section/value = stringarray
```

This is for the most part true although there is a class of System Parameters which may be read from the Registry but that do not appear in the INI file. These are detailed in a following section of this document. These *System Parameters* have names that always begin with a '$' character are not contained within a specific section. System Parameters are intended to convey system related information such as the firmware version.

In fact, the '$' character may be used as a prefix with a Registry key name anywhere within the structure (any section). This indicates a *Dynamic Value* which will not appear in the INI file. One example of this occurs with the use of external sensors on the one-wire port. For example, a temperature sensor would report its value as follows:

```
OneWire/9600080010C84D10/$Value = 73.4, 23.0
```

This provides the current temperature of 73.4 degrees Fahrenheit or 23.0 degrees Celsius from the device on the Sensor Port whose address is "9600080010C84D10". This `$Value` Registry entry does not appear in the INI file and it does change dynamically as in this case it would track the temperature in the area of the sensor (updated about 4 times a second).

By the way, the `OneWire/Devices` Registry entry lists the addresses of all active sensors found on the Sensor Port. An application would query that key first to locate a specific device and using that information it may then read (and subscribe to) the related `$Value` key. If a subscription is used, the application would be notified whenever the temperature value changed.

6. Configuration Examples

6.1. Simple Email Notification on Power-On Reset

When a JNIOR is used in an application requiring continuous 24/7 operation instances of power loss or other interruption may be of some concern. You might want to be alerted when such events occur. The JNIOR may be easily configured to provide a notification by email when this happens. This feature can be enabled through the Registry.

```
Events/OnBoot/Email = enabled
Email/ToAddress = jdoe@integpg.com
```

Using the Registry Editor make sure that the above two keys are defined. The first enables the feature and the second specifies your email address. If the first key is absent then the feature is disabled by default. If the second key is not defined JNIOR might not know where to send the message.

You must also make sure that the IP configuration of the JNIOR is correct. In particular JNIOR must know how to find the MailHost. Use the `ipconfig` command or set the appropriate Registry keys in the IpConfig section. If the Registry is edited you will have to reboot the JNIOR in order for the settings to be applied.

Note also that some email servers may accept email only from valid members of the domain. The server may not recognize the default FromAddress used by the JNIOR and reject the mail posting. This error will be logged to the jniorsys.log file. To work around this you can supply a valid From address. For instance the following will appear as though you are sending a message to yourself.

```
Email/FromAddress = jdoe@integpg.com
```

With the IP configuration and the above keys set, a very simple email will be transmitted at the completion of the Boot (power-on reset process). Using additional Registry entries this email may be highly customized. You can specify multiple recipients and even attach files like jniorboot.log conveying the information of interest.

6.2. Simple Alarm Notification

Certain situations may be set to trigger an Alarm condition. This might be when a digital input turns on, when an input counter reaches a certain count or when an input/output usage reaches a defined number of hours. When an Alarm condition occurs JNIOR may be configured to send an email notification.

```
Events/OnAlarm/Email = enabled
Email/ToAddress = jdoe@integpg.com
```

Using the Registry Editor make sure that the above two keys are defined. The first enables the alarm notification and the second specifies your email address. If the first key is absent then the feature may be disabled. This feature may be applied to individual inputs and the operation is dependant on the presence of other Registry settings. The above enables the feature generically (all alarms). If the second key is not defined JNIOR might not know where to send the message.

You must also make sure that the IP configuration of the JNIOR is correct. In particular JNIOR must know how to find the MailHost. Use the `ipconfig` command or set the appropriate Registry keys in the IpConfig section. If the Registry is edited you will have to reboot the JNIOR in order for the settings to be applied.

One of the JNIOR digital inputs must then be configured to generate an alarm. With the following setting an alarm will be triggered when a voltage is applied to the Digital Input number 1.

```
IO/Inputs/din1/Alarming = enabled
```

An alarm condition must exist for at least ½ second before it will trigger the notification. Input latching may be used to capture shorter pulses if those are intended to trigger the alarm. Once an alarm notification is sent the specific alarm condition must be removed for at least 5 minutes before any additional notification would be sent.

The resulting email messages may be highly customized on an input-by-input basis. These may be addressed to multiple recipients and include file attachments such as a jniorio.log file.

6.3. High Speed Relay Control Feature

The JNIOR provides for remote monitoring and it also supports eight relays which can be controlled remotely. Typically these are not applied in high speed or time critical applications. The JNIOR does support one feature that can be considered useful for high speed control.

Each of the relay outputs can be made to close upon reaching a defined trigger count. For instance if you want Relay 2 to close after counting 5,000 pulses on the associated Digital Input 2, you can define the following Registry Key:

```
IO/Outputs/rout2/TriggerCount = 5000
```

Assuming that your application has cleared the counter on Digital Input 2, the relay will close after 5,000 input pulses. The relay closes immediately upon seeing the 5000th low-to-high transition of the input. See the Registry Key definition for more information.

7. Dynamic Registry Keys

Any Registry Key or Path that begins with the ‘$’ character is considered a *dynamic entry* which is temporary and present only for the current session. These keys are not saved to the .INI file and do not persist through a system reset or reboot.

Certain Dynamic Keys are defined during boot and therefore appear to always be present. These are typically System Parameters as documented in the following section.

Other Dynamic Keys are created by the system to provide access to frequently changing data. One example of this is the $HourMeter key providing Usage information such as the amount of time that a particular input is active.

Entire Registry Paths may be defined wherein all the keys are dynamic in nature. If the ‘$’ begins any level of a Registry structure all of the keys contained therein are dynamic and will not be preserved. The individual entries themselves need not include the ‘$’.

Applications can use dynamic keys as a form of inter-process communications.

Only the system can define keys at the root level of the Registry. In all cases those defined are dynamic keys.

8. System Parameters

These '$' keys do not appear in the *jnior.ini* file and are OS related. These are special dynamic keys.

`$BootTime`

This returns a string representing the time according to the JNIOR clock at the completion of the latest power-up boot sequence.

`$Model`

This returns the product Model number. For example: "310"

`$SerialNumber`

This returns the product serial number as a String. For example: "104060134"

`$Version`

This returns the current Version string for the product release. For example: "Version 2.13.83"

9. Registry Key Reference

The following are the current assignments as coded into the OS firmware. The Registry may contain other keys as assigned by the user or applet/application developer.

9.1. General Device Identification

The following keys apply to the JNIOR in general.

Device/Desc

The entry provides a textual description for this JNIOR. This might be displayed by external applets or applications as identification. It may also be included in e-mail notifications to identify the specific source. This description may be optionally added by the user.

9.2. Drivers and Boot Programs

Drivers and programs may be initialized and run at boot. Such programs may be developed to handle custom capabilities that would not normally be part of the JNIOR OS. The development of these programs is beyond the scope of this document. Contact INTEG process group, inc. for further information if you are considering such an undertaking.

Run/<driver>

During the boot process JNIOR will scan this section of the Registry and attempt to execute drivers/programs listed here. The `<driver>` group name is entirely arbitrary and each such entry specifies the driver file and any arguments that may be associated with it. For example suppose that the following entry were defined:

```
Run/MyProgram = BootStuff.tini -timer 100
```

Near the end of the boot process JNIOR would execute the program BootStuf.tini as a background process if the file can be located in the file system. The parameters shown would be passed to the program for its use. JNIOR would refer to the instance of this program as “MyProgram”.

Note that only the keys defined at this node will be executed and any substructure will be ignored. If it were useful the programmer could design the program to retrieve configuration from this location. For example the additional parameters in the above could be conveyed as follows:

```
Run/MyProgram = BootStuff.tini
Run/MyProgram/timer = 100
```

Again, the program BootStuff.tini would have to be designed to either retrieve either the arguments either from the command line or from the Registry (or both). Of course the program configuration could be located elsewhere in the Registry. The point is that the above usage would not interfere with the execution of the drivers/programs or otherwise confuse the operating system into trying to execute a non-existent program named like an argument.

9.3. Network Configuration

The following keys work in conjunction with `ipconfig`, `hostname` and `jrconfig` to provide for network configuration.

IpConfig/DHCP

JNIOR by default will require that IP addressing be defined by the user through the `ipconfig` command. Initially this requires a serial connection to the JNIOR and access to the command prompt. This registry key may be used to enable DHCP allowing the JNIOR to lease an IP address during boot from the local DHCP server. It is preferred that DHCP be enabled directly rather than manually through the Registry by using the command:

```
ipconfig -d
```

which will immediately enable DHCP not requiring reboot. The associated registry key will be updated automatically by the command.

Since JNIOR is a *server* the IP address or some related name is required to access the unit. When DHCP is used and the DHCP server is configured to cache the JNIOR hostname the hostname can conveniently be used to find the JNIOR.

IpConfig/IPAddress

This defines the network IP Address to be used with this JNIOR. The address may be defined through changes to this Registry key, by using the `ipconfig` command, or through DHCP. Changes take effect on reboot. Use `ipconfig` to make immediate changes. This key is not valid and not used when DHCP is ‘enabled’.

`IpConfig/SubnetMask`

This defines the network Subnet Mask to be used with this JNIOR. The mask may be defined through changes to this Registry key or by using the `ipconfig` command. Changes take effect on reboot. Use `ipconfig` to make immediate changes. This key is not valid and not used when DHCP is ‘enabled’.

`IpConfig/GatewayIP`

This defines the network Gateway IP Address. This address is only required if JNIOR is to communicate outside its home network. This would be the case if JNIOR is to synchronize its clock with an external time server (see IpConfig/NTPServer). Changes take effect on reboot. Use `ipconfig` to make immediate changes. This key is not valid and not used when DHCP is ‘enabled’.

`IpConfig/PrimaryDNS`

This defines the Primary DNS IP Address used for name resolution on the network. This would be required if JNIOR is to synchronize its clock with an external time server as DNS is used to resolve “clock.isc.org” into the appropriate IP Address for communication (see IpConfig/NTPServer). Changes take effect on reboot. Use `ipconfig` to make immediate changes. This key is not valid and not used when DHCP is ‘enabled’.

`IpConfig/SecondaryDNS`

This defines the Secondary DNS IP Address used for name resolution on the network should the Primary DNS not be available. Changes take effect on reboot. Use `ipconfig` to make immediate changes. This key is not valid and not used when DHCP is ‘enabled’.

`IpConfig/HostName`

This defines a machine name for the device. By default “jr<$SerialNumber>” will be used. When email is constructed the default “From” address will be HostName@Domain.

`IpConfig/Domain`

Defines the Domain Name associated with the local network. This may be used to generate an email address such as jnior@integpg.com where the Domain is “integpg.com”. Changes take effect on reboot. Use `ipconfig` to make immediate changes.

IpConfig/MailHost

This specifies the address of the SMTP Mail Server that accepts email from this network. This must be specified if JNIOR is going to send email messages. Changes take effect on reboot. Use `ipconfig` to make immediate changes. Any MailHost defined by a DHCP server will override this key.

IpConfig/Username

Specifies the Username required for SMTP Authentication. This may or may not include the domain as this depends on the requirements of the particular server. SMTP Authentication is used ONLY when a MailHost is defined and when both the Username and Password keys are valid.

Jnior OS v4.1.708 and later

IpConfig/Password

Specifies the Password required for SMTP Authentication. The password is encoded in the Registry and is not displayed by the JNIOR except during initial entry using the Registry Editor. The jnior.ini file may be edited and the Password defined there in clear text however, once an email is issued the Password will be encoded. SMTP Authentication is used ONLY when a MailHost is defined and when both the Username and Password keys are valid.

Jnior OS v4.1.708 and later

IpConfig/DNSTimeout

This defines the timeout in milliseconds to be used in waiting for a response from the specified DNS servers. The default is 0 and this allows JNIOR to internally specify an optimized timeout.

IpConfig/NTPServer

JNIOR can synchronize with a network time server supporting Network Time Protocol (NTP). To utilize this capability JNIOR must be connected to a network with access to such a server and a Gateway and DNS Server (unless absolute IP address is given) must be defined. The format for the IpConfig/NTPServer key is as follows:

```
IpConfig/NTPServer = URLAddress [, NTPPort [, Timeout]]
```

By default the URLAddress is `clock.isc.org` and the NTPPort is the standard 123. A Timeout of 2000 milliseconds is used by default.

If the URLAddress specifies the IP Address of the server, such as `204.152.184.72` in the case of the default server, then the DNS Server configuration is not required.

The NTPPort and Timeout are seldom changed from the standard settings and need not be included in the Registry value. For example the following entry is sufficient to utilize an alternate server.

```
IpConfig/NTPServer = time.nist.gov
```

Only one time server can be specified. If that server is not available then the synchronization will be bypassed. Note that JNIOR's clock is maintained by a battery during periods without power. Synchronization is not required but useful periodically as the clock may drift in accuracy very slowly over long periods.

Time synchronization occurs during power-on boot. If synchronization was successful during boot then it is attempted every four (4) hours thereafter to maintain clock alignment. JNIOR may also be commanded to synchronize using the DATE command through a Telnet connection. In this manner the clock can be accurately set in the absence of a network connection or time server.

IpConfig/MTU

The `IpConfig/MTU` Registry key defines the maximum size of any packet transmitted over the Ethernet port. The Maximum Segment Size (MSS) is defined as MTU - 40 (40 bytes less than the MTU setting) and no packet will be transmitted with a payload exceeding this size. The default MTU setting is 1500. Regardless of the MTU setting JNIOR will properly receive packets of any size up to the standard network MTU of 1500.

Connections that utilize PPPoE (modems, VPN) require additional header bytes and therefore cannot carry as large of a payload. In these cases it may be necessary to reduce the maximum transmitted segment size in order to achieve proper operation. For instance the following Registry key may correct PPPoE problems:

