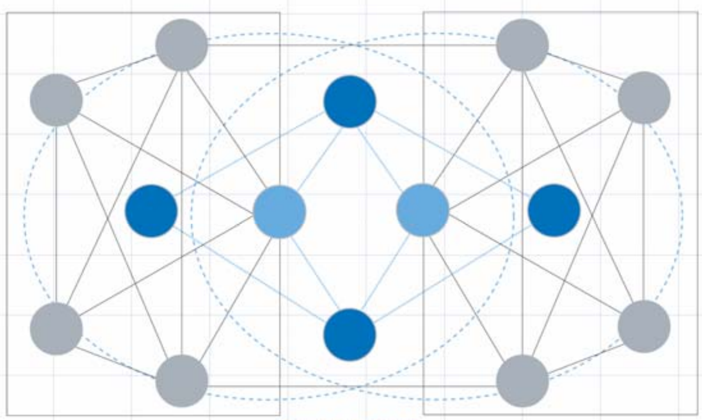
INSTEON
Developer's Guide 2nd Edition



smartlabs

August 16, 2007

Contents at a Glance

Full Table of Contents

Publication Dates

Date	Author	Version
06-21-05	Bill Morgan	Initial release.
10-14-05	Paul Darbee	First Edition.
03-15-07	Paul Darbee	Second Edition released for internal review.
08-16-07	Paul Darbee	Second Edition.

Preface to the Second Edition

Since the publication of the first edition of this *INSTEON Developer's Guide*, SmartLabs has made many improvements to INSTEON technology, thanks to real-world experience shipping over 300,000 INSTEON products, and constructive feedback from developers like you.

Because it has grown to book length, this second edition is now organized into chapters. Chapter 10 covering *INSTEON Modems* is all new. The glossary at the end can serve as a quick introduction to INSTEON for those new to the terminology.

The release of this second edition coincides with the release of the i2 INSTEON Engine. The most notable feature of the i2 Engine is an all-new i2/RF protocol documented in Chapter 6, *INSTEON Signaling Details*. Second-generation i2/RF replaces the original i1/RF protocol, which only the SmartLabs SignaLinc™ RF Signal Enhancer uses. Both i2/RF and i1/RF devices can coexist in the same INSTEON network because they operate on different radio frequencies.

The i2 Engine also fully enables Extended-length INSTEON messages. Other improvements include more robust broadcast messaging, an improved retry method following data collisions, and 115.2 KBaud, 32-byte-buffered serial communications with a host device.

SmartLabs is very happy to present this second edition of the *INSTEON Developer's Guide*. It is lengthy because it is comprehensive, but as with most reference works, you will not have to read the whole book through. Use the table of contents and the hyperlinks to get to the information you need quickly.

INSTEON is finding rapid acceptance among home builders, installers, and consumers alike. Most popular home-control software supports it, and INSTEON now powers a multitude of sensing and control devices. The momentum for INSTEON grows daily.

We are eager to hear back from you. Our goal is to make INSTEON so easy to use that it just becomes 'part of the plumbing,' enabling the end-to-end solutions that consumers really want and you are developing.

INTRODUCTION

A TV automatically turns on the surround sound amplifier, a smart microwave oven downloads new cooking recipes, a thermostat automatically changes to its energy saving setpoint when the security system is enabled, bathroom floors and towel racks heat up when the bath runs, an email alert goes out when there is water in the basement. When did the Jetson-style home of the future become a reality? When INSTEON™—the new technology standard for advanced home control—arrived. INSTEON enables product developers to create these distinctive solutions for homeowners, and other advantages yet unimagined, by delivering on the promise of a truly connected 'smart home.'

INSTEON is a cost-effective Dual Mesh™ network technology optimized for home management and control. INSTEON-networked Electronic Home Improvement™ products can interact with one another, and with people, in new ways that will improve the comfort, safety, convenience and value of homes around the world.

For a brief introduction to INSTEON see *Chapter 3 — INSTEON Overview* 14.

This Developer's Guide is part of the INSTEON Software and Hardware Development Kits that SmartLabs provides to Independent Software Vendors (ISVs) and Original Equipment Manufacturers (OEMs) who wish to create software and hardware systems that work with INSTEON.

In This INSTEON Developer's Guide

PART I — INSTEON BASICS

In Part I

Chapter 1 — Getting Started Quickly

INSTEON devices communicate by sending INSTEON messages over an INSTEON network. You can connect to an INSTEON network in two ways—with an INSTEON Modem (IM) module, such as the *The SmartLabs Powerline Modem* 29 (PLM) or chip (see *Chapter 10 — INSTEON Modems* 217), or with *The SmartLabs PowerLinc Controller* 28 (PLC).

The SmartLabs Powerline Modem™ (PLM) is an INSTEON device that also has a serial port that you connect to your PC (an Ethernet interface is under development). It uses an IN2680A Powerline Modem chip that offers a simple set of ASCII commands for interacting with INSTEON devices.

If you wish to build a custom INSTEON device using IM technology, you can interface an IN2680A Powerline Modem chip or an IN2682A RF Modem chip to a microcontroller of your choice. As an alternative, you can build a custom daughter board that fits within a PLM module. You can find hardware reference designs for such custom devices in *Chapter 13 — INSTEON Hardware Documentation* 358.

The SmartLabs PowerLinc V2 Controller™ (PLC) is an INSTEON network interface device that also has a serial port (RS232 or USB) that you connect to your PC. You can write applications that run on the PLC using tools documented in the *SALad Integrated Development Environment User's Guide* 287. If you wish, you can create applications that will run on the PLC in standalone mode without any connection to a PC.

In This Chapter

INSTEON Modem (IM) Quick Start

What to Look at First

For an accelerated introduction to using the SmartLabs Powerline Modem™ (PLM) or one of the INSTEON Modem chips to control and program INSTEON devices, follow these steps in sequence:

1. Review the *INSTEON Modem Applications*32 section and the *INSTEON Device Communication*21 diagram to see how things fit together.
2. Review the *IM Serial Communication Protocol*219 section to see how the serial protocol works.
3. Review the *IM Serial Commands*222 section to see how to use IM Serial Commands directly.
4. Review Chapter 5 — INSTEON Messages38 for more detailed information on the INSTEON protocol.
5. Review *Chapter 7 — INSTEON Device Networking*82 for details about INSTEON device categories, device ALL-Linking, and security issues.

IM-Related Summary Tables

Sections of this Developer's Guide that you will reference often are:

1. The *INSTEON Message Summary Table*46, which enumerates all possible INSTEON message types.
2. The *INSTEON Command Set Tables*124, which enumerate all of the INSTEON Commands that can appear in INSTEON messages.
3. The *IM Serial Command Summary Table*223 and *IM Serial Command Charts*227, which enumerate all of the commands for interacting serially with an INSTEON Modem.

PowerLinc Controller (PLC) Quick Start

What to Look at First

For an accelerated introduction to using the PLC and SALad to control and program INSTEON devices, follow these steps in sequence:

4. Review the *INSTEON SALad and PowerLinc Controller Architecture*35 nd *INSTEON Device Communication*21 diagrams to see how things fit together.
5. Review the *IBIOS Serial Communication Protocol and Settings*192 ection to see how the serial protocol works.
6. Review the *IBIOS Serial Command Examples*201 ection to see how to use IBIOS Serial Commands directly.
7. Review the *SALad IDE Quickstart*288 ection.
8. Review Chapter 5 — INSTEON Messages38 for more detailed information on the INSTEON protocol.
9. Review *Chapter 7 — INSTEON Device Networking*82 for details about INSTEON device categories, device ALL-Linking, and security issues.

PLC-Related Summary Tables

Sections of this Developer's Guide that you will reference often are:

1. The *INSTEON Message Summary Table*46, which enumerates all possible INSTEON message types.
2. The *INSTEON Command Set Tables*124, which enumerate all of the INSTEON Commands that can appear in INSTEON messages.
3. The *IBIOS Event Summary Table*185, which enumerates all of the events that IBIOS can generate.
4. The *IBIOS Serial Command Summary Table*197, which enumerates all of the commands for interacting serially with the PLC.
5. The *Flat Memory Map*170, which shows where everything is in the PLC's memory.
6. The *SALad Instruction Summary Table*281, which lists the SALad instruction set.

Chapter 2 — About This Developer's Guide

In This Chapter

Other Documents Included by Reference

Although this Developer's Guide is largely self-contained, there are aspects of INSTEON technology, such as listings of INSTEON Commands, INSTEON Device Categories, and INSTEON Product Keys, that require continuous updating as developers create new INSTEON products. Accordingly, SmartLabs maintains separate documents for that kind of information.

Readers should consider the documents listed in this section as part of this document. They are available for downloading at www.insteon.net.

INSTEON Conformance Specification

The *INSTEON Conformance Specification* identifies those aspects of INSTEON that assure interoperability with other INSTEON products. The Conformance Spec assumes that readers have already gained familiarity with INSTEON technology by reading this Developer's Guide.

INSTEON Command Tables Document

The current tables of INSTEON Commands are contained in a separate document titled *INSTEON Command Tables*, which is integral to both the *INSTEON Conformance Specification* and this Developer's Guide.

The filename for that document is *INSTEON Command Tables yyyymmddx.doc*, where *yyyy* is the year, *mm* is the month, *dd* is the day, and *x* is a daily version letter beginning with *a*. Be sure to refer to the document with the latest date.

As a convenience, the tables contained in the version of that document that was current as of the publication date of this Developer's Guide are reprinted herein, in the section *INSTEON Command Set Tables* 124 of *Chapter 8 — INSTEON Command Set* 114.

INSTEON Device Categories and Product Keys Document

The current table of INSTEON Device Categories (DevCats), Subcategories (SubCats), and INSTEON Product Keys (IPKs) is contained in a separate document titled *INSTEON Device Categories and Product Keys*, which is also integral to both the *INSTEON Conformance Specification* and this Developer's Guide.

The filename for that document is *INSTEON DevCats and Product Keys yyyymmddx.doc*, where *yyyy* is the year, *mm* is the month, *dd* is the day, and *x* is a daily version letter beginning with *a*. Be sure to refer to the document with the latest date.

As a convenience, the tables contained in the version of that document that was current as of the publication date of this Developer's Guide are reprinted herein, in the sections *Currently Defined Device Categories* 83 and *INSTEON Product Key and SubCat Assignments* 88 of *Chapter 7 — INSTEON Device Networking* 82.

INSTEON Modem Spec Sheets

Developers will find the latest specifications for INSTEON modem ICs at www.insteon.net.

IN2680A INSTEON Direct Powerline Modem Interface

The IN2680A is a one-chip solution that uses a simple ASCII serial interface to connect a host device or system to an INSTEON network via the powerline.

IN2682A INSTEON Direct RF Modem Interface

The IN2682A is similar to the IN2680A Powerline Modem except that it connects to an INSTEON network via radio.

Other INSTEON Documents of Interest

Developers can find additional information about INSTEON in the following white papers.

INSTEON, the Details

This white paper (downloadable from www.insteon.net/pdf/insteondetails.pdf) is an earlier account of the inner workings of INSTEON technology. For the latest information covering all aspects of INSTEON in greater depth, however, developers should refer to this Developer's Guide.

INSTEON Compared

This white paper (downloadable from www.insteon.net/pdf/insteoncompared.pdf) compares INSTEON to other technologies for home control, such as ZigBee, Z-Wave, WiFi, HomePlug, and X10.

Document Conventions

The following table shows the typographic conventions used in this *INSTEON Developer's Guide*.

Convention	Description	Example
Monospace	Indicates source code, code examples, code lines embedded in text, and variables and code elements	`DST EQU 0x0580`
Angle Brackets < and >	Indicates user-supplied parameters	<Address MSB>
Vertical Bar |	Indicates a choice of one of several alternatives	<0x06 (ACK) | 0x15 (NAK)>
Ellipsis …	Used to imply additional text	"Text …"
0xNN	Hexadecimal number	0xFF, 0x29, 0x89AB
⇒	Range of values, *including* the beginning and ending values	0x00 ⇒ 0x0D
Hyperlinks	Links to other parts of this document or to the Internet	Document Conventions
Subscripts	Page number references for hyperlinks	Document Conventions$_{11}$

Getting Help

INSTEON Support

SmartLabs is keenly interested in supporting the community of INSTEON developers. If you are having trouble finding the answers you need in this *INSTEON Developer's Guide*, you can get more help by accessing the INSTEON Developer's Forum or by emailing sdk@insteon.net.

INSTEON Developer's Forum

When you purchased the INSTEON Software Developer's Kit, you received a username and password for accessing the Developer's Forum at http://insteon.net/sdk/forum. The Forum contains a wealth of information for developers, including

- Frequently asked questions
- Software downloads, including the SALad IDE (Integrated Development Environment), and SmartLabs Device Manager
- Documentation updates
- Sample code
- Discussion forums

Providing Feedback

To provide feedback about this documentation, go to the INSTEON Developer's Forum or send email to sdk@insteon.net.

Legal Information

Terms of Use

This *INSTEON Developer's Guide* is supplied to you by SmartLabs, Inc. (SmartLabs) in consideration of your agreement to the following terms. Your use or installation of this *INSTEON Developer's Guide* constitutes acceptance of these terms. If you do not agree with these terms, please do not use or install this *INSTEON Developer's Guide*.

In consideration of your agreement to abide by the following terms, and subject to these terms, SmartLabs grants you a personal, non-exclusive license, under SmartLabs' intellectual property rights in this *INSTEON Developer's Guide*, to use this *INSTEON Developer's Guide*; provided that no license is granted herein under any patents that may be infringed by your works, modifications of works, derivative works or by other works in which the information in this *INSTEON Developer's Guide* may be incorporated. No names, trademarks, service marks or logos of SmartLabs, Inc. or INSTEON may be used to endorse or promote products derived from the *INSTEON Developer's Guide* without specific prior written permission from SmartLabs, Inc. Except as expressly stated herein, no other rights or licenses, express or implied, are granted by SmartLabs and nothing herein grants any license under any patents except claims of SmartLabs patents that cover this *INSTEON Developer's Guide* as originally provided by SmartLabs, and only to the extent necessary to use this *INSTEON Developer's Guide* as originally provided by SmartLabs. SmartLabs provides this *INSTEON Developer's Guide* on an "AS IS" basis.

SMARTLABS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THIS *INSTEON DEVELOPER'S GUIDE* OR ITS USE, ALONE OR IN COMBINATION WITH ANY PRODUCT.

IN NO EVENT SHALL SMARTLABS BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) ARISING IN ANY WAY OUT OF THE USE, REPRODUCTION, MODIFICATION AND/OR DISTRIBUTION OF THIS *INSTEON DEVELOPER'S GUIDE*, HOWEVER CAUSED AND WHETHER UNDER THEORY OF CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY OR OTHERWISE, EVEN IF SMARTLABS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Trademarks and Patents

SmartLabs, Smarthome, INSTEON, Dual Mesh, BiPHY, ALL-Link, Powerline Modem, PowerLinc, ControLinc, LampLinc, SwitchLinc, RemoteLinc, Electronic Home Improvement, SmartLabs Device Manager, Home Network Language, and Plug-n-Tap are trademarks of SmartLabs, Inc.

INSTEON networking technology is covered by pending U.S. and foreign patents.

Copyright

Revision History

Release Date	Author	Description
03-15-07	PVD	2nd Edition printed for review, 20 copies.
03-27-07	PVD	Fixed bytecount in IM Command **0x62** *Send INSTEON Standard or Extended Message*.
03-28-07	PVD	Added IM Command **0x58** *ALL-Link Cleanup Status Report*.
03-29-07	PVD	Updated explanation of IM Command **0x6F** *Manage ALL-Link Record*.
04-02-07	PVD	Updated explanation of IM Commands **0x61** *Send ALL-Link Command*, **0x56** *ALL-Link Cleanup Failure Report*, and **0x58** *ALL-Link Cleanup Status Report*.
04-06-07	PVD	IM Command **0x58** *ALL-Link Cleanup Status Report* also sent when IM interrupts its own Cleanup sequence.
04-17-07	PVD	Corrected <X10 Flag> value in IM Commands **0x63** *Send X10* and **0x52** *X10 Received*.
06-06-07	PVD	Clarified IBIOS Command **0x4F** *INSTEON Message Received* after NAK.
06-14-07	PVD	Corrected nominal INSTEON powerline packet timing (-876 to 947 µs).
08-14-07	PVD	Updated *INSTEON Command Set Tables* section from *INSTEON Command Tables 20070816a.doc*, and *INSTEON Product Key and SubCat Assignments* section from *INSTEON DevCats and Product Keys 20060814a.doc*. INSTEON Command **ED 0x2F00** *Read/Write ALDB* is now required for i2.

Chapter 3 — INSTEON Overview

INSTEON enables simple, low-cost devices to be networked together using the powerline, radio frequency (RF), or both. All INSTEON devices are peers, meaning that any device can transmit, receive, or repeat[1] other messages, without requiring a master controller or complex routing software. Adding more devices makes an INSTEON network more robust, by virtue of a simple protocol for communication retransmissions and retries. On the powerline, INSTEON devices are compatible[2] with legacy X10 devices.

This chapter explains why INSTEON has these properties and explains them further without going into the details.

In This Chapter

Why INSTEON?

INSTEON is the creation of SmartLabs, the world's leading authority on electronic home improvement. SmartLabs is organized into three divisions—Smarthome.com, "the Amazon of electronic home improvement," SmartLabs Design, creators of best-in-class home control products, and SmartLabs Technology, the pioneering architects of INSTEON. With Smarthome.com's global distribution channel, SmartLabs Design's product development and manufacturing resources, and SmartLabs Technology's ongoing innovation, SmartLabs is uniquely positioned to support and encourage INSTEON product developers.

But why did SmartLabs undertake the complex task of creating an entirely new home-control networking technology in the first place?

SmartLabs has been a leading supplier of devices and equipment to home automation installers and enthusiasts since 1992. Now selling over 5,000 products into more than 130 countries, SmartLabs has first-hand experience dealing directly with people all over the world who have installed lighting control, whole-house automation, security and surveillance systems, pet care devices, gadgets, and home entertainment equipment. Over the years, by talking to thousands of customers through its person-to-person customer support operation, SmartLabs has become increasingly concerned about the mismatch between the dream of living in a responsive, aware, automated home and the reality of existing home-control technologies.

Today's homes are stuffed with high-tech appliances, entertainment gear, computers, and communications gadgets. Utilities, such as electricity, lighting, plumbing, heating and air conditioning are so much a part of modern life that they almost go unnoticed. But these systems and devices all act independently of each other—there still is nothing that can link them all together. Houses don't know that people live in them. Lights happily burn when no one needs them, HVAC is insensitive to the location and comfort of people, pipes can burst without anyone being notified, and sprinklers dutifully water the lawn even while it's raining.

For a collection of independent objects to behave with a unified purpose, the objects must be able to communicate with each other. When they do, new, sometimes-unpredictable properties often emerge. In biology, animals emerged when nervous systems evolved. The Internet emerged when telecommunications linked computers together. The global economy emerges from transactions involving a staggering amount of communication. But there is no such communicating infrastructure in our homes out of which we might expect new levels of comfort, safety and convenience to emerge. There is nothing we use routinely in our homes that links our light switches or our door locks, for instance, to our PCs or our remote controls.

It's not that such systems don't exist at all. Just as there were automobiles for decades before Henry Ford made cars available to everyone, there are now and have been for some time systems that can perform home automation tasks. On the high end, all kinds of customized systems are available for the affluent, just as the rich could buy a Stanley Steamer or a Hupmobile in the late 1800s. At the low end, X10 powerline signaling technology has been around since the 1970s, but its early adoption is its limiting factor—it is too unreliable and inflexible to be useful as an infrastructure network.

SmartLabs is a major distributor of devices that use X10 signaling. In 1997, aware of the reliability problems its customers were having with X10 devices available at the time, SmartLabs developed and began manufacturing its own ‘Linc’ series of improved X10 devices, including controllers, dimmers, switches, computer interfaces and signal boosters. Despite the enhanced performance enjoyed by Linc products, it was still mostly do-it-yourselfers and hobbyists who were buying and installing them.

SmartLabs knew that a far more robust and flexible networking standard would have to replace X10 before a truly intelligent home could emerge. SmartLabs wanted a technology that would meet the simplicity, reliability, and cost expectations of the masses—mainstream consumers who want immediate benefits, not toys.

In 2001, SmartLabs’ engineers were well aware of efforts by others to bring about the home of the future. The aging X10 protocol was simply too limiting with its tiny command set and unacknowledged, ‘press and pray’ signaling over the powerline. CEBus had tried to be everything to everybody, suffering from high cost due to overdesign by a committee of engineers. Although CEBus did become an official standard (EIA-600), developers did not incorporate it into real-world products.

Radio-only communication protocols, such as Z-Wave and ZigBee, not only required complex routing strategies and a confusing array of different types of network masters, slaves, and other modules, but radio alone might not be reliable enough when installed in metal switch junction boxes or other RF-blocking locations.

Bluetooth radio has too short a range, WiFi radio is too expensive, and high-speed powerline protocols are far too complex to be built into commodity products such as light switches, door locks, or thermostats. Overall, it seemed that everything proposed or available was too overdesigned and therefore would cost too much to become a commodity for the masses in the global economy.

So, in 2001, SmartLabs decided to take its destiny into its own hands and set out to specify an ideal home control network, one that would be simple, robust and inexpensive enough to link everything to everything else. INSTEON was born.

Hallmarks of INSTEON

These are the project pillars that SmartLabs decided upon to guide the development of INSTEON. Products networked with INSTEON had to be:

Instantly Responsive

INSTEON devices respond to commands with no perceptible delay. INSTEON's signaling speed is optimized for home control—fast enough for quick response, while still allowing reliable networking using low-cost components.

Easy to Install

Installation in existing homes does not require any new wiring, because INSTEON products communicate over powerline wires or they use the airwaves. Users never have to deal with network enrollment issues because all INSTEON devices have an ID number pre-loaded at the factory—INSTEON devices join the network as soon as they're powered up.

Simple to Use

Getting one INSTEON device to control another is very simple—just press and hold a button on each device for 10 seconds, and they're linked. This ALL-Linking™ procedure guarantees that any INSTEON Controller can operate any INSTEON Responder, now and in the future. Because INSTEON messaging is two-way, INSTEON Controllers can confirm that commands get through, making INSTEON products dependable and 'guest friendly.'

Reliable

An INSTEON network becomes more robust and reliable as it is expanded because every INSTEON device repeats[1] messages received from other INSTEON devices. Dual Mesh™ communications using both the powerline and the airwaves ensures that there are multiple pathways for messages to travel. Whether by radio or powerline, INSTEON messages get repeated *in unison* whenever multiple INSTEON devices hear them. This *message simulcasting* is like an entire chorus singing a melody at once instead of one singer at a time—the 'music' is much easier to hear.

Affordable

INSTEON software is simple and compact, because all INSTEON devices send and receive messages in exactly the same way, without requiring a special network controller or complex routing algorithms. The cost of networking products with INSTEON is held to at an absolute minimum because INSTEON is designed specifically for home control applications, and not for transporting large amounts of data.

Compatible with X10

INSTEON and X10 signals can coexist with each other on the powerline without mutual interference. Designers are free to create hybrid INSTEON/X10 products that operate equally well in both environments, allowing current users of legacy X10 products to easily upgrade to INSTEON without making their investment in X10 obsolete.

INSTEON Specifications

The most important property of INSTEON is its no-frills simplicity.

INSTEON messages are fixed in length and synchronized to the AC powerline zero crossings. They do not contain routing information beyond a source and destination address. INSTEON is reliable and affordable because it is optimized for command and control, not high-speed data transport. INSTEON allows infrastructure devices like light switches and sensors to be networked together in large numbers, at low cost. INSTEON stands on its own, but can also bridge to other networks, such as WiFi LANs, the Internet, telephony, and entertainment distribution systems. Such bridging allows INSTEON to be part of very sophisticated integrated home control environments.

The following table shows the main features of INSTEON at a glance.

INSTEON Property	Specification	
Network	Dual Mesh™ (RF and powerline) Peer-to-Peer Mesh Topology Unsupervised No routing tables	
Protocol	All devices are two-way simulcasting Repeaters[1] Messages acknowledged Retry if not acknowledged Synchronized to powerline	
X10 Compatibility[2]	INSTEON devices can send and receive X10 Commands INSTEON devices do not repeat or amplify X10	
Data Rate	Instantaneous	13,165 bits/sec
	Sustained	2,880 bits/sec
Message Types	Standard-length	10 Bytes
	Extended-length	24 Bytes
Message Format, Bytes	From Address	3
	To Address	3
	Flags	1
	Command	2
	User Data	14 (Extended Messages only)
	Message Integrity	1
Devices Supported	Unique IDs	16,777,216
	Product Keys	16,777,216
	Device Categories	256
	Commands	33,554,432
	ALL-Link Groups per Controller Device	256

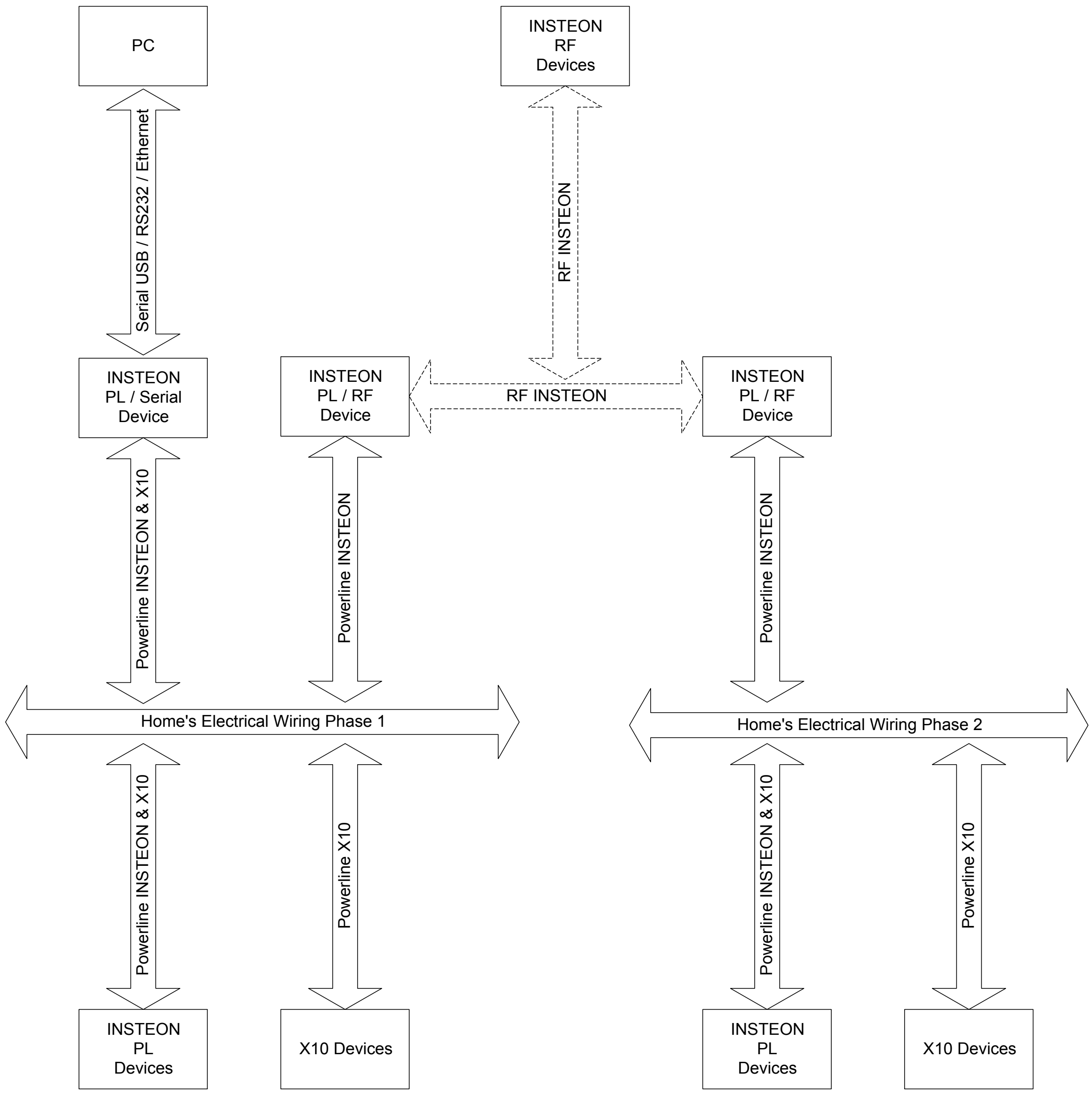
INSTEON Property	Specification	
	Members within an ALL-Link Group	Limited only by memory
INSTEON Engine Memory Requirements	RAM	80 Bytes
	ROM	3K Bytes
Typical INSTEON Application Memory Requirements (Light Switch, Lamp Dimmer)	RAM	256 Bytes
	EEPROM	256 Bytes
	Flash	7K Bytes
Device Installation	Plug-in Wire-in Battery Operated	
Device Setup	Plug-n-Tap™ manual ALL-Linking™ PC or Controller	
Security	Physical device possession Address masking Encrypted message payloads	
Application Development	INSTEON Modem chips and modules IDE (Integrated Development Environment) SALad interpreted language Software and Hardware Development Kits	
Powerline Physical Layer	Frequency	131.65 KHz
	Modulation	BPSK
	Min Transmit Level	3.16 Vpp into 5 Ohms
	Min Receive Level	10 mV
	Phase Bridging	INSTEON RF or hardware
i2/RF Physical Layer	Frequency	915 MHz
	Modulation	FSK
	Sensitivity	-103 dbm
	Range	300 ft unobstructed line-of-sight, half-wave dipole antenna, 0.1 raw bit-error rate

INSTEON Fundamentals

In This Section

INSTEON Device Communication

Devices communicate with each other using the INSTEON protocol over the air via radio frequency (RF) and over the powerline as illustrated below.



Electrical power is most commonly distributed to homes in North America as split-phase 220-volt alternating current (220 VAC). At the main electrical junction box to the home, the single three-wire 220 VAC powerline is split into a pair of two-wire 110 VAC powerlines, known as Phase 1 and Phase 2. Phase 1 wiring usually powers half the circuits in the home, and Phase 2 powers the other half.

INSTEON devices communicate with each other over the powerline using the INSTEON Powerline protocol, which will be described in detail below (see Chapter 5 — INSTEON Messages38 and *Chapter 6 — INSTEON Signaling Details*56).

Existing X10 devices also communicate over the powerline using the X10 protocol. The INSTEON Powerline protocol is compatible[2] with the X10 protocol, meaning that designers can create INSTEON devices that can also listen and talk to X10 devices. X10 devices, however, are insensitive to the INSTEON Powerline protocol.

INSTEON devices containing RF hardware may optionally communicate with other INSTEON RF devices using the INSTEON RF protocol.

INSTEON BiPHY™ devices (those that can use *both* the INSTEON Powerline protocol and the INSTEON RF protocol) solve a significant problem encountered by devices that can only communicate via the powerline. Powerline signals originating on the opposite powerline phase from a powerline receiver are severely attenuated, because there is no direct circuit connection for them to travel over.

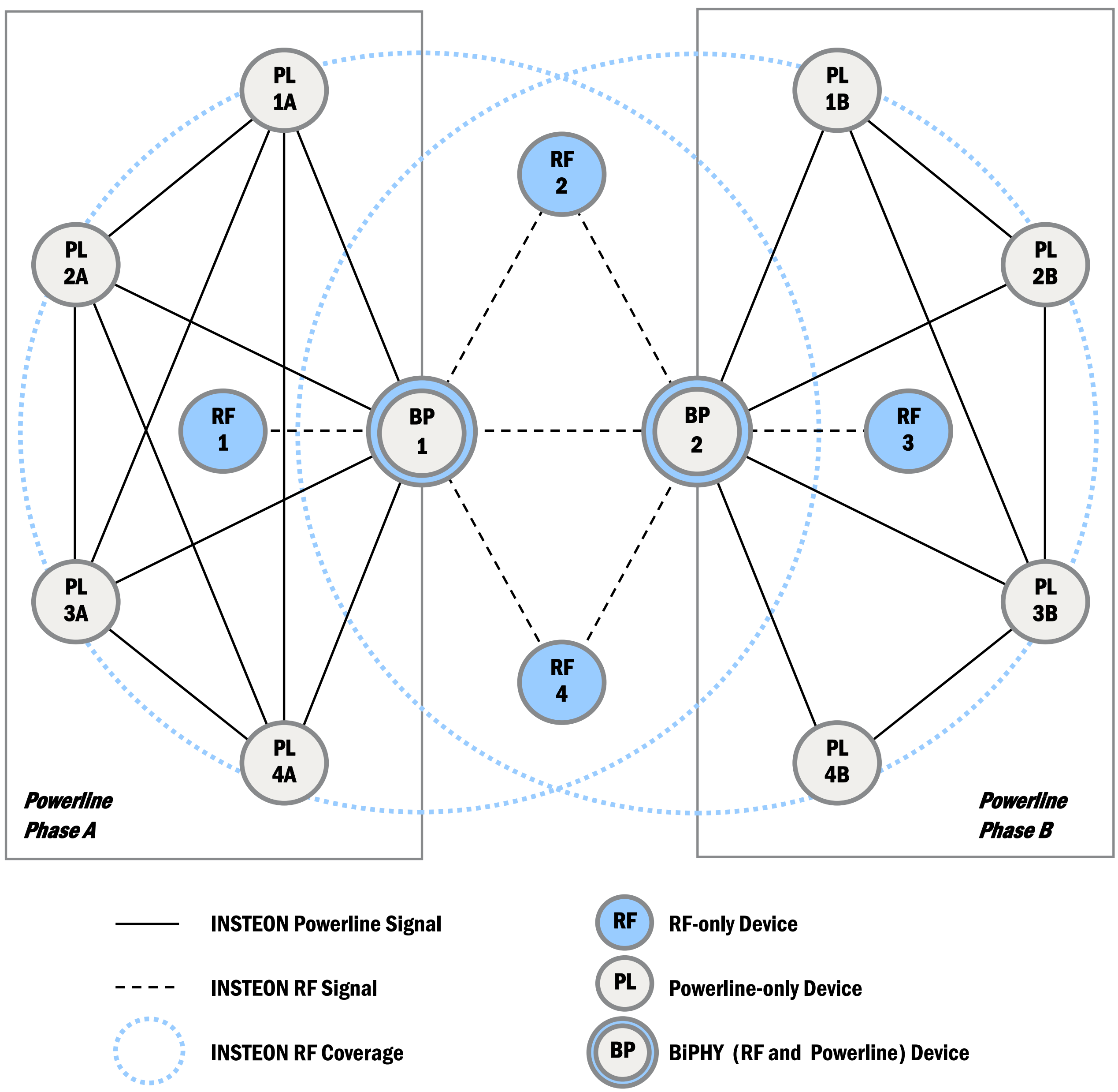
A traditional solution to this problem is to connect a signal coupling device between the powerline phases, either by hardwiring it in at a junction box or by plugging it into a 220 VAC outlet. INSTEON automatically solves the powerline phase coupling problem through the use of INSTEON BiPHY devices capable of both powerline and RF messaging. INSTEON RF messaging bridges the powerline phases whenever at least one INSTEON PL/RF device is installed on each powerline phase.

When suitably equipped with a dedicated serial interface, such as USB, RS232, or Ethernet, INSTEON devices can also interface with computers and other digital equipment. In the figure above, an INSTEON PL/Serial device is shown communicating with a PC using a serial link. In the Software Developer's Kit, that device is a SmartLabs PowerLinc™ V2 Controller with a USB or RS232 interface (see *The SmartLabs PowerLinc Controller* 28 and *Chapter 9 — INSTEON BIOS (IBIOS)* 166).

Serial communications can bridge networks of INSTEON devices to otherwise incompatible networks of devices in a home, to computers, to other nodes on a local-area network (LAN), or to the global Internet. Such connections to outside resources allow networks of INSTEON devices to exhibit complex, adaptive, people-pleasing behaviors. INSTEON devices capable of running downloadable SALad Applications (see *Chapter 11 — SALad Language Documentation* 263) can be upgraded to perform very sophisticated functions, including functions not envisioned at the time of manufacture or installation.

INSTEON Message Repeating

The figure below shows how network reliability improves when additional INSTEON devices are added. The drawing shows INSTEON devices that communicate by powerline-only (PL), RF-only (RF), and both (BiPHY™ or BP).



Every INSTEON device is capable of repeating[1] INSTEON messages. They will do this automatically as soon as they are powered up—they do not need to be specially installed using some network setup procedure. Adding more devices not only increases the number of available pathways for messages to travel, but it also increases the signal strength of repeated messages because every device that hears a message repeats it in unison with all other devices that heard the same message. *Path diversity* and *simulcasting* both result in a higher probability that a message will arrive at its intended destination, so the more devices in an INSTEON network, the better.

As an example, suppose RF device **RF1** desires to send a message to **RF3**, but **RF3** is out of range. The message will still get through, however, because devices within range of **RF1**, say **BP1** and **RF2**, will receive the message and retransmit it by simulcasting to other devices within range of themselves. In the drawing, **BP1** might reach **RF2**, **BP2**, and **RF4**, and devices **BP2** and **RF1** might be within range of the intended recipient, **RF3**. Therefore, there are many ways for a message to travel: **RF1** to **RF2** to **RF3** (1 retransmission), **RF1** to **BP1** to **BP2** to **RF3** (2 retransmissions), and **RF1** to **BP1** to **RF2** to **BP2** to **RF3** (3 retransmissions) are some examples.

On the powerline, path diversity has a similar beneficial effect. For example, the drawing shows powerline device **PL1B** without a direct communication path to device **PL4B**. In the real world, this might occur because of signal attenuation problems or because a direct path through the electric wiring does not exist. But a message from **PL1B** will still reach **PL4B** by taking a path through **BP2** (1 retransmission), through **PL2B** to **BP2** (2 retransmissions), or through **PL2B** to **BP2** to **PL3B** (3 retransmissions).

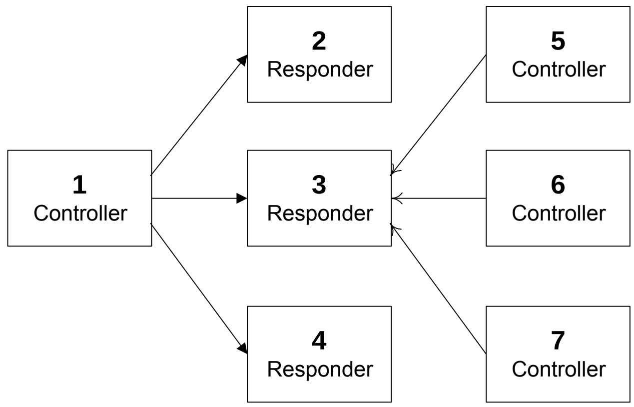
The figure also shows how messages can travel among powerline devices that are installed on different phases of a home's wiring. To accomplish phase bridging, at least one INSTEON BiPHY RF/powerline device must be installed on each powerline phase. In the drawing, BiPHY device **BP1** is installed on phase A and **BP2** is installed on phase B. Direct RF paths between **BP1** to **BP2**, or indirect paths using **RF2** or **RF4** (1 retransmission) allow messages to propagate between the powerline phases, even though there is no direct electrical connection.

With all devices repeating messages, there must be some mechanism for limiting the number of times that a message may be retransmitted, or else messages might propagate forever within the network. Network saturation by repeating messages is known as a 'data storm.' The INSTEON protocol avoids this problem by limiting the maximum number of times an individual message may be retransmitted to three (see *INSTEON Message Hopping* 49).

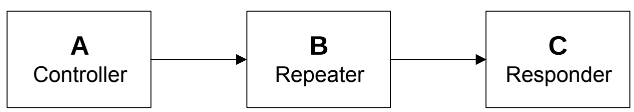
INSTEON Peer-to-Peer Networking

All INSTEON devices are peers, meaning that any device can act as a Controller (sending messages), Responder (receiving messages), or Repeater[1] (relaying messages).

This relationship is illustrated in the figure below, where INSTEON device **1**, acting as a Controller, sends messages to multiple INSTEON devices **2**, **3**, and **4** acting as Responders. Multiple INSTEON devices **5**, **6**, and **7** acting as Controllers can also send messages to a single INSTEON device **3** acting as a Responder.



Any INSTEON device can repeat[1] messages, as with device **B**, below, which is shown relaying a message from device **A** acting as a Controller to device **C** acting as a Responder.



INSTEON ALL-Linking

ALL-Linking allows any INSTEON Controller device to operate any INSTEON Responder device, even if the Controller does not know any of the Direct Commands that the Responder can execute. The principle is simple—during ALL-Linking to a button on a Controller, a Responder memorizes the state that it is in at the time. After ALL-Linking, pushing that button on the Controller causes the Responder to go back into the state that it memorized when it ALL-Linked.

When a button on a Controller ALL-Links to a Responder, the Controller creates an *ALL-Link Group*, which the Responder joins. Multiple Responders can join the same ALL-Link Group, so it is possible for a single button push to cause an entire ensemble of devices to recall their memorized states.

All of the Responder devices in the ALL-Link Group will recall their memorized states *simultaneously*, because when the Controller's button is pushed, the Controller first sends out an *ALL-Link Broadcast* message to all of the Group members at once, followed by individual *ALL-Link Cleanup* messages to each Group member in turn.

Chapter 4 — INSTEON Application Development Overview

INSTEON, with its no-nonsense emphasis on simplicity, reliability, and low cost, is optimized as an *infrastructure* network. Common devices in the home, such as light switches, door locks, thermostats, clocks, and entertainment systems currently do not communicate with one another. INSTEON can change all that.

When devices are networked together, there is a potential for coordinated, adaptive behavior that can bring a new, higher level of comfort, safety, and convenience to living. But networking devices together cannot by itself change the behavior of the devices. It is application-level software, created by developers, that transforms a network of previously unrelated devices into a coordinated, adaptive, lifestyle-enhancing system.

There are two basic kinds of applications that developers can create for INSTEON-networked devices: External Applications and Internal Applications.

External Applications run on a computing device such as a PC or PDA. A special type of INSTEON module called an INSTEON *Bridge* connects the computing device to an INSTEON network. *Manager Apps* are External Applications that exchange INSTEON messages directly with INSTEON devices via a Bridge.

Internal Applications run on INSTEON devices themselves. There are two ways to create an Internal Application for an INSTEON device: you can write it to run on the microcontroller of your choice and connect serially to an INSTEON network using an INSTEON Modem (IM) chip, or you can write it in SmartLabs' embedded interpreted language, called SALad, which resides in the firmware of SALad-enabled INSTEON devices.

In This Chapter

Interfacing to an INSTEON Network

An INSTEON device that connects an INSTEON network to the outside world is called an *INSTEON Bridge*. There can be many kinds of INSTEON Bridges. One kind, an INSTEON-to-Serial Bridge, connects an INSTEON network to a computing device like a PC, a PDA, or a dedicated home-control module with a user interface and a serial port. Another kind of Bridge, INSTEON-to-IP, connects an INSTEON network to a LAN or the Internet, either with wires (like Ethernet) or wirelessly (like WiFi). Still other INSTEON Bridges could connect to other networks such as wired or wireless telephony, Bluetooth, ZigBee, WiMax, or whatever else emerges in the future.

The SmartLabs PowerLinc Controller

The PowerLinc™ V2 Controller (PLC) from SmartLabs is an example of an INSTEON-to-Serial Bridge for connecting an INSTEON network to a computing device. PLCs are currently available with either a USB or an RS232 serial interface. An Ethernet interface, for connecting to a LAN or the Internet, is under development. For comprehensive information about the firmware capabilities of the PLC, see *Chapter 9 — INSTEON BIOS (IBIOS)*166.

Using the PLC, application developers can create high-level user interfaces to devices on an INSTEON network. *Manager Apps* are External Applications that run on a computing device and use the PLC to directly send and receive INSTEON messages to INSTEON devices. *SALad Apps* are Internal Applications that run on SALad-enabled INSTEON devices themselves. The PLC is a SALad-enabled INSTEON device, having a SALad language interpreter embedded in its firmware.

As shipped by SmartLabs, the PLC contains a 1200-byte *SALad coreApp Program*272 that performs a number of useful functions:

- When coreApp receives messages from INSTEON devices, it sends them to the computing device via its serial port, and when it receives INSTEON-formatted messages from the computing device via the serial port, it sends them out over the INSTEON network.
- CoreApp handles ALL-Linking to other INSTEON devices and maintains an ALL-Link Database.
- CoreApp is event-driven, meaning that it can send messages to the computing device based on the time of day or other occurrences.
- CoreApp can send and receive X10 Commands.

Source code for coreApp is available to developers to modify for their own purposes. Once programmed with an appropriately modified SALad App, the PLC can operate on its own without being connected to a computing device.

As described in the section *Masking Non-linked Network Traffic*112, the PLC hides the full addresses contained within INSTEON messages that it sees, unless the messages are from devices that it is already ALL-Linked to. In particular, SALad Apps that the PLC may be running cannot discover the addresses of previously unknown INSTEON devices, so a hacker cannot write a SALad App that violates INSTEON security protocols.

The SmartLabs Powerline Modem

The SmartLabs Powerline Modem™ (PLM) is an INSTEON-to-Serial Bridge module that plugs into a power outlet and also has a serial port that you connect to your PC (an Ethernet interface is under development). It uses an IN2680A Powerline Modem chip that offers a simple set of ASCII *IM Serial Commands*222 for interacting with INSTEON devices.

The main functions of a PLM are:

- Interfacing to a host via an RS232 serial port.
- Interfacing to the powerline using an isolated power supply.
- Sending and receiving INSTEON messages.
- Sending and receiving X10 messages.
- ALL-Linking to other INSTEON devices and managing an ALL-Link Database.
- Sending ALL-Link Commands and transparently handling ALL-Link Cleanups.
- Managing a SET Button and LED.

See *Chapter 10 — INSTEON Modems*217 for more information.

The PLM uses a daughter board to implement serial communications with the host. Daughter boards interface to the PLM's main board via an 8-pin connector using TTL-level serial communications. PLMs with RS232 daughter boards are currently available, with USB and Ethernet versions under development.

You may communicate to an RS232 PLM via USB by using a USB-to-Serial adapter. SmartLabs has found that Keyspan brand adapters, models USA-49WLC and USA-19HS, provide excellent protocol translation and PLM compatibility.

If you wish, you may create a custom daughter board that fits within a PLM module. You can find hardware reference designs for such custom devices in *Chapter 13 — INSTEON Hardware Documentation*358. To support custom daughter boards, SmartLabs offers a special version of the PLM with the following features:

- Uses the same case as the current PLM/PLC modules
- Has no labeling on the front cover or rear UL label.
- Does not have UL approval.
- Does not include a daughter board.
- Includes the plastic insert for an RJ-45 jack or a blank cover.
- Uses PLM firmware with auto EEPROM detection. When no external EEPROM is detected, the PLM is limited to 31 ALL-Links.

Comparing the Powerline Modem (PLM) to the PowerLinc Controller (PLC)

The PLM is an alternative to the PLC that uses an INSTEON Modem (IM) chip instead of a SALad program to implement an interface between a host device and an INSTEON network on the powerline. The PLM provides a simple set of ASCII *IM Serial Commands*222 that perform most of the same functions as the PLC, but also manage the details of ALL-Linking for the host.

Unlike the PLC, a PLM cannot operate in standalone mode because it cannot run application programs by itself. External applications designed to work with a PLC, such as SmartLabs Device Manager (SDM), will not work with a PLM.

In summary, these are the main differences between the PLC and the PLM:

- The PLM has a simplified command set compared to the PLC.
- The PLM does not dupport SmartLabs Device Manager (SDM) running on a host computer.
- The PLC runs a downloadable SALad application, such as the *SALad coreApp Program*272, but the PLM cannot run applications of any kind. An embedded host on a daughter card or else an always-on external host must be available full time to run applications and manage the PLM.
- The PLM does not have an internal realtime clock.
- If fewer than 32 ALL-Links need to be supported, the PLM can run without external EEPROM. The PLC must have external EEPROM to store a downloadable SALad program.

Manager Applications

An INSTEON Manager App is an External Application program that runs on a computing device, like a PC or PDA, connected to an INSTEON network via an INSTEON Bridge. Manager Apps can provide sophisticated user interfaces for INSTEON devices, they can interact in complex ways with the outside world, and they can orchestrate system behaviors that bring real lifestyle benefits to people.

A Manager App exchanges INSTEON messages directly with INSTEON devices, so it must contain a software module that can translate between a user's intentions and the rules for composing and parsing INSTEON messages.

An example of a Manager App that encapsulates these functions is SmartLabs' *Device Manager* (see *Chapter 12 — SmartLabs Device Manager (SDM) Reference* 336), a Windows program that connects to an INSTEON network via a PowerLinc™ Controller (PLC). SDM handles all the intricacies involved with sending and receiving INSTEON messages via a PLC. To the outside world, it exposes an interface that developers can connect their own custom top-level application layers to.

This topmost layer, often a user interface, communicates with SDM using the Internet HTTP protocol or Microsoft's ActiveX, so it can run on an Internet browser or within a Windows program. SDM and the top layer communicate using a simple text-based scripting language developed by SmartLabs called *Home Network Language™* (HNL).

SDM allows designers to concentrate on rapid application development of their end products without having to deal directly with INSTEON messaging issues. Product developers are encouraged to contact SmartLabs at info@insteon.net for more information about acquiring and using SDM.

INSTEON Modem Applications

INSTEON Modems (IMs) are single chips available from SmartLabs that use simple ASCII commands over a serial port to interface to an INSTEON network (see *Chapter 10 — INSTEON Modems*217). The *IN2680A INSTEON Direct Powerline Modem Interface*10 chip connects to an INSTEON network via the house wiring and the *IN2682A INSTEON Direct RF Modem Interface*10 connects via radio. A BiPHY™ Modem chip that interfaces to *both* the powerline and radio is under development.

SmartLabs also offers a self-contained module built around an IN2680A Powerline Modem chip: *The SmartLabs Powerline Modem*29 (PLM) communicates serially (using RS232) with a PC. USB and Ethernet interfaces are under development.

Developers can create INSTEON Internal or External Applications that run on whatever host device they choose, as long as the host can communicate serially with the IM using the RS232 serial protocol. A microcontroller chip is the most common choice for a host device in standalone INSTEON modules, although virtually any hardware capable of executing applications and communicating serially can use an IM to interface with an INSTEON network.

Perhaps the greatest advantage of using an IM is that developers can create applications in a development environment that they are already comfortable with. The ASCII *IM Serial Commands*222 are relatively few in number and easy to learn, so development cycles can be short.

SALad Applications

SALad is a language interpreter embedded in the firmware of SALad-enabled INSTEON devices (see *SALad Overview* 33). By writing and debugging SALad programs in SmartLabs' *SALad Integrated Development Environment* 33, developers can create INSTEON Internal Applications that run directly on SALad-enabled devices.

Devices running SALad Apps can exhibit very sophisticated behavior. Moreover, devices that have already been installed in the home can be upgraded by downloading new SALad Apps to them. With INSTEON upgradeability, the world of home control can dynamically adapt to people's expectations and needs as the marketplace evolves.

SALad Overview

Because the SALad instruction set is small, and addressing modes for the instructions are highly symmetrical, SALad programs run fast and SALad object code is very compact.

SALad is event driven. Events are triggered when a device receives an INSTEON message, a user pushes a button, a timer expires, an X10 Command is received, and so forth. As events occur, firmware in a SALad-enabled device posts event handles to an event queue, and then starts the SALad program. The SALad program determines what action to take based on the event that started it.

SALad programs can be downloaded into nonvolatile memory of INSTEON devices using the INSTEON network itself, or via a serial link if the device has one. SALad also contains a small debugger that allows programs to be started, stopped, and single-stepped directly over the INSTEON network.

SALad programming mostly consists of writing event handlers. By following examples in the INSTEON Software Development Kit, or by modifying SmartLabs' *SALad coreApp Program* 272, developers can rapidly create INSTEON devices with wide-ranging capabilities. For more information about the SALad Language, consult *Chapter 11 — SALad Language Documentation* 263 in this Developer's Guide, or contact SmartLabs at info@insteon.net.

SALad Integrated Development Environment

The SALad Integrated Development Environment (IDE) is a comprehensive, user-friendly tool for creating and debugging Internal Applications that run directly on SALad-enabled INSTEON devices. Using this tool, programmers can write, compile, download, and debug SALad programs without ever having to leave the IDE. The IDE is a Windows program that connects to an INSTEON network using a SmartLabs PowerLinc™ Controller (see *The SmartLabs PowerLinc Controller* 28).

The SALad IDE includes:

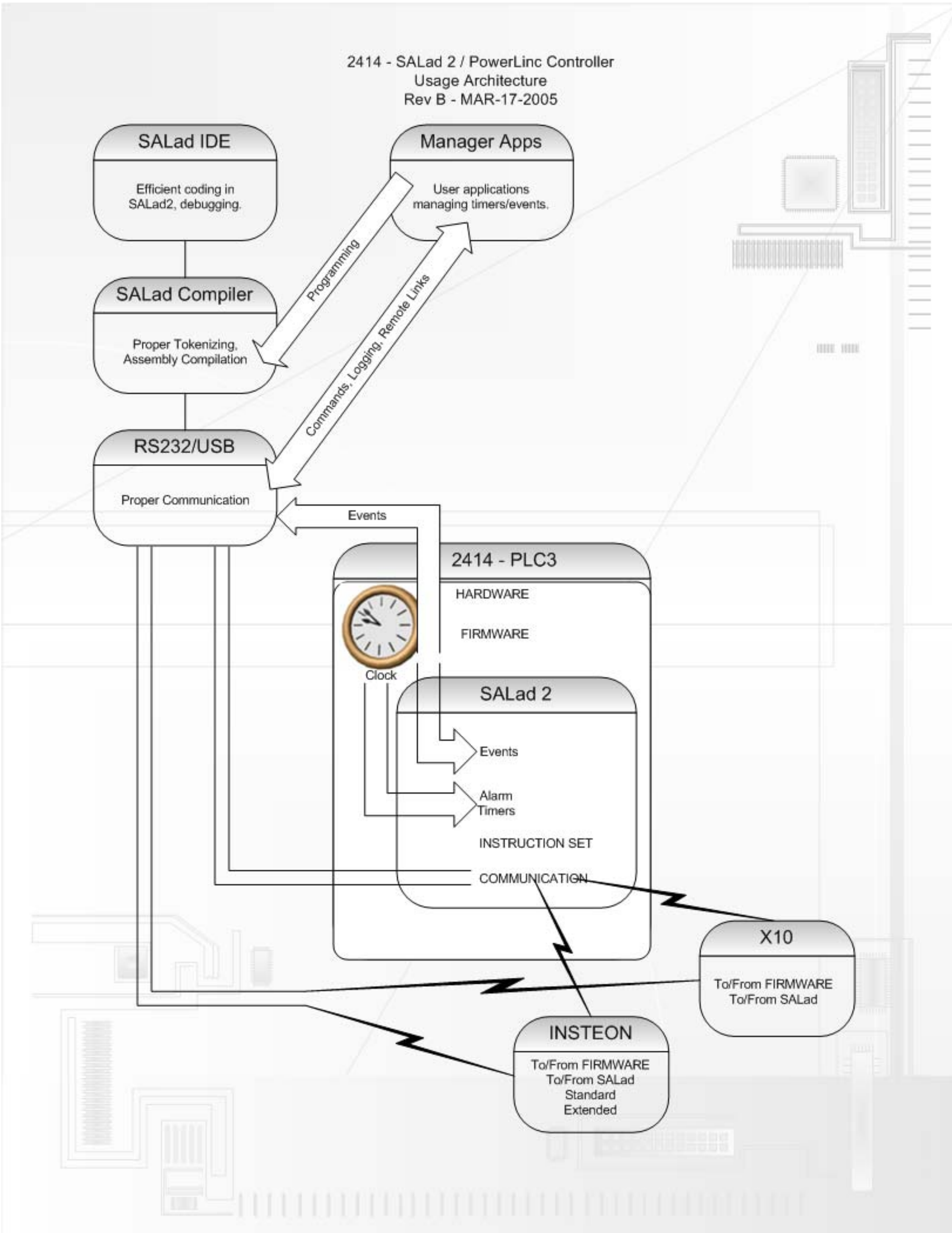
- A SALad Compiler that reads SALad language files and writes SALad object code, error listings, and variable maps
- A communications module that can download SALad object code to an INSTEON device via USB, RS232, or the INSTEON network itself

- A multiple-file, color-contextual source code editor that automatically compiles SALad programs on the fly
- Code templates for common tasks
- A real-time debugger based upon instantaneous feedback from a SALad-enabled device
- A program tracer
- An interactive device conversation window for sending and receiving INSTEON, X10, or ASCII messages
- A raw data window
- A PLC simulator for writing and debugging SALad Apps without actually being connected to an INSTEON network
- INSTEON device diagnostics
- INSTEON network diagnostics
- A device ALL-Link Database manager
- A program listing formatter

For complete information on installing and using the SALad IDE, consult the *SALad Integrated Development Environment User's Guide* 287 below.

INSTEON SALad and PowerLinc Controller Architecture

This diagram shows how software solutions can be implemented using [The SmartLabs PowerLinc Controller](#)28 as an INSTEON gateway/controller device.



INSTEON Developer's Kits

SmartLabs is committed to making the development process as easy as possible for those who create products that can profit from INSTEON networking. For designers who will be crafting new INSTEON devices, adding INSTEON networking to existing devices, or developing External Applications for a network of INSTEON devices, SmartLabs offers both a Software Developer's Kit (SDK) and a series of Hardware Development Modules, as well as extensive technical support.

Software Developer's Kit

To encourage as many developers as possible to join the community of INSTEON product creators, SmartLabs offers a comprehensive Software Developer's Kit (SDK). The INSTEON SDK includes:

- The INSTEON Integrated Development Environment (IDE)
- A SmartLabs PowerLinc™ V2 Controller (PLC) with either a USB or RS232 serial interface
- A SmartLabs LampLinc™ V2 Dimmer module
- This *INSTEON Developer's Guide*
- Access to technical support and peer networking on the INSTEON Internet Forum
- Source code to the *SALad coreApp Program*272 that runs on the PLC
- Sample SALad Applications
- Version maps for product upgrades
- Header files

Hardware Development Modules

SmartLabs has released a series of Hardware Development Modules. Currently available modules include an isolated powerline Hardware Development Kit (HDK) and a Powerline Modem™ (PLM). An RF development module is under development.

The isolated powerline HDK is essentially a PowerLinc™ Controller (PLC) with an extender board that has a prototyping area and a hardware interface to internal circuitry, including the microcontroller. With this module, designers can build and debug hardware interfaces to controllers, sensors, or actuators that connect to an INSTEON network. The isolated power supply for this module ensures that no dangerous voltages are exposed. See *INSTEON Hardware Development Kit (HDK) Reference*359 for details.

The SmartLabs Powerline Modem™ module used an *IN2680A INSTEON Direct Powerline Modem Interface*10 chip on a Main Board, and a Daughter Board for host interfacing or custom development. See *SmartLabs Powerline Modem (PLM) Hardware Reference*367 for more information.

The RF development module uses an *IN2682A INSTEON Direct RF Modem Interface*10 chip. With this module, developers can create products that communicate via RF, and only optionally communicate via the powerline. RF-only devices can be battery operated, so this module is especially suited for developers of handheld INSTEON devices.

PART II — INSTEON REFERENCE

In Part II

Chapter 5 — INSTEON Messages

INSTEON devices communicate by sending messages to one another. In the interest of maximum simplicity, there are only two kinds of INSTEON messages: 10-byte Standard-length messages and 24-byte Extended-length messages. The only difference between the two is that Extended-length messages carry 14 bytes of arbitrary *User Data*. They both carry a *From Address*, a *To Address*, a *Message Flags* byte, two *Command* bytes, and a *Message Integrity* byte.

In This Chapter

INSTEON Message Structure

INSTEON devices communicate with each other by sending fixed-length messages. This section describes the two *Message Lengths*39 (Standard and Extended) and explains the contents of the *Message Fields*41 within the messages. The next section, *INSTEON Message Summary Table*46, presents this information more compactly.

Message Lengths

There are only two kinds of INSTEON messages, 10-byte Standard-length messages and 24-byte Extended-length messages.

The only difference between the two is that the Extended-length message contains 14 *User Data* bytes not found in the Standard-length message. The remaining information fields for both types of message are identical except for an *Extended Message Flag* bit.

INSTEON Standard-length Message – 10 Bytes				
3 Bytes	3 Bytes	1 Byte	2 Bytes	1 Byte
From Address	To Address	Flags	Command 1, 2	CRC[3]

INSTEON Extended-length Message – 24 Bytes					
3 Bytes	3 Bytes	1 Byte	2 Bytes	14 Bytes	1 Byte
From Address	To Address	Flags	Command 1, 2	User Data	CRC[3]

Standard-length Message

Standard-length messages are designed for direct command and control. The payload is just two bytes, *Command 1* and *Command 2*.

Data		**Bits**	**Contents**
From Address		24	Message Originator's address
To Address		24	For Direct messages: Intended Recipient's address For Broadcast messages: Device Category, Subcategory For ALL-Link Broadcast messages: ALL-Link Group Number [0 - 255]
Message Flags	Message Type	1	Broadcast/NAK
		1	ALL-Link
		1	Acknowledgement
	Extended Msg Flag	1	**0 (Zero) for Standard-length messages**
	Hops Left	2	Counted down on each retransmission
	Max Hops	2	Maximum number of retransmissions allowed
Command 1		8	Command to execute
Command 2		8	
CRC[3]		8	Cyclic Redundancy Check

Extended-length Message

In addition to the same fields found in Standard-length messages, Extended-length messages carry 14 bytes of arbitrary *User Data* for downloads, uploads, encryption, and advanced applications.

Data		Bits	Contents
From Address		24	Message Originator's address
To Address		24	For Direct messages: Intended Recipient's address For Broadcast messages: Device Category, Subcategory For ALL-Link Broadcast messages: ALL-Link Group Number [0 - 255]
Message Flags	Message Type	1	Broadcast/NAK
		1	ALL-Link
		1	Acknowledgement
	Extended Msg Flag	1	**1 (One) for Extended-length messages**
	Hops Left	2	Counted down on each retransmission
	Max Hops	2	Maximum number of retransmissions allowed
Command 1		8	Command to execute
Command 2		8	
User Data 1		8	User defined data
User Data 2		8	
User Data 3		8	
User Data 4		8	
User Data 5		8	
User Data 6		8	
User Data 7		8	
User Data 8		8	
User Data 9		8	
User Data 10		8	
User Data 11		8	
User Data 12		8	
User Data 13		8	
User Data 14		8	
CRC[3]		8	Cyclic Redundancy Check

Message Fields

All INSTEON messages contain source and destination Device Addresses41, a Message Flags41 byte, a 2-byte Command 1 and 244 payload, and a Message Integrity Byte44. INSTEON Extended-length messages also carry 14 bytes of User Data44.

Device Addresses

The first field in an INSTEON message is the *From Address*, a 24-bit (3-byte) number that uniquely identifies the INSTEON device originating the message being sent. There are 16,777,216 possible INSTEON devices identifiable by a 3-byte number. This number can be thought of as an ID Code or, equivalently, as an address for an INSTEON device. During manufacture, a unique ID Code is stored in each device in nonvolatile memory.

The second field in an INSTEON message is the *To Address*, also a 24-bit (3-byte) number. Most INSTEON messages are of the *Direct* type, where the intended recipient is another single, unique INSTEON device.

If the message is indeed Direct (as determined by the Flags Byte), the *To Address* contains the 3-byte unique ID Code for the intended recipient. However, INSTEON messages can also be sent to all recipients within range, as *Broadcast* messages, or they can be sent to all members of a group of devices, as *ALL-Link Broadcast* messages. In the case of Broadcast messages, the *To Address* field contains a 1-byte *Device Category*, a 1-byte *Device Subcategory*, and either 0xFF or a *Firmware Version* byte. For ALL-Link Broadcast messages, the *To Address* field contains an ALL-Link Group Number. ALL-Link Group Numbers only range from 0 to 255, given by one byte, so the two most-significant bytes of the three-byte field will be zero.

Message Flags

The third field in an INSTEON message, the *Message Flags* byte, not only signifies the Message Type but it also contains other information about the message. The three most-significant bits, the *Broadcast/NAK Flag* (bit 7), the *ALL-Link Flag* (bit 6), and the *ACK Flag* (bit 5) together indicate the Message Type. Message Types will be explained in more detail in the next section (see Message Type Flags42). Bit 4, the *Extended Message Flag*, is set to one if the message is an Extended-length message, i.e. contains 14 *User Data* bytes, or else it is set to zero if the message is a Standard-length message. The low nibble contains two two-bit fields, *Hops Left* (bits 3 and 2) and *Max Hops* (bits 1 and 0). These two fields control message retransmission as explained below (see Message Retransmission Flags43).

The table below enumerates the meaning of the bit fields in the *Message Flags* byte. The *Broadcast/NAK Flag* (bit 7, the most-significant byte), the *ALL-Link Flag* (bit 6), and the *ACK Flag* (bit 5) together denote the eight possible Message Types.

Bit Position	Flag	Meaning
Bit 7 (Broadcast /NAK) (MSB)	Message Type	100 = Broadcast Message 000 = Direct Message 001 = ACK of Direct Message 101 = NAK of Direct Message 110 = ALL-Link Broadcast Message 010 = ALL-Link Cleanup Message 011 = ACK of ALL-Link Cleanup Message 111 = NAK of ALL-Link Cleanup Message
Bit 6 (ALL-Link)		
Bit 5 (Acknowledgement)		
Bit 4	Extended	1 = Extended-length message 0 = Standard-length Message
Bit 3	Hops Left	00 = 0 message retransmissions remaining 01 = 1 message retransmission remaining 10 = 2 message retransmissions remaining 11 = 3 message retransmissions remaining
Bit 2		
Bit 1	Max Hops	00 = Do not retransmit this message 01 = Retransmit this message 1 time maximum 10 = Retransmit this message 2 times maximum 11 = Retransmit this message 3 times maximum
Bit 0 (LSB)		

Message Type Flags

There are eight possible INSTEON Message Types given by the three Message Type Flag Bits.

Message Types

To fully understand the eight Message Types, consider that there are five basic classes of INSTEON messages: *Broadcast*, *ALL-Link Broadcast*, *ALL-Link Cleanup*, *Direct*, and *Acknowledgement*.

Broadcast messages contain general information with no specific destination. Directed to the community of all devices within range, they are mainly used during device ALL-Linking (see SET Button Pressed *Broadcast Messages* 84, below). Broadcast messages are not acknowledged.

ALL-Link Broadcast messages are directed to a group of devices that have previously been ALL-Linked to the message originator (see *INSTEON ALL-Link Groups* 93, below). ALL-Link Broadcast messages are a means for speeding up the response to a command intended for multiple devices. They are not acknowledged directly. Instead, after sending an ALL-Link Broadcast message to an ALL-Link Group of devices, the message originator then sends an **ALL-Link Cleanup** message addressed to each member of the ALL-Link Group individually, with the expectation of an acknowledgement back from each device in turn.

Direct messages (sometimes referred to as Point-to-Point messages) are intended for a single specific recipient. The recipient responds to Direct messages by returning an Acknowledgement message.

Acknowledgement messages (ACK or NAK) are messages from the recipient to the message originator in response to a Direct or ALL-Link Cleanup message. There is no acknowledgement to a Broadcast or ALL-Link Broadcast message. In some cases, when a Direct message specifically requests returned data, an ACK message may return one or two data bytes to the originator, or a NAK message may return an error code.

Message Type Flag Bits

The *Broadcast/NAK Flag* (bit 7) will be set whenever the message is a Broadcast message or an ALL-Link Broadcast message. In those two cases the *Acknowledgement Flag* (bit 5) will be clear. If the *Acknowledgement Flag* is set, the message is an Acknowledgement message. In that case the *Broadcast/NAK Flag* will be set when the Acknowledgement message is a NAK, and it will be clear when the Acknowledgement message is an ACK.

The *ALL-Link Flag* (bit 6) will be set to indicate that the message is an ALL-Link Broadcast message or part of an ALL-Link Cleanup conversation. This flag will be clear for general Broadcast messages and Direct conversations.

Now all eight Message Types can be enumerated as follows, where the three-bit field is given in the order Bit 7, Bit 6, Bit 5.

- Broadcast messages are Message Type 100.
- Direct messages are 000.
- An ACK of a Direct message is 001
- A NAK of a Direct message is 101
- An ALL-Link Broadcast message is 110.
- ALL-Link Broadcasts are followed up by a series of ALL-Link Cleanup messages of Message Type 010 to each member of the ALL-Link Group.
- Each recipient of an ALL-Link Cleanup message will return an acknowledgement with an ALL-Link Cleanup ACK of Message Type 011 or an ALL-Link Cleanup NAK of Message Type 111.

See the INSTEON Message Summary Table 46 in the next section for a chart of all possible message types.

Extended Message Flag

Bit 4 is the *Extended Message Flag*. This flag is set for 24-byte Extended-length messages that contain a 14-byte *User Data* field, and the flag is clear for 10-byte Standard-length messages that do not contain User Data.

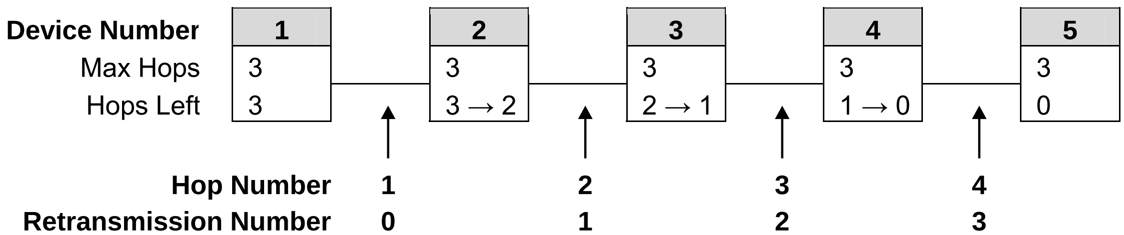
Message Retransmission Flags

The remaining two flag fields, *Max Hops* and *Hops Left*, manage message retransmission. As described above, all INSTEON devices are capable of repeating[1] messages by receiving and retransmitting them. Without a mechanism for limiting the number of times a message can be retransmitted, an uncontrolled 'data storm' of endlessly repeated messages could saturate the network. To solve this problem, INSTEON message originators set the 2-bit *Max Hops* field to a value of 0, 1, 2, or 3, and they also set the 2-bit *Hops Left* field to the same value.

The standard value of *Max Hops* for Broadcast and ALL-Link Broadcast messages is 3. For Direct and ALL-Link Cleanup messages, the standard initial value of *Max Hops* is 1. If INSTEON Message Retrying 54 is necessary, INSTEON Engine firmware will automatically increment *Max Hops* for each retry, up to the maximum value of 3.

A *Max Hops* value of zero tells other devices within range not to retransmit the message. A higher *Max Hops* value tells devices receiving the message to retransmit it depending on the *Hops Left* field. If the *Hops Left* value is one or more, the receiving device decrements the *Hops Left* value by one, then retransmits the message with the new *Hops Left* value. Devices that receive a message with a *Hops Left* value of zero will not retransmit that message. Also, a device that is the intended recipient of a message will not retransmit the message, no matter what the *Hops Left* value is. See *INSTEON Message Hopping*49 for more information.

Note that the designator *Max Hops* really means maximum *retransmissions* allowed. All INSTEON messages 'hop' at least once, so the value in the *Max Hops* field is one less than the number of times a message actually hops from one device to another. Since the maximum value in this field is three, there can be four actual hops, consisting of the original transmission and three retransmissions. Four hops can span a chain of five devices. This situation is shown schematically below.



Command 1 and 2

The fourth field in an INSTEON message is a two-byte Command, made up of *Command 1* and *Command 2*. The usage of this field depends on the Message Type as explained below (see *INSTEON Message Summary Table*46 and *Chapter 8 — INSTEON Command Set*114).

User Data

Only if the message is an Extended-length message, with the *Extended Message Flag* set to one, will it contain the fourteen-byte *User Data* field. Extended-length Direct Commands have a predefined User Data field, but developers may define their own User Data fields by employing so-called *FX Commands* (see *User-Defined FX Commands*121).

If more than 14 bytes of User Data need to be transmitted, multiple INSTEON Extended-length messages will have to be sent using FX Commands. Users can define a packetizing method for their data so that a receiving device can reliably reassemble long messages. Encrypting User Data can provide private, secure communications for sensitive applications such as security systems.

Message Integrity Byte

The last field in an INSTEON message is a one-byte CRC, or Cyclic Redundancy Check. The INSTEON transmitting device computes the CRC over all the bytes in a message beginning with the *From Address*. INSTEON uses a software-implemented 7-bit linear-feedback shift register with taps at the two most-significant bits. The CRC covers 9 bytes for Standard-length messages and 23 bytes for Extended-length messages. An INSTEON receiving device computes its own CRC over the same message bytes as it receives them. If the message is corrupt, the receiver's CRC will not match the transmitted CRC.

Firmware in the INSTEON Engine handles the CRC byte automatically, appending it to messages that it sends, and comparing it within messages that it receives. Applications post messages to and receive messages from the INSTEON Engine without the CRC byte being appended.

Detection of message integrity allows for highly reliable, verified communications. The INSTEON ACK/NAK (acknowledge, non-acknowledge) closed-loop messaging protocol based on this detection method is described below (see *INSTEON Message Retrying*54).

INSTEON Message Summary Table

The table below summarizes all the fields in every possible type of INSTEON message. The *From Address*, the *To Address*, the *Message Flags*, and the CRCs are as explained above.

The table introduces two-letter abbreviations denoting message types that appear often in this Developer's Guide and in other INSTEON documentation. The first letter is either **S** for Standard-length messages or **E** for Extended-length messages. The second letter is **D** for Direct messages, **A** for ALL-Link Broadcast messages, **C** for ALL-Link Cleanup messages, or **B** for Broadcast messages.