```
IpConfig/MTU = 1480
```

Valid MTU settings are 400 to 1500 inclusive. A change in MTU setting applies to all Ethernet connections and takes effect only upon reboot.

9.4. Event Management

There are various events which occur during the operation of JNIOR. Some are normal occurrences and others not so normal. JNIOR can be configured to perform certain actions when these events occur. The following Registry keys define how JNIOR is to respond to such events.

`Events/Services`

JNIOR monitors events and responds to certain situations depending on the configuration established by the Registry. JNIOR also can generate an audit trail of events and otherwise routinely log changes in data. By default these services are “enabled”. This Registry key can be used to completely disable all such services. A setting of “disabled” will stop processing without affecting the configuration. This key takes effect immediately and reboot is not required.

`Events/OnBoot`

This key can be used to globally enable or disable the activities related to Power On Reset (Boot) defined below. If set to “disabled” this key will override and disable each of the `OnBoot` actions. If set to “enabled” this will enable all of the `OnBoot` activities not specifically set within the section.

`Events/OnBoot/Email`

When “enabled” this instructs JNIOR to send an Email on Boot. This key can also specify a named `Email` section that defines the content and addressing of the particular message to be used. This Registry key has two parts where the Email name is optional. The format is as follows.

```
Events/OnBoot/Email = enabled [, EmailBlock]
```

For example, the following will send an (admittedly basic) email to an INTEG employee when JNIOR boots.

```
Events/OnBoot/Email = enabled
Email/ToAddress = jdoe@integpg.com
```

This uses the “default” Email message definition. A custom or “named” Email message might optionally be referenced by including its block name.

```
Events/OnBoot/Email = enabled, AdminNotify
Email/AdminNotify/ToAddress = jdoe@integpg.com
Email/AdminNotify/Attachment = jniorboot.log
```

The key in this case defines the subsection of the `Email` section or that part following “`Email/`” and this can be any descriptive name that may be helpful. It must match that used in the Email specification block or the Email will not be generated. By default if this part of the key is undefined then general Email defined by the `Email` key will be sent. See Email Specification below for more information.

`Events/OnAlarm`

This key may be optionally defined and set to “disabled” to override and disable the individual Digital Input state alarm services defined by the individual `IO/inputs/din#/Alarm/` Registry sections. If any state alarm services are desired then this key may be left undefined as it is “enabled” by default.

`Events/OnAlarm1`

This key may be optionally defined and set to “disabled” to override and disable the individual Digital Input Counter alarm Type 1 services defined by the individual `IO/inputs/din#/Alarm1/` Registry sections. If any counter alarm Type 1 services are desired then this key may be left undefined as it is “enabled” by default. Counter Type 1 and Type 2 alarm services differ only in the set points used.

`Events/OnAlarm2`

This key may be optionally defined and set to “disabled” to override and disable the individual Digital Input Counter alarm Type 2 services defined by the individual `IO/inputs/din#/Alarm2/` Registry sections. If any counter alarm Type 2 services are desired then this key may be left undefined as it is “enabled” by default. Counter Type 1 and Type 2 alarm services differ only in the set points used.

`Events/OnAlarm/Email`

This key may be optionally defined and has two separate values. It is formatted as follows.

```
Events/OnAlarm/Email = enabled [, EmailBlock]
```

This key may be optionally set to “disabled” to override and disable the transmission of Email Notifications in response to individual Digital Input state Alarms as defined by the individual `IO/inputs/din#/Alarm/Email` Registry keys. If state alarms Email Notifications are desired then this key may be left undefined as it is “enabled” by default. Counter Type 1 and Type 2 alarm services differ only in the set points used.

An optional EmailBlock may be defined specifying the alternate default Email format to be used for Digital Input state alarms. If the individual `IO/inputs/din#/Alarm/Email` keys omit an Email Block specification the one defined here would be used. If no block is given in either location then the default Email format defined in the `Email/` section would apply. See the Email Specification below.

Events/OnAlarm1/Email

This key may be optionally defined and has two separate values. It is formatted as follows.

```
Events/OnAlarm1/Email = enabled [, EmailBlock]
```

This key may be optionally set to “disabled” to override and disable the transmission of Email Notifications in response to individual Digital Input counter Type 1 Alarms as defined by the individual `IO/inputs/din#/Alarm1/Email` Registry keys. If counter Type 1 alarms Email Notifications are desired then this key may be left undefined as it is “enabled” by default. Counter Type 1 and Type 2 alarm services differ only in the set points used.

An optional EmailBlock may be defined specifying the alternate default Email format to be used for Digital Input counter Type 1 alarms. If the individual `IO/inputs/din#/Alarm1/Email` keys omit an Email Block specification the one defined here would be used. If no block is given in either location then the default Email format defined in the `Email/` section would apply. See the Email Specification below.

Events/OnAlarm2/Email

This key may be optionally defined and has two separate values. It is formatted as follows.

```
Events/OnAlarm2/Email = enabled [, EmailBlock]
```

This key may be optionally set to “disabled” to override and disable the transmission of Email Notifications in response to individual Digital Input counter Type 2 Alarms as defined by the individual `IO/inputs/din#/Alarm2/Email` Registry keys. If counter Type 2 alarms Email Notifications are desired then this key may be left undefined as it is “enabled” by default. Counter Type 1 and Type 2 alarm services differ only in the set points used.

An optional EmailBlock may be defined specifying the alternate default Email format to be used for Digital Input counter Type 2 alarms. If the individual `IO/inputs/din#/Alarm1/Email` keys omit an Email Block specification the one defined here would be used. If no block is given in either location then the default Email format defined in the `Email/` section would apply. See the Email Specification below.

Events/OnUsage

This key may be optionally defined and set to “disabled” to override and disable the individual Usage Alarm services defined by the individual `IO` Registry sections. If any usage alarm services are desired then this key may be left undefined as it is “enabled” by default.

Events/OnUsage/Email

This key may be optionally defined and has two separate values. It is formatted as follows.

```
Events/OnUsage/Email = enabled [, EmailBlock]
```

This key may be optionally set to “disabled” to override and disable the transmission of Email Notifications in response to individual Usage Alarms as defined by the individual `IO` Registry keys. If Usage Alarm Email Notifications are desired then this key may be left undefined as it is “enabled” by default.

An optional EmailBlock may be defined specifying the alternate default Email format to be used for Usage Alarms. If the individual `IO` keys omit an Email Block specification the one defined here would be used. If no block is given in either location then the default Email format defined in the `Email/` section would apply. See the Email Specification below.

Events/OnConfig

Certain services may be performed when JNIOR updates the jnior.ini file in response to changes in the Registry. Set this key to “disabled” to override and disable all services related to a detected configuration change. By default this key is “enabled”.

Events/OnConfig/Email

When configuration changes are made and settings altered in the Registry the jnior.ini file will be updated to reflect those changes. JNIOR may be configured to send a Configuration Change Notification when this occurs. This key contains two optional entries. The format is as follows.