	Message Type	3 Bytes From Address			3 Bytes To Address			1 Byte Message Flags Type			X	HL	MH	1 Byte Cmd 1	1 Byte Cmd 2	1 Byte CRC[3]
Standard	**SB** [Broadcast]	ID1_2	ID1_1	ID1_0	DevCat	SubCat	0xFF	1	0	0	**0**	Message retransmissions left	Maximum message retransmissions allowed	SB Cmd	0xFF	CRC
Standard	**SA** Broadcast	ID1_2	ID1_1	ID1_0	0x00	0x00	Group #	1	1	0	**0**			SA Cmd	0x00	CRC
Standard	**SC** Cleanup	ID1_2	ID1_1	ID1_0	ID2_2	ID2_1	ID2_0	0	1	0	**0**			SA Cmd	Group #	CRC
Standard	**SC** Cleanup ACK	ID1_2	ID1_1	ID1_0	ID2_2	ID2_1	ID2_0	0	1	1	**0**			SA Cmd	Group #	CRC
Standard	**SC** Cleanup NAK	ID1_2	ID1_1	ID1_0	ID2_2	ID2_1	ID2_0	1	1	1	**0**			SA Cmd	Error #	CRC
Standard	**SD** [Direct]	ID1_2	ID1_1	ID1_0	ID2_2	ID2_1	ID2_0	0	0	0	**0**			SD Cmd		CRC
Standard	**SD** ACK	ID1_2	ID1_1	ID1_0	ID2_2	ID2_1	ID2_0	0	0	1	**0**			ACK Echo or Data		CRC
Standard	**SD** NAK	ID1_2	ID1_1	ID1_0	ID2_2	ID2_1	ID2_0	1	0	1	**0**			NAK Error #		CRC

	Message Type	From Address			To Address			Type			X	HL	MH	14 Bytes D1 ⇒ D14		1 Byte CRC[3]
Extended	**EB** [Broadcast]	Unused						1	0	0	**1**			Unused		
Extended	**EA** Broadcast	Unused						1	1	0	**1**			Unused		
Extended	**EC** Cleanup	Unused						0	1	0	**1**			Unused		
Extended	**EC** Cleanup ACK	Unused						0	1	1	**1**			Unused		
Extended	**EC** Cleanup NAK	Unused						1	1	1	**1**			Unused		
Extended	**ED** [Direct]	ID1_2	ID1_1	ID1_0	ID2_2	ID2_1	ID2_0	0	0	0	**1**			ED Cmd	D1 ⇒ D14	CRC
Extended	**ED** ACK	Unused						0	0	1	**1**			Unused		
Extended	**ED** NAK	Unused						1	0	1	**1**			Unused		
								Broadcast / NAK	**ALL-Link**	**Acknowledge**	**Extended Flag**	**Hops Left, 2 bits**	**Max Hops, 2 bits**			

The top section of the table shows the possible Standard-length messages and the bottom section shows Extended-length messages. Extended-length messages have the same structure as Standard-length messages, except that Extended-length messages have their *Extended Message Flag* set to one and they possess a 14-byte *User Data* field.

Although there are eight possible Extended-length message types, the only one in actual use is **ED** (Extended Direct). The reason is that Acknowledgement (ACK or NAK) messages are always Standard-length, and ALL-Link and Broadcast messages do not require the 14-byte *User Data* field.

The *Command 1* and *Command 2* fields contain different information depending on the INSTEON message type.

SD and ED Messages

In the case of Direct messages, the two *Command* fields together comprise a 2-byte Command chosen from a possible 65,536 Commands suitable for controlling an individual device within an INSTEON *Device Category*, or *DevCat*. Each set of 65,536 possible **SD** or **ED** Commands can have a different interpretation, depending on the DevCat of the Responder. For example, a Direct Command of 0x11AA tells a device belonging to DevCat 0x01 (Dimmable Lighting Controls) to turn on the lamp it operates to brightness level 0xAA. Every INSTEON Responder device belongs to a DevCat and contains a database of Direct Commands specific to that DevCat that it is capable of executing (see *Chapter 8 — INSTEON Command Set*114).

SD ACK and SD NAK Messages

In the interest of maximum communications reliability, the INSTEON protocol requires that recipients of **SD** and **ED** (Direct) messages acknowledge successful message reception by sending either an **SD** ACK or an **SD** NAK message back to the message originator in a particular timeslot following successful reception of the original message. Note that ACK and NAK messages are always Standard-length (**SD**), even if the message being acknowledged is Extended-length (**ED**).

When the originator of an **SD** or **ED** message receives an **SD** ACK or an **SD** NAK, it knows that the receiving device got the original message without corruption. If a receiving device fails to send an **SD** ACK or an **SD** NAK back to the originating device, the INSTEON Engine in the originating device will automatically retry sending the message up to five times (see *INSTEON Message Retrying*54).

By default, when an INSTEON device receives an uncorrupted **SD** or **ED** message, its INSTEON Engine firmware sends an **SD** ACK back to the originator by

7. swapping the *From Address* and the *To Address* in the message it received,
8. setting the *Acknowledgement Flag* (bit 5 of the *Message Type* field) to one, and
9. echoing the received *Command 1* and *Command 2* fields.

Application-level software in a receiving device may alter the echoed bytes that the INSTEON Engine put into the *Command 1* or *Command 2* fields, and it may switch the default **SD** ACK message to an **SD** NAK message.

Certain **SD** Commands function as requests for just one or two bytes of data from a receiving device. When an INSTEON device receives one of these Commands, its application software either puts a single byte of data into the *Command 2* field, or else two bytes of data into the *Command 1* and *Command 2* fields of the **SD** ACK message.

When certain error conditions occur after reception of an **SD** or **ED** Command, the receiving device's application software may change the message type to **SD** NAK by setting the *Broadcast/NAK* (bit 7 of the *Message Type* field) to one. It may also put one of the *NAK Error Codes*119 into the *Command 2* field of the resulting **SD** NAK message.

SB Messages

In the case of **SB** (Broadcast) messages, the *Command 1* field contains one of 256 possible **SB** Commands suitable for sending to all devices at once. (*Command 2* should be set to 0xFF.) The main purpose of **SB** Commands is to support ALL-Linking of Controllers with Responders. For example, a Controller invites Responder devices to ALL-Link to one of its buttons by sending a *SET Button Pushed Controller* **SB** Command of 0x02 (see SET Button Pressed *Broadcast Messages*84). Every INSTEON device contains a database of Broadcast Commands that it is capable of executing.

Recipients do not acknowledge **SB** messages.

SA ALL-Link Broadcast Messages

The remaining INSTEON message types are for dealing with ALL-Link Groups of one or more devices (see *INSTEON ALL-Link Groups*93). *INSTEON ALL-Linking*26 not only allows universal INSTEON device interoperability, but it also allows multiple INSTEON devices to respond simultaneously to an **SA** ALL-Link Broadcast Command.

While it is true that all the members of an ALL-Link Group of devices could be sent individual **SD** or **ED** (Direct) messages with the same Command (to turn on, for example), it would take a noticeable amount of time for all the messages to be transmitted in sequence. The members of the ALL-Link Group would not execute the Command all at once, but rather in the order received. INSTEON solves this problem by first sending an **SA** ALL-Link Broadcast message to all members of an ALL-Link Group at once, then following it up with individual **SC** ALL-Link Cleanup messages directed to each member of the ALL-Link Group in turn.

SA ALL-Link Broadcast messages contain an ALL-Link Group Number in the *To Address* field, and a one-byte **SA** Command in the *Command 1* field (*Command 2* should be set to 0x00). During the **SC** Cleanup messages that will follow, the **SA** Command will still occupy the *Command 1* field but the ALL-Link Group Number will move to the *Command 2* field. Because these are both one-byte fields, there can only be 256 **SA** Commands and only 256 ALL-Link Group Numbers. (There is one legacy **SA** Broadcast Command, *Light Start Manual Change*, that uses the Command 2 field as a parameter. See *INSTEON Standard-length ALL-Link Commands*152 for more information.)

Recipients of an **SA** ALL-Link Broadcast message check the ALL-Link Group Number in the *To Address* field against their own ALL-Link Group memberships recorded in an ALL-Link Database (see *INSTEON ALL-Link Database*101). This database, stored in nonvolatile memory, is established during a prior *ALL-Linking* process (see *Methods for ALL-Linking INSTEON Devices*96). If the recipient is a member of the ALL-Link Group being broadcast to, it executes the Command in the *Command 1* field.

Recipients do not acknowledge **SA** messages.

SC ALL-Link Cleanup Messages

SA ALL-Link Broadcast Command recipients can expect an individually-addressed **SC** ALL-Link Cleanup message to follow. If the recipient has already executed the **SA** Command, it will not execute the **SA** Command a second time. However, if the recipient missed the **SA** ALL-Link Broadcast Command for any reason, it will not have executed it, so it will execute the Command after receiving the **SC** ALL-Link Cleanup message.

SC ACK and SC NAK Messages

After receiving the **SC** ALL-Link Cleanup message and executing the **SA** ALL-Link Command, the recipient device will respond with an **SC** ACK or an **SC** NAK message. The mechanism for handling **SC** ACK and **SC** NAK messages is the same as for *SD ACK and SD NAK Messages*47, except that the *ALL-Link Flag* (bit 6 of the *Message Flags* field) is set.

INSTEON Message Repetition

To maximize communications reliability, the INSTEON messaging protocol includes two kinds of message repetition: message hopping and message retrying.

*INSTEON Message Hopping*49 is the mechanism whereby INSTEON devices, all of which can retransmit INSTEON messages, aid each other in delivering a message from a message originator to a message recipient.

*INSTEON Message Retrying*54 occurs when the originator of an **SD** or **ED** Direct or **SC** ALL-Link Cleanup message does not receive an acknowledgement message from the intended recipient.

INSTEON Message Hopping

In order to improve reliability, the INSTEON messaging protocol includes message retransmission, or hopping. Hopping enables other INSTEON devices, all of which can repeat[1] messages, to help relay a message from an originator to a recipient.

When INSTEON devices repeat messages, multiple devices can end up simulcasting the same message, meaning that they can repeat the same message at the same time. To ensure that simulcasting is synchronous (so that multiple devices do not jam each other), INSTEON devices adhere to specific rules given below (see *Timeslot Synchronization*49).

Message Hopping Control

Two 2-bit fields in the *Message Flags* byte manage INSTEON message hopping (see *Message Retransmission Flags*43, above). One field, *Max Hops*, contains the maximum number of hops, and the other, *Hops Left*, contains the number of hops remaining.

To avoid 'data storms' of endless repetition, messages can be retransmitted a maximum of three times only. A message originator sets the *Max Hops* for a message. The larger the number of *Max Hops*, the longer the message will take to complete being sent, whether or not the recipient hears the message early.

The standard value of *Max Hops* for **SB** Broadcast and **SA** ALL-Link Broadcast messages is 3. For **SD** and **ED** Direct and **SC** ALL-Link Cleanup messages, the standard initial value of *Max Hops* is 1. If *INSTEON Message Retrying*54 is necessary, INSTEON Engine firmware will automatically increment *Max Hops* for each retry, up to the maximum value of 3.

If the *Hops Left* field in a message is nonzero, every device that hears the message synchronously repeats it, thus increasing the signal strength, path diversity, and range of the message. An INSTEON device that repeats a message decrements *Hops Left* before retransmitting it. When a device receives a message with zero *Hops Left*, it does not retransmit the message.

Timeslot Synchronization

There is a specific pattern of transmissions, retransmissions and acknowledgements that occurs when an INSTEON message is sent, as shown in the examples below.

An INSTEON message on the powerline occupies either six or thirteen zero crossing periods, depending on whether the message is Standard- or Extended-length. This message transmission time, six or thirteen powerline half-cycles, is called a *timeslot* in the following discussion.

During a single timeslot, an INSTEON message can be transmitted, retransmitted, or acknowledged. The entire process of communicating an INSTEON message, which may involve retransmissions and acknowledgements, will occur over integer multiples of timeslots. See *INSTEON Full Message Cycle Times* 63 below for a table that gives these times.

The following examples show how INSTEON messages propagate in a number of common scenarios. The examples use these symbols:

Legend	**T**	Transmission by Message Originator
	R	Message Retransmission
	A	Acknowledgement by Intended Recipient
	C	Confirmation received by Message Originator
	L	Listening State
	W	Waiting State

	Max Hops	Timeslot	1	2	3	4	5	6	7	8
Example 1	**0**	Sender	**T**							

Example 1, the simplest, shows a Broadcast message with a *Max Hops* of zero (no retransmissions). The **T** indicates that the Sender has originated and transmitted a single message. There is no acknowledgement that intended recipients have heard the message. The message required one timeslot of six or thirteen powerline zero crossings to complete.

	Max Hops	Timeslot	1	2	3	4	5	6	7	8
Example 2	**1**	Sender	**T**							
		Repeater 1	L	**R**						

Example 2 shows a Broadcast message with a *Max Hops* of one. *Max Hops* can range from zero to three as explained above. The Sender transmits a Broadcast message as signified by the **T**. Another INSTEON device, functioning as a Repeater, listens to the message, as signified by an **L**, and then retransmits it in the next timeslot as indicated by the **R**.

	Max Hops	Timeslot	1	2	3	4	5	6	7	8
Example 3	**3**	Sender	**T**	L	L	L	L			
		Repeater 1	L	**R**	L	**R**	L			
		Repeater 2	L	L	**R**	L	L			
		Repeater 3	L	L	L	**R**	L			

Up to three retransmissions are possible with a message. **Example 3** shows the progression of the message involving an originating Sender and three repeating devices, with a *Max Hops* of three. Example 3 assumes that the range between Repeaters is such that only adjacent Repeaters can hear each other, and that only

Repeater 1 can hear the Sender. Note that the Sender will not retransmit its own message.

	Max Hops	Timeslot	1	2	3	4	5	6	7	8
Example 4	**0**	Sender	**T**	**C**						
		Recipient	L	**A**						

When a Sender transmits a Direct message, it expects an acknowledgement from the Recipient. **Example 4** shows what happens if the *Max Hops* value is zero. The **A** designates the timeslot in which the Recipient acknowledges receipt of the Direct message. The **C** shows the timeslot when the Sender finds that the message is confirmed.

	Max Hops	Timeslot	1	2	3	4	5	6	7	8
Example 5	**1**	Sender	**T**	L	L	**C**				
		Repeater 1	L	**R**	L	**R**				
		Recipient	L	L	**A**	L				

When *Max Hops* is set to one, a Direct message propagates as shown in **Example 5**. Repeater 1 will retransmit both the original Direct message and the acknowledgement from the Recipient.

	Max Hops	Timeslot	1	2	3	4	5	6	7	8
Example 6	**1**	Sender	**T**	L	**C**	W				
		Repeater 1	L	**R**	L	**R**				
		Recipient	L	W	**A**	L				

If *Max Hops* is set to one, but no retransmission is needed because the Recipient is within range of the Sender, messages flow as shown in **Example 6**. The **W** in the Sender and Recipient rows indicates a wait. The Recipient immediately hears the Sender since it is within range. However, the Recipient must wait one timeslot before sending its acknowledgement, because it is possible that a repeating device will be retransmitting the Sender's message. Repeater 1 is shown doing just that in the example, although the Recipient would still have to wait even if no Repeaters were present. Only when all of the *possible* retransmissions of the Sender's message are complete, can the Recipient send its acknowledgement. Being within range, the Sender hears the acknowledgement immediately, but it must also wait until possible retransmissions of the acknowledgement are finished before it can send another message.

	Max Hops	Timeslot	1	2	3	4	5	6	7	8
Example 7	**3**	Sender	**T**	L	L	L	L	L	L	**C**
		Repeater 1	L	**R**	L	**R**	L	**R**	L	**R**
		Repeater 2	L	L	**R**	L	L	L	**R**	L
		Repeater 3	L	L	L	**R**	L	**R**	L	**R**
		Recipient	L	L	L	L	**A**	L	**R**	L

Example 7 shows what happens when *Max Hops* is three and three retransmissions are in fact needed for the message to reach the Recipient. Note that if the Sender or Recipient were to hear the other's message earlier than shown, it still must wait until *Max Hops* timeslots have occurred after the message was originated before being free to send its own message. If devices did not wait, they would jam each other by sending different messages in the same timeslot. A device can calculate how many timeslots have passed prior to receiving a message by subtracting the *Hops Left* number in the received message from the *Max Hops* number.

All seven of the above examples are given again in the table below in order to show the patterns more clearly.

	Max Hops	Timeslot	1	2	3	4	5	6	7	8
Example 1	**0**	Sender	**T**							
Example 2	**1**	Sender	**T**							
		Repeater 1	L	**R**						
Example 3	**3**	Sender	**T**	L	L	L	L			
		Repeater 1	L	**R**	L	**R**	L			
		Repeater 2	L	L	**R**	L	L			
		Repeater 3	L	L	L	**R**	L			
Example 4	**0**	Sender	**T**	**C**						
		Recipient	L	**A**						
Example 5	**1**	Sender	**T**	L	L	**C**				
		Repeater 1	L	**R**	L	**R**				
		Recipient	L	L	**A**	L				
Example 6	**1**	Sender	**T**	L	**C**	W				
		Repeater 1	L	**R**	L	**R**				
		Recipient	L	W	**A**	L				
Example 7	**3**	Sender	**T**	L	L	L	L	L	L	**C**
		Repeater 1	L	**R**	L	**R**	L	**R**	L	**R**
		Repeater 2	L	L	**R**	L	L	L	**R**	L
		Repeater 3	L	L	L	**R**	L	**R**	L	**R**
		Recipient	L	L	L	L	**A**	L	**R**	L

Legend

T	Transmission by Message Originator
R	Message Retransmission
A	Acknowledgement by Intended Recipient
C	Confirmation received by Message Originator
L	Listening State
W	Waiting State

INSTEON Message Retrying

If the originator of an INSTEON Direct message does not receive an acknowledgement from the intended recipient, the message originator will automatically try resending the message up to five times.

Firmware in the INSTEON Engine handles message retrying. In case a message did not get through because *Max Hops* was set too low, each time the INSTEON Engine retries a message, it also increases *Max Hops* up to the limit of three. A larger number of *Max Hops* can achieve greater range for the message by allowing more devices to retransmit it.

The tables below give the time in seconds to retry messages five times, taking into account the starting *Max Hops* value.

Time for Five Direct (Acknowledged) Message Retries, Seconds		
Beginning Max Hops	**Standard-length Messages**	**Extended-length Messages**
0	1.40	2.22
1	1.70	2.69
2	1.90	3.01
3	2.00	3.17

When an application uses the INSTEON Engine to send a Direct message to an intended recipient, it can conclude that the recipient did not get the message after five retries if the INSTEON Engine does not return the expected acknowledgement message within the pertinent time limit given in the table. In other words, when an application sends an **SD** or **ED** Direct or **SC** ALL-Link Cleanup message, it should set a timer with the appropriate value from the table. If the timer expires before the application receives the acknowledgement, then the message did not get through.

Because message retrying is automatic, it is important to unlink INSTEON Responder devices from INSTEON Controller devices when an ALL-Linked device is removed from an INSTEON network. Otherwise, the Controller will needlessly retry communicating with the missing Responder. See the *INSTEON ALL-Link Database*101 section, below, for more information.

i2 Engine Message Retrying

The i2 INSTEON Engine improves retries in two ways.

First, if the i2 Engine sends an **SB** or **SA** Broadcast message with *Max Hops* greater than zero, and then does not hear another INSTEON device hopping the message, then it will begin a retry sequence until it does hear a hop or until it has retried five times.

Second, the i2 Engine waits for a randomized number of powerline zero crossing times before it retries sending a message. Both i1 and i2 INSTEON Engines wait for any existing INSTEON traffic to complete before sending an INSTEON message. However, it is still possible that two or more INSTEON devices may attempt to send their messages at the same time after the channel is free. When this happens, the messages will 'clobber' each other and receivers will detect a corrupted message. In the case of Direct (**SD**, **ED**, or **SC**) messages, receivers will not send an acknowledgement message, and in the case of Broadcast (**SB** or **SA**) messages, other devices will not hop the message. In either case, the senders will begin a retry sequence. With the randomized retry delay, each sender will begin the retry at a different time with high probability. Whichever device starts first will gain the channel. The other devices will abort their retry attempts and signal the retry failure by setting the *_MsgFail* flag (bit 4) in the *I_Error* byte (see the *i2 Engine Memory Map*170).

Chapter 6 — INSTEON Signaling Details

This chapter gives complete information about how the data in INSTEON messages actually travels over the powerline or the airwaves. Unlike other mesh networks, INSTEON does not elaborately route its traffic in order to avoid data collisions—instead, INSTEON devices *simulcast* according to simple rules explained below. Simulcasting by multiple devices is made possible because INSTEON references a global clock, the powerline zero crossing.

In This Chapter

INSTEON Powerline Signaling

This section covers low-level INSTEON messaging over the powerline.

In This Section

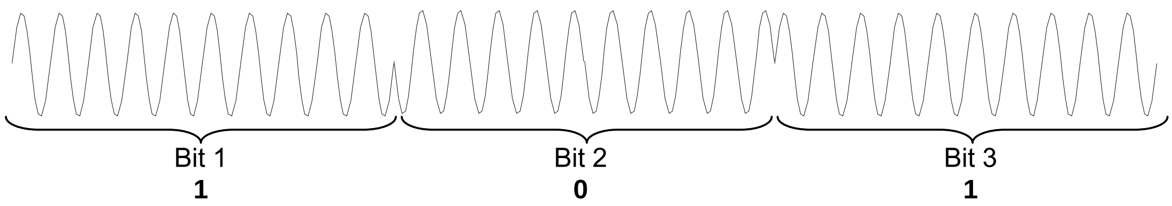
Powerline BPSK Modulation

INSTEON devices communicate on the powerline by adding a signal to the powerline voltage. In the United States, powerline voltage is nominally 110 VAC RMS, alternating at 60 Hz.

An INSTEON powerline signal uses a carrier frequency of 131.65 KHz, with a nominal amplitude of 4.64 volts peak-to-peak into a 5 ohm load. In practice, the impedance of powerlines varies widely, depending on the powerline configuration and what is plugged into it, so measured INSTEON powerline signals can vary from sub-millivolt to more than 5 volts.

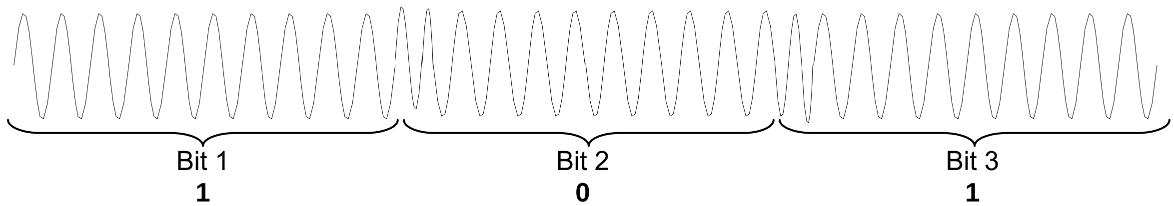
INSTEON data is modulated onto the 131.65 KHz carrier using binary phase-shift keying, or BPSK, chosen for reliable performance in the presence of noise.

The figure below shows an INSTEON 131.65 KHz powerline carrier signal with alternating binary phase-shift keying (BPSK) bit modulation.



INSTEON uses 10 cycles of carrier for each bit. Bit 1, interpreted as a one, begins with a positive-going carrier cycle. Bit 2, interpreted as a zero, begins with a negative-going carrier cycle. Bit 3 begins with a positive-going carrier cycle, so it is interpreted as a one. Note that the sense of the bit interpretations is arbitrary. That is, ones and zeros could be reversed as long as the interpretation is consistent. Phase transitions only occur when a bitstream changes from a zero to a one or from a one to a zero. A one followed by another one, or a zero followed by another zero, will not cause a phase transition. This type of coding is known as NRZ, or non-return to zero.

Note the abrupt phase transitions of 180 degrees at the bit boundaries. Abrupt phase transitions introduce troublesome high-frequency components into the signal's spectrum. Phase-locked detectors can have trouble tracking such a signal. To solve this problem, INSTEON uses a gradual phase change to reduce the unwanted frequency components.



The figure above shows the same BPSK signal with gradual phase shifting. The transmitter introduces the phase change by inserting 1.5 cycles of carrier at 1.5 times the 131.65 KHz frequency. Thus, in the time taken by one cycle of 131.65 KHz, three half-cycles of carrier will have occurred, so the phase of the carrier will be reversed at the end of the period due to the odd number of half-cycles. Note the smooth transitions between the bits.

INSTEON Powerline Packets

Messages sent over the powerline are broken up into packets, with each packet sent in conjunction with a zero crossing of the AC voltage on the powerline.

The bytes in an INSTEON powerline message are transmitted most-significant byte first, and the bits are transmitted most-significant bit first.

Standard-length messages use five packets and Extended-length messages use eleven packets, as shown below.

Standard-length Message – 5 Packets

120 total bits = 15 bytes
84 Data bits = 10½ bytes, 10 usable

SP	BP	BP	BP	BP

Extended-length Message - 11 Packets

264 total bits = 33 bytes
192 Data bits = 24 bytes

SP	BP	BP	BP	BP	BP	BP	BP	BP	BP	BP

A Start Packet appears as the first packet in an INSTEON message, as shown by the symbol **SP** in both the Standard- and Extended-length messages. The remaining packets in a message are Body Packets, as shown by the symbols **BP**.

Each packet contains 24 bits of information, but the information is interpreted in two different ways, as shown below.

SP **Start Packet**

1	0	1	0	1	0	1	0	1	0	0	1	x	x	x	x	x	x	x	x	x	x	x	x
8 Sync bits								4 Start Code bits				12 Data bits											

BP **Body Packet**

1	0	1	0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2 Sync bits		4 Start Code bits				18 Data Bits																	

Powerline packets begin with a series of *Sync Bits*. There are eight Sync Bits in a Start Packet and there are two Sync Bits in a Body Packet. The alternating pattern of ones and zeros allows the receiver to detect the presence of a signal.

Following the Sync Bits are four *Start Code Bits*. The 1001 pattern indicates to the receiver that Data bits will follow.

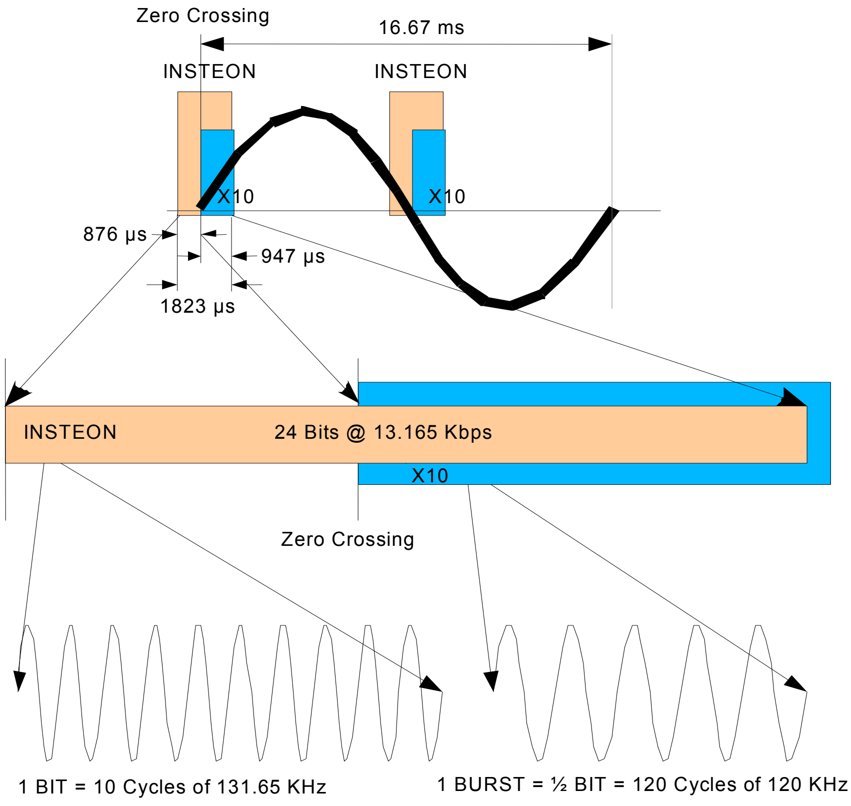
The remaining bits in a packet are *Data Bits*. There are twelve Data Bits in a Start Packet, and there are eighteen Data Bits in a Body Packet.

The total number of Data Bits in a Standard-length message is 84, or 10½ bytes. The last four data bits in a Standard-length message are ignored, so the usable data is 10 bytes. The total number of Data Bits in an Extended-length message is 192, or 24 bytes.

Powerline Packet Timing

All INSTEON powerline packets contain 24 bits. Since a bit takes 10 cycles of 131.65 KHz carrier, there are 240 cycles of carrier in an INSTEON packet. An INSTEON powerline packet therefore lasts 1.823 milliseconds.

The powerline environment is notorious for uncontrolled noise, especially high-amplitude spikes caused by motors, dimmers and compact fluorescent lighting. This noise is minimal during the time that the current on the powerline reverses direction, a time known as the powerline zero crossing. Therefore, INSTEON packets are transmitted during the zero crossing quiet time, as shown in the figure below.



The top of the figure shows a single powerline cycle, which possesses two zero crossings. An INSTEON packet is shown at each zero crossing. INSTEON packets nominally begin 876 microseconds before a zero crossing and last until 947 microseconds after the zero crossing. To allow for hardware zero crossing detector component tolerances and for load-dependent powerline phase shifts, the INSTEON signal may begin up to 300 microseconds early or late with respect to the zero crossing detected by a particular INSTEON device.

X10 Compatibility

The figure also shows how X10 signals are applied to the powerline. X10 is the signaling method used by many devices already deployed on powerlines around the world. Compatibility[2] with this existing population of legacy X10 devices is an important feature of INSTEON. At a minimum, X10 compatibility means that INSTEON and X10 signals can coexist with each other, but compatibility also allows designers to create hybrid devices that can send and receive both INSTEON and X10 signals.

The X10 signal uses a burst of approximately 120 cycles of 120 KHz carrier beginning at the powerline zero crossing and lasting about 1000 microseconds. A burst followed by no burst signifies an X10 one bit and no burst followed by a burst signifies an X10 zero bit. An X10 message begins with two bursts in a row followed by a one bit, followed by nine data bits. The figure shows an X10 burst at each of the two zero crossings.

The X10 specification also allows for two copies of the zero crossing burst located one-third and two-thirds of the way through a half-cycle of power. These points correspond to the zero crossings of the other two phases of three-phase power. INSTEON is insensitive to those additional X10 bursts and does not transmit them when sending X10.

The middle of the figure shows an expanded view of an INSTEON packet with an X10 burst superimposed. The X10 signal begins at the zero crossing, 876 microseconds after the beginning of the INSTEON packet, and ends 1000 microseconds after the zero crossing.

INSTEON devices achieve compatibility with X10 by listening for an INSTEON signal beginning 876 microseconds before the zero crossing. INSTEON receivers implemented in software can be very sensitive, but at the cost of having to receive a substantial portion of a packet before being able to validate that a true INSTEON packet is being received. Reliable validation may not occur until as much as 450 microseconds after the zero crossing, although an INSTEON device will still begin listening for a possible X10 burst right at the zero crossing. If at the 450-microsecond mark the INSTEON receiver validates that it is not receiving an INSTEON packet, but that there *is* an X10 burst present, the INSTEON receiver will switch to X10 mode and listen for a complete X10 message over the next 11 powerline cycles. If instead the INSTEON device detects that it is receiving an INSTEON packet, it will remain in INSTEON mode and not listen for X10 until it receives the rest of the complete INSTEON message.

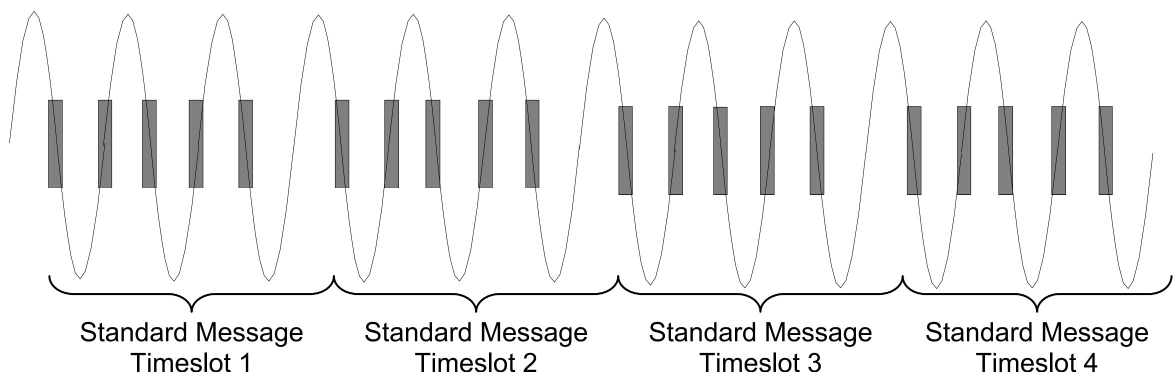
The bottom of the figure shows that the raw bitrate for INSTEON is much faster for INSTEON than for X10. An INSTEON bit requires ten cycles of 131.65 KHz carrier, or 75.96 microseconds, whereas an X10 bit requires *two* 120-cycle bursts of 120 KHz. One X10 burst takes 1000 microseconds, but since each X10 burst is sent at a zero crossing, it takes 16,667 microseconds to send the two bursts in a bit, resulting in a sustained bitrate of 60 bits per second. INSTEON packets consist of 24 bits, and an INSTEON packet can be sent during each zero crossing, so the nominal raw sustained bitrate for INSTEON is 2880 bits per second, 48 times faster than X10. Note that this nominal INSTEON bitrate must be derated to account for packet and message overhead, as well as message retransmissions. See *INSTEON Full Message Cycle Times* 63, below, for details.

Powerline Message Timeslots

To allow time for processing messages and potential retransmission of a message by INSTEON i1/RF devices, an INSTEON transmitter waits for one additional zero crossing after sending a Standard-length message, or for two zero crossings after sending an Extended-length message. Therefore, the total number of zero crossings needed to send a Standard-length message is 6, or 13 for an Extended-length message. This number, 6 or 13, constitutes an INSTEON message timeslot.

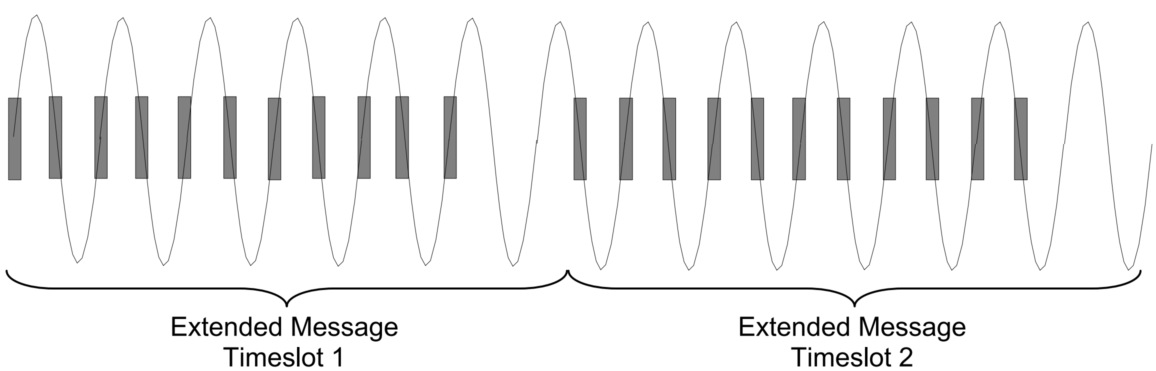
Standard-length Message Timeslots

The figure below shows a series of 5-packet Standard-length INSTEON messages being sent on the powerline. INSTEON transmitters wait for one zero crossing after each Standard-length message before sending another message, so the Standard-length message timeslot is 6 zero crossings, or 50 milliseconds, in length.



Extended-length Message Timeslots

The next figure shows a series of 11-packet Extended-length INSTEON messages being sent on the powerline. INSTEON transmitters wait for two zero crossings after each Extended-length message before sending another message, so the Extended-



length message timeslot is 13 zero crossings, or 108.333 milliseconds, in length.

INSTEON Full Message Cycle Times

An INSTEON full message cycle encompasses all of the *Powerline Message Timeslots*62 required for hopping the outgoing INSTEON message and receiving acknowledgement messages, if any. Responders return acknowledgement messages with the same number of hops as the outgoing message.

SD and **ED** Direct and **SC** ALL-Link Cleanup messages require acknowledgement, but **SB** Broadcast and **SA** ALL-Link Broadcast messages do not. All acknowledgement messages are Standard-length, even if the outgoing message is Extended-length.

The table below gives the overall time required to complete a full message cycle, depending on whether the message is Standard- or Extended-length, on the value of the *Max Hops* field within the *Message Flags*41 byte, and on whether or not there is an acknowledgement message. See *INSTEON Message Hopping*49 above for examples of messages propagating in timeslots.

INSTEON Full Message Cycle Times

Message Length	Max Hops	ACK?	Timeslots S = Standard E = Extended	Powerline Zero Crossings	Time (ms)
Standard	0	No	1 **S**	6	50
Standard	1	No	2 **S**	12	100
Standard	2	No	3 **S**	18	150
Standard	3	No	4 **S**	24	200
Standard	0	Yes	2 **S**	12	100
Standard	1	Yes	4 **S**	24	200
Standard	2	Yes	6 **S**	36	300
Standard	3	Yes	8 **S**	48	400
Extended	0	No	1 **E**	13	108.333
Extended	1	No	2 **E**	26	216.667
Extended	2	No	3 **E**	39	325
Extended	3	No	4 **E**	52	433.333
Extended	0	Yes	1 **E** + 1 **S**	19	158.333
Extended	1	Yes	2 **E** + 2 **S**	38	316.667
Extended	2	Yes	3 **E** + 3 **S**	57	475
Extended	3	Yes	4 **E** + 4 **S**	76	633.333

INSTEON Powerline Data Rates

INSTEON Standard-length messages contain 120 raw data bits and require 6 zero crossings, or 50 milliseconds to send. Extended-length messages contain 264 raw data bits and require 13 zero crossings, or 108.33 milliseconds to send. Therefore, the actual raw bitrate for INSTEON is 2400 bits per second for Standard-length messages, or 2437 bits per second for Extended-length messages, instead of the 2880 bits per second it would be without waiting for the extra zero crossings.