```
Events/OnConfig/Email = enabled [, EmailBlock ]
```

If a Configuration Change Notification email is desired this key must be set to “enabled” and the `Events/OnConfig` key must not be set to “disabled”. Optionally an EmailBlock can be specified to select the email format to use (*see Email Specification*).

9.5. Configuring Email Notifications

JNIOR can send email messages in response to certain events. Any number of unique Email messages can be defined for use as the situation requires. A generic (not situation specific) Email is defined by the following keys. Unique Email construct can be defined and assigned to unique Registry sections. These may be separately referenced and used as the situation may require. To create a situation specific E-mail message replace <block> with a unique message identifier in those keys where it appears.

Email/<block>/FromAddress

This specifies a single email address to indicate the source of the message. By default the HostName@Domain address will be used. Care should be taken to use a valid "From" address and one that can legally post email through the defined MailHost as some Mail Servers impose strict security requirements.

Email/<block>/ToAddress

This defines one or more destination email addresses of the form user@domain.com. Multiple addresses are separated by commas. For example:

```
Email/AdminNotify/ToAddress = jdoe@integpg.com, admin@integpg.com
```

Note that Registry key values are limited to 255 characters in length and this would place a limit on the number of email address that may be defined. It is the user's responsibility to ensure that these addresses are valid and are updated as needed.

Email/<block>/CcAddress

This defines one or more destination email addresses of the form user@domain.com. These addresses will also receive the defined message but as a CC: recipient.

Email/<block>/BccAddress

This defines one or more destination email addresses of the form user@domain.com. These addresses will also receive the defined message but as a BCC: recipient.

Email/<block>/Subject

This defines the Subject line to be used with the message. JNIOR requires that a Subject be defined for all messages although this is not strictly a requirement for email itself. If the Subject key is not given, JNIOR will utilize a default Subject as appropriate to the purpose of the email.

`Email/<block>/Message`

This defines text Message content to be sent in the email. JNIOR does not require that this message content be supplied. This may be used in conjunction with a Message File and the text defined here will appear as a prefix to the content of the file. This is not as applicable in combination with HTML formatted file content as in that case this text would have to comply with the HTML structure as well.

`Email/<block>/MessageFile`

This defines the file containing the textual Message content to be included in the email. If separate Message text is supplied the content of this file will be appended to that text in the message.

`Email/<block>/HTMLMessageFile`

This defines the file containing the Message content which is assumed to be defined using valid HTML coding. If the file exists this supersedes the MessageFile specification. If separate message text is supplied the HTML content will be appended to that text and therefore the separate text must comply with the HTML structure.

`Email/<block>/Attachments`

This lists one or more attachments (file names) to be sent with the email message.

`Email/FromAddress`
`Email/ToAddress`
`Email/CcAddress`
`Email/BccAddress`
`Email/Subject`
`Email/Message`
`Email/MessageFile`
`Email/HTMLMessageFile`
`Email/Attachments`

A Default Email Format may be specified by omitting the <block> section. The above keys define the format for an email that would be used when a uniquely named Email Block is not specified.

`Email/RetryCount`

There may be difficulties in delivering an e-mail to the destination server. By default after an attempt has failed JNIOR will schedule the delivery of the message and will do so up to 12 times. Failures on an initial attempt are typical these days as servers are implementing greylisting techniques to reduce the amount of unsolicited spam e-mail. In general the Internet is a lossy network and retries are not unusual. Set RetryCount to 0 or 1 to disable retries.

Email/RetryDelay

After a failed e-mail delivery attempt JNIOR will reschedule another delivery at a later time. By default this is configured for 10 minutes. The RetryDelay key defines this time in minutes. E-mail servers implementing greylisting my routinely reject initial e-mail deliveries for typically 30 minutes. These techniques are designed to cause spammers some difficulties and help to cut down the amount of unsolicited e-mail. The JNIOR should be set to attempt repeated deliveries for at least a couple of hours to increase chances of success.

Email/Port

The default Simple Mail Transfer Protocol SMTP port is 25. This key may be used to specify an alternate port as may be required by your MailHost. Note that the MailHost and any associated SMTP Authentication settings (Username and Password) are set by keys in the IpConfig section.

9.6. Web (HTTP) Server

The following configuration settings control the WWW interface to JNIOR.

WebServer/Server

This entry can be used to disable the HTTP Server. This may be desirable if communications with JNIOR will be through some other means and connections to the JNIOR HTTP Port are to be ignored. The default setting is “enabled”. Changes take effect on reboot.

WebServer/Port

This specifies the TCP/IP port to use for HTTP services. The default is standard port 80.

WebServer/Root

This specifies the folder within the JNIOR file system that represents the root of the website. This is the folder that would contain the unit’s home page and related pages are located in this folder or in subfolders. The default is “/www/”. A useful alternative might be “/flash/www/” which would place the root in the Flash File System. This folder must be specified from the root of the file system (starting with a ‘/’) and the trailing ‘/’ is optional.

`WebServer/Index`

This specifies the page to be served as the unit’s home page. This is the HTML document that would appear if only the JNIOR’s IP address were referenced from the browser for instance. The default is “index.html”.

`WebServer/Path`

This may be used to specify alternate search paths for web content. The JNIOR first searches the Root of the WebServer for the requested content. If the page is not located then each path defined in this key is searched in sequence. Paths must be specified from the root of the file system (starting with a ‘/’) and the trailing ‘/’ is optional. Multiple paths are separated by a semicolon ‘;’.

9.7. Jnior Protocol Server

The following are configuration settings for the JNIOR protocol server.

`JniorServer/Server`

This entry can be used to disable the JNIOR Server. This may be desirable if communications with JNIOR will be through some other means and connections to the JNIOR I/O Port are to be ignored. The default setting is “enabled”. Changes take effect on reboot.

`JniorServer/Port`

This defines the IP port on which JNIOR will listen for connections. The default port is 9200. Changes take effect on reboot. Use `jrconfig` to make immediate changes.

`JniorServer/Login`

By default the JNIOR protocol requires a successful login. This is achieved through a function as part of the protocol. It is highly recommended. If the JNIOR is connected to a private secure network this login requirement can be removed. Set this Registry key to ‘disabled’. The change takes effect immediately.

JniorServer/Anonymous

Defines the user ID (integer 0-254) applied to anonymous logins. When the JNIOR protocol requires a login (default) the login must reference a defined username using the correct password for that account. In order to accommodate a scheme whereby data monitoring would not require login but control or configuration would, the JNIOR OS allows for *anonymous* login. When the JniorServer/Anonymous key is defined (exists) anonymous login is allowed. To prevent anonymous login this key must be removed from the Registry. A user ID of 0 is recommended.

User IDs of 128 through 254 are administrators and have complete control through the JNIOR protocol. If you use a JniorServer/Anonymous setting of 128 or greater you might as well set JniorServer/Login to disabled as the level of security is the same (none).

An anonymous login is one where both the username and password are omitted (null) in the associated login request.

JniorServer/RemoteIP

The JNIOR protocol server can be configured to make a connection to one remote host. This is configured through the `JniorServer/RemoteIP` and `JniorServer/RemotePort` registry keys. The `JniorServer/RemoteIP` key defines the IP Address of the Remote Host that the JNIOR will connect to.

JniorServer/RemotePort

The JNIOR protocol server can be configured to make a connection to one remote host. This is configured through the `JniorServer/RemoteIP` and `JniorServer/RemotePort` registry keys. The `JniorServer/RemotePort` key defines the Port of the Remote Host that the JNIOR will connect to.

JniorServer/LogHostName

The JNIOR protocol connection will log an entry when the connection is established, when the login is either successful or unsuccessful or when the connection is closed. This key specifies whether to resolve and prepend the HostName to the log entry. It is disabled by default.

9.8. Modbus Protocol Server

JNIOR can communicate through Modbus. The following are configuration settings that pertain to Modbus.

`ModbusServer/Server`

This entry can be used to disable the Modbus Server. This may be desirable if communications with JNIOR will be through some other means and connections to the Modbus I/O Port are to be ignored. The default setting is “enabled”. Changes take effect on reboot.

`ModbusServer/Port`

This defined the TCP/IP port to be used for Modbus communications. The default is the standard Modbus Port 502.

`ModbusServer/Login`

By default the JNIOR Modbus protocol implementation requires a successful login. This is achieved through a function as part of the protocol implementation. It is highly recommended. If the JNIOR is connected to a private secure network this login requirement can be removed. Set this Registry key to ‘disabled’. The change takes effect immediately.

9.9. I/O Logging

The following keys apply specifically to the JNIOR Digital Inputs and Relay Outputs in general.

`IO/Log`

The JNIOR can provide a record of changes to its inputs and outputs. When changes are detected an entry is made in the `/jniorio.log` file. By default this logging feature is disabled. To enable logging set this key to “enabled”. Logging can be enabled or disabled on an input or output group basis or for each individual input or output. This is controlled by separate key entries in the associated sections as described below.

9.10. Configuring Digital Inputs

The following keys are associated with the Digital Inputs. Note: In each of the following Keys replace the ‘#’ with the appropriate channel number 1-8.

`IO/Inputs/Log`

If the IO/Log key is set to “enabled” a record of input changes is generated. This key can be optionally used to specifically disable logging in general for the Digital Inputs as a group. By default it is enabled.

`IO/Inputs/din#/Desc`

This defines the textual description for the associated Digital Input (String). The defaults should be “Digital In #” where ‘#’ is replaced by ‘1’ through ‘8’ as appropriate. (Used by the applets.)

`IO/Inputs/din#/OnDesc`

This defines the textual description used when the associated input is in the “On” state. “On” should be the default. (Used by the applets.)

`IO/Inputs/din#/OffDesc`

This defines the textual description used when the associated input is in the “Off” state. “Off” should be the default. (Used by the applets.)

`IO/Inputs/din#/Inversion`

Set to “enabled” to invert the sense of the Digital Input. When enabled the input will be considered “On” when no voltage is applied to the external circuit (LED off). The input will be considered “Off” when voltage is applied (LED on). The default is “disabled”. This inversion is applied immediately to the input and affects all other functions (Alarming, Counting, etc.).

`IO/Inputs/din#/Debounce`

Relays and switches have mechanical contacts which physically make or break a circuit. Rarely will the contacts come together solidly or separate decisively without bouncing (briefly making and breaking the circuit). This can raise havoc with digital latching and counting circuits that might be monitoring through the relay/switch contact. It can result in latching at the wrong time (when the relay opens for instance) or in extra counts. Both are undesirable.

By default the JNIOR digital inputs are *debounced*. The default debouncing delay is 200 milliseconds. An input must remain quiet (not change) for 200ms before any transition on that input will be processed (latched, counted or logged). This is sufficient to eliminate almost all of the issues arising from contact bounce.

This Registry Key optionally allows for the adjustment of the debounce filter. A setting of 0 (zero milliseconds) disables the software debounce. In this case the JNIOR is capable of counting transitions occurring at rates up to 2,000/sec. If the Registry Key is absent the JNIOR will use whatever setting had been previously given. It does not fall back to factory default if you remove the key. The filter setting is non-volatile. Valid settings are 0 thru 255 (milliseconds).

IO/Inputs/din#/Latching

Set to “enabled” when the associated input is to be latched. When enabled, the input will be considered to remain in the “On” state after voltage is applied to the external input even when it is removed. The input is considered to be latched. If the LatchTime is set to 0 seconds the User must manually reset the input. This can be done through the Monitoring applet or other external application. The default is “disabled”. This does not affect Counting.

IO/Inputs/din#/LatchTime

This defines the time in seconds (0.1 equals 100 milliseconds) that an input remains latched before being automatically reset. A value of 0.0 will require the User to separately reset the latched input. The default is 0.0 seconds.

IO/Inputs/din#/Log

If the IO/Log key is set to “enabled” and IO/Inputs/Log key has not been set to “disabled” a record of input condition changes will be written to the /jniorio.log file. This key can be optionally used to disable logging on an input by input basis. If an input is going to be rapidly changing the time spent in the logging process can degrade system performance. In such circumstances it is recommended that logging be disabled for the input.

IO/Inputs/din#/$HourMeter

This dynamically reports the total number of hours that the individual digital input has physically been in the “On” state. This value is non-volatile maintaining content through power loss and until it is specifically reset. It is reported here in hours to the one-hundredth. The Hour Meter is accurate to the millisecond and this high resolution value may be read through the JNIOR Protocol or through the JRMON command.

IO/Inputs/din#/Alarming

Set to “enabled” when the associated input is to generate a state alarm.

IO/Inputs/din#/Alarm/Email

This key may be optionally defined and has two separate values. It is formatted as follows.