INSTEON Standard-length messages contain 9 bytes (72 bits) of usable data, not counting packet sync and start code bits, nor the message CRC byte. Extended-length messages contain 23 bytes (184 bits) of usable data using the same criteria. Therefore, the bitrates for usable data are further reduced to 1440 bits per second for Standard-length messages and 1698 bits per second for Extended-length messages. If one only counts the 14 bytes (112 bits) of User Data in Extended-length messages, the User Data bitrate is 1034 bits per second.

The above data rates assume that messages are sent with *Max Hops* set to zero and that there are no message retries. They also do not take into account the time it takes for a message to be acknowledged. The table below shows net data rates when multiple hops and message acknowledgement are taken into account. To account for retries, divide the given data rates by one plus the number of retries (up to a maximum of 5 possible retries).

Condition			**Bits per Second**		
Max Hops	**ACK**	**Retries**	**Standard Message Usable Data**	**Extended Message Usable Data**	**Extended Message User Data Only**
0	No	0	1440	1698	1034
1	No	0	720	849	517
2	No	0	480	566	345
3	No	0	360	425	259
0	Yes	0	720	849	517
1	Yes	0	360	425	259
2	Yes	0	240	283	173
3	Yes	0	180	213	130

INSTEON Second Generation i2/RF Signaling

Second generation i2/RF replaces first generation i1/RF for wireless INSTEON communications. Because i2/RF and i1/RF use different frequencies, they operate independently. (There is only one legacy product that implemented i1/RF, the SignaLinc™ RF Signal Enhancer introduced in May 2005.)

i2/RF Physical Layer

The table below gives the specifications for second generation INSTEON i2/RF physical radios.

i2/RF Specification	Value
Center Frequency	915.0000 MHz
Modulation Method	FSK
FSK Deviation	200 KHz peak-to-peak
Data Encoding Method	Manchester
Symbol Rate	9124 symbols per second
Data Rate	4562 bits per second
Symbol Time	109.600 microseconds
Bit Time	219.200 microseconds
Range	400 ft unobstructed line-of-sight, half-wave dipole antenna, 0.1 raw bit-error rate

i2/RF Center Frequency

The center frequency, 915.0000 MHz, lies in the band 902 to 924 MHz, which is permitted for unlicensed operation in the United States. i2/RF radios cannot communicate with i1/RF radios because they operate at different frequencies.

i2/RF Modulation

Symbols are modulated onto the carrier using frequency-shift keying (FSK), where a zero-symbol modulates the carrier half the FSK deviation frequency downward and a one-symbol modulates the carrier half the FSK deviation frequency upward. The FSK deviation frequency chosen for i2/RF is 200 KHz.

i2/RF Data Encoding

Each data bit is Manchester encoded, meaning that two symbols are sent for each bit. A one-symbol followed by a zero-symbol designates a *One-Bit*, and a zero-symbol followed by a one-symbol designates a *Zero-Bit*.

The two illegal Manchester codes play special roles. Two zero-symbols in a row designate a *Start-Bit*, and two one-symbols in a row designate a *Sync-Bit*. i2/RF transmitters begin packets with *Start-Bits*, but they never send *Sync-Bits*. i2/RF receivers, however, can use *Sync-Bit* detection in a symbol stream to adjust the phase of their bit clocks.

The table below shows the four possible symbol combinations.

Symbols	Bit	Name	Usage
00	**S**	Start-Bit	Designates beginning of i2/RF Packet
01	**0**	Zero-Bit	Data bit of 0
10	**1**	One-Bit	Data bit of 1
11	**—**	Sync-Bit	Never transmitted, receivers can use to sync bit clock

i2/RF Timing

Symbols are modulated onto the carrier at 9,124 symbols per second, resulting in a raw data rata of half that, or 4,562 bits per second.

The master symbol clock derives from counting 274 ticks of a 400-nanosecond timer, giving a symbol period of 109.600 microseconds, or a bit period of 219.200 microseconds.

i2/RF Range

The typical range for free-space reception is 400 feet to achieve a raw bit-error rate of 0.1% using a half-wave dipole antenna. The presence of walls and other RF energy absorbers will reduce this range.

i2/RF Data Packets

Each byte in an i2/RF message is transported in a separate *i2/RF Data Packet*. Accordingly, a Standard-length message requires 10 *i2/RF Data Packets* and an Extended-length message requires 24 *i2/RF Data Packets*.

i2/RF Data Packets always begin with a *Start-Bit*, which is an illegal Manchester code consisting of two zero symbols in a row. Thirteen Manchester-encoded *One-Bits* or *Zero-Bits* follow the *Start-Bit*. The first five bits constitute a *Sleep Code*, and the last eight bits make up the INSTEON message *i2/RF Data Byte*.

Counting the *Start-Bit*, there are 140 bits (280 symbols) in the 10 *i2/RF Data Packets* of a Standard-length message, and 336 bits (672 symbols) in the 24 *i2/RF Data Packets* of an Extended-length message. This bitstream for the entire message is transmitted continuously—there is no space between packets.

Both the *Sleep Code* and the *i2/RF Data Byte* are transmitted least-significant bit (LSB) first.

The table below shows the contents of an *i2/RF Data Packet*.

i2/RF Data Packet Structure				
Bit #	**Bit**	**Symbols**	**Field**	
1	**S**	00	*Start-Bit*	
2	**0** or **1**	01 or 10	*Sleep Code*	Bit 0 (LSB)
3	**0** or **1**	01 or 10		Bit 1
4	**0** or **1**	01 or 10		Bit 2
5	**0** or **1**	01 or 10		Bit 3
6	**0** or **1**	01 or 10		Bit 4 (MSB)
7	**0** or **1**	01 or 10	*i2/RF Data Byte*	Bit 0 (LSB)
8	**0** or **1**	01 or 10		Bit 1
9	**0** or **1**	01 or 10		Bit 2
10	**0** or **1**	01 or 10		Bit 3
11	**0** or **1**	01 or 10		Bit 4
12	**0** or **1**	01 or 10		Bit 5
13	**0** or **1**	01 or 10		Bit 6
14	**0** or **1**	01 or 10		Bit 7 (MSB)

At a bitrate of 4,562 bits per second (219.200 microseconds per bit), it takes 3.0688 milliseconds to send a 14-bit *i2/RF Data Packet*.

i2/RF Sync Pattern

A *Sync Pattern* is a nibble of 0x5, or 0101 in binary. As with the other fields in an i2/RF packet, the *Sync Pattern* is also sent LSB first, so it looks like this:

i2/RF Sync Pattern		
Bit #	**Bit**	**Symbols**
1	**1**	10
2	**0**	01
3	**1**	10
4	**0**	01

A full byte (two nibbles) of *Sync Pattern* precedes the first packet in an i2/RF message. Its main function is to synchronize the decoder's bit clock so that the *Start-Bit* immediately following the *Sync Pattern* can be properly detected

The *Sync Pattern* also fills in the gaps between the end of the first transmission of an i2/RF message and any hops of the same message that the originating transmitter may send. The *Sync Pattern* is contained in a number of *Sync Filler Packets* followed by a special *Sync Gap Packet*.

Standard-length messages will have three *Sync Filler Packets* and Extended-length messages will have six. *Sync Filler Packets* are similar to *i2/RF Data Packets* except that the data byte consists of two nibbles of *Sync Pattern*, like this:

i2/RF Sync Filler Packet Structure				
Bit #	**Bit**	**Symbols**	**Field**	
1	**S**	00	*Start-Bit*	
2	**0**	01	*Sleep Code* **1** through **6**	Bit 0 (LSB)
3	**0**	01		Bit 1
4	**0** or **1**	01 or 10		Bit 2
5	**0** or **1**	01 or 10		Bit 3
6	**0** or **1**	01 or 10		Bit 4 (MSB)
7	**1**	10	*Sync Pattern*	Bit 0 (LSB)
8	**0**	01		Bit 1
9	**1**	10		Bit 2
10	**0**	01		Bit 3 (MSB)
11	**1**	10		Bit 0 (LSB
12	**0**	01		Bit 1
13	**1**	10		Bit 2
14	**0**	01		Bit 3 (MSB)

The *Sync Gap Packet* that comes after the *Sync Filler Packets* has a *Sleep Code* of **0** followed by 15 nibbles of *Sync Pattern*. *Sync Gap Packets* look like this:

i2 / RF Sync Gap Packet Structure				
Bit #	**Bit**	**Symbols**	**Field**	
1	**S**	00	*Start-Bit*	
2	**0**	01	*Sleep Code* **0**	Bit 0 (LSB)
3	**0**	01		Bit 1
4	**0**	01		Bit 2
5	**0**	01		Bit 3
6	**0**	01		Bit 4 (MSB)
7	**1**	10	*Sync Pattern*	Bit 0 (LSB)
8	**0**	01		Bit 1
9	**1**	10		Bit 2
10	**0**	01		Bit 3 (MSB)
. . .	. . .			. . .
63	**1**	10		Bit 0 (LSB
64	**0**	01		Bit 1
65	**1**	10		Bit 2
66	**0**	01		Bit 3 (MSB)

i2/RF Sleep Codes

Battery powered INSTEON i2/RF devices must conserve as much energy as possible in order to prolong battery life. The 5-bit *Sleep Code* at the beginning of each i2/RF packet allows i2/RF devices to rapidly determine where the packet lies in a message sequence so that they can power down and wake up again during the *Sync Pattern* preceding the next message timeslot.

When an i2/RF device receives a *Start-Bit*, the *Sleep Code* follows immediately. If the *Sleep Code* is **31** (0x1F), then the packet is an *i2/RF Data Packet* containing the INSTEON *Message Flags*41. If the *Sleep Code* is **0** (0x00), then the packet is a *Sync Gap Packet* containing 15 nibbles of *Sync Pattern*. Any other *Sleep Code* value (**1** through **30**) tells the receiver that it is in the middle of an i2/RF message and that it is too late to parse that repetition of the message.

A receiver that decodes a *Sleep Code* from **1** through **30** can calculate the maximum time that it can go to sleep before waking up in time to receive either a retransmission of the message it is currently in the middle of or else a new message. The calculation is simple—multiply the *Sleep Code* value by the i2/RF packet duration of 3.0688 milliseconds and then add 14 milliseconds. The result is the time that will elapse between receiving the last bit of the *Sleep Code* and the first bit of the *Sync Pattern* nibble that precedes the next message timeslot.

Max Sleep Time = (*Sleep Code* X 3.0688) + 14.0000 milliseconds

Designers should deduct from these maximum sleep times any additional processing time before going to sleep plus any additional time it will take to wake up and begin reliably detecting a *Sync Pattern*. To wake up two *Sync Pattern* nibbles before the next message timeslot, deduct another 0.8768 milliseconds (four 219.2 microsecond bit times).

Note that when waking up after the calculated sleep time, there may not be an i2/RF message occupying the following timeslot. This will happen when the receiver parses a *Sleep Code* from the last repetition of a message and no transmitter begins sending a message in the next timeslot. See *i2/RF Wakeup Strategies*75 below for various actions to take in this case.

The table below summarizes the above information.

i2 / RF Sleep Codes		
Sleep Code	**Meaning**	**Sleep Time Until Next Message Sync**
31 (0x1F)	*First i2/RF Packet* designator. This packet contains the *Message Flags* byte.	Don't sleep right away; parse first.
30 (0x1E)	30 X 3.0688 = 92.0640 ms	106.0640 ms
29 (0x1D)	29 X 3.0688 = 88.9952 ms	102.9952 ms
28 (0x1C)	28 X 3.0688 = 85.9264 ms	99.9264 ms
27 (0x1B)	27 X 3.0688 = 82.8576 ms	96.8576 ms
26 (0x1A)	26 X 3.0688 = 79.7888 ms	93.7888 ms
25 (0x19)	25 X 3.0688 = 76.7200 ms	90.7200 ms
24 (0x18)	24 X 3.0688 = 73.6512 ms	87.6512 ms
23 (0x17)	23 X 3.0688 = 70.5824 ms	84.5824 ms
22 (0x16)	22 X 3.0688 = 67.5136 ms	81.5136 ms
21 (0x15)	21 X 3.0688 = 64.4448 ms	78.4448 ms
20 (0x14)	20 X 3.0688 = 61.3760 ms	75.3760 ms
19 (0x13)	19 X 3.0688 = 58.3072 ms	72.3072 ms
18 (0x12)	18 X 3.0688 = 55.2384 ms	69.2384 ms
17 (0x11)	17 X 3.0688 = 52.1696 ms	66.1696 ms
16 (0x10)	16 X 3.0688 = 49.1008 ms	63.1008 ms
15 (0x0F)	15 X 3.0688 = 46.0320 ms	60.0320 ms
14 (0x0E)	14 X 3.0688 = 42.9632 ms	56.9632 ms
13 (0x0D)	13 X 3.0688 = 39.8944 ms	53.8944 ms
12 (0x0C)	12 X 3.0688 = 36.8256 ms	50.8256 ms
11 (0x0B)	11 X 3.0688 = 33.7568 ms	47.7568 ms
10 (0x0A)	10 X 3.0688 = 30.6880 ms	44.6880 ms
9 (0x09)	9 X 3.0688 = 27.6192 ms	41.6192 ms
8 (0x08)	8 X 3.0688 = 24.5504 ms	38.5504 ms
7 (0x07)	7 X 3.0688 = 21.4816 ms	34.4816 ms
6 (0x06)	6 X 3.0688 = 18.4128 ms	31.4128 ms
5 (0x05)	5 X 3.0688 = 15.3440 ms	29.3440 ms
4 (0x04)	4 X 3.0688 = 12.2752 ms	26.2752 ms
3 (0x03)	3 X 3.0688 = 9.2064 ms	23.2064 ms
2 (0x02)	2 X 3.0688 = 6.1376 ms	20.1376 ms
1 (0x01)	1 X 3.0688 = 3.0688 ms	17.0688 ms
0 (0x00)	*Sync Gap Packet* designator. A continuous Sync Pattern follows this Sleep Code until the start of the next message hop	A new message timeslot will follow.

i2/RF Messages

INSTEON i2/RF messages contain the same information as INSTEON powerline messages, but with the data reordered so that a battery-powered device can quickly determine whether or not it is the addressee of an incoming message, and if it is not, go back to sleep.

i2/RF Message Timing

i2/RF messages occupy timeslots that are very close in duration to the *Powerline Message Timeslots* 62 specified above. Each Standard-length i2/RF message occupies a 49.9776 ms timeslot corresponding to 6 powerline zero crossing intervals (50.0000 ms), and each Extended-length i2/RF message occupies a 108.2848 ms timeslot corresponding to 13 powerline zero crossing intervals (108.3333 ms).

The actual time to transmit an i2/RF message is shorter than the message interval, in order to allow time for the message receiver to process the message. Accordingly, an i2/RF message interval time consists of the i2/RF message duration time plus an i2/RF message gap time.

The following table summarizes this information. All times are in milliseconds.

i2/RF Message Timing		
Message Property	**Standard-length Messages**	**Extended-length Messages**
Powerline Zero Crossings	6	13
Powerline Message Interval Time	50.0000	108.3333
i2/RF Message Interval Time	49.9776	108.2848
i2/RF Message Duration Time	30.6880	73.6512
i2/RF Message Gap Time	19.2896	34.6336

i2/RF Message Retransmission

i2/RF messages are subject to the same *INSTEON Message Hopping* 49 rules as powerline messages. The section *INSTEON Full Message Cycle Times* 63 above gives a table showing how long a full message cycle takes, depending on the message length, how many times the message originator specified for the message to hop, and whether the message is acknowledged or not.

On the powerline, message originators only transmit a message once no matter how many times the message will be hopped (although if they hear their own message hopped by another INSTEON device and there are still some more hops remaining to be done, then they will hop the message as any other device would). In contrast, i2/RF message originators *do* "hop their own messages" by transmitting the same message more than once, depending on the *Max Hops* field in the message. Of course, they decrement the *Hops Left* field each time they retransmit the message.

i2/RF message originators always set the *Max Hops* field in a message to a minimum of one so that the message will be retransmitted (hopped) at least once. Thus, receivers that may wake up in the middle of the first transmission will have at least one more chance to receive the complete message.

i2/RF Message Structure

All i2/RF messages begin with two nibbles of *Sync Pattern* (0x55), followed by one i2/RF packet for each byte in the message. If a message originator is going to retransmit a message due to remaining hops, then it will fill the space between the two messages with a number of *Sync Filler Packets* followed by a special *Sync Gap Packet* containing a long *Sync Pattern*. The *Sync Pattern* in the *Sync Gap Packet* then serves as the lead-in to the next repetition of the message in lieu of the two nibbles of *Sync Pattern* that normally precede the first repetition of a message in a message cycle.

The table below shows all of the packets in both a Standard-length and an Extended-length i2/RF message. Note that the table does not show the *Sleep Pattern* byte preceding the first message in a message cycle.

Standard-length Message			**Extended-length Message**		
Sleep Code	**Broadcast Message Byte**	**Direct Message Byte**	**Sleep Code**	**Broadcast Message Byte**	**Direct Message Byte**
31 (0x1F)	Flags	Flags	**31** (0x1F)	Flags	Flags
11 (0x0B)	From ID Low	To ID Low	**30** (0x1E)	From ID Low	To ID Low
10 (0x0A)	From ID Mid	To ID Mid	**29** (0x1D)	From ID Mid	To ID Mid
9 (0x09)	From ID Hi	To ID Hi	**28** (0x1C)	From ID Hi	To ID Hi
8 (0x08)	To ID Low	From ID Low	**27** (0x1B)	To ID Low	From ID Low
7 (0x07)	To ID Mid	From ID Mid	**26** (0x1A)	To ID Mid	From ID Mid
6 (0x06)	To ID Hi	From ID Hi	**25** (0x19)	To ID Hi	From ID Hi
5 (0x05)	Command 1	Command 1	**24** (0x18)	Command 1	Command 1
4 (0x04)	Command 2	Command 2	**23** (0x17)	Command 2	Command 2
3 (0x03)	CRC	CRC	**22** (0x16)	User Data 14	User Data 14
2 (0x02)	Sync Filler	Sync Filler	**21** (0x15)	User Data 13	User Data 13
1 (0x01)	Sync Filler	Sync Filler	**20** (0x14)	User Data 12	User Data 12
0 (0x00)	Sync Gap	Sync Gap	**19** (0x13)	User Data 11	User Data 11
			18 (0x12)	User Data 10	User Data 10
			17 (0x11)	User Data 9	User Data 9
			16 (0x10)	User Data 9	User Data 9
			15 (0x0F)	User Data 8	User Data 8
			14 (0x0E)	User Data 7	User Data 7
			13 (0x0D)	User Data 6	User Data 6
			12 (0x0C)	User Data 5	User Data 5
			11 (0x0B)	User Data 4	User Data 4
			10 (0x0A)	User Data 3	User Data 3
			9 (0x09)	User Data 2	User Data 2
			8 (0x08)	User Data 1	User Data 1
			7 (0x07)	CRC	CRC
			6 (0x06)	Sync Filler	Sync Filler
			5 (0x05)	Sync Filler	Sync Filler
			4 (0x04)	Sync Filler	Sync Filler
			3 (0x03)	Sync Filler	Sync Filler
			2 (0x02)	Sync Filler	Sync Filler
			1 (0x01)	Sync Filler	Sync Filler
			0 (0x00)	Sync Gap	Sync Gap

The *Message Flags*41 byte of the INSTEON message appears first so that receivers can reject a message and go back to sleep at the earliest opportunity. By comparing the *Max Hops* value to the *Hops Left* value in the *Message Retransmission Flags*43, a receiver can determine how many repetitions of the current message remain in a message cycle. Inspection of the *Message Type Flags*42 allows a receiver to determine if the message cycle includes an acknowledgement, or if the current message is itself an acknowledgement message. Receivers can establish the length of a timeslot for a message repetition by looking at the *Extended Message Flag*43. Along with the *Sleep Code*, this information is sufficient for a receiver to calculate how much time will elapse before a new message cycle will begin. If a receiver further determines that there is nothing relevant to it in the current message cycle, then it can go back to sleep and wake up again during a later message cycle.

To help a receiver determine message relevancy quickly, either the *From Address* appears next in the case of Broadcast Messages, or else the *To Address* appears next in the case of Direct messages. Receivers can reject **SD** and **ED** Direct and **SC** ALL-Link Cleanup messages with a *To Address* that does not match the receiver's INSTEON address. They can also reject **SA** ALL-Link Broadcast messages with a *From Address* and an ALL-Link Group Number in the *To Address* low byte that do not match any ALL-Link Records in the receiver's *INSTEON ALL-Link Database*101.

i2/RF Wakeup Strategies

When a battery-powered i2/RF receiver wakes up, it will either find nothing being transmitted, or else it will hear i2/RF traffic.

Wakeup During i2/RF Traffic

If there is i2/RF traffic, then most likely the receiver woke up somewhere in the middle of a message cycle, in which case every fourteenth bit will be a *Start-Bit*, followed by a five-bit *Sleep Code*. On average, a receiver will be able to wake up and determine if there is a valid *Sleep Code* in a little more than one packet time, or about 3.2 milliseconds.

If the receiver finds a *Sleep Code*, it can go to sleep until the beginning of the next message timeslot. If there is a message in the next timeslot, then it is either part of an ongoing message cycle or else it is the beginning of a new message cycle.

From the beginning of a message timeslot, the receiver will be able to reject an irrelevant message in 5.9 milliseconds best case, but no more than 14.7 milliseconds worst case. The best case occurs for **SD** or **ED** Direct or **SC** ALL-Link Cleanup messages where the least-significant bits of the message *To Address* and the receiver's INSTEON address do not match. In the worst case, the receiver must parse the *Message Flags*, the *From Address*, and the *To Address* low byte, and then search its ALL-Link Database to determine that an ALL-Link Broadcast message is not for it.

When a receiver rejects a message, it can go back to sleep for the rest of the message cycle. If it woke up at the beginning of an Extended-length Direct message cycle, this could be over 600 milliseconds. The table below gives all possible *INSTEON Full Message Cycle Times*63, depending on the message length, *Max Hops*, and whether the message is acknowledged or not.

INSTEON Full Message Cycle Times

Message Length	Max Hops	ACK?	Timeslots S = Standard E = Extended	Powerline Zero Crossings	Time (ms)
Standard	0	No	1 **S**	6	50
Standard	1	No	2 **S**	12	100
Standard	2	No	3 **S**	18	150
Standard	3	No	4 **S**	24	200
Standard	0	Yes	2 **S**	12	100
Standard	1	Yes	4 **S**	24	200
Standard	2	Yes	6 **S**	36	300
Standard	3	Yes	8 **S**	48	400
Extended	0	No	1 **E**	13	108.333
Extended	1	No	2 **E**	26	216.667
Extended	2	No	3 **E**	39	325
Extended	3	No	4 **E**	52	433.333
Extended	0	Yes	1 **E** + 1 **S**	19	158.333
Extended	1	Yes	2 **E** + 2 **S**	38	316.667
Extended	2	Yes	3 **E** + 3 **S**	57	475
Extended	3	Yes	4 **E** + 4 **S**	76	633.333

Wakeup with No i2/RF Traffic

When an i2/RF receiver wakes up and does not detect any i2/RF traffic, then the optimum sleep strategy is a tradeoff between maximum battery life and quickest response to commands.

For longest battery life, a receiver should sleep as much as possible in order to achieve as low a duty cycle as possible. When an INSTEON device sends a Direct message to another INSTEON device, it expects an acknowledgement message. If there is no acknowledgement, the sender's INSTEON Engine will automatically retry the message up to five times, each time incrementing the *Max Hops* value up to the maximum of three. The table below (reprinted from *INSTEON Message Retrying*54, above) shows the overall message cycle times given the initial *Max Hops* value and the message length.

Time for Five Direct (Acknowledged) Message Retries, Seconds		
Beginning Max Hops	**Standard-length Messages**	**Extended-length Messages**
0	1.40	2.22
1	1.70	2.69
2	1.90	3.01
3	2.00	3.17

If there is still no response after five automatic engine-level retries, an INSTEON application can start the overall message cycle again an arbitrary number of times. Therefore, an i2/RF receiver can go to sleep for longer than an overall message cycle time and still catch messages from applications that retry enough times.

The table below shows the duty cycle and estimated battery life for various i2/RF device wakeup intervals. The table assumes that the i2/RF device consumes 25 milliamps when running from AA batteries capable of supplying 2000 milliamp-hours over their lifetime. The duty cycle is the ratio of running time to sleeping time. Assuming that the i2/RF receiver runs for 3.2 milliseconds on average when it wakes up to sample i2/RF traffic, then its duty cycle is

3.2 milliseconds / Wakeup Interval in seconds.

If the i2/RF device were running all of the time, then the batteries would last just 3.3 days (i.e., 2000 mAh / 25 mA = 80 hours). By decreasing the duty cycle, the battery life increases as shown in the table. Batteries capable of supplying more or less than 2000 mAh will, of course, last a proportionately longer or shorter time.

Note that the table does not take into account standby current while the i2/RF device is sleeping.

Battery Life Estimate (2000 mAh Batteries)			
Wakeup Interval, Seconds	**Duty Cycle**	**Days**	**Years**
0	1	3.3	-
0.1	1/31	103	0.28
0.2	1/63	210	0.58
0.3	1/94	313	0.86
0.4	1/125	417	1.14
0.5	1/156	520	1.42
1.0	1/313	1043	2.86
1.5	1/469	1563	4.28
2.0	1/625	2083	5.71
2.5	1/781	2603	7.13
3.0	1/938	3126	8.56

The table shows a clear tradeoff between command-response latency and battery life.

i2/RF Powerline Synchronization

INSTEON BiPHY™ devices communicate via *both* powerline and radio. INSTEON powerline traffic uses the powerline zero crossing as a global clock to orchestrate message cycle timing, but i2/RF traffic may start at any time, since it can originate from handheld devices that have no powerline zero crossing reference.

Nevertheless, i2/RF devices use the same message timeslot concept as powerline devices do. In lieu of a hardware powerline zero crossing detector, they employ an internal timer that counts out *virtual* zero crossings starting from the beginning of the first i2/RF message in a cycle. An i2/RF message cycle therefore takes the same amount of time as a powerline message cycle with the same number of *Max Hops* and retries, except that the message cycles most probably begin at different times. In other words, i2/RF and powerline messages are synchronous but not phase-locked.

When a BiPHY device receives an i2/RF message, it retransmits the message over the powerline at the next powerline zero crossing, but it retransmits the message via RF according to the incoming 12/RF message timing.

When a BiPHY device originates a message, it transmits the message over powerline at the next powerline zero crossing, and then it transmits the message via i2/RF as soon thereafter as it can.

INSTEON First Generation i1/RF Signaling

SmartLabs introduced first generation INSTEON RF (i1/RF) in May 2005. The only product using i1/RF is the SignaLinc™ RF Signal Enhancer.

Second generation i2/RF replaces first generation i1/RF for wireless INSTEON communications. Because i2/RF and i1/RF use different frequencies, they operate independently.

First generation i1/RF sends and receives the same messages that appear on the powerline. There are two i1/RF message lengths: Standard-length 10-byte messages and Extended-length 24-byte messages. Unlike powerline messages, however, messages sent by i1/RF are not broken up into smaller packets sent at powerline zero crossings, but instead are sent whole.

This section describes the *i1/RF Physical Layer*78, *i1/RF Messages*79, and *i1/RF Timing*79.

i1/RF Physical Layer

The table below gives the physical layer specifications for first generation INSTEON i1/RF radios.

i1/RF Specification	Value
Center Frequency	904 MHz
Modulation Method	FSK
FSK Deviation	64 KHz
Data Encoding Method	Manchester
Symbol Rate	76,800 symbols per second
Data Rate	38,400 bits per second
Range	150 feet outdoors

The center frequency, 904 MHz, lies in the band 902 to 924 MHz, which is permitted for unlicensed operation in the United States. i1/RF radios cannot communicate with i2/RF radios because they operate at different frequencies.

Symbols are modulated onto the carrier using frequency-shift keying (FSK), where a zero-symbol modulates the carrier half the FSK deviation frequency downward and a one-symbol modulates the carrier half the FSK deviation frequency upward. The FSK deviation frequency chosen for INSTEON is 64 KHz.

Each bit is Manchester encoded, meaning that two symbols are sent for each bit. A one-symbol followed by a zero-symbol designates a one-bit, and a zero-symbol followed by a one-symbol designates a zero-bit.

Symbols are modulated onto the carrier at 76,800 symbols per second, resulting in a raw data rata of half that, or 38,400 bits per second.

The typical range for free-space reception is 150 feet, which is reduced in the presence of walls and other RF energy absorbers.

i1/RF Messages

Referring to the figures below, i1/RF messages begin with two sync bytes consisting of AAAA in hexadecimal, followed by a start code byte of C3 in hexadecimal. Ten data bytes (80 bits) follow in Standard-length messages, or twenty-four data bytes (192 bits) in Extended-length messages. The last data byte in a message is a CRC[3] over the data bytes as explained above (see *Message Integrity Byte*44).

The bytes in an INSTEON i1/RF message are transmitted most-significant byte first, and the bits are transmitted most-significant bit first.

i1/RF Standard-length Message – 1 Packet

112 total bits = 14 bytes
80 Data bits = 10 bytes

AA	AA	C3	x	x	x	X	x	x	x	x	x	x	n
2 Sync bytes		1 Start Code byte	80 Data Bits (10 Data bytes)										CRC[3]

i1/RF Extended-length Message – 1 Packet

224 total bits = 28 bytes

192 Data bits = 24 bytes

AA	AA	C3	x	x	x	x	x	x	x	X	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	n
2 Sync bytes		1 Start Code byte	192 Data Bits (24 Data bytes)																								CRC[3]

i1/RF Timing

It takes 2.708 milliseconds to send a 104-bit Standard-length message, and 5.625 milliseconds to send a 216-bit Extended-length message. Zero crossings on the powerline occur every 8.333 milliseconds, so a Standard or Extended i1/RF message can be sent during one powerline half-cycle. The waiting times after sending powerline messages, as shown in the section *Powerline BPSK Modulation*58, are to allow sufficient time for INSTEON i1/RF devices, if present, to retransmit a powerline message.

INSTEON Simulcasting

By following the above rules for message propagation, INSTEON systems achieve a marked increase in the reliability of communications. The reason is that multiple INSTEON devices can transmit the same message at the same time within a given timeslot. INSTEON devices within range of each other thus "help each other out." Most networking protocols for shared physical media prohibit multiple devices from simultaneously transmitting within the same band by adopting complex routing algorithms. In contrast, INSTEON turns what is usually a problem into a benefit by ensuring that devices transmitting simultaneously will be sending the same messages in synchrony with each other.

Powerline Simulcasting

One might think that multiple INSTEON devices transmitting on the powerline could easily cancel each other out rather than boost each other. In practice, even if one were *trying* to nullify one signal with another, signal cancellation by multiple devices would be extremely difficult to arrange. The reason is that for two signals to cancel at a given receiver, the two transmitters would have to send carriers such that the receiver would see them as exactly equal in amplitude and very nearly 180 degrees out of phase. The probability of this situation occurring and persisting for extended periods is low.

The crystals used on typical INSTEON devices to generate the powerline carrier frequency of 131.65 KHz run independently of each other with a frequency tolerance of a few tenths of a percent. Phase relationships among multiple powerline carriers therefore will drift, although slowly with respect to the 1823 microsecond duration of an INSTEON packet. Even if the phases of two transmitters happened to cancel, it is very unlikely that the amplitudes would also be equal at the location of a receiver, so a receiver would very likely still see some signal even in the worst-case transient phase relationship. INSTEON receivers have a wide dynamic range, from millivolts to five volts or so, which will allow them to track signals even if they fade temporarily. Adding more transmitters reduces the probability of signal cancellation even more. With source diversity, the probability that the sum of all the signals will increase in signal strength rises significantly.

The INSTEON powerline carrier is modulated using binary phase-shift keying (BPSK), meaning that receivers are looking for 180-degree phase shifts in the carrier to detect changes in a string of bits from a one to a zero or vice-versa. Multiple transmitters, regardless of the absolute phase of their carriers, will produce signals whose sum still possesses 180-degree phase reversals at bit-change boundaries, so long as their relative carrier frequencies do not shift more than a few degrees over a packet time. Of course, bit timings for each transmitter need to be fairly well locked, so INSTEON transmitters are synchronized to powerline zero crossings. An INSTEON bit lasts for ten cycles of the 131.65 KHz powerline carrier, or 76 microseconds. The powerline zero crossing detector should be accurate within one or two carrier periods so that bits received from multiple transmitters will overlay each other.

In practice, multiple INSTEON powerline transmitters simulcasting the same message will improve the strength of the powerline signal throughout a building.

RF Simulcasting

Since RF signaling is used as an extension to powerline signaling, it also is based on simulcasting. However, because of the short wavelength of 900 MHz RF carrier signals, standing wave interference patterns can form where the RF carrier signal is reduced, even when the carrier and data are ideally synchronized.

As with powerline, for a cancellation to occur, two carriers must be 180 degrees out of phase and the amplitudes must be the same. Perfect cancellation is practically impossible to obtain. In general, two co-located carriers on the same frequency with random phase relationships and the same antenna polarization will sum to a power level greater than that of just one transmitter 67% of the time. As one of the transmitters is moved away from a receiver, the probability of cancellation drops further because the signal amplitudes will be unequal. As the number of transmitters increases, the probability of cancellation becomes nearly zero.

Mobile INSTEON RF devices, such as handheld controllers, are battery operated. To conserve power, mobile devices are not configured as RF repeaters, but only as message originators, so RF simulcasting is not an issue for them. INSTEON devices that do repeat RF messages are attached to the powerline, so most of them will not be moved around after initial setup. During setup, such RF devices can be located, and their antennas adjusted, so that no signal cancellation occurs. With the location of the transmitters fixed, the non-canceling configuration will be maintained indefinitely.

Chapter 7 — INSTEON Device Networking

INSTEON messaging technology can be used in many different ways in many kinds of devices. To properly utilize the full set of possible INSTEON message types, devices must share a common set of specific, preassigned number values for the one- and two-byte Commands, one-byte Device Categories, one-byte Device Subcategories, and one-byte NAK Error codes. SmartLabs maintains the database of allowable values for these parameters.

Because INSTEON devices are individually preassigned a three-byte Address at the time of manufacture, complex procedures for assigning network addresses in the field are not needed. Instead, INSTEON devices are logically ALL-Linked together in the field using a simple, uniform procedure.

INSTEON Extended-length messages allow programmers to devise all kinds of meanings for the User Data that can be exchanged among devices. For example, some INSTEON devices include an interpreter for an application language, called SALad, which is compiled into token strings and downloaded into devices using Extended-length messages. Also, secure messaging can be implemented by sending encrypted payloads in Extended-length messages.

In This Chapter

INSTEON Device Categories

All INSTEON Devices belong to a Device Category, or *DevCat*, denoted by a one-byte hexadecimal number stored in the device's nonvolatile read-only memory. The primary reason for the DevCat is to allow INSTEON **SD** and **ED** Direct Command Numbers to be reused for each category of device. In other words, each DevCat has a *separate* list of **SD** and **ED** Direct Commands applicable to it. It is therefore possible for a number designating a particular Direct Command to be interpreted differently by devices belonging to different DevCats.

This section outlines the *Currently Defined Device Categories*83, , discusses *Device Categories and Subcategories*84, describes the methods for *Determining an INSTEON Device's DevCat Number*84, and explains *Using DevCats to Qualify INSTEON Commands*86.

Currently Defined Device Categories

The following table shows all of the DevCats defined as of the publication date of this Developer's Guide. Although reprinted here for convenience, the official table is contained in the *INSTEON Device Categories and Product Keys Document*9 described in the *Other Documents Included by Reference*9 section above.

Dev Cat #	Device Category	Examples of Devices
0x00	**Generalized Controllers**	ControLinc, RemoteLinc, SignaLinc, etc.
0x01	**Dimmable Lighting Control**	Dimmable Light Switches, Dimmable Plug-In Modules
0x02	**Switched Lighting Control**	Relay Switches, Relay Plug-In Modules
0x03	**Network Bridges**	PowerLinc Controllers, TRex, Lonworks, ZigBee, etc.
0x04	**Irrigation Control**	Irrigation Management, Sprinkler Controllers
0x05	**Climate Control**	Heating, Air conditioning, Exhausts Fans, Ceiling Fans, Indoor Air Quality
0x06	**Pool and Spa Control**	Pumps, Heaters, Chemicals
0x07	**Sensors and Actuators**	Sensors, Contact Closures
0x08	**Home Entertainment**	Audio/Video Equipment
0x09	**Energy Management**	Electricity, Water, Gas Consumption, Leak Monitors
0x0A	**Built-In Appliance Control**	White Goods, Brown Goods
0x0B	**Plumbing**	Faucets, Showers, Toilets
0x0C	**Communication**	Telephone System Controls, Intercoms
0x0D	**Computer Control**	PC On/Off, UPS Control, App Activation, Remote Mouse, Keyboards
0x0E	**Window Coverings**	Drapes, Blinds, Awnings
0x0F	**Access Control**	Automatic Doors, Gates, Windows, Locks
0x10	**Security, Health, Safety**	Door and Window Sensors, Motion Sensors, Scales
0x11	**Surveillance**	Video Camera Control, Time-lapse Recorders, Security System Links
0x12	**Automotive**	Remote Starters, Car Alarms, Car Door Locks
0x13	**Pet Care**	Pet Feeders, Trackers
0x14	**Toys**	Model Trains, Robots
0x15	**Timekeeping**	Clocks, Alarms, Timers
0x16	**Holiday**	Christmas Lights, Displays
0x17 ⇒ 0xFE	**Reserved**	
0xFF	**Unassigned**	For devices that will be assigned a DevCat and SubCat by software

Device Categories and Subcategories

In the past, a one-byte DevCat and a one-byte Device Subcategory (SubCat) have been sufficient to identify an INSTEON product uniquely. Going forward, however, it is very likely that there will not be enough SubCat numbers to uniquely identify all of the different devices within a given DevCat. Therefore, applications should not rely on the SubCat number for product identification, but instead they should use the *INSTEON Product Key* to look up the product in the INSTEON Product Database. See the section *INSTEON Product Database*87 for more information, including a table of *INSTEON Product Key and SubCat Assignments*88.

Determining an INSTEON Device's DevCat Number

Devices disclose their DevCat and SubCat numbers during ALL-Linking within SET Button Pressed *Broadcast Messages*84, or by *Responding to a* Product Data Request *Message*85.

SET Button Pressed Broadcast Messages

INSTEON devices disclose their one-byte Device Category Number (DevCat) and one-byte Device Subcategory Number (SubCat) whenever they send an **SB** (**S**tandard-length **B**roadcast) message with a *Command 1* field of **0x01** (*SET Button Pressed Responder*) or **0x02** (*SET Button Pressed Controller*). During ALL-Linking sessions INSTEON devices send one of these messages and go into ALL-Linking Mode whenever a user physically presses and holds the SET Button on a device. INSTEON devices will also send one of the *Set Button Pressed* **SB** messages after receiving an *ID Request* **SD** (**S**tandard-length **D**irect) Command (**0x10**), although in that case they will not actually go into ALL-Linking Mode.

Whenever an INSTEON device sends one of the *Set Button Pressed* **SB** messages, the high byte of the 3-byte *To Address* field in the message contains the DevCat, and the middle byte contains the SubCat. The low byte of the *To Address* field has contained a firmware version in the past (see the next section, *Responding to a* Product Data Request *Message*85, for an alternative way to communicate the firmware version number). Always set this byte to 0xFF to ensure future compatibility.

The table below shows an INSTEON *SET Button Pressed* **SB** message sent by a device with an INSTEON ID of 0xCCCCCC, a DevCat of 0x01, and a SubCat of 0x03. The numbers are in hexadecimal.

***SET Button Pressed* Broadcast Message**		
From Address		0xCCCCCC
To Address H	**Device Category**	0x01
To Address M	**Device Subcategory**	0x03
To Address L	**Reserved (Firmware Version)**	0xFF (Used in the past for Firmware Version Number. Always set to 0xFF.)
Flags		0x8F (Broadcast Message, 3 Max Hops, 3 Hops Left)
Command 1		0x01 (*SET Button Pressed Responder*), or 0x02 (*SET Button Pressed Controller*)
Command 2	**Reserved**	0xFF (Unused. Always set to 0xFF.)

Responding to a *Product Data Request* Message

INSTEON devices may request other INSTEON devices to return certain product data, including their DevCat and SubCat numbers, by sending a *Product Data Request* **SD** (**S**tandard-length **D**irect) Command. The addressee will respond with a *Product Data Response* **ED** (**E**xtended-length **D**irect) Command with the following information in the 14-byte *User Data* field.

***Product Data Response* Extended-length Direct Message *User Data* Field**	
Byte	**Data**
D1	Reserved (always set to 0x00)
D2	INSTEON Product Key MSB
D3	INSTEON Product Key 2MSB
D4	INSTEON Product Key LSB
D5	Device Category (DevCat)
D6	Device Subcategory (SubCat)
D7	Reserved (always set to 0xFF) (Matches byte in LSB of *To Address* of *SET Button Pushed* Broadcast Commands)
D8	Reserved (always set to 0xFF) (Matches byte in *Command 2* of *SET Button Pushed* Broadcast Commands)
D9 ⇒ D14	User-defined

Note that the six bytes D9 through D14 are user-defined. If desired, the firmware version number that previously appeared in the LSB of the *To Address* field of *SET Button Pressed* **SB** messages may appear here, along with any other data that the device manufacturer may require.

Using DevCats to Qualify INSTEON Commands

The primary reason that INSTEON DevCats exist is to expand the space of possible INSTEON **SD** and **ED** Direct Commands. INSTEON **SD** and **ED** Commands consist of two bytes occupying the *Command 1* and *Command 2* fields of INSTEON **SD** (**S**tandard-length **D**irect) and **ED** (**E**xtended-length **D**irect) messages, respectively.

Two bytes can enumerate only 65,536 possible different Commands. Considering that many Commands use the *Command 2* field as a parameter (for example, to give a brightness level for turning on a lamp), INSTEON would soon run out of Command space as new Commands are defined. By making **SD** and **ED** Command interpretation dependent on the DevCat, the number of Commands is potentially multiplied by 256, giving 16,777,216 possible **SD** Commands and another 16,777,216 possible **ED** Commands, for a total of 33,554,432 possible Direct Commands.

An INSTEON Controller capable of sending Direct Commands will know at least some of the available Direct Commands for controlling one or more DevCats. It is up to an INSTEON Controller's *application program* to validate that the DevCats for which it knows the Direct Commands match the DevCats of any INSTEON Responder devices that it ALL-Links to. After ALL-Linking, Controllers may send Direct Commands to Responders only if the DevCat of the Direct Command matches the DevCat of the Responder.

Responders are *not* required to validate Direct Commands that they receive. It is assumed that the Controller sending the Direct Command validated that the DevCats matched during *INSTEON Device ALL-Linking*93, at the same time that the Controller learned the INSTEON ID (IID) number of the Responder. Therefore, Responders are free to accept Direct Commands from any Controller that knows the Responder's IID.

INSTEON Product Database

All INSTEON devices are assigned a 3-byte (24-bit) number, called the INSTEON Product Key (IPK), which functions as a lookup key to the INSTEON Product Database (IPDB). The IPDB is currently under construction and will be maintained by SmartLabs. Sufficiently advanced INSTEON devices that are part of an INSTEON network with access to the Internet will be able to query the online IPDB to determine the features and capabilities of other INSTEON devices. It will also be possible for one or more devices in an INSTEON network to maintain a local offline copy of the IPDB.

INSTEON Product Database information is most useful for determining Direct Command compatibility between INSTEON Controller and Responder devices during ALL-Linking sessions. *INSTEON Device ALL-Linking* 93 is always permitted whether or not the devices have access to the IPDB. If, however, there is a match between the DevCats of ALL-Linked Controllers and Responders, Controllers are also permitted to send **SD** (**S**tandard-length **D**irect) and **ED** (**E**xtended-length **D**irect) Commands to the Responders. But even if the DevCats match, Controllers may not know some or all of the **SD** and **ED** Commands that Responders can execute, or Controllers may possibly send Commands that ALL-Linked Responders do not recognize. By querying the IPDB, intelligent Controllers will be able to determine the capabilities of Responder devices during ALL-Linking and adapt appropriately. INSTEON devices with sufficient user interface resources will also be able to use information in the IPDB to provide feedback to users when they interact with the device, for example by displaying appropriate button labels.

This section gives the *IPK Support Requirements* 87, reprints the most recently available *INSTEON Product Key and SubCat Assignments* 88 table, and describes the *INSTEON Product Database (IPDB)* 91.

IPK Support Requirements

Although legacy INSTEON devices did not support IPKs, new INSTEON devices that ship after February 1, 2007 will be required to support them.

New INSTEON Devices

After February 1, 2007, all INSTEON devices will be required to have their 3-byte INSTEON Product Key loaded into nonvolatile memory during manufacturing, and to support the *Product Data Request* **SD** and *Product Data Response* **ED** Commands. Applications may then use the *Product Data Request* **SD** Command at any time to fetch a device's IPK in a *Product Data Response* **ED** message. See the section *Responding to a* Product Data Request *Message* 85 above for more information.

Legacy INSTEON Devices without IPKs

INSTEON devices manufactured before the requirement to support IPKs will not respond to a *Product Data Request* **SD** Command with a *Product Data Response*. The DevCat, SubCat, and (possible) Firmware Version numbers that appear in *SET Button Pressed* **SB** Commands issued by these devices are the only way that they can identify themselves to other devices. See the section SET Button Pressed *Broadcast Messages* 84 above for more information.

INSTEON Product Key and SubCat Assignments

The following table was current as of the publication date of this Developer's Guide. Although reprinted here for convenience, the official table is contained in the *INSTEON Device Categories and Product Keys Document*9 described in the *Other Documents Included by Reference*9 section above. *INSTEON DevCats and Product Keys 20070814a.doc* is the source for the table reprinted below.

The table gives examples of devices that belong to the various Device Categories (DevCats). Devices that are already developed or under development also have a Subcategory (SubCat) defined.

Model numbers (if known) are given in square brackets.

The table also gives INSTEON Product Keys (IPKs) that have been assigned. Legacy products that did not have an IPK defined at the time of manufacture are marked *Legacy*. IPKs are assigned sequentially.

Dev Cat	Device Category Name	Sub Cat	Product Key	Device Description [Model]
0x00	**Generalized Controllers** ControLinc, RemoteLinc, SignaLinc, etc.	0x04	Legacy	ControLinc [2430]
		0x05	0x000034	RemoteLinc [2440]
		0x06	Legacy	Icon Tabletop Controller [2830]
		0x09	Legacy	SignaLinc RF Signal Enhancer [2442]
		0x0A	0x000007	Balboa Instruments Poolux LCD Controller
		0x0B	0x000022	Access Point [2443]
		0x0C	0x000028	IES Color Touchscreen
0x01	**Dimmable Lighting Control** Dimmable Light Switches, Dimmable Plug-In Modules	0x00	Legacy	LampLinc V2 [2456D3]
		0x01	Legacy	SwitchLinc V2 Dimmer 600W [2476D]
		0x02	Legacy	In-LineLinc Dimmer [2475D]
		0x03	Legacy	Icon Switch Dimmer [2876D]
		0x04	Legacy	SwitchLinc V2 Dimmer 1000W [2476DH]
		0x06	Legacy	LampLinc 2-Pin [2456D2]
		0x07	Legacy	Icon LampLinc V2 2-Pin [2456D2]
		0x09	0x000037	KeypadLinc Dimmer [2486D]
		0x0A	Legacy	Icon In-Wall Controller [2886D]
		0x0D	0x00001E	SocketLinc [2454D]
		0x13	0x000032	Icon SwitchLinc Dimmer for Lixar/Bell Canada [2676D-B]
		0x17	Legacy	ToggleLinc Dimmer [2466D]
0x02	**Switched Lighting Control** Relay Switches, Relay Plug-In Modules	0x09	Legacy	ApplianceLinc [2456S3]
		0x0A	Legacy	SwitchLinc Relay [2476S]
		0x0B	Legacy	Icon On Off Switch [2876S]
		0x0C	Legacy	Icon Appliance Adapter [2856S3]
		0x0D	Legacy	ToggleLinc Relay [2466S]
		0x0E	Legacy	SwitchLinc Relay Countdown Timer [2476ST]
		0x10	0x00001B	In-LineLinc Relay [2475D]
		0x13	0x000033	Icon SwitchLinc Relay for Lixar/Bell Canada [2676R-B]
0x03	**Network Bridges** PowerLinc Controllers, TRex, Lonworks, ZigBee, etc.	0x01	Legacy	PowerLinc Serial [2414S]
		0x02	Legacy	PowerLinc USB [2414U]
		0x03	Legacy	Icon PowerLinc Serial [2814 S]
		0x04	Legacy	Icon PowerLinc USB [2814U]

Dev Cat	Device Category Name	Sub Cat	Product Key	Device Description [Model]
		0x05	0x00000C	Smartlabs Power Line Modem Serial [2412S]
0x04	**Irrigation Control** Irrigation Management, Sprinkler Controllers	0x00	0x000001	Compacta EZRain Sprinkler Controller
0x05	**Climate Control** Heating, Air conditioning, Exhausts Fans, Ceiling Fans, Indoor Air Quality	0x00	Legacy	Broan SMSC080 Exhaust Fan
		0x01	0x000002	Compacta EZTherm
		0x02	Legacy	Broan SMSC110 Exhaust Fan
		0x03	0x00001F	Venstar RF Thermostat Module
		0x04	0x000024	Compacta EZThermx Thermostat
0x06	**Pool and Spa Control** Pumps, Heaters, Chemicals	0x00	0x000003	Compacta EZPool
0x07	**Sensors and Actuators** Sensors, Contact Closures	0x00	0x00001A	IOLinc
		0x01	0x000004	Compacta EZSns1W Sensor Interface Module
		0x02	0x000012	Compacta EZIO8T I/O Module
		0x03	0x000005	Compacta EZIO2X4 #5010D INSTEON/X10 I/O Module for Dakota Alerts Products
		0x04	0x000013	Compacta EZIO8SA I/O Module
		0x05	0x000014	Compacta EZSnsRx RF #5010E Receiver Interface Module
		0x06	0x000015	Compacta EZISnsRf Sensor Interface Module
0x08	**Home Entertainment** Audio/Video Equipment			
0x09	**Energy Management** Electricity, Water, Gas Consumption, Leak Monitors	0x00	0x000006	Compacta EZEnergy
		0x01	0x000020	OnSitePro Leak Detector
		0x02	0x000021	OnSitePro Control Valve
		0x03	0x000025	Energy Inc. TED Measuring Transmitting Unit (MTU)
		0x04	0x000026	Energy Inc. TED Receiving Display Unit (RDU)
0x0A	**Built-In Appliance Control** White Goods, Brown Goods			
0x0B	**Plumbing** Faucets, Showers, Toilets			
0x0C	**Communication** Telephone System Controls, Intercoms			
0x0D	**Computer Control** PC On/Off, UPS Control, App Activation, Remote Mouse, Keyboards			
0x0E	**Window Coverings** Drapes, Blinds, Awnings	0x00	0x00000B	Somfy Drape Controller RF Bridge
0x0F	**Access Control** Automatic Doors, Gates, Windows, Locks	0x00	0x00000E	Weiland Doors Central Drive and Controller
		0x01	0x00000F	Weiland Doors Secondary Central Drive
		0x02	0x000010	Weiland Doors Assist Drive
		0x03	0x000011	Weiland Doors Elevation Drive
0x10	**Security, Health, Safety** Door and Window Sensors, Motion Sensors, Scales	0x00	0x000027	First Alert ONELink RF to INSTEON Bridge
0x11	**Surveillance** Video Camera Control, Time-lapse Recorders, Security System Links			
0x12	**Automotive** Remote Starters, Car Alarms, Car Door Locks			
0x13	**Pet Care** Pet Feeders, Trackers			
0x14	**Toys** Model Trains, Robots			

Dev Cat	Device Category Name	Sub Cat	Product Key	Device Description [Model]
0x15	**Timekeeping** Clocks, Alarms, Timers			
0x16	**Holiday** Christmas Lights, Displays			
0x17 ⇒ 0xFE	**Reserved**			
0xFF	**Unassigned** For devices that will be assigned a DevCat and SubCat by software			

INSTEON Product Database (IPDB)

The INSTEON Product Database (IPDB) will contain up to 16,777,216 records, each of which will refer to a distinct INSTEON product. The primary lookup key to a record in the IPDB will be the 3-byte INSTEON Product Key (IPK).

SmartLabs will host and maintain the online IPDB server.

Local IPDB Server

Devices on an INSTEON network that have sufficient resources may download full or partial copies of the IPDB in order to function as a local IPDB server. When a local IPDB server is available, the INSTEON network will only need to connect to the Internet intermittently. The rules for refreshing a stale local IPDB are not yet defined.

IPDB Record Fields

Some of the fields associated with an IPDB record may include:

- INSTEON device manufacturer
- Manufacturer's part number
- Device description
  - Device Category
    - Device Category (DevCat) Number (1 byte)
    - Device Subcategory (SubCat) Number (1 byte)
    - Device Category text description
  - Text description
  - Powerline, radio, or both?
  - Market
  - Power requirements
  - User interface
  - Link to User Guide
  - Link to photo
- Device capabilities (associated with Firmware Version number)
  - Release date
  - List of INSTEON Commands supported
    - FX Commands supported?
    - FX Username
  - List of FX Commands
  - SALad enabled?
  - Controller, Responder, or both?
  - Secure?

IPDB Query Response

When the IPDB is accessed with an INSTEON Product Key, the information in the IPDB record will be returned in XML format, so that it is both machine and human readable. The schema for this XML file is not yet defined.

INSTEON Device ALL-Linking

When a user adds a new device to an INSTEON network, the newcomer device joins the network automatically, in the sense that it can hear INSTEON messages and will repeat[1] them automatically according to the INSTEON protocol. So, no user intervention is needed to establish an INSTEON network of communicating devices.

However, for one INSTEON device to control other INSTEON devices, the devices must be logically ALL-Linked together. INSTEON provides two very simple methods for ALL-Linking devices—manual ALL-Linking using button pushes, and electronic ALL-Linking using INSTEON messages.

INSTEON ALL-Link Groups

During ALL-Linking, users create associations between events that can occur in an INSTEON Controller, such as a button press or a timed event, and the actions of an ALL-Link Group of one or more Responders. This section defines *ALL-Link Group Behavior*93, explains the *Number of ALL-Links Supported*93, shows how to handle *Controllers with Multiple Buttons per ALL-Link Group*94, differentiates between *ALL-Link Groups and ALL-Links*94, and gives *Examples of ALL-Link Groups*95.

ALL-Link Group Behavior

When an INSTEON Responder device ALL-Links to an INSTEON Controller device by joining one of the Controller's ALL-Link Groups, the Responder memorizes the state that it is in at the time of ALL-Linking, and associates that state with the Controller's INSTEON ID (IID) number and the ALL-Link Group Number that it is joining.

After ALL-Linking, the Responder goes back into the previously memorized state whenever it receives an ALL-Link Broadcast message with

10. A *From Address* matching the stored IID of the Controller,
11. An ALL-Link Group Number in the *To Address* low byte matching a stored ALL-Link Group Number, and
12. A *Command 1* field containing an *ALL-Link Recall* Command.

NOTE 1: The 'time of ALL-Linking' is the time that the user pushes the Responder's SET Button.

NOTE 2: An ALL-Link Group Number of 0xFF denotes *all* devices linked to a Controller. Responders interpret an ALL-Link Group Number of 0xFF in the *To Address* low byte as matching *any* stored ALL-Link Group Number.

NOTE 3: If the Responder is a Dimmable Lighting Control (DevCat 0x01), then the first time that it ALL-Links to a Controller, it must be in a fully on state, to avoid inadvertently linking in an off state. If a dimmer were to link in an off state, then it would appear that the dimmer was not working when the user first tried to turn it on.

Number of ALL-Links Supported

Each Controller device (with a unique IID) may create up to 256 ALL-Link Groups that one or more Responders can ALL-Link to. The minimum number of ALL-Link Groups for a Controller IID is one.

The maximum number of Responders that may ALL-Link to a given ALL-Link Group in a Controller depends only on the available memory for the Link Database in the Controller. Similarly, the maximum number of ALL-Link Groups that a Responder may join depends only on the available memory for the ALL-Link Database in the Responder. The minimum number of ALL-Link Groups that a Responder may join is one.

The time it takes to search a Responder's ALL-Link Database can present a practical limit on how many ALL-Link Groups a Responder may belong to. Before executing the Command in an **SA** ALL-Link Broadcast or **SC** ALL-Link Cleanup message, a Responder must search its Link Database for an IID and ALL-Link Group Number match, and this search takes time. The search time depends on the design of the database—a *Threaded ALL-Link Database (ALDB/T)*105, as used in devices like the SmartLabs PowerLinc™ Controller, is significantly faster to search than a *Linear ALL-Link Database (ALDB/L)*102, as used in simpler devices. A safe practical limit is fifty ALL-Link Group memberships for a Responder that uses linear searches.

Controllers with Multiple Buttons per ALL-Link Group

An ALL-Link Group typically corresponds to a single button on a Controller. For example, a Controller might have one button labeled *Scene 1 On* and another button labeled *Scene 1 Off*. A user could then ALL-Link the *Scene 1 On* button to one or more lamp dimmers in the ON state (forming ALL-Link Group 1), and ALL-Link the *Scene 1 Off* button to the same lamp dimmers in the OFF state (forming ALL-Link Group 2).

However, many INSTEON Controllers have toggle buttons or pairs of buttons that correspond to a *single* ALL-Link Group. A toggle button typically alternates between sending an *ALL-Link Recall* Command and an *ALL-Link Alias 1 Low* Command each time a user presses it. (See the next section for an explanation of *ALL-Link Alias Commands*116.)

In the case of paired buttons, the most common configuration is an ON/OFF pair. The *ON* button usually sends an *ALL-Link Recall* Command and the *OFF* button sends an *ALL-Link Alias 1 Low* Command. There are other configurations as shown in the table below.

Toggle or Paired Button ALL-Link Commands	
HIGH State (Toggle ON or Button 1)	**LOW State** (Toggle OFF or Button 2)
ALL-Link Recall	ALL-Link Alias 1 Low
ALL-Link Alias 2 High	ALL-Link Alias 2 Low
ALL-Link Alias 3 High	ALL-Link Alias 3 Low
ALL-Link Alias 4 High	ALL-Link Alias 4 Low

ALL-Link Groups and ALL-Links

An ALL-Link Group is a set of logical ALL-Links between INSTEON devices. An ALL-Link is an association between a Controller and a Responder or Responders. Controllers originate ALL-Link Groups, and Responders join ALL-Link Groups.

Internally, in an *INSTEON ALL-Link Database*[101] maintained by INSTEON devices, an ALL-Link Group ID consists of 4 bytes—the 3-byte address of the Controller, and a 1-byte ALL-Link Group Number. A Controller assigns ALL-Link Group Numbers as needed to the various physical or logical events that it supports. For example, a single press of a certain button could send commands to one ALL-Link Group, and a double press of the same button could send commands to another ALL-Link Group. The Controller determines which commands are sent to which ALL-Link Groups.

An ALL-Link Group can have one or many members, limited only by the memory available for the ALL-Link Database.

Examples of ALL-Link Groups

A device configured as a wall switch with a paddle could be designed to support one, two, or three ALL-Link Groups, as shown in the following examples.

One ALL-Link Group		
Controller Event	**Group**	**Action of ALL-Link Group Responders**
Tap Top	1	Turn On
Tap Bottom	1	Turn Off
Hold Top	1	Brighten
Hold Bottom	1	Dim

Two ALL-Link Groups		
Controller Event	**Group**	**Action of ALL-Link Group Responders**
Tap Top	1	Turn On
Tap Top Again	1	Turn Off
Tap Bottom	2	Turn On
Tap Bottom Again	2	Turn Off

Three ALL-Link Groups		
Controller Event	**Group**	**Action of ALL-Link Group Responders**
Tap Top	1	Turn On
Tap Bottom	1	Turn Off
Double Tap Top	2	Turn On
Double Tap Bottom	2	Turn Off
Triple Tap Top	3	Turn On
Triple Tap Bottom	3	Turn Off

Methods for ALL-Linking INSTEON Devices

There are two ways to create ALL-Links among INSTEON devices, *Manual ALL-Linking*96 and *Electronic ALL-Linking*96. This section also gives an *Example of an INSTEON ALL-Linking Session*97.

Manual ALL-Linking

Easy setup is very important for products sold to a mass market. INSTEON devices can be ALL-Linked together very simply:

- Push and hold for 10 seconds the button that will control an INSTEON device.
- Push and hold a button on the INSTEON device to be controlled.

This kind of manual ALL-Linking implements a form of security. Devices cannot be probed by sending messages to discover their addresses—a user must have physical possession of INSTEON devices in order to ALL-Link them together.

Designers are free to add to this basic ALL-Linking procedure. For example, when multiple devices are being ALL-Linked to a single button on a Controller, a *multilink* mode could enable a user to avoid having to press and hold the button for 10 seconds for each new device.

There must also be procedures to unlink devices from a button, and ways to clear ALL-Links from buttons in case devices ALL-Linked to them are lost or broken. See the *INSTEON ALL-Link Database*101 section below for more information on this point.

Electronic ALL-Linking

As the example below shows (see *Example of an INSTEON ALL-Linking Session*97), ALL-Linking is actually accomplished by sending INSTEON messages, so a PC or other device can create ALL-Links among devices if the device addresses are known and if devices can respond to the necessary commands.

To maintain security, PC-INSTEON interface devices such as SmartLabs' PowerLinc™ V2 Controller (PLC) mask the two high bytes of the address fields in INSTEON messages received from unknown devices. Devices are only known if there is an ALL-Link to the device stored in the ALL-Link Database of the PLC, or if the message's *To Address* matches that of the PLC. Such ALL-Links must have been previously established by manual button pushing or else by manually typing in the addresses of ALL-Linked devices (see *Masking Non-linked Network Traffic*112, below).

Example of an INSTEON ALL-Linking Session

This section outlines the message exchange that occurs when a Controller and Responder set up an ALL-Link relationship. In this scenario, a SmartLabs ControLinc™ V2 is the Controller, and a SmartLabs LampLinc™ V2 is the Responder. Numbers are in hexadecimal.

Message 1	**ControLinc: "I'm looking for ALL-Link Group members"**		
`00 00 CC 00 04 0C 8F` **`02`** `00`	ControLinc, with address of 00 00 CC, sends a *SET Button Pressed Controller* Broadcast message indicating it is now listening for Responders to be added to ALL-Link Group 1.		
	From Address		00 00 CC (ControLinc)
	To Address	Device Type	00 0A (ControLinc)
		Firmware Version	0C
	Flags		8F (Broadcast Message, 3 Max Hops, 3 Hops Left)
	Command 1		**02** (*SET Button Pressed Controller*)
	Command 2	Device Attributes	00 (Not used)

Message 2	**LampLinc: "My SET Button has been pressed"**		
`00 00 AA 00 02 30 8F` **`01`** `00`	LampLinc, with address of 00 00 AA, sends a *SET Button Pressed Responder* Broadcast message. When the ControLinc hears this, it will respond with a message to join ALL-Link Group 1.		
	From Address		00 00 AA (LampLinc)
	To Address	Device Type	00 02 (LampLinc)
		Firmware Version	30
	Flags		8F (Broadcast Message, 3 Max Hops, 3 Hops Left)
	Command 1		**01** (*SET Button Pressed Responder*)
	Command 2	Device Attributes	00 (Not used)

Message 3	**ControLinc: "Okay, join ALL-Link Group 1"**	
`00 00 CC 00 00 AA 0F` **`01 01`**	ControLinc (00 00 CC) sends message to LampLinc (00 00 AA) to join Group 1.	
	From Address	00 00 CC (ControLinc)
	To Address	00 00 AA (LampLinc)
	Flags	0F (Direct Message, 3 Max Hops, 3 Hops Left)
	Command 1	**01** (*Assign to ALL-Link Group*)
	Command 2	**01** (ALL-Link Group 1)

Message 4	**LampLinc: “I joined ALL-Link Group 1”**	
`00 00 AA 00 00 CC 2F` **`01 01`**	LampLinc (00 00 31) sends ACK to ControLinc (00 00 10).	
	From Address	00 00 AA (LampLinc)
	To Address	00 00 CC (ControLinc)
	Flags	2F (ACK of Direct Message, 3 Max Hops, 3 Hops Left)
	Command 1	**01** (*Assign to ALL-Link Group*)
	Command 2	**01** (ALL-Link Group 1)

Example of an ALL-Link Command Sequence

This example illustrates how messages containing ALL-Link Commands are passed from device to device in an ALL-Link Group. In this scenario, a SmartLabs ControLinc™ V2 ALL-Linked to two SmartLabs LampLinc™ V2 Dimmers in ALL-Link Group 1 commands them to turn on. Numbers are in hexadecimal.

Note that the **SA** ALL-Link Broadcast message (which both LampLinc Dimmers should respond to immediately) is followed by an acknowledged **SC** ALL-Link Cleanup message to each LampLinc Dimmer (in case they didn't get the Broadcast).

Message 1	**ControLinc: "ALL-Link Group 1, turn on"**		
`00 00 CC 00 00` **`01`** `CF` **`11`** `00`	ControLinc, with address of 00 00 CC, sends an ALL-Link Broadcast message to ALL-Link Group 1, with a Command of *On*.		
	From Address		00 00 CC (ControLinc)
	To Address	Unused	00 00
		ALL-Link Group Number	**01**
	Flags		CF (ALL-Link Broadcast Message, 3 Max Hops, 3 Hops Left)
	Command 1		**11** (*On*)
	Command 2		00 (Unused)

Message 2	**ControLinc: "LampLinc A, turn on"**		
`00 00 CC 00 00 AA 4F` **`11 01`**	ControLinc (00 00 CC) sends an ALL-Link Cleanup message to LampLinc A (00 00 AA) in ALL-Link Group 1, with a Command of *On*.		
	From Address		00 00 CC (ControLinc)
	To Address		00 00 AA (LampLinc A)
	Flags		4F (ALL-Link Cleanup Message, 3 Max Hops, 3 Hops Left)
	Command 1		**11** (*On*)
	Command 2	ALL-Link Group Number	**01**

Message 3	LampLinc A: "I turned on"		
00 00 AA 00 00 CC 2F **11 01**	LampLinc A (00 00 AA) sends ACK to ControLinc (00 00 CC).		
	From Address		00 00 AA (LampLinc A)
	To Address		00 00 CC (ControLinc)
	Flags		2F (ACK of Direct Message, 3 Max Hops, 3 Hops Left)
	Command 1		**11** (*On*)
	Command 2	ALL-Link Group Number	**01**

Message 4	ControLinc: "LampLinc B, turn on"		
00 00 CC 00 00 BB 4F **11 01**	ControLinc (00 00 CC) sends an ALL-Link Cleanup message to LampLinc B (00 00 BB) in ALL-Link Group 1, with a Command of *On*.		
	From Address		00 00 CC (ControLinc)
	To Address		00 00 BB (LampLinc B)
	Flags		4F (ALL-Link Cleanup Message, 3 Max Hops, 3 Hops Left)
	Command 1		**11** (*On*)
	Command 2	ALL-Link Group Number	**01**

Message 5	LampLinc B: "I turned on"		
00 00 BB 00 00 CC 2F **11 01**	LampLinc B (00 00 BB) sends ACK to ControLinc (00 00 CC).		
	From Address		00 00 BB (LampLinc B)
	To Address		00 00 CC (ControLinc)
	Flags		2F (ACK of ALL-Link Cleanup Message, 3 Max Hops, 3 Hops Left)
	Command 1		**11** (*On*)
	Command 2	ALL-Link Group Number	**01**

An INSTEON Controller will send **SC** ALL-Link Cleanup Commands to all Responder devices in an ALL-Link Group, unless other INSTEON traffic interrupts the cleanup, in which case the ALL-Link Cleanups will stop.

INSTEON ALL-Link Database

Every INSTEON device stores an ALL-Link Database in nonvolatile memory, representing Controller/Responder relationships with other INSTEON devices. Controllers know which Responders they are ALL-Linked to, and Responders know which Controllers they are ALL-Linked to. ALL-Link data is therefore distributed among devices in an INSTEON network.

If a Controller is ALL-Linked to a Responder, and the Responder is removed from the network without updating the Controller's ALL-Link Database, then the Controller will retry messages intended for the missing Responder. The retries, which are guaranteed to fail, will add unnecessary traffic to the network. It is therefore very important for users to unlink INSTEON Responder devices from Controllers when unused Responders are removed. Unlinking is normally accomplished in the same way as ALL-Linking—press and hold a button on the Controller, then press and hold a button on the Responder.

Because lost or broken Responder devices cannot be unlinked using a manual unlinking procedure, Controllers must also have an independent method for unlinking missing Responders. Providing a 'factory reset' procedure for a single Controller button, or for the entire Controller all at once, is common.

When a Controller is removed from the network, it should likewise be unlinked from all of its Responder devices before removal, or else the ALL-Link Databases in the Responders will be cluttered up with obsolete links. A 'factory reset' should be provided for Responder devices for this purpose.

There are two forms of ALL-Link Database Record, a high-performance threaded one for devices with a large number of ALL-Links such as SmartLabs' PowerLinc™ V2 Controller, and another, linear one for devices with limited memory. This section describes both.

In This Section

Linear ALL-Link Database (ALDB/L)

INSTEON devices with limited memory, such as SmartLabs' ControLinc™ V2, SwitchLinc™ V2, LampLinc™ V2, or ApplianceLinc™ V2, contain an ALL-Link Database whose records are stored sequentially, rather than in separate linked lists as in a *Threaded ALL-Link Database (ALDB/T)*105. Nevertheless, the data contained in a given record is similar.

SmartLabs does not recommend that you write your own ALDB/L routines. Instead, you can use new i2 INSTEON Commands for reading and writing ALDB/L records without having to know anything about the ALDB/L's internal organization. At a higher level, both the SmartLabs Device Manager and INSTEON Modems have ALDB/L utility routines that insulate you from the details.

Nevertheless, if you still need to write your own routines, the information below should be sufficient for you to manipulate an ALDB/L directly using the INSTEON *Peek* and *Poke* Commands discussed in *Using Peek and Poke Commands for One Byte*162.

ALDB/L Overview

The ALDB/L starts at the top of external (serial) EEPROM and grows downward. In most limited-memory INSTEON devices, top of memory is 0x0FFF. Each ALDB/L Record is 8 bytes long, so the first record starts at 0x0FF8, the second record starts at 0x0FF0, and so on. The ALDB/L starts out containing only one 8-byte physical record.

In what follows, the 3-byte INSTEON Address contained in a record is called the *Device ID* or sometimes just the *ID*. The high byte (MSB) of the Device ID is *ID2*, the middle byte is *ID1*, and the low byte (LSB) is *ID0*.

ALDB/L Record Format

The table below gives the format of an ALDB/L record. The explanation following the table walks you through the meaning of bits 1, 6, and 7 of the *Record Control* byte. (Bits 2 through 5 are product dependent. Contact the product manufacturer for the specific interpretation.) The *Device ID* is stored as three contiguous bytes, *ID2* first. The other fields, *Group*, *Data 1*, *Data 2*, and *Data 3*, serve the same purpose as the corresponding fields in a *Threaded ALL-Link Database (ALDB/T)*105. The table lists the record contents in the same order that they appear in memory, i.e. the first byte, *Record Control*, is stored at the lowest memory address.

Linear ALL-Link Database (ALDB/L) Record Format

Field	Length (bytes)	Description
Record Control	1	Record Control Flag Bits: Bit 7: 1 = Record is in use, 0 = Record is available Bit 6: 1 = Controller (Master) of Device ID, 0 = Responder to (Slave of) Device ID Bit 5: Product dependent Bit 4: Product dependent Bit 3: Product dependent Bit 2: Product dependent Bit 1: 1 = Record has been used before, 0 = 'High-water Mark' Bit 0: Reserved
Group	1	ALL-Link Group Number this Device ID belongs to (see *INSTEON ALL-Link Groups*93)
ID	3	Device ID (ID2, ID1, ID0 in that order)
Data 1	1	Link-specific data (e.g. *On-Level*)
Data 2	1	Link-specific data (e.g. Ramp Rates, Setpoints, etc.)
Data 3	1	Link-specific data (normally unused)

Adding Records to an ALDB/L

To add a record to an ALDB/L, you search for an existing record that is marked available. (Available means the same as empty, unused or deleted.) If none is available, you create a new record at the end of the ALDB/L.

An unused record will have bit 7 of the *Record Control* byte set to zero. The last record in an ALDB/L will have bit 1 of the *Record Control* byte set to zero.

Overwriting an Empty ALDB/L Record

If you found an empty record, you simply overwrite it with your new record data.

Change bit 7 of the *Record Control* byte from zero to one to show that the record is now in use.

Set bit 6 of the *Record Control* byte to one if the device containing the ALDB/L is an INSTEON Controller of the INSTEON Responder Device whose *ID* is in the record. If instead the device containing the ALDB/L is an INSTEON Responder to the INSTEON Controller Device whose *ID* is in the record, then clear bit 6 of the *Record Control* byte to zero. In other words, within an ALDB/L, setting bit 6 means "I'm a Controller," and clearing bit 6 means "I'm a Responder."

Put the ALL-Link Group number in the *Group* field, and put the *Device ID* in the *ID* field. Finally, set the *Data 1*, *Data 2*, and *Data 3* fields appropriately for the *Record Class* you are storing.

Creating a New ALDB/L Record

To create a new record at the end of the ALDB/T, find the record with bit 1 of the *Record Control* byte set to zero, indicating that it is the last record in the ALDB/L. Flip that bit to one.

Next, subtract 8 from the address of the *Record Control* byte in the record you just altered, and write a new *Record Control* byte there with bit 1 set to zero to show that this record is the new last record. Write all of the other information in the new record just as you would when *Overwriting an Empty ALDB/L Record*103.

Deleting Records from an ALDB/L

Deleting an existing record from an ALDB/L is simple—just set bit 7 of the *Record Control* byte to zero.

If the record you just deleted is the one immediately preceding the last record (i.e. the record with bit 1 of its Record Control byte set to one), then you should mark the newly-deleted record as the *new* last record, by flipping bit 1 of its Record Control byte to one.