```
IO/Inputs/din#/Alarm/Email = enabled [, EmailBlock]
```

This key may be set to "enabled" to enable the transmission of Email Notifications in response to a Digital Input state change Alarm. An optional EmailBlock may be defined specifying the Email format to be used.

IO/Inputs/din#/Alarm/Inversion

Set to "enabled" when the associated input is to alarm upon entering the "Off" state. The default is "disabled" and normally when alarming is enabled the alarm is generated when the input turns "On".

IO/Inputs/din#/Alarm/OnAlarm

When set to "enabled" JNIOR will provide services related to the occurrence of Input Alarms generated by a state change on this Digital Input. Counter Alarms are handled separately. If the key is set to "disabled" no Digital Input state alarm services will be provide. By default services are "enabled".

IO/Inputs/din#/Alarm/HoldOff

This defines the amount of time in milliseconds that the Digital Input state alarm must remain clear before any subsequent state alarm on this input will be acted upon. The default is 300000 or 5 minutes. When an alarm occurs the services associated with that event are performed. The alarm must reset and remain so for this amount of time before those actions would be performed again.

IO/Inputs/din#/Count/Units

The defines the "units" text to be displayed with the associated input counter. The default should be "counts".

IO/Inputs/din#/Count/Multiplier

When the Count Multiplier is set to 0.0 (default) the absolute counter value should be displayed. When a non-zero Multiplier is specified then the value is used to scale the counter value for display. The scaled counter value is used for the count alarm trigger points.

IO/Inputs/din#/Count/SampleTime

Counts accumulate until reset separately by the User using the applet or other application. This is the case when SampleTime is 0.0 (default). When a non-zero SampleTime is used the counter displays the total count accumulated during that period (in seconds). For instance, with the appropriate combination of Multiplier and SampleTime the counter can display RPM for a strobe input. The alarm trigger points may be set to monitor either high or low count ranges.

IO/Inputs/din#/Count/Alarm1

Set to enable an alarm when the absolute count exceeds the Limit1 count value.

IO/Inputs/din#/Count/Limit1

This defines the trigger point for the count alarm. An alarm (type 1) can be generated with the scaled counter exceeds this value.

IO/Inputs/din#/Count/Alarm2

Set to enable an alarm when the absolute count exceeds the Limit2 count value.

IO/Inputs/din#/Count/Limit2

This defines the trigger point for the count alarm. An alarm (type 2) can be generated with the scaled counter exceeds this value.

IO/Inputs/din#/Alarm1/OnAlarm

This key may be optionally defined and set to “disabled” to disable services related to the occurrence of Digital Input Type 1 Counter Alarms on this input. Counter Type 1 and Type 2 Alarms differ only in the set points used.

IO/Inputs/din#/Alarm1/HoldOff

This defines the amount of time in milliseconds that the Digital Input counter type 1 alarm must remain clear before any subsequent type 1 alarm on this input will be acted upon. The default is 30000 or 5 minutes. When an alarm occurs the services associated with that event are performed. The alarm must reset and remain so for this amount of time before those actions would be performed again.

`IO/Inputs/din#/Alarm1/Email`

This key may be optionally defined and has two separate values. It is formatted as follows.