Searching an ALDB/L

The most common search of an ALDB/L is for a particular 3-byte INSTEON ID and 1-byte ALL-Link Group number matching the *Device ID* and *Group* fields in a record. You will have to search the ALDB/L from the beginning until you find what you are looking for, or until you get to the end without finding it.

If you are searching the database for something that may occur multiple times, such as all records with a given ALL-Link Group Number, then you will have to look at all of the records in the ALDB/L.

Threaded ALL-Link Database (ALDB/T)

Because a Threaded ALL-Link Databases (ALDB/T) is 128 times faster to search than a *Linear ALL-Link Database (ALDB/L)*102, INSTEON devices such as SmartLabs' PowerLinc™ V2 Controller (PLC) employ the threaded version in order to support ALL-Linking to a large number of other INSTEON devices.

High performance in the ALDB/T comes at the cost of some increase in complexity. SmartLabs does not recommend that you write your own ALDB/T routines. Instead, you can use new i2 INSTEON Commands for reading and writing ALDB/T records without having to know anything about the ALDB/T's internal organization. At a higher level, both the SmartLabs Device Manager and the *SALad coreApp Program*272 have ALDB/T utility routines that insulate you from the details.

During SALad code development, you can directly read and write records to an ALDB/T if you are using the SALad Integrated Development Environment (IDE). See the *PLC Database*326 section of the *SALad Integrated Development Environment User's Guide*287.

Although not for the faint of heart, the information below should be sufficient for you to write your own routines for manipulating an ALDB/T directly using *IBIOS Serial Commands*196 or the INSTEON *Peek* and *Poke* Commands discussed in *Using Peek and Poke Commands for One Byte*162.

ALDB/T Overview

An ALDB/T starts at the top of external (serial) EEPROM and grows downward. Because of the way *Flat Memory Addressing*168 works, top of memory can always be found at 0xFFFF.

Each ALDB/T record is 8 bytes long, so the first physical record starts at 0xFFF8, the second physical record starts at 0xFFF0, and so on. The ALDB/T starts out containing a minimum of 128 physical records, so it occupies the top 1024 bytes of external EEPROM. The ALDB/T can grow larger than 1024 bytes, until it bumps up against the SALad application or other code that grows upward from the bottom of EEPROM.

In what follows, the 3-byte INSTEON Address contained in a record is called the *Device ID* or sometimes just the *ID*. The high byte (MSB) of the Device ID is *ID2*, the middle byte is *ID1*, and the low byte (LSB) is *ID0*. *MSb* and *LSb* refer to most and least significant *bits*, respectively. All addresses refer to the *Flat Memory Map*170.

ALDB/T Record Format

The table below gives the format of an ALDB/T record. The explanation following the table walks you through the meaning of the *Record Control* field. The other fields, *Group*, *Data 1*, *Data 2*, and *Data 3*, serve the same purpose as the corresponding fields in a *Linear ALL-Link Database (ALDB/L)*102. The table lists the record contents in the same order that they appear in memory, i.e. the first byte of *Record Control* is stored at the lowest memory address.

<table>
<tr><th colspan="3">Threaded ALL-Link Database (ALDB/T) Record Format</th></tr>
<tr><th>Field</th><th>Length (bytes)</th><th>Description</th></tr>
<tr><td>Record Control</td><td>2</td><td><pre>
          1st Byte 2nd Byte
          76543210 76543210   Record Class:
          XXXXXXXX XXXXXXXX   00 Deleted (ID0 LSb = 1 indicates end of ALDB/T)
Link----++++++++--+++++||| 01 Other (extended class)
Record Class-----------++| 10 INSTEON Responder to (Slave of) Device ID
ID0 LSb------------------+   11 INSTEON Controller (Master) of Device ID
</pre>
Link gives the 13 MSbs of the 16-bit address of the next record in this one of 128 possible database threads. The 3 LSbs of this address are 0.
If the 8 MSbs of Link are 0, this is the last record in the thread.
If the 8 MSbs of Link are 0, the Record Class is 00, and ID0 LSb is 1, then this is the last physical record in the database.
The 16-bit address of the first record in a thread is computed from ID0 by shifting ID0 left 2 (i.e. multiplying ID0 by 4), complementing the 16-bit result, then setting the 3 LSbs to 0.
Record Class indicates whether the record is available (Deleted), for an INSTEON Controller (Master), for an INSTEON Responder (Slave), or user-defined (Other).
ID0 LSb is the LSb of ID0 in the Device ID. </td></tr>
<tr><td>ID1</td><td>1</td><td>Middle byte of the Device ID</td></tr>
<tr><td>ID2</td><td>1</td><td>High (MSB) byte of the Device ID</td></tr>
<tr><td>Group</td><td>1</td><td>ALL-Link Group Number this Device ID belongs to (see INSTEON ALL-Link Groups 93)</td></tr>
<tr><td>Data 1</td><td>1</td><td>Link-specific data (e.g. On-Level)</td></tr>
<tr><td>Data 2</td><td>1</td><td>Link-specific data (e.g. Ramp Rates, Setpoints, etc.)</td></tr>
<tr><td>Data 3</td><td>1</td><td>Link-specific data (normally unused)</td></tr>
</table>

Each record in the ALDB/T contains the *ID* of an INSTEON device that the PLC is ALL-Linked to. The PLC may be ALL-Linked to the same *ID* multiple times, each time in a different *ALL-Link Group*. To search for or to store a record in the ALDB/T, you use the least-significant byte of the record's *ID*, i.e. *ID0*, as a lookup key.

ALDB/T Threads

The ALDB/T is organized as a set of 128 *linked lists* of records. In the following discussion, each linked list is called a *thread*.

A record's *ID0* tells which thread to store the record in. All records with the same *ID0* will be stored somewhere in the same thread. There are only 128 threads, but the value of *ID0* can range from 0 to 255 (0x00 to 0xFF), so both even and odd *ID0* numbers are stored in the same thread. Thus, records with an *ID0* of 0x00 or 0x01 will be stored in the first thread, records with an *ID0* of 0x02 or 0x03 will be stored in the second thread, and so forth, up to records with an *ID0* of 0xFE or 0xFF, which will be stored in the 128th thread.

The first record in the first thread is located at the top of memory, occupying 8 memory locations from 0xFFF8 to 0xFFFF. The first record in the second thread starts 8 bytes below the first record, at 0xFFF0. The first records for the remaining 126 threads each occupy the next lower 8 bytes, down to a starting point of 0xFC00 for the first record in the 128th thread.

Now, given a particular *ID0*, you can calculate the memory address of the first record in the thread for that *ID0* very simply. Just multiply *ID0* by four (that is, shift it left by 2 into a 16-bit value), complement the 16-bit value, and then set the 3 LSbs of the 16-bit value to 0 by ANDing the 16-bit value with 0xFFF8.

As an example, let's try an *ID0* of 0x01. Shifting left 2 gives 0x0004. Complementing gives 0xFFF83. ANDing with 0xFFF8 gives 0xFFF8, which is the correct starting address of the first record in the first thread, as expected. Note that starting with an *ID0* of 0x00, you would get the same record starting address. The table below shows this and a few other examples.

Calculating an ALDB/T Record Address from ID0				
ID0	**Shifted Left 2**	**Complemented**	**ANDed with 0xFFF8 = ALDB/T Thread Address**	**Thread Number**
0x00	0x0004	0xFF83	**0xFFF8**	0
0x01	0x0006	0xFF81	**0xFFF8**	0
…	**…**	**…**	**…**	**…**
0x37	0x00DC	0xFF23	**0xFF20**	27
…	**…**	**…**	**…**	**…**
0xA2	0x0288	0xFD77	**0xFD70**	81
…	**…**	**…**	**…**	**…**
0xFE	0x03F8	0xFC07	**0xFC00**	127
0xFF	0x03FC	0xFC03	**0xFC00**	127

ALDB/T Record Control Field

The first two bytes of a record contain a 16-bit value called the *Record Control* field.

Link to Next Record

When you take the 3 LSbs of the *Record Control* field to be zero, the full 16 bits of the field make up a memory address that points to the first byte of another 8-byte ALDB/T record. In other words, the top 13 bits of the *Record Control* field, along with 3 more low bits set to zero, constitute a memory pointer, or *Link*, that always takes the form 0xXXX0 or 0xXXX8.

The *Link* within an ALDB/T record gives the memory address (i.e. points to) the *next* ALDB/T record in a thread. If there is no next record, then the top 8 bits of *Link* will be set to zero to designate the last record in a thread.

ID0 Least-Significant Bit

Bit 0, the LSb, of the *Record Control* field indicates whether *ID0* for this record is even or odd. Called *ID0 LSb*, this bit is just the LSb of *ID0* for the record, as advertised. We need this bit because a given thread contains all of the records whose *ID0* is the same except for the LSb. In other words, knowing which thread we're in tells what the top seven bits of *ID0* are, and *ID0 LSb* tells what the low bit is.

Record Class

Bits 2 and 1 of the *Record Control* field designate the ALDB/T record's *Record Class*.

If the *Record Class* is 10, then the *ID* in this ALDB/T record belongs to an INSTEON Responder (Slave) Device. The device containing the ALDB/T is therefore an INSTEON Controller (Master) of the Responder in the ALDB/T record.

Similarly, if the *Record Class* is 11, then the *ID* in the ALDB/T record belongs to an INSTEON Controller (Master) Device. The device containing this ALDB/T is therefore an INSTEON Responder (Slave) to the Controller in the ALDB/T record.

(Another way to explain this is, within an ALDB/T, a *Device Class* of 10 means "I'm a Controller," and 11 means "I'm a Responder.")

A *Record Class* of 01 indicates that the record contains information that may be interpreted in different ways, depending on the application.

If both of the *Record Class* bits are zero, then this record is *deleted*, i.e. no longer in use. In this discussion, deleted means the same as empty, unused, or available. Deleted records are not removed from the ALDB/T. Instead, they are merely marked as available for future use by setting the *Record Class* to 00.

The last *physical* record in the ALDB/T has a *Record Control* field with a high byte of 0x00 (last record in a thread), a *Record Class* of 00 (deleted), and an *ID0 LSb* of 1. Records that are deleted but are not the last physical record will therefore have an *ID0 LSb* set to zero.

An Empty ALDB/T

An empty ALDB/T starts out looking like this:

Empty ALDB/T											
Thread Number	Record's Address	Addr + 0	Addr + 1		Addr + 2	Addr + 3	Addr + 4	Addr + 5	Addr + 6	Addr + 7
		Record Control			ID1	ID2	Group	Data 1 ⇒ Data 3		
		Link, 13 bits	Class, 2 bits	LSb, 1 bit	8 bits	8 bits	8 bits	24 bits		
0	**0xFFF8**	0x0000 / 8	0b00	0b0	0xXX	0xXX	0xXX	0xXXXXXX		
1	**0xFFF0**	0x0000 / 8	0b00	0b0	0xXX	0xXX	0xXX	0xXXXXXX		
…	**…**	…	…	…	…	…	…	…		
…	**…**	…	…	…	…	…	…	…		
126	**0xFFC8**	0x0000 / 8	0b00	0b0	0xXX	0xXX	0xXX	0xXXXXXX		
127	**0xFFC0**	0x0000 / 8	0b00	0b0	0xXX	0xXX	0xXX	0xXXXXXX		
N/A	**0xFFB8**	0xXXXX / 8	0b00	0b1	0xXX	0xXX	0xXX	0xXXXXXX		

In the table, the prefix 0x designates a hexadecimal number and 0b designates a binary number. 0xX… means the hex digits don't matter, and 0bB… means the binary bits don't matter. The notation 0xXXXX / 8 means just take the most significant 13 bits (i.e. ignore the low 3 bits).

There are 128 threads in the ALDB/T, each containing one record. Each of those records is marked empty (each *Record Class* is 00) and also designated the last record in a thread (each *Link* high byte is 0x00). None of these records is the last physical record in the ALDB/T because even though the *Record Class* is 00, the *ID0 LSb* is zero.

Note the additional record at address 0xFFB8. This record is also empty (its *Record Class* is 00) but it is the last physical record in the ALDB/T (because it is empty with an *ID0 LSb* of 1). To avoid having to search the entire ALDB/T for the last physical record each time you need to add a new record, you should keep a variable, *LastALDBRecordAddress*, for saving the address of the last physical record. Thus, in an empty ALDB/T, *LastALDBRecordAddress* would contain 0xFFB8.

Adding Records to an ALDB/T

To add a record to an ALDB/T, you first calculate the address of the first record in the thread corresponding to *ID0*. (Remember, shift *ID0* left by two into a 16-bit value, complement the result, and then zero out the three low bits.)

Next, search that thread for an empty record. (Empty means the same as available, unused, or deleted.) Starting at the address you calculated, look at the *Record Control* field in the first two bytes at that address. If bits 2 and 1 (the *Record Class*) are 00, then the record is empty. If the record is not empty, go to the next record in the thread, which you will find at the *Link* address that you get by zeroing out the three low bits of the *Record Control* field.

Follow the links until you find an empty record (*Record Class* 00) or until the high byte of *Link* is 0x00, signifying the end of the thread. If you got to the end of the thread without finding an empty record, you will have to create a new physical record at the physical end of the ALDB/T.

Overwriting an Empty ALDB/T Record

If you found an empty record, you simply overwrite it with your new record data, except for the *Link* portion (top 13 bits) of the *Record Control* field, which will remain unchanged.

Put the low bit of the *ID0* that you are storing into the *ID0 LSb* bit of the *Record Control* field, then put *ID1* and *ID2* into the *ID1* and *ID2* fields of the record, respectively. Change the *Record Class* bits of the *Record Control* field from 00 to one of 11, 10, or 01, depending on the type of record you are storing. Put the ALL-Link Group number in the *Group* field. Finally, set the *Data 1*, *Data 2*, and *Data 3* fields appropriately for the *Record Class* you are storing.

Creating a New ALDB/T Record

To create a new record at the end of the ALDB/T, fetch the address saved in *LastALDBRecordAddress*. Write that address into the *Link* portion (the top 13 bits) of the *Record Control* field in the last record of the thread you just searched (the record with a *Link* high byte of 0x00). Be careful not to alter the three low bits of that *Record Control* field.

Now go to the new record at *LastALDBRecordAddress*. Set the high byte of the *Record Control* field at that address to 0x00 to signify that this record is the new last record in the thread. Set the remaining information in the record just as you would if you were Overwriting an Empty ALDB/T Record 109.

Finally, you must create a new, empty, last physical record. Subtract 8 from the value in *LastALDBRecordAddress*, and store that new value in *LastALDBRecordAddress*.

Set the *Record Class* bits in the *Record Control* field at that new address to 00 to show that that the new record is empty, and set the *ID0 LSb* bit one to show that the new empty record is also the last physical record in the ALDB/T. It does not matter what the other bits in the record are, because they will be overwritten if the record gets used.

In most applications, memory is limited, so you should do bounds-checking to avoid overwriting whatever is in lower memory as the ALDB/T grows downward.

Deleting Records from an ALDB/T

Deleting an existing record from an ALDB/T is simple—just set the *Record Class* (bits 2 and 1 of the *Record Control* field) to 00.

Of course, this method does not physically remove the record, so there will be gaps in the ALDB/T. The gaps should not be a problem, though, because the next time you add a record, it could go in any thread with equal probability, since the *ID0* of INSTEON devices is effectively random.

It is possible to write a defragmentation algorithm that would close up the gaps in the ALDB/T threads, but it is far simpler just to have adequate memory available.

Searching an ALDB/T

If you are searching the ALDB/T for a record with a particular *ID*, use *ID0* to calculate the starting address for the thread containing the *ID*, and then follow the links in the thread until *ID1*, *ID2*, and *ID0 LSb* match. (Remember, to find the first record in the *ID*'s thread, shift *ID0* left by two into a 16-bit value, complement the result, and then zero out the three low bits. A *Link* is just the *Record Control* field with the three low bits set to zero.) If you get to the end of the thread without finding the *ID*, then the *ID* is not in the ALDB/T. The last record in a thread has a *Link* high byte of 0x00.

If you are searching the database for something that may occur multiple times, such as all records with a given ALL-Link Group Number and/or all records in a given *Record Class*, then you will have to look at all 128 threads. As you increment through the threads, keep a *ThreadIndex* counter running from 0x00 to 0x7F. To recover *ID0* of the *Device ID* in a given record, multiply *ThreadIndex* by 2 and add in the *ID0 LSb* bit.

ALDB Performance Comparison

For this comparison, we assume that the ALDB is stored in external serial EEPROM, and that a serial EEPROM transaction takes 25 µs (microseconds). Reading a serial EEPROM requires four overhead transactions, plus one transaction per byte read. The overhead transactions are:

1. Tell the EEPROM you are writing an address.
2. Write the address high byte
3. Write the address low byte.
4. Put the EEPROM into read mode.

You can then read bytes sequentially, with the EEPROM automatically incrementing the read address after each byte that you read.

Let's assume that we want to perform the most common search, which is for a match to a given 3-byte INSTEON ID and 1-byte ALL-Link Group number. In an ALDB/L, we will have to read from 2 to 5 bytes—the *Record Control* byte and the *Group*, and possibly three of the *ID* bytes—before discovering a mismatch. In an ALDB/T, we will also have to read from 2 to 5 bytes—in this case the 2-byte *Record Control* field, then possibly *ID1*, *ID2*, and the *Group* bytes—before discovering a mismatch.

Taking the average to be four bytes for either type of ALDB, and adding in the four overhead transactions, it takes an average of 200 µs to search a record and eliminate it as a match.

How much time we have available to perform a search depends on how incoming INSTEON messages are buffered. Worst case, if there is only a single buffer, a received message can be overwritten by new INSTEON traffic as it occurs. In that case, we have only 14 milliseconds (ms) to perform the search. 14 ms is the time between the completion of Standard-length message reception and the possible arrival of a new message (which could be an acknowledgement of the message just received). If there is a double buffer, then we can process a received Standard-length message during the entire time that a new message is coming in. With double buffering, we have 50 ms to perform the search.

At 200 µs per record, we can search 70 records in an ALDB/L in 14 ms, or 250 records in 50 ms. In an ALDB/T, we can search 128 times as many records in the same time, because we immediately know which one of the 128 threads to look in. Thus, with the ALDB/T we effectively search 8,960 records in 14 ms, or 32,000 records in 50 ms. The table below summarizes this result.

Average ALDB Records Searchable		
ALDB Type	**Single Buffer (14 ms)**	**Double Buffer (50 ms)**
ALDB/L	70	250
ALDB/T	8,960	32,000

INSTEON Security

INSTEON network security is maintained at two levels. *ALL-Linking Control*112 ensures that users cannot create ALL-Links that would allow them to control their neighbors' INSTEON devices, even though those devices may be repeating each other's messages. *Encryption within Extended-length Messages*113 permits completely secure communications for applications that require it.

ALL-Linking Control

INSTEON enforces ALL-Linking Control by requiring that users have *Physical Possession of Devices*112 in order to create ALL-Links, and by *Masking Non-linked Network Traffic*112 when messages are relayed outside the INSTEON network itself.

Physical Possession of Devices

Firmware in INSTEON devices prohibits them from identifying themselves to other devices unless a user physically presses a button on the device. That is why the Command in the network identification Broadcast message is called *SET Button Pressed*. As shown above in the section *Example of an INSTEON ALL-Linking Session*97, a user has to push buttons on both the Controller device and the Responder device in order to establish an ALL-Link between them. A Responder will not act on Commands from an unlinked Controller.

ALL-Linking by sending INSTEON messages requires knowledge of the 3-byte addresses of INSTEON devices. These addresses, unique for each device, are assigned at the factory and displayed on printed labels attached to the device. Users who have physical possession of a device can read the device address from the label and manually enter it when prompted by a computer program.

Masking Non-linked Network Traffic

As described in the section *Interfacing to an INSTEON Network*28, above, there can be many kinds of INSTEON devices, called Bridges, that connect an INSTEON network to the outside world. But since an INSTEON Bridge is itself just another INSTEON device, it must be ALL-Linked to other devices on the INSTEON network in order to exchange messages with them. A user must establish these ALL-Links in the same way as for any other INSTEON device—by pushing buttons or by typing in addresses.

SmartLabs' PowerLinc™ Controller (PLC) is an example of an INSTEON-certified Bridge device that monitors INSTEON traffic and relays it to a computer via a serial link. For security, the PLC's firmware masks the all but the two low-bytes of the *From Address* and *To Address* fields of INSTEON messages unless the traffic is from an INSTEON device already ALL-Linked to the PLC, or the traffic is from a device that already knows the address of the PLC. In this way, software can take into account the existence of INSTEON traffic without users being able to discover the addresses of devices that they never had physical access to.

To avoid 'spoofing,' where an attacker poses as someone else (by causing the PLC to send messages with bogus From Addresses), the PLC's firmware always inserts the true PLC ID number in the *From Address* field of messages that it sends.

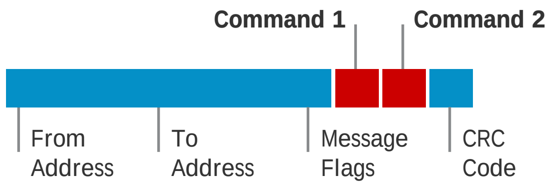
Encryption within Extended-length Messages

For applications such as door locks and security systems, INSTEON Extended-length messages can contain encrypted payloads. Possible encryption methods include rolling-code, managed-key, and public-key algorithms. In keeping with INSTEON's hallmark of simplicity, rolling-code encryption, as used by garage door openers and radio keyfobs for cars, is the method preferred by SmartLabs. The encryption method that will be certified as the INSTEON standard is currently under development.

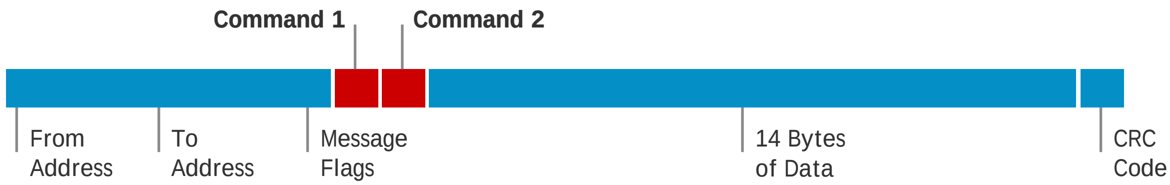
Chapter 8 — INSTEON Command Set

All INSTEON messages, whether Standard-length or Extended-length, contain two one-byte fields called *Command 1* and *Command 2* in the eighth and ninth byte positions respectively, as shown below.

Standard-length INSTEON Message



Extended-length INSTEON Message



Both fields may be used together to contain a single two-byte Command in the case of **SB** Broadcast or **SD** or **ED** Direct messages. In the case of **SA** ALL-Link Broadcast or **SC** ALL-Link Cleanup messages, however, only the *Command 1* field is available because the ALL-Link Group Number occupies the *Command 2* field within **SC** ALL-Link Cleanup messages. (In **SA** ALL-Link Broadcast messages the ALL-Link Group Number appears as the low byte in the *To Address* field, and the *Command 2* field is set to 0x00.)

In This Chapter

INSTEON Command Categories

INSTEON Command Numbers may be interpreted six different ways, depending on the type of INSTEON message in which they appear. The following table shows the six possibilities, although the two in the darkened rows are not currently used.

Command Type	Command Designator	Valid for These Message Types	Message Length	Command Bytes
Direct Commands	**SD**	**SD**	Standard	2
	ED	**ED**	Extended	16
ALL-Link Commands	**SA**	**SA, SC**	Standard	1
	EA	**EA, EC**	Extended	15
Broadcast Commands	**SB**	**SB**	Standard	1
	EB	**EB**	Extended	16

The *Command Designator* and *Valid for These Message Types* columns use the same abbreviations as first introduced in the *INSTEON Message Summary Table*46 above. The first letter is the message length, either **S** for Standard-length or **E** for Extended-length. The second letter is **D** for Direct, **A** for ALL-Link Broadcast, **C** for ALL-Link Cleanup, or **B** for Broadcast. The text below and the tables of Commands all use these same Command Designators.

SD and **ED** Direct Commands appear in **SD** and **ED** Direct Messages, respectively. **SA** ALL-Link Commands appear in both **SA** ALL-Link Broadcast and **SC** ALL-Link Cleanup Messages. **SB** Broadcast Commands appear in **SB** Broadcast Messages.

ALL-Link Commands

ALL-Linking allows any INSTEON Controller device to operate any INSTEON Responder device, even if the Controller does not know any of the Direct Commands that the Responder can execute. The principle is simple—during ALL-Linking to a button on a Controller, a Responder memorizes the state that it is in at the time. After ALL-Linking, pushing that button on the Controller causes the Responder to go back into the state that it memorized when it ALL-Linked.

During *INSTEON Device ALL-Linking*93, when a button on a Controller ALL-Links to a Responder, the Controller creates an *ALL-Link Group*, which the Responder joins (see *INSTEON ALL-Link Groups*93, below). Multiple Responders can join the same ALL-Link Group, so it is possible for a single button push to cause an entire ensemble of devices to recall their memorized states. All of the Responder devices in the ALL-Link Group will recall their memorized states simultaneously, because when the Controller's button is pushed, the Controller first sends out an **SA** ALL-Link Broadcast message to all of the Group members at once, followed by individual **SC** ALL-Link Cleanup messages to each Group member in turn, as described in the sections *SA ALL-Link Broadcast Messages*48 and *SC ALL-Link Cleanup Messages*48 above.

Note that although **EA** and **EC** Extended-length ALL-Link Broadcast and ALL-Link Cleanup Commands are logically possible, INSTEON does not currently use them.

Universally-Required ALL-Link Command

A basic requirement for INSTEON conformance is ALL-Link support.

All INSTEON devices must implement an *ALL-Link Recall* **SA** Command, no matter what DevCat the device belongs to, in order to support ALL-Linking. In the *INSTEON Command Set Tables*124, universally-required Commands are listed in **bold type** and color-coded yellow.

The required *ALL-Link Recall* Command is reprinted in this document in the section *Required Commands for All INSTEON Devices*157 below.

ALL-Link Alias Commands

The only required ALL-Link Command is *ALL-Link Recall*, but there are several additional ALL-Link Commands, called *ALL-Link Alias* Commands, that Responders may optionally execute.

When a Responder receives one of the ALL-Link Alias Commands in an ALL-Link Broadcast message, it checks to see if it has previously stored a substitute Direct Command to execute in place of the ALL-Link Alias Command. If the Responder does find a substitute Direct Command, then it executes the substitute Command just as it would if it had received the Direct Command within an INSTEON Direct message. The substitute Direct Command may be Standard-length or Extended-length (**SD** or **ED**). Because a Direct Command of 0x0000 will never be defined, a substitute Direct Command of 0x0000 means 'do nothing.'

The substitute Direct Command may be pre-programmed into the Responder as a default. Defaults may be altered over the INSTEON network via *Set ALL-Link Command Alias* Commands, or by the use of an appropriate user interface.

Lighting control devices have pre-programmed default substitute **SD** Commands as shown in the table below. Lighting control devices are those with DevCats of 0x01 (Dimmable Lighting Control), or 0x02 (Switched Lighting Control).

ALL-Link Command		Default Substitute SD Commands for Lighting Controls	
HIGH State	**LOW State**	**HIGH State**	**LOW State**
ALL-Link Recall	ALL-Link Alias 1 Low	N/A	Light OFF
ALL-Link Alias 2 High	ALL-Link Alias 2 Low	Light ON Fast	Light OFF Fast
ALL-Link Alias 3 High	ALL-Link Alias 3 Low	Light Brighten One Step	Light Dim One Step
ALL-Link Alias 4 High	ALL-Link Alias 4 Low	Light Start Manual Change	Light Stop Manual Change

Note that the *ALL-Link Recall* Command never has a substitute Direct Command, because *ALL-Link Recall* is the basic required Command to support ALL-Linking. In the case of lighting controls, the effect will be the same as executing a *Light ON* **SD** Command, because an ALL-Linked light will go to a saved *On-level* at a saved *Ramp Rate*.

Direct Commands

INSTEON **SD** (**S**tandard-length **D**irect) Commands consist of two bytes, *Command 1* and *Command 2*. INSTEON **ED** (**E**xtended-length **D**irect) Commands consist of the same *Command 1* and *Command 2* bytes plus fourteen additional bytes, *D1* through *D14*.

The interpretation of any given Direct Command Number depends on the DevCat (Device Category) that the Direct Command is associated with. See *Using DevCats to Qualify INSTEON Commands*86 above for more information.

Two-byte **SD** Commands are the payload within Standard-length INSTEON messages. **SD** Commands are intended for frequent use, fast response, or both. **ED** Commands, which require Extended-length INSTEON messages to transport, can be more elaborate but they take more time to transmit.

Required Direct Commands

Although the **SD** and **ED** *INSTEON Command Set Tables*124 list a large (and growing) number of Direct Commands, only a small subset of them will typically be required for a given INSTEON device, as explained below.

Universally-Required Direct Commands

All INSTEON devices must implement a small number of Direct Commands, no matter what DevCat the device belongs to, in order to support ALL-Linking and fetching product data. In the *INSTEON Command Set Tables*124, universally-required Commands are listed in **bold type** and color-coded yellow.

Universally-required Direct Commands are reprinted in this document in the section *Required Commands for All INSTEON Devices*157 below.

Conditionally-Required Direct Commands

Some Direct Commands are required only under certain conditions. For example, products that utilize *User-Defined FX Commands*121 must support *FX Username Request* and *FX Username Response* Commands. In the *INSTEON Command Set Tables*124, conditionally-required Commands are also listed in **bold type** and color-coded yellow, except that the condition for requirement is given in red type.

Conditionally-required Direct Commands are reprinted in this document in the section *Required Commands for Some INSTEON Devices*160 below.

Required Direct Commands within a DevCat

Within a DevCat, a small set of Direct Commands may be required in order to guarantee basic functionality within the DevCat. For example, all lighting controls must support *Light On* and *Light Off*, and dimmable lighting controls must also support *Light Brighten* and *Light Dim*. Required Direct Commands within a DevCat are given in the *INSTEON Command Set Tables*124 in underline type.

Returned Data Following a Direct Command

All **SD** and **ED** Commands from a sender to an addressee are followed by a Standard-length acknowledgement message from the addressee back to the sender (see SD ACK and SD NAK Messages47 above). The acknowledgement message serves as a confirmation that the addressee received the outgoing **SD** or **ED** message without error. Normally, the addressee simply echoes the received *Command 1* and *Command 2* fields in the ACK message. However, some **SD** or **ED** Commands specifically request one or two bytes of returned data, which may be contained in the acknowledgement message. (For completeness, note that **SC** ALL-Link Cleanup messages also receive acknowledgements.)

Returning a NAK

When a Responder receives a Direct Command from a Controller, and the Responder cannot execute the Command because the Command is not in its repertoire, then the Responder may return a *Direct NAK* message instead of a *Direct ACK* message to the Controller by altering the message flag bits.

NAK Error Codes

If the recipient of an **SD** or **ED** Direct or **SC** ALL-Link Cleanup message responds to the message originator with a NAK, the **SD** or **SC** NAK message will contain the reason for the NAK in the *Command 2* field (see INSTEON Message Summary Table46). These are the NAK Error codes:

NAK Code	Error
0x00 ⇒ 0xFC	Reserved
0xFD	Unknown INSTEON Command
0xFE	No load detected
0xFF	Not in ALL-Link Group

Returning an ACK

When a Responder receives an **SD** or **ED** Direct message or an **SC** ALL-Link Cleanup message from a Controller, and validates that the received message is error-free, then the Responder's INSTEON Engine automatically returns an **SD** or **SC** ACK message to the Controller (see SD ACK and SD NAK Messages47 and SC ACK and SC NAK Messages48, above). Normally, the *Command 1* and *Command 2* fields of the ACK message simply echo the *Command 1* and *Command 2* fields of the received message. However, if the received Command is one that requests just one or two bytes of returned data, an application may return the data in those fields. Only selected **SD** and **ED** Direct Commands expect returned data, and that data is normally one byte in the *Command 2* field.

Because ACK messages are part of a timed INSTEON message cycle, an application only has a limited amount of time to insert the returned bytes in the ACK message. Worst case (when the message is received on the last hop), that time is 15 milliseconds.

Returning Data Using Request/Response Commands

When more than one or two bytes of data must be returned, the data may be contained in an Extended-length message. The **SD** and **ED** Direct *INSTEON Command Set Tables* 124 contain several request/response pairs, where the request is an **SD** or **ED** Command, and the response is an **ED** Command. Because the response is an independent, asynchronous message, and not part of a timed cycle, applications have more time to compose the response. However, applications that request a response should set a timer and not block further processing after a timeout in case there is no response for whatever reason.

User-Defined FX Commands

INSTEON supports user-defined Direct Commands known as *FX Commands*, so named because the *Command 1* field ranges from 0xF0 to 0xFF (and because these Commands may create special *effects*).

Matching FX Usernames

In order to use FX Commands, both the Controller and Responder devices must be pre-programmed with an 8-byte *FX Username* in nonvolatile read-only memory, and both FX Usernames must match before a Controller may send an FX Command to a Responder. The Controller's *application program* has the responsibility to check that the FX Username in the Controller matches the FX Username in any Responder devices *before* it sends FX Commands to them.

A Controller may check for an FX Username match just after ALL-Linking to a Responder, or it may check at any other time as needed. To check an INSTEON device's FX Username, another device may send it an *FX Username Request* **SD** Command. The queried device will respond with an *FX Username Response* **ED** Command. The first eight data bytes, *D1* through *D8*, in the *FX Username Response* Command contains the FX Username. The remaining six data bytes, *D9* through *D14*, may be user-defined.

To ensure that all 8-byte FX Usernames are unique, SmartLabs maintains an FX Username database. Manufacturers who wish to use FX Commands need to submit their desired FX Usernames to Smartlabs for approval before building devices that use them.

All INSTEON devices that utilize FX Commands must implement the **ED** *FX Username Response* Command. Controller devices that can send FX Commands must also implement the **SD** *FX Username Request* Command. Devices that do not utilize FX Commands should respond to an *FX Username Request* Command with an **SD** NAK. Legacy devices, however, may respond with an **SD** ACK and then fail to send the **ED** *FX Username Response* message.

FX Command Definitions

The value from 0xF0 through 0xFF in the *Command 1* field of **SD** or **ED** messages may be interpreted in whatever way the device designer desires. The *Command 2* field, ranging from 0x00 to 0xFF, may be freely defined, and in the case of **ED** messages, the fourteen bytes *D1* through *D14* are also user-defined.

SmartLabs encourages manufacturers who utilize FX Commands to disclose them so that they may be published in a SmartLabs-maintained database. Popular FX Commands are candidates for standardized Direct Commands defined within a DevCat.

Data Transfer Commands

It is possible to implement Direct Commands that directly read and write memory in an INSTEON device. Commands that write to memory can be destructive if not used with extreme caution. A better (object oriented) method is to define new Commands that read or write device properties by name, without regard to where the data is located in memory.

A legacy mechanism for peeking and poking single bytes using **SD** Commands is given in the **SD** Command table, highlighted in blue. These Commands are deprecated, meaning that they should not be implemented in the future. An explanation and examples of how these Commands have been used in the past are given below in the section *About INSTEON Peek and Poke Commands*162.

A more advanced mechanism for performing block data transfers using **ED** Commands with a *Command 1* field of 0x2A is given in the **ED** Command table, also highlighted in blue. These Commands, if implemented, should not be used directly because of their dependence on specific memory addresses. Instead, named data transfers may be defined using a modified *Request Block Data Transfer* Command with a *Command 2* byte within the range 0x0E to 0xFE. Manufacturers who wish to implement block data transfers should contact SmartLabs Technology for assistance.

Data transfer Commands are *not* required for INSTEON conformance. To determine if a given INSTEON device supports data transfer, try reading known data. If the device has implemented the Command, then it will return the expected data.

Broadcast Commands

By definition, INSTEON **SB** Broadcast Commands are addressed to all INSTEON devices. Accordingly, the *To Address* field may contain three bytes of data that pertain to the particular **SB** Broadcast Command. **SB** Broadcast Commands are not acknowledged.

Note that the **SB** Broadcast Commands described in this section are *not* the same as **SA** ALL-Link Broadcast Commands, which are first broadcast and then sent sequentially as **SC** ALL-Link Cleanup messages. See the sections *SB Messages* 47, *SA ALL-Link Broadcast Messages* 48, and *SC ALL-Link Cleanup Messages* 48 above for clarification.

Note that although **EB** Extended-length Broadcast Commands are logically possible, INSTEON does not currently use them.

Required Broadcast Commands

The *INSTEON Command Set Tables* 124 list only a few required Broadcast Commands.

Universally-Required Broadcast Commands

All INSTEON devices are required to implement a small number of Broadcast Commands, no matter what DevCat the device belongs to, in order to support ALL-Linking. See the section SET Button Pressed *Broadcast Messages* 84 above for more information. In the *INSTEON Command Set Tables* 124, universally-required Commands are listed in **bold type** and color-coded yellow.

Universally-required Broadcast Commands are reprinted in this document in the section *Required Commands for All INSTEON Devices* 157 below.

Conditionally-Required Broadcast Commands

Some Broadcast Commands are required only under certain conditions. For example, SALad-enabled products must support a *SALad Debug Report* Command. In the *INSTEON Command Set Tables* 124, conditionally-required Commands are also listed in **bold type** and color-coded yellow, except that the condition for requirement is given in red type.

Conditionally-required Broadcast Commands are reprinted in this document in the section *Required Commands for Some INSTEON Devices* 160 below.

INSTEON Command Set Tables

The following six tables show all of the INSTEON Commands defined as of the publication date of this Developer's Guide. Although reprinted here for convenience, the official table is contained in the *INSTEON Command Tables Document*[9] described in the *Other Documents Included by Reference*[9] section above. *INSTEON Command Tables 20070816a.doc* is the source for the tables reprinted below.

The tables in the following six sections contain:

- **SD**, Standard-length Direct Commands
- **ED**, Extended-length Direct Commands
- **SA**, Standard-length ALL-Link Commands
- **EA**, Extended-length ALL-Link Commands
- **SB**, Standard-length Broadcast Commands
- **EB**, Extended-length Broadcast Commands

The tables utilize Note Keys, text conventions, and color-codes to designate the following conditions:

Note Key	Text Sample	Description
Req-All	**Enter Linking Mode**	Required Commands for INSTEON conformance
Req-Ex	(Required after 2/1/07)	Required Commands with exceptions
Req-DC	Light ON	Required Commands for specific DevCats
-	Light ON Fast	Optional Commands
DataTr	Peek One Byte	Data Transfer Commands
FX	FX Commands	FX Commands
-	Reserved	Reserved for future use, currently unassigned
Dupl	**0x45**	Duplicated command number definitions for different DevCats
Prop	**0x2F**	Proposed command does not yet have final approval
NClar	Get Temperature	Needs further clarification
Depr	Deprecated	Deprecated command—do not use in the future

INSTEON Direct Commands

This section lists **SD** Standard-length and **ED** Extended-length INSTEON Direct Commands in two separate tables.

INSTEON Standard-length Direct Commands

The table below lists the existing INSTEON **SD** Standard-length Direct Commands.

The Note Key **Req-All** denotes INSTEON commands that must be supported by INSTEON devices in all Device Categories. **Req-All** command names appear in **bold type**.

The Note Key **Req-Ex** (…) denotes INSTEON commands that must be supported by INSTEON devices in all Device Categories except as noted within the parentheses. **Req-Ex** command names appear in **bold type**.

The Note Key **Req-DC** denotes INSTEON commands that must be supported only by those INSTEON devices in the Device Categories given in the **DevCat** and **SubCat** columns. **Req-DC** command names appear in underlined type.

SD Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
Reserved			**0x00**	**0x00**	Must be undefined in all INSTEON devices because this is the default command to execute using **ED 0x0304** *Set ALL-Link Command Alias*
Reserved			**0x00**	**0x01⇒ 0xFF**	
Assign to ALL-Link Group	**All**	**All**	**0x01**	**0x00 ⇒ 0xFF** Group Number	**Req-All** Used during INSTEON device linking session.
Delete from ALL-Link Group	**All**	**All**	**0x02**	**0x00 ⇒ 0xFF** Group Number	**Req-All** Used during unlinking session.
Product Data Request	**All**	**All**	**0x03**	**0x00**	**Req-All, Req-Ex** (Required after 2/1/07) Addressee responds with an **ED 0x0300** *Product Data Response* message
FX Username Request	**All**	**All**	**0x03**	**0x01**	**Req-Ex** (Only required for devices that support FX Commands) , **FX** Addressee responds with an **ED 0x0301** *FX Username Response* message
Device Text String Request	**All**	**All**	**0x03**	**0x02**	Addressee responds with an **ED 0x0302** *Device Text String Response* message
Reserved			**0x03**	**0x03 ⇒ 0xFF**	
Reserved			**0x04 ⇒ 0x08**		
Enter Linking Mode	**All**	**All**	**0x09**	**0x00 ⇒ 0xFF** Group Number	**Req-All** Same as holding down *SET Button* for 10 seconds **NOTE:** Not supported by i1 devices
Enter Unlinking Mode	**All**	**All**	**0x0A**	**0x00 ⇒ 0xFF** Group Number	**Req-All** **NOTE:** Not supported by i1 devices
Reserved			**0x0B ⇒ 0x0C**		

SD Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
Get INSTEON Engine Version	**All**	**All**	**0x0D**	**0x00**	**Req-All** Returned ACK message will contain the INSTEON Engine Version in Command 2. **0x00** = i1 (default echo for legacy devices) **0x01** = i2
Reserved			**0x0D**	**0x01** ⇒ **0xFF**	Do not use so that legacy devices will echo **0x00** in Command 2
Reserved			**0x0E**		
Ping	**All**	**All**	**0x0F**	**0x00** (**0x01** ⇒ **0xFF** Not Parsed in legacy devices. Use only 0x00 in the future.)	**Req-All** Addressee returns an ACK message but performs no operation.
ID Request	**All**	**All**	**0x10**	**0x00** (**0x01** ⇒ **0xFF** Not Parsed in legacy devices. Use only 0x00 in the future.)	**Req-All** Addressee first returns an ACK message, then it sends an **SB 0x01** *SET Button Pressed Responder* or **SB 0x02** *SET Button Pressed Controller* Broadcast message, but it does not enter Linking Mode.
Light ON	**0x01**	**All**	**0x11**	**0x00** ⇒ **0xFF** On-Level	**Req-DC** Go to *On-Level* at saved *Ramp Rate*
Light ON	**0x02**	**All**	**0x11**	**0x00** ⇒ **0xFF** Not Parsed	**Req-DC** Switch to full on
Light ON Fast	**0x01**	**All**	**0x12**	**0x00** ⇒ **0xFF** On-Level	Go to saved *On-Level* instantly
Light ON Fast	**0x02**	**All**	**0x12**	**0x00** ⇒ **0xFF** Not Parsed	Switch to full on
Light OFF	**0x01**	**All**	**0x13**	**0x00** ⇒ **0xFF** Not Parsed	**Req-DC** Go to full off at saved *Ramp Rate*
Light OFF	**0x02**	**All**	**0x13**	**0x00** ⇒ **0xFF** Not Parsed	**Req-DC** Switch to full off
Light OFF Fast	**0x01**	**All**	**0x14**	**0x00** ⇒ **0xFF** Not Parsed	Go to full off instantly
Light OFF Fast	**0x02**	**All**	**0x14**	**0x00** ⇒ **0xFF** Not Parsed	Switch to full off
Light Brighten One Step	**0x01**	**All**	**0x15**	**0x00** ⇒ **0xFF** Not Parsed	**Req-DC** Brighten one step. There are 32 steps from off to full brightness.
Light Dim One Step	**0x01**	**All**	**0x16**	**0x00** ⇒ **0xFF** Not Parsed	**Req-DC** Dim one step. There are 32 steps from off to full brightness.
Light Start Manual Change	**0x01**	**All**	**0x17**	Direction **0x00** Down **0x01** Up **0x02** ⇒ **0xFF** Unused	Begin changing *On-Level*.
Light Stop Manual Change	**0x01**	**All**	**0x18**	**0x00** ⇒ **0xFF** Not Parsed	Stop changing *On-Level*.
Light Status Request (SmartLabs 2486D KeypadLinc Dimmer, SmartLabs 2886D Icon In-Wall Controller)	**0x01**	**0x09** **0x0A**	**0x19**	**0x00**	Returned ACK message will contain the *On-Level* in Command 2. Command 1 will contain an *ALL-Link Database Delta* number that increments every time there is a change in the addressee's ALL-Link Database.
				0x01	Returned ACK message will contain the *LED Bit Flags* in Command 2. Command 1 will contain an *ALL-Link Database Delta* number that increments every time there is a change in the addressee's ALL-Link Database.

SD Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
Light Status Request	**0x01**	**All But 0x09 0x0A**	**0x19**	**0x00 ⇒ 0xFF** Not Parsed	Returned ACK message will contain the *On-Level* in Command 2. Command 1 will contain an *ALL-Link Database Delta* number that increments every time there is a change in the addressee's ALL-Link Database.
Light Status Request (SmartLabs 2486S KeypadLinc Relay)	**0x02**	**0x0F**	**0x19**	**0x00**	Returned ACK message will contain the *On-Level* (**0x00** or **0xFF** only) in Command 2. Command 1 will contain an *ALL-Link Database Delta* number that increments every time there is a change in the addressee's ALL-Link Database.
				0x01	Returned ACK message will contain the *LED Bit Flags* in Command 2. Command 1 will contain an *ALL-Link Database Delta* number that increments every time there is a change in the addressee's ALL-Link Database.
Light Status Request	**0x02**	**All But 0x0F**	**0x19**	**0x00 ⇒ 0xFF** Not Parsed	Returned ACK message will contain the *On-Level* (**0x00** or **0xFF** only) in Command 2. Command 1 will contain an *ALL-Link Database Delta* number that increments every time there is a change in the addressee's ALL-Link Database.
Reserved			**0x1A ⇒ 0x1E**		
Get Operating Flags (SmartLabs 2430 ControLinc and 2830 Icon Tabletop Controller)	**0x00**	**0x04 0x06**	**0x1F**	Flags Requested **0x00**: Bit 0: 0 = Program Lock Off; 1 = Program Lock On Bit 1: 0 = LED Off; 1 = LED On Bit 2: 0 = Beeper Off; 1 = Beeper On Bit 3-7 = Unused **0x01**: *ALL-Link Database Delta* number **0x02 ⇒ 0xFF**: Unused	Returned ACK message will contain the requested data in Command 2.
Get Operating Flags (SmartLabs 2843 RemoteLinc)	**0x00**	**0x05**	**0x1F**	Flags Requested **0x00**: Bit 0: 0 = Program Lock Off; 1 = Program Lock On Bit 1: 0 = LED Off; 1 = LED On Bit 2: 0 = Beeper Off; 1 = Beeper On Bit 3: 0 = Allow Sleep; 1 = Stay Awake Bit 4: 0 = Allow Transmit; 1 = Receive Only Bit 5: 0 = Allow Heartbeat; 1 = No Heartbeat	Returned ACK message will contain the requested data in Command 2.

SD Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2			Note Keys, Description
					Bit 6-7 = Unused		
				0x01	*ALL-Link Database Delta* number		
				0x02 ⇒ 0xFF	Unused		
Get Operating Flags (SmartLabs 2486D KeypadLinc Dimmer, SmartLabs 2886D Icon In-Wall Controller)	**0x01**	**0x09 0x0A**	**0x1F**	Flags Requested			Returned ACK message will contain the requested data in Command 2.
				0x00	Bit 0	0 = Program Lock Off	
						1 = Program Lock On	
					Bit 1	0 = LED Off	
						1 = LED On During Transmit	
					Bit 2	0 = Resume Dim Disabled	
						1 = Resume Dim Enabled	
					Bit 3	0 = 6 Keys	
						1 = 8 Keys	
					Bit 4	0 = Backlight Off	
						1 = Backlight On	
					Bit 5	0 = Key Beep Off	
						1 = Key Beep On	
					Bit 6-7 = Unused		
				0x01	*ALL-Link Database Delta* number		
				0x02 ⇒ 0xFF	Unused		
Get Operating Flags	**0x01**	**All But 0x09 0x0A**	**0x1F**	Flags Requested			Returned ACK message will contain the requested data in Command 2.
				0x00	Bit 0	0 = Program Lock Off	
						1 = Program Lock On	
					Bit 1	0 = LED Off	
						1 = LED On During Transmit	
					Bit 2	0 = Resume Dim Disabled	
						1 = Resume Dim Enabled	
					Bit 3 = Unused		
					Bit 4	0 = LED Off	
						1 = LED On	
					Bit 5	0 = Load Sense Off	
						1 = Load Sense On	
					Bit 6-7 = Unused		
				0x01	*ALL-Link Database Delta* number		
				0x02	Signal-to-Noise Value		
				0x03 ⇒ 0xFF	Unused		
Get Operating Flags	**0x02**	**0x0F**	**0x1F**	Flags Requested			Returned ACK message will contain the

SD Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**			**Note Keys, Description**
(SmartLabs 2486S KeypadLinc Relay)				**0x00**	Bit 0	0 = Program Lock Off	requested data in Command 2.
						1 = Program Lock On	
					Bit 1	0 = LED Off	
						1 = LED On During Transmit	
					Bit 2	0 = Resume Dim Disabled	
						1 = Resume Dim Enabled	
					Bit 3	0 = 6 Keys	
						1 = 8 Keys	
					Bit 4	0 = Backlight Off	
						1 = Backlight On	
					Bit 5	0 = Key Beep Off	
						1 = Key Beep On	
					Bit 6-7 = Unused		
				0x01	*ALL-Link Database Delta* number		
				0x02	Signal-to-Noise Value		
				0x03 ⇒ 0xFF	Unused		
Get Operating Flags	**0x02**	**All But 0x0F**	**0x1F**	Flags Requested			Returned ACK message will contain the requested data in Command 2.
				0x00	Bit 0	0 = Program Lock Off	
						1 = Program Lock On	
					Bit 1	0 = LED Off	
						1 = LED On During Transmit	
					Bit 2	0 = Resume Dim Disabled	
						1 = Resume Dim Enabled	
					Bit 3 = Unused		
					Bit 4	0 = LED Off	
						1 = LED On	
					Bit 5	0 = Load Sense Off	
						1 = Load Sense On	
					Bit 6-7 = Unused		
				0x01	*ALL-Link Database Delta* number		
				0x02 ⇒ 0xFF	Unused		
Set Operating Flags (SmartLabs 2430 ControLinc and 2830 Icon Tabletop Controller)	**0x00**	**0x04 0x06**	**0x20**	Flag to Alter			Defaults given in **bold**.
				0x00	Program Lock On		
				0x01	**Program Lock Off**		
				0x02	**LED On**		
				0x03	LED Off		
				0x04	Beeper On		
				0x05	**Beeper Off**		

SD Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**		**Note Keys, Description**
				0x06 ⇒ 0xFF	Unused	
Set Operating Flags (SmartLabs 2843 RemoteLinc)	**0x00**	**0x05**	**0x20**	Flag to Alter		Defaults given in **bold**.
				0x00	Program Lock On	
				0x01	**Program Lock Off**	
				0x02	**LED On**	
				0x03	LED Off	
				0x04	**Beeper On**	
				0x05	Beeper Off	
				0x06	Stay Awake On	
				0x07	**Stay Awake Off**	
				0x08	Listen Only On	
				0x09	**Listen Only Off**	
				0x0A	No I'm Alive On	
				0x0B	**No I'm Alive Off**	
				0x0C ⇒ 0xFF	Unused	
Set Operating Flags (SmartLabs 2486D KeypadLinc Dimmer, SmartLabs 2886D Icon In-Wall Controller)	**0x01**	**0x09** **0x0A**	**0x20**	Flag to Alter		Defaults given in **bold**.
				0x00	Program Lock On	
				0x01	**Program Lock Off**	
				0x02	LED On during TX	
				0x03	**LED Off during TX**	
				0x04	Resume Dim On	
				0x05	**Resume Dim Off**	
				0x06	8-Key KeypadLinc	
				0x07	**6-Key KeypadLinc**	
				0x08	LED Backlight Off	
				0x09	**LED Backlight On**	
				0x0A	Key Beep On	
				0x0B	**Key Beep Off**	
				0x0C ⇒ 0xFF	Unused	
Set Operating Flags	**0x01**	**All But 0x09 0x0A**	**0x20**	Flag to Alter		Defaults given in **bold**.
				0x00	Program Lock On	
				0x01	**Program Lock Off**	
				0x02	LED On during TX Default for SubCat **0x00** (SmartLabs LampLinc V2 Dimmer 2456D3)	
				0x03	LED Off during TX Default for SubCat **0x01** (SmartLabs SwitchLinc V2 Dimmer 2476D)	
				0x04	Resume Dim On	
				0x05	**Resume Dim Off**	
				0x06	**Load Sense On**	
				0x07	Load Sense Off	
				0x08	LED Off	
				0x09	**LED On**	
				0x0A ⇒ 0xFF	Unused	
Set Operating Flags	**0x02**	**0x0F**	**0x20**	Flag to Alter		Defaults given in **bold**.
				0x00	Program Lock On	

SD Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2		Note Keys, Description
(SmartLabs 2486S KeypadLinc Relay)				0x01	**Program Lock Off**	
				0x02	LED On during TX	
				0x03	**LED Off during TX**	
				0x04	Resume Dim On	
				0x05	**Resume Dim Off**	
				0x06	8-Key KeypadLinc	
				0x07	**6-Key KeypadLinc**	
				0x08	LED Backlight Off	
				0x09	**LED Backlight On**	
				0x0A	Key Beep On	
				0x0B	**Key Beep Off**	
				0x0C ⇒ 0xFF	Unused	
Set Operating Flags	0x02	All But 0x0F	0x20	Flag to Alter		Defaults given in **bold**.
				0x00	Program Lock On	
				0x01	**Program Lock Off**	
				0x02	LED On during TX Default for SubCat **0x09** (SmartLabs ApplianceLinc 2456S3)	
				0x03	LED Off during TX Default for SubCat **0x0A** (SmartLabs SwitchLinc Relay 2476S)	
				0x04	Resume Dim On	
				0x05	**Resume Dim Off**	
				0x06	**Load Sense On**	
				0x07	Load Sense Off	
				0x08	LED Off	
				0x09	**LED On**	
				0x0A ⇒ 0xFF	Unused	
Light Instant Change	0x01	All	0x21	**0x00** ⇒ **0xFF** On-Level		Set light to *On-Level* at next zero crossing. [Added 20060420]
Light Manually Turned Off	0x01	All	0x22	**0x00** ⇒ **0xFF** Not Parsed		Indicates manual load status change.
Light Manually Turned Off	0x02	All	0x22	**0x00** ⇒ **0xFF** Not Parsed		Indicates manual load status change.
Light Manually Turned On	0x01	All	0x23	**0x00** ⇒ **0xFF** Not Parsed		Indicates manual load status change.
Light Manually Turned On	0x02	All	0x23	**0x00** ⇒ **0xFF** Not Parsed		Indicates manual load status change.
Reread Init Values (SmartLabs 2486D KeypadLinc Dimmer, SmartLabs 2886D Icon In-Wall Controller)	0x01	0x09 0x0A	0x24	**0x00** ⇒ **0xFF** Not Parsed		**Depr** Deprecated (do not use in the future). For KeypadLinc only, reread initialization values from EEPROM, so that they will take effect after being poked.
Reread Init Values (SmartLabs 2486S KeypadLinc Relay)	0x02	0x0F	0x24	**0x00** ⇒ **0xFF** Not Parsed		**Depr** Deprecated (do not use in the future). For KeypadLinc only, reread initialization values from EEPROM, so that they will take effect after being poked.
Remote SET Button Tap	0x01	All	0x25	Number of Taps		Cause a device to respond as if its SET Button were tapped once or twice.
				0x00	Unused	
				0x01	1 Tap	

SD Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2		Note Keys, Description
				0x02	2 Taps	
				0x03 ⇒ 0xFF	Unused	
Reserved			0x26			
Light Set Status	0x01	N/A	0x27	**0x00 ⇒ 0xFF** On-Level		Updates SwitchLinc Companion's LEDs.
Set Address MSB	All	All	0x28	**0x00 ⇒ 0xFF** High byte of 16-bit address		**DataTr, Depr** Deprecated (do not use in the future). Set Most-significant Byte of EEPROM address for peek or poke.
Poke One Byte	All	All	0x29	**0x00 ⇒ 0xFF** Byte to write		**DataTr, Depr** Deprecated (do not use in the future). Poke Data byte into address previously loaded with *Set Address MSB* and *Peek* commands (*Peek One Byte* sets LSB).
Reserved	All	All	0x2A	**0x00 ⇒ 0xFF**		**DataTr, Depr** These are the Block Data Transfer commands in **ED** messages.
Peek One Byte	All	All	0x2B	**0x00 ⇒ 0xFF** LSB of address to peek or poke		**DataTr, Depr** Deprecated (do not use in the future). The returned ACK message will contain the peeked byte in Command 2. *Peek One Byte* is also used to set the LSB for *Poke One Byte*.
Peek One Byte Internal	All	All	0x2C	**0x00 ⇒ 0xFF** LSB of internal memory address to peek or poke		**DataTr, Depr** Deprecated (do not use in the future). Works like *Peek One Byte*, except only used to read from internal EEPROM of a Smarthome ControLinc V2.
Poke One Byte Internal	All	All	0x2D	**0x00 ⇒ 0xFF** Byte to write		**DataTr, Depr** Deprecated (do not use in the future). Works like *Poke One Byte*, except only used to write into internal EEPROM of a Smarthome ControLinc V2.
Light ON at Ramp Rate	0x01	All	0x2E	**0x00 ⇒ 0xFF** On-Level and Ramp Rate Combined		Bits 0-3 = 2 x *Ramp Rate* + 1 Bits 4-7 = *On-Level* + 0x0F
Reserved			0x2F ⇒ 0x3F			
Sprinkler Valve ON	0x04	All	0x40	**0x00 ⇒ 0xFF** Valve Number		
Sprinkler Valve OFF	0x04	All	0x41	**0x00 ⇒ 0xFF** Valve Number		
Sprinkler Program ON	0x04	All	0x42	**0x00 ⇒ 0xFF** Program Number		
Sprinkler Program OFF	0x04	All	0x43	**0x00 ⇒ 0xFF** Program Number		
Sprinkler Control	0x04	All	0x44	Subcommand		
				0x00	Load Initialization Values	
				0x01	Load EEPROM from RAM	Load RAM parameters from RAM EEPROM
				0x02	Get Valve Status	ACK contains 1-byte valve status in Command 2 0 = Off 1 = On
				0x03	Inhibit Command Acceptance	Stop accepting commands
				0x04	Resume Command Acceptance	Resume accepting commands
				0x05	Skip Forward	Turn off active valve and continue with next valve in program

SD Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2		Note Keys, Description
				0x06	Skip Back	Turn off active valve and continue with previous valve in program
				0x07	Enable Pump on V8	Enable pump control on V8
				0x08	Disable Pump on V8	Disable pump control on V8
				0x09	Broadcast ON	Enable **SB 0x27** *Device Status Changed* broadcast on valve status change
				0x0A	Broadcast OFF	Disable **SB 0x27** *Device Status Changed* broadcast on valve status change
				0x0B	Load RAM from EEPROM	Load RAM parameters from EEPROM
				0x0C	Sensor ON	Enable sensor reading
				0x0D	Sensor OFF	Disable sensor reading
				0x0E	Diagnostics ON	Put device in self-diagnostics
				0x0F	Diagnostics OFF	Take device out of self-diagnostics
				0x10 ⇒ 0xFF	Unused	
Flash LED (SmartLabs 2676D-B ICON Dimmer)	**0x01**	**0x13**	**0x45**	Subcommand		**Dupl**
				0x00	Cancel LED Flashing	
				0x01 ⇒ 0xFF	Begin LED Flashing	Device's LED flashes ½ second on, ½ second off, until canceled
Flash LED (SmartLabs 2676R-B ICON Relay)	**0x02**	**0x13**	**0x45**	Subcommand		**Dupl**
				0x00	Cancel LED Flashing	
				0x01 ⇒ 0xFF	Begin LED Flashing	Device's LED flashes ½ second on, ½ second off, until canceled
Sprinkler Get Program Request	**0x04**	**All**	**0x45**	**0x00** ⇒ **0xFF** Program Number		**Dupl** Added 5/05/06 Addressee responds with **ED 0x41xx** *Sprinkler Get Program Response*
I/O Output On	**0x07**	**All**	**0x45**	**0x00** ⇒ **0xFF** Output Number		**Dupl** Turns *Output Number* On
I/O Output Off	**0x07**	**All**	**0x46**	**0x00** ⇒ **0xFF** Output Number		Turns *Output Number* Off
I/O Alarm Data Request	**0x07**	**All**	**0x47**	**0x00**		Addressee responds with an **ED 0x4C00** *Alarm Data Response* message
Reserved			**0x47**	**0x01** ⇒ **0xFF**		
I/O Write Output Port	**0x07**	**All**	**0x48**	**0x00** ⇒ **0xFF** Value to store (only output bits are affected)		ACK contains byte written to Output Port in Command 2
I/O Read Input Port	**0x07**	**All**	**0x49**	**0x00**		ACK contains byte read from Input Port in Command 2
I/O Get Sensor Value	**0x07**	**All**	**0x4A**	**0x00** ⇒ **0xFF** Sensor number		ACK contains Sensor Value in Command 2
I/O Set Sensor 1 Nominal Value	**0x07**	**All**	**0x4B**	**0x00** ⇒ **0xFF** Nominal Value		Set Nominal Value for Sensor 1 to reach. Other sensors can be set with **ED 0x4Bxx** *Set Sensor Nominal*
I/O Get Sensor Alarm Delta	**0x07**	**All**	**0x4C**	**Bits 0-3** Sensor number **Bits 4-6** Delta from nominal **Bit 7** Delta Direction (+ if 0)		**Dupl** When added to or subtracted from midpoint, these are the values to trigger **SB 0x27** *Device Status Changed* alarm messages
Fan Status Report	**0x05**	**0x00** **0x02**	**0x4C**	Fan Capacity		**Dupl** Sent to controller when fan state changes.
				0x00 ⇒ 0x7F	Bits 0 - 6 = Fan Capacity in CFM Bit 7 = 0	

SD Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2		Note Keys, Description
				0x80 ⇒ **0xFF**	Bits 0 - 6 = Fan Capacity in CFM Bit 7 = 1, fan was turned off, Fan Capacity is removed from total airflow	
I/O Write Configuration Port	**0x07**	**All**	**0x4D**	**Bits 0-1** 00 Analog Input not used 01 Analog Input used, convert upon command 10 Analog Input used, convert at fixed interval 11 Unused		Modifies command responses
				Bit 2 If 1, send **SB 0x27** *Device Status Changed* broadcast on Sensor Alarm		
				Bit 3 If 1, send **SB 0x27** *Device Status Changed* broadcast on Input Port change		
				Bit 4 If 1, Enable 1-Wire port (Sensors 1-8)		
				Bit 5 If 1, Enable ALL-Link aliasing to default set		
				Bit 6 If 1, send **SB 0x27** *Device Status Changed* broadcast on Output Port change		
				Bit 7 If 1, Enable Output Timers		
I/O Read Configuration Port	**0x07**	**All**	**0x4E**	**0x00**		ACK contains byte read from Configuration Port in Command 2. See **SD 0x4Dxx** *Write Configuration Port* above for port bit definitions.
I/O Module Control	**0x07**	**All**	**0x4F**	Subcommand		
				0x00	Load Initialization Values	Reset to factory default settings
				0x01	Load EEPROM from RAM	Makes permanent any changes to settings such as those made to parameters with a Poke command
				0x02	Status Request	ACK contains state of outputs in Command 2
				0x03	Read Analog Once	Starts the A/D conversion once
				0x04	Read Analog Always	Starts the A/D conversion at preset intervals
				0x05 ⇒ **0x08** Unused		
				0x09	Enable Status Change message	Enables **SB 0x27** *Device Status Changed* broadcast message each time the Input Port status changes
				0x0A	Disable Status Change message	Disables **SB 0x27** *Device Status Changed* broadcast message each time the Input Port status changes
				0x0B	Load RAM from EEPROM	Moves parameters from EEPROM into RAM
				0x0C	Sensor On	Enable sensor reading
				0x0D	Sensor Off	Disable sensor reading
				0x0E	Diagnostics On	Put device in self-diagnostics mode
				0x0F	Diagnostics Off	Take device out of self-diagnostics mode
				0x10 ⇒ **0xFF** Unused		

SD Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**		**Note Keys, Description**
Pool Device ON	**0x06**	**All**	**0x50**	**0x00 ⇒ 0xFF** Device Number		0 = Unused 1 = Pool 2 = Spa 3 = Heat 4 = Pump 5 - 255 Aux
Pool Device OFF	**0x06**	**All**	**0x51**	**0x00 ⇒ 0xFF** Device Number		0 = All OFF 1 = Pool 2 = Spa 3 = Heat 4 = Pump 5 - 255 Aux
Pool Temperature Up	**0x06**	**All**	**0x52**	**0x00 ⇒ 0xFF** Temperature Change		Increase current temperature setting by *Temperature Change* x 0.5
Pool Temperature Down	**0x06**	**All**	**0x53**	**0x00 ⇒ 0xFF** Temperature Change		Decrease current temperature setting by *Temperature Change* x 0.5
Pool Control	**0x06**	**All**	**0x54**	Subcommand		
				0x00	Load Initialization Values	
				0x01	Load EEPROM from RAM	
				0x02	Get Pool Mode	ACK contains 1-byte thermostat mode in Command 2 0 = Pool 1 = Spa 2 - 255 Unused
				0x03	Get Ambient Temperature	**NClar** ACK contains ambient temperature in Command 2
				0x04	Get Water Temperature	**NClar** ACK contains water temperature in Command 2
				0x05	Get pH	ACK contains pH value in Command 2
				0x06 ⇒ 0xFF	Unused	
Reserved			**0x55 ⇒ 0x57**			
Door Move	**0x0F**	**All**	**0x58**	Subcommand		
				0x00	Raise Door	
				0x01	Lower Door	
				0x02	Open Door	
				0x03	Close Door	
				0x04	Stop Door	
				0x05	Single Door Open	
				0x06	Single Door Close	
				0x07 ⇒ 0xFF	Unused	
Door Status Report	**0x0F**	**All**	**0x59**	Subcommand		
				0x00	Raise Door	
				0x01	Lower Door	
				0x02	Open Door	
				0x03	Close Door	
				0x04	Stop Door	
				0x05	Single Door Open	
				0x06	Single Door Close	

SD Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2		Note Keys, Description
				0x07 ⇒ 0xFF	Unused	
Reserved			**0x5A ⇒ 0x5F**			
Window Covering	**0x0E**	**All**	**0x60**	Subcommand		
				0x00	Open	
				0x01	Close	
				0x02	Stop	
				0x03	Program	
				0x04 ⇒ 0xFF	Unused	
Window Covering Position	**0x0E**	**All**	**0x61**	**0x00** ⇒ **0xFF** Position		0x00 is closed, 0xFF is open.
Reserved			**0x62 ⇒ 0x67**			
Thermostat Temperature Up	**0x05**	**All**	**0x68**	**0x00** ⇒ **0xFF** Temperature Change x 2 (unsigned byte)		Increase current temperature setting by *Temperature Change* x 0.5
Thermostat Temperature Down	**0x05**	**All**	**0x69**	**0x00** ⇒ **0xFF** Temperature Change x 2 (unsigned byte)		Decrease current temperature setting by *Temperature Change* x 0.5
Thermostat Get Zone Information	**0x05**	**All**	**0x6A**	**Bits 0-4** Zone Number 0-31		ACK contains Zone Temperature, Setpoint, Deadband, or Humidity as an unsigned byte in Command 2
				Bits 5,6	00 = Temperature	
					01 = Setpoint	
					10 = Deadband	
					11 = Humidity	
				Bit 7 Unused		
Thermostat Control	**0x05**	**All**	**0x6B**	Subcommand		
				0x00	Load Initialization Values	
				0x01	Load EEPROM from RAM	
				0x02	Get Thermostat Mode	ACK contains 1-byte thermostat mode in Command 2 0x00 = Off 0x01 = Heat 0x02 = Cool 0x03 = Auto 0x04 = Fan 0x05 = Program 0x06 = Program Heat 0x07 = Program Cool 0x08 ⇒ 0xFF Unused
				0x03	Get Ambient Temperature	**NClar** ACK contains ambient temperature in Command 2
				0x04	ON Heat	Set mode to Heat
				0x05	ON Cool	Set mode to Cool
				0x06	ON Auto	Set mode to Auto
				0x07	ON Fan	Turn fan on
				0x08	OFF Fan	Turn fan off
				0x09	OFF All	Turn everything off
				0x0A	Program Heat	Set mode to Program Heat
				0x0B	Program Cool	Set mode to Program Cool
				0x0C	Program Auto	Set mode to Program Auto

SD Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2		Note Keys, Description
				0x0D	Get Equipment State	Bit 0 = Cool active Bit 1 = Heat active Bit 2 = Programmable output available Bit 3 = Programmable output state Bits 4-7 Unused
				0x0E	Set Equipment State	Bit 0 = Programmable output state Bits 1-7 Unused
				0x0F	Get Temperature Units	ACK contains Units in Command 2 0x00 = Fahrenheit 0x01 = Celsius 0x02 ⇒ 0xFF Unused
				0x10	Set Fahrenheit	Set Temperature Units to Fahrenheit
				0x11	Set Celsius	Set Temperature Units to Celsius
				0x12	Get Fan-On Speed	ACK contains speed fan will run at when turned on, in Command 2 0x00 = Single-speed Fan 0x01 = Low Speed 0x02 = Medium Speed 0x03 = High Speed 0x04 ⇒ 0xFF Unused
				0x13	Set Fan-On Speed Low	Fan will run at low speed when on (ignored by single-speed fans)
				0x14	Set Fan-On Speed Medium	Fan will run at medium speed when on (Ignored by single-speed fans)
				0x15	Set Fan-On Speed High	Fan will run at high speed when on (Ignored by single-speed fans)
				0x16	Enable Status Change message	Enables **SB 0x27** *Device Status Changed* broadcast message each time the Thermostat Mode status changes
				0x17	Disable Status Change message	Disables **SB 0x27** *Device Status Changed* broadcast message each time the Thermostat Mode status changes
				0x18 ⇒ 0xFF	Unused	
Thermostat Set Cool Setpoint	**0x05**	**All**	**0x6C**	**0x00** ⇒ **0xFF** Temperature Setpoint x 2 (unsigned byte)		Set current cool temperature setpoint to *Temperature Setpoint* x 0.5
Thermostat Set Heat Setpoint	**0x05**	**All**	**0x6D**	**0x00** ⇒ **0xFF** Temperature Setpoint x 2 (unsigned byte)		Set current heat temperature setpoint to *Temperature Setpoint* x 0.5
Reserved			**0x6E ⇒ 0x6F**			
Leak Detector Announce	**0x09**	**All**	**0X70**	**0x00** Leak Detected		
				0x01 No Leak Detected		
				0x02 Battery Low		
				0x03 Battery OK		
Reserved			**0x70**	**0x04** ⇒ **0xFF**		
Reserved			**0x71 ⇒ 0x80**			
Assign to Companion Group	**0x01**	**0x01** **0x04**	**0x81**	**0x00** ⇒ **0xFF** Not Parsed		Deprecated (do not use in the future). For SwitchLinc only, allows Slaves of a Master to follow the Master when the Master is controlled by a companion device.
Reserved			**0x82 ⇒ 0xEF**			

SD Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description
FX Commands	**All**	**All**	**0xF0 ⇒ 0xFF**	User-specific	**FX** These commands only function if *FX Usernames* in a Controller and Responder device match during linking.

INSTEON Extended-length Direct Commands

The table below lists the existing INSTEON **ED** Extended-length Direct Commands.

The Note Key **Req-All** denotes INSTEON commands that must be supported by INSTEON devices in all Device Categories. **Req-All** command names appear in **bold type**.

The Note Key **Req-Ex** (…) denotes INSTEON commands that must be supported by INSTEON devices in all Device Categories except as noted within the parentheses. **Req-Ex** command names appear in **bold type**.