```
        IO/Inputs/din#/Alarm1/Email = enabled [, EmailBlock]
```

This key may be set to "enabled" to enable the transmission of Email Notifications in response to a Digital Input counter Type 1 Alarm. If a counter Type 1 alarm Email Notification is desired then this key or the `Events/OnAlarm1/Email` must be defined and "enabled". Counter Type 1 and Type 2 alarm services differ only in the set points used. An optional EmailBlock may be defined specifying the Email format to be used.

`IO/Inputs/din#/Alarm2/OnAlarm`

This key may be optionally defined and set to "disabled" to disable services related to the occurrence of Digital Input Type 2 Counter Alarms on this input. Counter Type 1 and Type 2 Alarms differ only in the set points used.

`IO/Inputs/din#/Alarm2/HoldOff`

This defines the amount of time in milliseconds that the Digital Input counter type 2 alarm must remain clear before any subsequent type 2 alarm on this input will be acted upon. The default is 30000 or 5 minutes. When an alarm occurs the services associated with that event are performed. The alarm must reset and remain so for this amount of time before those actions would be performed again.

`IO/Inputs/din#/Alarm2/Email`

This key may be optionally defined and has two separate values. It is formatted as follows.

```
        IO/Inputs/din#/Alarm2/Email = enabled [, EmailBlock]
```

This key may be set to "enabled" to enable the transmission of Email Notifications in response to a Digital Input counter Type 2 Alarm. If a counter Type 2 alarm Email Notification is desired then this key or the `Events/OnAlarm2/Email` must be defined and "enabled". Counter Type 1 and Type 2 alarm services differ only in the set points used. An optional EmailBlock may be defined specifying the Email format to be used.

`IO/Inputs/din#/Usage/Alarm`

Set to enable an alarm when the associated $HourMeter usage meter reaches a specified number of hours.

`IO/Inputs/din#/Usage/Limit`

Defines the alarm setpoint in hours and may include a decimal portion. The associated input goes into alarm when the $HourMeter usage meter reaches or exceeds this setpoint.

`IO/Inputs/din#/Usage/Email`

This key may be optionally defined and has two separate values. It is formatted as follows.

```
IO/Inputs/din#/Usage/Email = enabled [, EmailBlock]
```

This key may be set to “enabled” to enable the transmission of Email Notifications in response to a Digital Input Usage Alarm. If a Usage Alarm Email Notification is desired then this key or the `Events/OnUsage/Email` key must be defined and “enabled”. An optional EmailBlock may be defined specifying the Email format to be used.

`IO/Inputs/din#/Usage/HoldOff`

This defines the amount of time in milliseconds that the Digital Input usage alarm must remain clear before any subsequent usage alarm on this input will be acted upon. The default is 30000 or 5 minutes. When an alarm occurs the services associated with that event are performed. The alarm must reset and remain so for this amount of time before those actions would be performed again.

`IO/Inputs/din#/Usage/OnAlarm`

This key may be optionally defined and set to “disabled” to disable services related to the occurrence of Digital Input Usage Alarms on this input.

9.11. Configuring Relay Outputs

The following keys are associated with the Relay Outputs. Note: In each of the following Keys replace the ‘#’ with the appropriate channel number 1-8.

`IO/Outputs/Log`

If the IO/Log key is set to “enabled” a record of output changes is generated. This key can be optionally used to specifically disable logging in general for the Relay Outputs as a group. By default it is enabled.

IO/Outputs/Rout_9-12

The JNIOR model 310 has 8 internal outputs. You can expand the number of outputs by purchasing the 4 Relay Output Expansion Module. You can connect up to 2 of them per JNIOR to obtain output channels 9 – 16. This registry key defines while module address corresponds to channels 9 – 12. This key is automatically assigned when you connect 1 or modules and reboot. If more then one module is present during the reboot then one module is arbitrarily assigned. This key assures that the same module is used for these channels after every reboot.

IO/Outputs/Rout_13-16

The JNIOR model 310 has 8 internal outputs. You can expand the number of outputs by purchasing the 4 Relay Output Expansion Module. You can connect up to 2 of them per JNIOR to obtain output channels 9 – 16. This registry key defines while module address corresponds to channels 13 – 16. This key is automatically assigned when you connect 2 or modules and reboot. If more then two modules are present during the reboot then one module is arbitrarily assigned. This key assures that the same module is used for these channels after every reboot even if the module representing channels 9 – 12 is removed.

IO/Outputs/rout#/Desc

This defines the textual description for the associated Relay Output (String). The defaults should be “Relay Out #” where ‘#’ is replaced by ‘1’ through ‘8’ as appropriate. (Used by the applets.)

IO/Outputs/rout#/ClosedDesc

This defines the textual description used when the associated relay has been activated and is in the “Closed” state. “Closed” should be the default. (Used by the applets.)

IO/Outputs/rout#/OpenDesc

This defines the textual description used when the associated relay is in the “Open” state. “Open” should be the default. (Used by the applets.)

IO/Outputs/rout#/Log

If the IO/Log key is set to “enabled” and IO/Outputs/Log key has not been set to “disabled” a record of output condition changes will be written to the /jniorio.log file. This key can be optionally used to disable logging on an output by output basis.

IO/Outputs/rout#/InitialState

This key is used to define what the state of the output should be set to on boot up. This state is set after the registry is loaded. This might be 60 seconds after initial power is applied. Setting the key to a value of 0 would effectively close the output. Setting the key to a positive integer value would cause the output to pulse for the duration defined by the value given. Any negative or non integer value will result in no action.

IO/Outputs/rout#/$HourMeter

This dynamically reports the total number of hours that the individual relay output has physically been in the “Closed” state. This value is non-volatile maintaining content through power loss and until it is specifically reset. It is reported here in hours to the one-hundredth. The Hour Meter is accurate to the millisecond and this high resolution value may be read through the JNIOR Protocol or through the JRMON command.

IO/Outputs/rout#/Usage/Alarm

Set to enable an alarm when the associated $HourMeter usage meter reaches a specified number of hours.

IO/Outputs/rout#/Usage/Limit

Defines the alarm setpoint in hours and may include a decimal portion. The associated relay output goes into alarm when the $HourMeter usage meter reaches or exceeds this setpoint.

IO/Outputs/rout#/Usage/Email

This key may be optionally defined and has two separate values. It is formatted as follows.

```
IO/Outputs/rout#/Usage/Email = enabled [, EmailBlock]
```

This key may be set to “enabled” to enable the transmission of Email Notifications in response to a Relay Output Usage Alarm. If a Usage Alarm Email Notification is desired then this key or the `Events/OnUsage/Email` key must be defined and “enabled”. An optional EmailBlock may be defined specifying the Email format to be used.

IO/Outputs/rout#/Usage/HoldOff

This defines the amount of time in milliseconds that the Relay Output usage alarm must remain clear before any subsequent usage alarm on this input will be acted upon. The default is 30000 or 5 minutes. When an alarm occurs the services associated with that event are performed. The alarm must reset and remain so for this amount of time before those actions would be performed again.

IO/Outputs/rout#/Usage/OnAlarm

This key may be optionally defined and set to “disabled” to disable services related to the occurrence of Relay Output Usage Alarms on this output.

IO/Outputs/rout#/Slave

Each **JNIOR** Relay Output can track another Digital Input or Relay Output located on a remote **JNIOR**. In this case the relay output that is tracking is said to be *slaved* to the other I/O point. With proper configuration two **JNIOR** devices can be set up as an I/O relay through the network. In this case perhaps an input on one controls the relay output on the other and vice versa. This function is suitable for very low frequency signals (switches) as there can be considerable delay between the change of an input state and the reaction of the slaved relay. Within a small LAN this might be roughly ½ second but through Internet over some distance there may be a few seconds before the slave response occurs.

The I/O Slave Registry key is used to establish the link with the remote I/O point. This is done as part of the **JNIOR** boot. The connection is maintained and reestablished as would be necessary to continually achieve the tracking. Changes in Slave keys do not take effect until the **JNIOR** is rebooted. Use of the `reboot` command is recommended as this unloads the Registry and insures that all changes are saved prior to restarting the **JNIOR**.

The `IO/Outputs/rout#/Slave` key has the following multi-part format:

```
IO/Outputs/rout#/Slave = IPAddress, nPort, UserName, Password[, IOPoint]
```

Where:

`IPAddress`	Defines the hostname or IP address of the remote JNIOR. This will be processed through DNS if necessary. It is a required entry.
`nPort`	The TCP/IP Port number on the remote JNIOR servicing the JNIOR Protocol. If left blank, 9200 will be assumed.
`UserName`	The username of a valid user on the remote JNIOR. This is a required entry.

`Password`	The valid password associated with the above username. JNIOR must be able to successfully log into the remote JNIOR in order to track its I/O. This is a required entry.

`IOPoint`	Defines the remote I/O point to track on the above JNIOR. This has to be precisely one of the following:
	`din1` `rout1`
	`din2` `rout2`
	`din3` `rout3`
	`din4` `rout4`
	`din5` `rout5`
	`din6` `rout6`
	`din7` `rout7`
	`din8` `rout8`
	If omitted the Digital Input corresponding to the associated `rout#` will be assumed.

For instance the following entries cause Relay Output 2 and Relay Output 3 to track the state of Digital Input 1 on the **JNIOR** at 10.0.0.223:

```
IO/Outputs/rout2/Slave = 10.0.0.223, 9200, jnior, jnior, din1
IO/Outputs/rout3/Slave = 10.0.0.223, 9200, jnior, jnior, din1
```

Note that multiple relays can track the same I/O point and relays can track I/O points on multiple **JNIOR** units. The requirement is that all of the **JNIOR**s are running a version 2.01.323 or later Operating System and have the JNIOR Protocol enabled.

Should communications be lost and the **JNIOR** is unable to reestablish the connection with several seconds the Relay Outputs affected will go into Safe Mode and turn off (open). These outputs will again reflect the status of the remote I/O Points once the connection can be reestablished. Refer to the `jniorsys.log` file for a record of events associated with connection problems.

IO/Outputs/rout#/OnDelay

Each relay output may be configured with a Turn-On Delay. The value defines the number of milliseconds of delay. This may be used for example to insure that the signal driving the relay is stable for a minimum period of time before closing the relay. The On-Delay affects only activating/closing the relay. Any command to open the relay takes immediate effect.

For example, suppose that a **JNIOR** is configured to monitor the signal driving a control system status lamp. When a process begins the lamp flashes for a start-up period and then illuminates continuously while processing proceeds. If we require a single relay closure during processing but not during the start-up period you could *slave* (See Slave registry key) a relay output to the input connected to the lamp signal and set the following. This value was selected assuming that the lamp flashes with roughly a ½ second period.

```
IO/Outputs/rout#/OnDelay = 750
```

In this case the lamp must be on for at least ¾ of a second before the relay will activate/close. Since the lamp flashes only briefly during start-up the relay will not activate. Note that this relay activates 750 milliseconds after the lamp signal goes continuous. This delay may need to be considered if large OnDelay settings are used.

OnDelay is accurate to +/- 0.5 milliseconds. Delays of several hours may be configured. This delay is not operative in conjunction with pulsing functions. To disable OnDelay operation set this registry key to 0. If the key is removed a reboot will be required to insure that any previously active OnDelay setting is disabled.

Jnior OS v4.1.830 and later

IO/Outputs/rout#/$TriggerCount

NOTE: This key was IO/Outputs/rout#/TriggerCount prior to release 3.3.x.y

Each **JNIOR** Relay Output can be configured to close upon reaching a defined number of counts on its associated Digital Input (din1 for rout1, din2 for rout2, etc.). If defined this Registry Key enables this high speed control feature and specifies the desired trigger count. For instance:

```
IO/Outputs/rout8/$TriggerCount = 1500
```

Assuming that the input counter for Digital Input 8 has been cleared, Relay Output 8 will close immediately upon seeing the 1500th low-to-high transition on Digital Input 8. The relay closes in well under 1 millisecond of receipt of the triggering transition. **JNIOR** can process input pulses at a rate as high as 2,000 per second. Input counters are 4-byte integer values which can tally as many as 4.3 billion transitions.

Once the relay closes it will remain closed until commanded by the user's application to open (or power is lost). If the user's application does not increase the TriggerCount value or reset the Digital Input Counter, the relay will again close on the next low-to-high transition of the associated input. The relay is commanded to close on each and every low-to-high transition resulting in an input counter value equal to or greater than the defined trigger count.

Setting TriggerCount to 0 or removing the key entirely for any relay disables this feature for that relay. By default the feature is disabled on all relays. Relays will remain closed until commanded separately by the user's application or manually through the applets or other means. The trigger count is configured by writing the Registry Key. This occurs at the moment the key is written. This is true now for all I/O configurations whereas previously (prior to v2.13.83) it may have taken up to a minute after adjusting Registry settings for the I/O configuration to update. This is true no matter how the Registry Key is written either through a protocol, the applets, Registry Editor, or by command at the command prompt.

`IO/Outputs/rout#/ForcePulseDuration`

Each **JNIOR** Relay Output can be configured to pulse for a configured number of milliseconds instead of performing the requested close open or toggle commands. This will act like a fail safe and was implemented to act as a pulse when a pulse command cannot be sent.

This key, if defined, causes the JNIOR to pulse the output for the specified number of milliseconds instead of doing the requested open or close. If an open was requested then a pulse low occurs. Alternatively if a close was requested a pulse high will occur.

9.12. Sensor Port

One or more One-Wire devices may be connected through the external one-wire port or *Sensor Port*. JNIOR detects these devices during boot and updates the Registry to identify new devices and information regarding those currently available. The following Registry keys apply. *Note: <address> represents the 16-character device address where appropriate below.*

`OneWire/Devices`

This entry contains the list of currently active (detected as of the most recent boot) One-Wire devices. This is a list of <address> separated by commas. If there are no devices available then the key will not be present.

`OneWire/<address>/Desc`

This provides a text description for the device whose address is <address>. The entry is created automatically when a new device is detected during boot. It can be edited as appropriate. The entry will remain in the Registry even after this device is removed. It can be manually deleted.

`OneWire/<address>/Name`

This provides a list of device names (DS1920 for a temperature sensor for instance) that are appropriate for the device type. The entry is created automatically when a new device is detected during boot. The entry will remain in the Registry even after this device is removed. It can be manually deleted.

`OneWire/<address>/Type`

This indicates the *type* of One-Wire device. For instance `Type=10` for a class of Temperature sensors.

`OneWire/<address>/Mount`

This defines the Flash Memory mount name for external devices containing Flash Memory support. For example a setting of ‘`panel`’ for a User Panel with the Flash Memory option will locate the files contained therein under the folder `panel/`. Note that if left undefined the external flash memory devices will be mounted in folders named: `extrn/` `extrn1/`, `extrn2/`, etc. External flash devices are automatically mounted during boot when they are discovered. See the `jrflash` command for information on maintaining flash memory devices and mounting devices at other times.

`OneWire/<address>/ModbusAddress`

This maps the associated Sensor Port device into the Modbus data memory. For instance `ModbusAddress=1000` will place the device’s ReadBlock starting at Modbus address 1000. An address must be manually assigned if you need access to the device through the Modbus protocol. Connected devices are discovered and entered into the Registry during boot. When new devices are added they will appear after reboot. The ModbusAddress mapping and device availability is established when connecting to the Modbus port.

`OneWire/<address>/$Value`

For certain devices (notably the Type 10 Temperature Sensors) this key will indicate the current value. In the case of a temperature sensor it will contain both the current Fahrenheit and Celsius readings. Note that ‘$’ keys are not posted to the jnior.ini file. These are dynamic values.

`OneWire/<address & 0x28>/TempResolution`

This key is used to set the resolution for the DS18B20 one wire device, type 0x28. The valid values are 9, 10, 11, and 12. They represent the number of bits used to calculate the temperature. A smaller value will cause the conversion time to be faster but less precise. The following table shows the effect of changing the resolution.

Resolution	9 bit	10 bit	11 bit	12 bit
Conversion Time (ms)	93.75	187.5	375	750
LSB (°C)	0.5	0.25	0.125	0.0625

9.13. File Compression

The JNIOR supports compressing files into zip file format. This can be enabled to achieve better logging capabilities. If compression is enabled then when a file reaches its maximum size and rolls over to a backup file, the backup file will then be compressed. Up to 5 compressed archive files can be contained within one zip file.

`Device/Compression`

Set this keys value to “enabled” to enable file compression. The compress.jnior application must also be resident on the JNIOR in the flash/ folder.

9.14. SNMP

SNMP, or Simple Network Management Protocol, is a protocol used for monitoring and configuring network devices. Network JNIOR discovery is possible with SNMP. Refer to the MIB file to see what objects are available.

`Snmp/Server`

Set this keys value to “enabled” to enable SNMP functionality.

`Snmp/CommunityName`

This key will define the community string used for JNIOR specific keys.

`IO/Inputs/SnmpTrapMask`

This key enables an SNMP Trap to be sent when the specified Inputs change states. This key represents an integer mask. 0 represents no inputs where 255 represents all 8 inputs. SNMP traps should not be used on an input that rapidly changes state. SNMP I/O traps have been implemented in the I/O logging portion of the JNIOR OS and therefore directly affect the speed of the logging capabilities. Set this keys value to “enabled” to have a trap sent on I/O change.

`IO/Inputs/din#/SnmpTrap`

This key enables an SNMP Trap to be sent when the Input state changes. SNMP traps should not be used on an input that rapidly changes state. SNMP I/O traps have been implemented in the I/O logging portion of the JNIOR OS and therefore directly affect the speed of the logging capabilities. Set this keys value to “enabled” to have a trap sent on I/O change.

`IO/Outputs/SnmpTrapMask`

This key enables an SNMP Trap to be sent when the specified Inputs change states. This key represents an integer mask. 0 represents none of the outputs will send a trap where 255 represents all 8 outputs will send trap messages. SNMP traps should not be used on an input that rapidly changes state. SNMP I/O traps have been implemented in the I/O logging portion of the JNIOR OS and therefore directly affect the speed of the logging capabilities. Set this keys value to “enabled” to have a trap sent on I/O change.

`IO/Outputs/rout#/SnmpTrap`

This key enables an SNMP Trap to be sent when the Output state changes. SNMP traps should not be used on an input that rapidly changes state. SNMP I/O traps have been implemented in the I/O logging portion of the JNIOR OS and therefore directly affect the speed of the logging capabilities. Set this keys value to “enabled” to have a trap sent on I/O change.

9.15. Default Key Settings

Registry Keys may be defined or undefined. An *undefined* key is one that has not been specifically assigned a value in the current Registry. In many cases the absence of a value for a key implies a *default*. In other situations the absence of the value defers the function to another key. Certain key settings may *override* others.

Index