ED Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
Reserved			**0x00**	**0x00**	Must be undefined in all INSTEON devices because this is the default Command to execute using **ED 0x0304** *Set ALL-Link Command Alias*
Reserved			**0x00**	**0x01⇒ 0xFF**	
Reserved			**0x01 ⇒ 0x02**		
Product Data Response [Response to **SD 0x0300** *Product Data Request*]	**All**	**All**	**0x03**	**0x00**	**Req-All, Req-Ex** (Required after 2/1/07)
					D1 **0x00** Reserved (always set to 0x00)
					D2 **0x00 ⇒ 0xFF** INSTEON Product Key MSB
					D3 **0x00 ⇒ 0xFF** INSTEON Product Key 2MSB
					D4 **0x00 ⇒ 0xFF** INSTEON Product Key LSB
					D5 **0x00 ⇒ 0xFF** Device Category (DevCat)
					D6 **0x00 ⇒ 0xFF** Device Subcategory (SubCat)
					D7 **0xFF** Reserved (always set to 0xFF) (Matches byte in LSB of *To Address* of **SB 0x01** *SET Button Pressed Responder* or **SB 0x02** *SET Button Pressed Controller* commands)
					D8 **0xFF** Reserved (always set to 0xFF) (Matches byte in Command 2 of **SB 0x01** *SET Button Pressed Responder* or **SB 0x02** *SET Button Pressed Controller* commands)
					D9 ⇒ D14 User-defined
FX Username Response [Response to **SD 0x0301** *FX Username Request*]	**All**	**All**	**0x03**	**0x01**	**Req-Ex** (Only required for devices that support FX Commands), **FX**
					D1 ⇒ D8 **0x00 ⇒ 0xFF** FX Command Username Used for *FX Commands*, which are user-specific **SD** or **ED** commands numbered **0xFF00 ⇒ 0xFFFF**
					D9 ⇒ D14 User-defined

ED Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description
Device Text String Response [Response to **SD 0x0302** *Device Text String Request*]	**All**	**All**	**0x03**	**0x02**	D1 ⇒ D14 **ASCII** Text string describing device Null (0x00) terminated unless 14 bytes long
Set Device Text String	**All**	**All**	**0x03**	**0x03**	D1 ⇒ D14 **ASCII** Text string describing device Null (0x00) terminated unless 14 bytes long
Set ALL-Link Command Alias	**All**	**All**	**0x03**	**0x04**	D1 **0x11** ⇒ **0xFF** ALL-Link Command Number to replace with **SD** or **ED** Direct Command in D2, D3.
					D2, D3 **0x0000** ⇒ **0xFFFF** **SD** or **ED** Direct Command to execute in place of ALL-Link Command in D1. Set to **0x0000** to ignore ALL-Link Command.
					D4 **0x00**, **0x01** Flag
					0x00 Direct Command is **SD** (Standard-length).
					0x01 Direct Command is **ED** Extended-length), **ED 0x0305** *Set ALL-Link Command Alias Extended Data* message follows.
					D5 ⇒ D14 Unused
Set ALL-Link Command Alias Extended Data	**All**	**All**	**0x03**	**0x05**	D1 ⇒ D14 **0x00** ⇒ **0xFF** Data field of **ED** Command to execute in place of ALL-Link Command in D1 of previous **ED 0x0304** *Set ALL-Link Command Alias* message.
Reserved			**0x03**	**0x06** ⇒ **0xFF**	
Reserved			**0x04** ⇒ **0x29**		
Block Data Transfer	**All**	**All**	**0x2A**	**0x00** Transfer Failure	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D14 Unused
				0x01 Transfer Complete, 1 byte in this last message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 Final 1 byte
					D4 ⇒ D14 Unused
				0x02 Transfer Complete, 2 bytes in this last message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D4 Final 2 bytes
					D5 ⇒ D14 Unused
				0x03 Transfer Complete, 3 bytes in this last message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D5 Final 3 bytes
					D6 ⇒ D14 Unused
				0x04 Transfer Complete, 4 bytes in this last message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D6 Final 4 bytes

ED Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description
					D7 ⇒ D14 Unused
				0x05 Transfer Complete, 5 bytes in this last message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D7 Final 5 bytes
					D8 ⇒ D14 Unused
				0x06 Transfer Complete, 6 bytes in this last message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D8 Final 6 bytes
					D9 ⇒ D14 Unused
				0x07 Transfer Complete, 7 bytes in this last message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D9 Final 7 bytes
					D10 ⇒ D14 Unused
				0x08 Transfer Complete, 8 bytes in this last message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D10 Final 8 bytes
					D11 ⇒ D14 Unused
				0x09 Transfer Complete, 9 bytes in this last message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D11 Final 9 bytes
					D12 ⇒ D14 Unused
				0x0A Transfer Complete, 10 bytes in this last message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D12 Final 10 bytes
					D13 ⇒ D14 Unused
				0x0B Transfer Complete, 11 bytes in this last message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D13 Final 11 bytes
					D13 Unused
				0x0C Transfer Complete, 12 bytes in this last message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D14 Final 12 bytes
				0x0D Transfer Continues, 12 bytes in this message	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 ⇒ D14 12 bytes
				0x0E ⇒ **0xFE** Reserved	
				0xFF Request Block Data Transfer	**DataTr**
					D1 **0x00** ⇒ **0xFF** Source address MSB
					D2 **0x00** ⇒ **0xFF** Source address LSB
					D3 **0x00** ⇒ **0xFF** Destination addr MSB
					D4 **0x00** ⇒ **0xFF** Destination addr LSB
					D5 **0x00** ⇒ **0xFF** Block length MSB
					D6 **0x00** ⇒ **0xFF** Block length LSB

ED Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**	
					D7 **0x00** ⇒ **0xFF** Destination ID MSB	
					D8 **0x00** ⇒ **0xFF** Destination ID 2MSB	
					D9 **0x00** ⇒ **0xFF** Destination ID LSB	
					D10 ⇒ D14 Unused	
Reserved			**0x2B** ⇒ **0x2D**			
Extended Set/Get (SmartLabs 2430 ControLinc and 2830 Icon Tabletop Controller)	**0x00**	**0x04** **0x06**	**0x2E**	**0x00**	D1 **0x00** ⇒ **0xFF** Button/Group Number	
					D2 **0x00** Data Request [Addressee responds with *Data Response*]	D3 ⇒ D14 Unused
					D2 **0x01** Data Response [Response to *Data Request*]	D3 **0x00** ⇒ **0x0F** X10 House Code #1 (**0x20** = none)
						D4 **0x00** ⇒ **0x0F** X10 Unit Code #1
						D5 **0x00** ⇒ **0x0F** X10 House Code #2 (**0x20** = none)
						D6 **0x00** ⇒ **0x0F** X10 Unit Code #2
						D7 **0x00** ⇒ **0x0F** X10 House Code #3 (**0x20** = none)
						D8 **0x00** ⇒ **0x0F** X10 Unit Code #3
						D9 **0x00** ⇒ **0x0F** X10 House Code #4 (**0x20** = none)
						D10 **0x00** ⇒ **0x0F** X10 Unit Code #4
						D11 **0x00** ⇒ **0x0F** X10 House Code #5 (**0x20** = none)
						D12 **0x00** ⇒ **0x0F** X10 Unit Code #5
						D13 ⇒ D14 Unused
					D2 **0x02** ⇒ **0x03** Unused	
					D2 **0x04** Set X10 Address	D3 **0x00** ⇒ **0x0F** X10 House Code (**0x20** = none)
						D4 **0x00** ⇒ **0x0F** X10 Unit Code
						D5 ⇒ D14 Unused
					D2 **0x05** ⇒ **0xFF** Unused	
Extended Set/Get (SmartLabs 2843 RemoteLinc)	**0x00**	**0x05**	**0x2E**	**0x00**	D1 **0x00** ⇒ **0xFF** Button/Group Number	
					D2 **0x00** Data Request [Addressee responds with *Data Response*]	D3 ⇒ D14 Unused
					D2 **0x01** Data Response [Response to *Data*	D3 **0x00** ⇒ **0xFF** Awake Time Upon Heartbeat, seconds

ED Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description	
					Request]	D4 **0x00** ⇒ **0xFF** Heartbeat Interval X 755.2 seconds (12.5 minutes)
						D5 **0x00** ⇒ **0xFF** Number of **SB 0x04** *Heartbeat* messages to send upon Heartbeat
						D6 **0x00** ⇒ **0xFF** Button Trigger-ALL-Link Bitmap If bit = 0, associated button sends normal Command If bit = 0, associated button sends **ED 0x30** *Trigger ALL-Link* Command to first device in ALDB
						D7 ⇒ D14 Unused
					D2 **0x02** Set Awake Time Upon Heartbeat	D3 **0x00** ⇒ **0xFF** Awake Time Upon Heartbeat, seconds
						D4 ⇒ D14 Unused
					D2 **0x03** Set Heartbeat Interval	D3 **0x00** ⇒ **0xFF** Heartbeat Interval X 755.2 seconds (12.5 minutes)
						D4 ⇒ D14 Unused
					D2 **0x04** Set Number of **SB 0x04** *Heartbeat* messages to send upon Heartbeat	D3 **0x00** ⇒ **0xFF** Number of **SB 0x04** *Heartbeat* messages to send upon Heartbeat
						D4 ⇒ D14 Unused
					D2 **0x05** Set Trigger-ALL-Link State for Button	D3 **0x00** ⇒ **0x01** 0 = Button sends normal Command 1 = Button sends **ED 0x30** *Trigger ALL-Link* Command to first device in ALDB
						D4 ⇒ D14 Unused
					D2 **0x06** ⇒ **0xFF** Unused	
Extended Set/Get (SmartLabs 2486D KeypadLinc Dimmer, SmartLabs 2886D Icon In-Wall Controller)	**0x01**	**0x09** **0x0A**	**0x2E**	**0x00**	D1 **0x00** ⇒ **0xFF** Button/Group Number	
					D2 **0x00** Data Request [Addressee responds with *Data Response*]	D3 ⇒ D14 Unused
					D2 **0x01** Data Response [Response to *Data Request*]	D3 **0x00** ⇒ **0xFF** Button's LED-Follow Mask
						D4 **0x00** ⇒ **0xFF** Button's LED-Off Mask
						D5 **0x00** ⇒ **0xFF** Button's X10 House Code

ED Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description	
						D6 **0x00** ⇒ **0xFF** Button's X10 Unit Code
						D7 **0x00** ⇒ **0x1F** Button's *Ramp Rate*
						D8 **0x00** ⇒ **0xFF** Button's *On- Level*
						D9 **0x11** ⇒ **0x7F** Global LED Brightness
						D10 **0x00** ⇒ **0xFF** Non-toggle Bitmap If bit = 0, associated button is Toggle If bit = 1, associated button is Non-toggle
						D11 **0x00** ⇒ **0xFF** Button-LED State Bitmap If bit = 0, associated button's LED is Off If bit = 1, associated button's LED is On
						D12 **0x00** ⇒ **0xFF** X10-All Bitmap If bit = 0, associated button sends X10 On/Off If bit = 1, associated button sends X10 All-On/All-Off
						D13 **0x00** ⇒ **0xFF** Button Non-toggle On/Off Bitmap If bit = 0, associated button, if Non-toggle, sends Off If bit = 0, associated button, if Non-toggle, sends On
						D14 **0x00** ⇒ **0xFF** Button Trigger-ALL-Link Bitmap If bit = 0, associated button sends normal Command If bit = 0, associated button sends **ED 0x30** *Trigger ALL-Link* Command to first device in ALDB
					D2 **0x02** Set LED-Follow Mask for Button	D3 **0x00** ⇒ **0xFF** If bit = 0, associated button's LED is not affected If bit = 1, associated button's LED follows this button's LED
						D4 ⇒ D14 Unused

ED Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description	
					D2 **0x03** Set LED-Off Mask for Button	D3 **0x00** ⇒ **0xFF** If bit = 0, associated button' LED is not affected If bit = 1, associated button's LED turns off when this button is pushed
						D4 ⇒ D14 Unused
					D2 **0x04** Set X10 Address for Button	D3 **0x00** ⇒ **0xFF** X10 House Code
						D4 **0x00** ⇒ **0xFF** X10 Unit Code
						D5 ⇒ D14 Unused
					D2 **0x05** Set Ramp Rate for Button	D3 **0x00** ⇒ **0x1F** *Ramp Rate* (0.1 second to 9 minutes)
						D4 ⇒ D14 Unused
					D2 **0x06** Set On-Level for Button	D3 **0x00** ⇒ **0xFF** *On-Level*
						D4 ⇒ D14 Unused
					D2 **0x07** Set Global LED Brightness (ignores D1)	D3 **0x11** ⇒ **0x7F** Brightness for all LEDs when on
						D4 ⇒ D14 Unused
					D2 **0x08** Set Non-toggle State for Button	D3 **0x00** ⇒ **0x01** 0 = Button is Toggle 1 = Button is Non-toggle
						D4 ⇒ D14 Unused
					D2 **0x09** Set LED State for Button	D3 **0x00** ⇒ **0x01** 0 = Turn button's LED Off 1 = Turn button's LED On
						D4 ⇒ D14 Unused
					D2 **0x0A** Set X10 All-On State for Button	D3 **0x00** ⇒ **0x01** 0 = Button sends X10 On/Off 1 = Button sends X10 All-On/All-Off
						D4 ⇒ D14 Unused
					D2 **0x0B** Set Non-toggle On/Off State for Button	D3 **0x00** ⇒ **0x01** 0 = If Non-toggle, Button sends Off Command 1 = If Non-toggle, Button sends On Command
						D4 ⇒ D14 Unused
					D2 **0x0C** Set Trigger-ALL-Link State for Button	D3 **0x00** ⇒ **0x01** 0 = Button sends normal Command 1 = Button sends **ED 0x30** *Trigger ALL-Link* Command to first device in ALDB
						D4 ⇒ D14 Unused
					D2 **0x0D** ⇒ **0xFF** Unused	
Extended Set/Get	**0x01**	**All**	**0x2E**	**0x00**	D1 **0x00** ⇒ **0xFF** Button/Group Number	

ED Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description	
		But 0x09 0x0A			D2 **0x00** Data Request [Addressee responds with *Data Response*]	D3 ⇒ D14 Unused
					D2 **0x01** Data Response [Response to *Data Request*]	D3 Unused
						D4 Unused
						D5 **0x00** ⇒ **0x0F** X10 House Code (**0x20** = none)
						D6 **0x00** ⇒ **0x0F** X10 Unit Code
						D7 **0x00** ⇒ **0x1F** *Ramp Rate*
						D8 **0x00** ⇒ **0xFF** *On-Level*
						D9 **0x00** ⇒ **0xFF** *Signal-to-Noise Threshold*
						D10 ⇒ D14 Unused
					D2 **0x02** ⇒ **0x03** Unused	
					D2 **0x04** Set X10 Address	D3 **0x00** ⇒ **0x0F** X10 House Code (**0x20** = none)
						D4 **0x00** ⇒ **0x0F** X10 Unit Code
						D5 ⇒ D14 Unused
					D2 **0x05** Set Ramp Rate	D3 **0x00** ⇒ **0x1F** *Ramp Rate* (0.1 second to 9 minutes)
						D4 ⇒ D14 Unused
					D2 **0x06** Set On-Level	D3 **0x00** ⇒ **0xFF** *On-Level*
						D4 ⇒ D14 Unused
					D2 **0x07** ⇒ **0xFF** Unused	
Extended Set/Get (SmartLabs 2486S KeypadLinc Relay)	**0x02**	**0x0F**	**0x2E**	**0x00**	D1 **0x00** ⇒ **0xFF** Button/Group Number	
					D2 **0x00** Data Request [Addressee responds with *Data Response*]	D3 ⇒ D14 Unused
					D2 **0x01** Data Response [Response to *Data Request*]	D3 **0x00** ⇒ **0xFF** Button's LED-Follow Mask
						D4 **0x00** ⇒ **0xFF** Button's LED-Off Mask
						D5 **0x00** ⇒ **0xFF** Button's X10 House Code
						D6 **0x00** ⇒ **0xFF** Button's X10 Unit Code
						D7 **0x00** ⇒ **0x1F** Button's *Ramp Rate* (ignore for relay)
						D8 **0x00** ⇒ **0xFF** Button's *On- Level*

ED Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description	
						D9 **0x11** ⇒ **0x7F** Global LED Brightness
						D10 **0x00** ⇒ **0xFF** Non-toggle Bitmap If bit = 0, associated button is Toggle If bit = 1, associated button is Non-toggle
						D11 **0x00** ⇒ **0xFF** Button-LED State Bitmap If bit = 0, associated button's LED is Off If bit = 1, associated button's LED is On
						D12 **0x00** ⇒ **0xFF** X10-All Bitmap If bit = 0, associated button sends X10 On/Off If bit = 1, associated button sends X10 All-On/All-Off
						D13 **0x00** ⇒ **0xFF** Button Non-toggle On/Off Bitmap If bit = 0, associated button, if Non-toggle, sends Off If bit = 0, associated button, if Non-toggle, sends On
						D14 **0x00** ⇒ **0xFF** Button Trigger-ALL-Link Bitmap If bit = 0, associated button sends normal Command If bit = 0, associated button sends **ED 0x30** *Trigger ALL-Link* Command to first device in ALDB
					D2 **0x02** Set LED-Follow Mask for Button	D3 **0x00** ⇒ **0xFF** If bit = 0, associated button's LED is not affected If bit = 1, associated button's LED follows this button's LED
						D4 ⇒ D14 Unused
					D2 **0x03** Set LED-Off Mask for Button	D3 **0x00** ⇒ **0xFF** If bit = 0, associated button' LED is not affected If bit = 1, associated button's LED turns off when this button is pushed
						D4 ⇒ D14 Unused
					D2 **0x04** Set X10 Address for Button	D3 **0x00** ⇒ **0xFF** X10 House Code

ED Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description	
						D4 **0x00** ⇒ **0xFF** X10 Unit Code
						D5 ⇒ D14 Unused
					D2 **0x05** Set Ramp Rate for Button	D3 **0x00** ⇒ **0x1F** *Ramp Rate* (0.1 second to 9 minutes)
						D4 ⇒ D14 Unused
					D2 **0x06** Set On-Level for Button	D3 **0x00** ⇒ **0xFF** *On-Level*
						D4 ⇒ D14 Unused
					D2 **0x07** Set Global LED Brightness (ignores D1)	D3 **0x11** ⇒ **0x7F** Brightness for all LEDs when on
						D4 ⇒ D14 Unused
					D2 **0x08** Set Non-toggle State for Button	D3 **0x00** ⇒ **0x01** 0 = Button is Toggle 1 = Button is Non-toggle
						D4 ⇒ D14 Unused
					D2 **0x09** Set LED State for Button	D3 **0x00** ⇒ **0x01** 0 = Turn button's LED Off 1 = Turn button's LED On
						D4 ⇒ D14 Unused
					D2 **0x0A** Set X10 All-On State for Button	D3 **0x00** ⇒ **0x01** 0 = Button sends X10 On/Off 1 = Button sends X10 All-On/All-Off
						D4 ⇒ D14 Unused
					D2 **0x0B** Set Non-toggle On/Off State for Button	D3 **0x00** ⇒ **0x01** 0 = If Non-toggle, Button sends Off Command 1 = If Non-toggle, Button sends On Command
						D4 ⇒ D14 Unused
					D2 **0x0C** Set Trigger-ALL-Link State for Button	D3 **0x00** ⇒ **0x01** 0 = Button sends normal Command 1 = Button sends **ED 0x30** *Trigger ALL-Link* Command to first device in ALDB
						D4 ⇒ D14 Unused
					D2 **0x0D** ⇒ **0xFF** Unused	
Extended Set/Get	**0x02**	**All But 0x0F**	**0x2E**	**0x00**	D1 **0x00** ⇒ **0xFF** Button/Group Number	
					D2 **0x00** Data Request [Addressee responds with *Data Response*]	D3 ⇒ D14 Unused
					D2 **0x01** Data Response [Response to *Data Request*]	D3 Unused
						D4 Unused
						D5 **0x00** ⇒ **0x0F** X10 House Code (**0x20** = none)

ED Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description	
						D6 **0x00** ⇒ **0x0F** X10 Unit Code
						D7 ⇒ D14 Unused
					D2 **0x02** ⇒ **0x03** Unused	
					D2 **0x04** Set X10 Address	D3 **0x00** ⇒ **0x0F** X10 House Code (**0x20** = none)
						D4 **0x00** ⇒ **0x0F** X10 Unit Code
						D5 ⇒ D14 Unused
					D2 **0x05** ⇒ **0xFF** Unused	
Reserved			**0x2E**	**0x01** ⇒ **0xFF**		
Read/Write ALL-Link Database (ALDB)	**All**	**All**	**0x2F**	**0x00**	**Req-All, Req-Ex, DataTr** (Required for all i2 devices) Not implemented in i1 devices	
					D1 Unused	
					D2 **0x00** ALDB Record Request [Addressee responds with ALDB Record Response(s)] **NOTE:** Set address to 0x0000 to start at first record in ALDB. (Actual memory address is 0x0FFF in SmartLabs devices.)	D3 **0x00** ⇒ **0xFF** Address High Byte
						D4 **0x00** ⇒ **0xFF** Address Low Byte
						D5 **0x00** Dump all records D5 **0x01** ⇒ **0xFF** Dump one record
						D6 ⇒ D14 Unused
					D2 **0x01** ALDB Record Response [Response to ALDB Record Request] If D5 of ALDB Record Request was 0x00, return one record, else return all records until end of ALDB is reached. (Flag Byte in last record will be 0x00). Address is automatically decremented by 8 for each record returned.	D3 **0x00** ⇒ **0xFF** Address High Byte
						D4 **0x00** ⇒ **0xFF** Address Low Byte
						D5 Unused
						D6 ⇒ D13 **0x00** ⇒ **0xFF** Returned 8-byte Record
						D14 Unused
					D2 **0x02** Write ALDB Record	D3 **0x00** ⇒ **0xFF** Address High Byte
						D4 **0x00** ⇒ **0xFF** Address Low Byte
						D5 **0x01** ⇒ **0x08** Number of Bytes (**0x09** ⇒ **0xFF** is the same as **0x08**)

ED **Commands**	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
					D6 ⇒ D13 **0x00** ⇒ **0xFF** 8-byte Record to Write D14 Unused D2 **0x03** ⇒ **0xFF** Unused
Reserved			**0x2F**	**0x01** ⇒ **0xFF**	
Trigger ALL-Link Command (SmartLabs 2476D SwitchLinc i2 Dimmer 600 W, 2476DH SwitchLinc i2 Dimmer 1000 W, 2486D KeypadLinc Dimmer, 2886D Icon In-Wall Controller)	**0x01**	**0x01** **0x04** **0x09** **0x0A**	**0x30**	**0x00**	D1 **0x00** ⇒ **0xFF** Button/Group Number D2 On-Level Switch **0x00** Use On-Level stored in ALDB **0x01** Use On-Level in D3 **0x02** ⇒ **0xFF** Unused D3 **0x00** ⇒ **0xFF** On-Level if D2 = 0x01 D4 **0x00** ⇒ **0xFF** **SA** Command 1 to send D5 **0x00** ⇒ **0xFF** **SA** Command 2 to send D6 Ramp Rate Switch **0x00** Use Ramp Rate stored in ALDB **0x01** Use instant Ramp Rate **0x02** ⇒ **0xFF** Unused D7 ⇒ D14 Unused
Trigger ALL-Link Command (SmartLabs 2476S SwitchLinc i2 Relay, 2486S KeypadLinc Relay)	**0x02**	**0x0A** **0x0F**	**0x30**	**0x00**	D1 **0x00** ⇒ **0xFF** Button/Group Number D2 On-Level Switch **0x00** Use On-Level stored in ALDB **0x01** Use On-Level in D3 **0x02** ⇒ **0xFF** Unused D3 **0x00** ⇒ **0xFF** On-Level if D2 = 0x01 D4 **0x00** ⇒ **0xFF** **SA** Command 1 to send D5 **0x00** ⇒ **0xFF** **SA** Command 2 to send D6 Ramp Rate Switch **0x00** Use Ramp Rate stored in ALDB **0x01** Use instant Ramp Rate **0x02** ⇒ **0xFF** Unused D7 ⇒ D14 Unused
Reserved			**0x31** ⇒ **0x3F**		
Set Sprinkler Program	**0x04**	**All**	**0x40**	**0x00** ⇒ **0xFF** Program Number (0x00 is Default Program)	D1 to D14 contain program data to set
Sprinkler Get Program Response [Response to **SD** **0x45xx** *Sprinkler Get Program Request*]	**0x04**	**All**	**0x41**	**0x00** ⇒ **0xFF** Program Number (0x00 is Default Program)	Added 5/05/06 D1 to D14 contain program data
Reserved			**0x42** ⇒ **0x4A**		
I/O Set Sensor Nominal	**0x07**	**All**	**0x4B**	**0x00** ⇒ **0xFF** Sensor Number	D1 **0x00** ⇒ **0xFF** Sensor Nominal Value D2 ⇒ D14 Unused
I/O Alarm Data Response [Response to **SD** **0x4700** *I/O Alarm Data Request*]	**0x07**	**All**	**0x4C**	**0x00**	D1 ⇒ D14 Alarm 1-14 Data
Reserved			**0x4C**	**0x01** ⇒ **0xFF**	
Reserved			**0x4D** ⇒ **0x4F**		

ED Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
Pool Set Device Temperature	**0x06**	**All**	**0x50**	**0x00**	D1 **0x00** Unused D1 **0x01** Pool D1 **0x02** Spa D1 **0x03** ⇒ **0xFF** Unused D2 **0x00** ⇒ **0xFF** Temperature D3 ⇒ D14 Unused
Pool Set Device Hysteresis	**0x06**	**All**	**0x50**	**0x01**	D1 **0x00** Unused D1 **0x01** Pool D1 **0x02** Spa D1 **0x03** ⇒ **0xFF** Unused D2 **0x00** ⇒ **0xFF** Hysteresis D3 ⇒ D14 Unused
Reserved			**0x50**	**0x02** ⇒ **0xFF**	
Reserved			**0x51** ⇒ **0x67**		
Thermostat Zone Temperature Up	**0x05**	**All**	**0x68**	**0x00** ⇒ **0xFF** Zone Number	D1 **0x00** ⇒ **0xFF** Temperature Change x 2 D2 ⇒ D14 Unused
Thermostat Zone Temperature Down	**0x05**	**All**	**0x69**	**0x00** ⇒ **0xFF** Zone Number	D1 **0x00** ⇒ **0xFF** Temperature Change x 2 D2 ⇒ D14 Unused
Reserved			**0x6A** ⇒ **0x6B**		
Thermostat Set Zone Cool Setpoint	**0x05**	**All**	**0x6C**	**0x00** ⇒ **0xFF** Zone Number	D1 **0x00** ⇒ **0xFF** Temperature Setpoint x 2 D2 **0x00** ⇒ **0xFF** Deadband x 2 D3 ⇒ D14 Unused
Thermostat Set Zone Heat Setpoint	**0x05**	**All**	**0x6D**	**0x00** ⇒ **0xFF** Zone Number	D1 **0x00** ⇒ **0xFF** Temperature Setpoint x 2 D2 **0x00** ⇒ **0xFF** Deadband x 2 D3 ⇒ D14 Unused
Reserved			**0x6E** ⇒ **0xEF**		
FX Commands	**All**	**All**	**0xF0** ⇒ **0xFF**	User-specific	**FX** These commands only function if *FX Usernames* in a Controller and Responder device match during linking. D1 to D14 are user-specific.

INSTEON ALL-Link Commands

This section lists **SA** Standard-length and **EA** Extended-length INSTEON ALL-Link Commands in two separate tables. Because **EA** commands are not currently used, the **EA** table is blank.

SA ALL-Link Commands are sent twice, first in an **SA** ALL-Link Broadcast message to all of the members of an ALL-Link Group, followed by separate **SC** ALL-Link Cleanup messages sent to each individual member of the ALL-Link Group.

In the **SA** ALL-Link Broadcast message, the ALL-Link Group Number appears in the *To Address* field, and the *Command 2* field contains 0x00 (with one exception for certain legacy devices as noted in the table below for the *Light Start Manual Change* Command **0x17**).

In **SC** ALL-Link Cleanup messages, the ALL-Link Group Number moves to the *Command 2* field, because the *To Address* field contains the INSTEON Address of the individual ALL-Link Group member.

INSTEON Standard-length ALL-Link Commands

The table below lists the existing INSTEON **SA** Standard-length ALL-Link Commands.

The Note Key **Req-All** denotes INSTEON commands that shall be supported by INSTEON devices in all Device Categories. **Req-All** command names appear in **bold type**.

These same commands are used in both **SA** ALL-Link Broadcast messages and **SC** ALL-Link Cleanup messages.

SA Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
Reserved			**0x00 ⇒ 0x10**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups	
ALL-Link Recall (Used as *ALL-Link Light ON* by legacy controllers)	**All**	**All**	**0x11**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups	**Req-All** Responder reverts to state remembered during ALL-Linking.
ALL-Link Alias 2 High (Used as *Light ON Fast* by legacy controllers)	**All**	**All**	**0x12**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups	Ignore Command unless, if previously set up by default or by using **ED 0x0304** *Set ALL-Link Command Alias*, then execute substitute Direct Command. For DevCats **0X01** and **0x02**, defaults to **SD 0x1200** *Light ON Fast*, which goes to saved *On-Level* instantly.
ALL-Link Alias 1 Low (Used as *Light OFF* by legacy controllers)	**All**	**All**	**0x13**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups	Ignore Command unless, if previously set up by default or by using **ED 0x0304** *Set ALL-Link Command Alias*, then execute substitute Direct Command. For DevCats **0X01** and **0x02**, defaults to **SD 0x1300** *Light OFF*, which goes full off at saved *Ramp Rate.*

SA Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
ALL-Link Alias 2 Low (Used as *Light OFF Fast* by legacy controllers)	**All**	**All**	**0x14**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups	Ignore Command unless, if previously set up by default or by using **ED 0x0304** *Set ALL-Link Command Alias*, then execute substitute Direct Command. For DevCats **0X01** and **0x02**, defaults to **SD 0x1400** *Light OFF Fast*, which goes full off instantly.
ALL-Link Alias 3 High (Used as *Light Brighten One Step* by legacy controllers)	**All**	**All**	**0x15**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups	Ignore Command unless, if previously set up by default or by using **ED 0x0304** *Set ALL-Link Command Alias*, then execute substitute Direct Command. For DevCats **0X01** and **0x02**, defaults to **SD 0x1500** *Light Brighten One Step*. There are 32 steps from off to full brightness.
ALL-Link Alias 3 Low (Used as *Light Dim* by legacy controllers)	**All**	**All**	**0x16**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups	Ignore Command unless, if previously set up by default or by using **ED 0x0304** *Set ALL-Link Command Alias*, then execute substitute Direct Command. For DevCats **0X01** and **0x02**, defaults to **SD 0x1500** *Light Dim One Step*. There are 32 steps from off to full brightness.
ALL-Link Alias 4 High (Used as *Light Start Manual Change* by legacy controllers)	**All**	**All**	**0x17**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups **NOTE:** Certain legacy SmartLabs Controllers and Responders (ControLinc V2, SwitchLinc V2, KeypadLinc V2, and LampLinc V2) use this *Command 2* field to hold a *direction* parameter during the **SA** Broadcast. **0x01** means *Increase* and **0x00** means *Decrease*. Those legacy Controllers do *not* follow up the **SA** Broadcast of this Command with an **SC** Cleanup sequence.	Ignore Command unless, if previously set up by default or by using **ED 0x0304** *Set ALL-Link Command Alias*, then execute substitute Direct Command. For DevCats **0X01** and **0x02**, defaults to **SD 0x1700** *Light Start Manual Change*, which starts changing the *On-Level*.
ALL-Link Alias 4 Low (Used as *Light Stop Manual Change* by legacy controllers)	**All**	**All**	**0x18**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups **NOTE:** Certain legacy SmartLabs Controllers (ControLinc V2, SwitchLinc V2, and KeypadLinc V2) do *not* follow up the **SA** Broadcast of this Command with an **SC** Cleanup sequence.	Ignore Command unless, if previously set up by default or by using **ED 0x0304** *Set ALL-Link Command Alias*, then execute substitute Direct Command. For DevCats **0X01** and **0x02**, defaults to **SD 0x1800** *Light Stop Manual Change*, which stops changing the *On-Level*.
Reserved			**0x19** ⇒ **0x20**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups	Do not add any new commands in this interval because legacy devices do not parse message type flags or DevCats.

SA **Commands**	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
ALL-Link Alias 5	**All**	**All**	**0x21**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups	Ignore Command unless, if previously set up by default or by using **ED 0x0304** *Set ALL-Link Command Alias*, then execute substitute Direct Command. For DevCats **0X01** and **0x02**, defaults to **SD 0x2100** *Light Instant Change*, which restores light to *On-Level* in ALL-Link Database at next zero crossing. [Added 20060420]
Reserved			**0x22** ⇒ **0xFF**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups	

INSTEON Extended-length ALL-Link Commands

The table below lists the existing INSTEON Extended-length ALL-Link Commands. Because **EA** commands are not currently used, this table is blank.

EA **Commands**	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
Reserved			**0x00** ⇒ **0xFF**	**0x00** for initial **EA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **EC** Cleanups	

INSTEON Broadcast Commands

This section lists **SB** Standard-length and **EB** Extended-length INSTEON Broadcast Commands in two separate tables. Because **EB** commands are not currently used, the **EB** table is blank.

INSTEON Standard-length Broadcast Commands

The table below lists the existing INSTEON **SB** Standard-length Broadcast Commands.

The Note Key **Req-All** denotes INSTEON commands that must be supported by INSTEON devices in all Device Categories. **Req-All** command names appear in **bold type**.

The Note Key **Req-Ex** (…) denotes INSTEON commands that must be supported by INSTEON devices in all Device Categories except as noted within the parentheses. **Req-Ex** command names appear in **bold type**.

The Note Key **Req-DC** denotes INSTEON commands that must be supported only by those INSTEON devices in the Device Categories given in the **DevCat** and **SubCat** columns. **Req-DC** command names appear in underlined type.

SB Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
Reserved			**0x00**		
SET Button Pressed Responder	**All**	**All**	**0x01**	Reserved (Set to **0xFF**)	**Req-Ex** (Required for Responder-only or Controller/Responder devices) Possible Linking Mode for a Responder or Controller/Responder device. Every INSTEON device must send *either* **SB 0x01** or **SB 0x02**
SET Button Pressed Controller	**All**	**All**	**0x02**	Reserved (Set to **0xFF**)	**Req-Ex** (Required for Controller-only devices) Possible Linking Mode for a Controller-only device. Every INSTEON device must send *either* **SB 0x01** or **SB 0x02**
Test Powerline Phase (Only sent by i2/RF devices, with *Max Hops* = 0)	**All**	**All**	**0x03**	**0x00**	Sender is on powerline phase A (low cycle). Receiver blinks LED fast for 10 seconds if on same phase. Receiver blinks LED slow for 10 seconds if on opposite phase.
				0x01	Sender is on powerline phase B (high cycle). Receiver blinks LED fast for 10 seconds if on same phase. Receiver blinks LED slow for 10 seconds if on opposite phase.
Reserved			**0x03**	**0x03 ⇒ 0xFF**	
<u>Heartbeat</u> (SmartLabs 2843 RemoteLinc)	**0x00**	**0x05**	**0x04**	**0x00 ⇒ 0xFF** Battery Level	**Req-DC** Periodic broadcast set up using **ED 0x2E** *Extended Set/Get*
Reserved			**0x05 ⇒ 0x26**		
Device Status Changed	**All**	**All**	**0x27**	Reserved (Set to **0xFF**)	Sent by a device when its status changes

SB Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description
Reserved			**0x28 ⇒ 0x48**		
SALad Debug Report	**All**	**All**	**0x49**	**0x00 ⇒ 0xFF** Not Parsed	**Req-Ex** (Only required for SALad-enabled devices) The two low bytes of the *To Address* contain the high and low bytes of the Program Counter for a SALad program being remotely debugged.
Reserved			**0x4A ⇒ 0xFF**		

INSTEON Extended-length Broadcast Commands

The table below lists the existing INSTEON Extended-length Broadcast Commands. Because **EB** commands are not currently used, this table is blank.

EB Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description
Reserved			**0x00 ⇒ 0xFF**		

Required INSTEON Commands

This section reprints in one table the Commands required for INSTEON conformance, excerpted from the *INSTEON Command Tables* document.

See the Universally-Required ALL-Link Command116, Required Direct Commands118, and Required Broadcast Commands123 sections above for more information about the required Commands, depending on whether the Command is ALL-Link, Direct, or Broadcast.

See the INSTEON Command Set Tables124 for an explanation of the color coding and abbreviations used in the Note Keys.

Required Commands for All INSTEON Devices

This table shows the Commands that all INSTEON devices must support, no matter which DevCat the device belongs to.

SD Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
Assign to ALL-Link Group	**All**	**All**	**0x01**	**0x00** ⇒ **0xFF** Group Number	**Req-All** Used during INSTEON device linking session.
Delete from ALL-Link Group	**All**	**All**	**0x02**	**0x00** ⇒ **0xFF** Group Number	**Req-All** Used during unlinking session.
Product Data Request	**All**	**All**	**0x03**	**0x00**	**Req-All, Req-Ex** (Required after 2/1/07) Addressee responds with an **ED 0x0300** *Product Data Response* message
Enter Linking Mode	**All**	**All**	**0x09**	**0x00** ⇒ **0xFF** Group Number	**Req-All** Same as holding down *SET Button* for 10 seconds **NOTE:** Not supported by i1 devices
Enter Unlinking Mode	**All**	**All**	**0x0A**	**0x00** ⇒ **0xFF** Group Number	**Req-All** **NOTE:** Not supported by i1 devices
Get INSTEON Engine Version	**All**	**All**	**0x0D**	**0x00**	**Req-All** Returned ACK message will contain the INSTEON Engine Version in Command 2. **0x00** = i1 (default echo for legacy devices) **0x01** = i2
Ping	**All**	**All**	**0x0F**	**0x00** (**0x01** ⇒ **0xFF** Not Parsed in legacy devices. Use only 0x00 in the future.)	**Req-All** Addressee returns an ACK message but performs no operation.
ID Request	**All**	**All**	**0x10**	**0x00** (**0x01** ⇒ **0xFF** Not Parsed in legacy devices. Use only 0x00 in the future.)	**Req-All** Addressee first returns an ACK message, then it sends an **SB 0x01** *SET Button Pressed Responder* or **SB 0x02** *SET Button Pressed Controller* Broadcast message, but it does not enter Linking Mode.
ED Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**
Product Data Response [Response to **SD 0x0300** *Product Data*	**All**	**All**	**0x03**	**0x00**	**Req-All, Req-Ex** (Required after 2/1/07) D1 **0x00** Reserved (always set to 0x00) D2 **0x00** ⇒ **0xFF** INSTEON Product Key MSB

Request]					D3 **0x00** ⇒ **0xFF** INSTEON Product Key 2MSB	
					D4 **0x00** ⇒ **0xFF** INSTEON Product Key LSB	
					D5 **0x00** ⇒ **0xFF** Device Category (DevCat)	
					D6 **0x00** ⇒ **0xFF** Device Subcategory (SubCat)	
					D7 **0xFF** Reserved (always set to 0xFF) (Matches byte in LSB of *To Address* of **SB 0x01** *SET Button Pressed Responder* or **SB 0x02** *SET Button Pressed Controller* commands)	
					D8 **0xFF** Reserved (always set to 0xFF) (Matches byte in Command 2 of **SB 0x01** *SET Button Pressed Responder* or **SB 0x02** *SET Button Pressed Controller* commands)	
					D9 ⇒ D14 User-defined	
Read/Write ALL-Link Database (ALDB)	**All**	**All**	**0x2F**	**0x00**	**Req-All, Req-Ex, DataTr** (Required for all i2 devices) Not implemented in i1 devices	
					D1 Unused	
					D2 **0x00** ALDB Record Request [Addressee responds with ALDB Record Response(s)] **NOTE:** Set address to 0x0000 to start at first record in ALDB. (Actual memory address is 0x0FFF in SmartLabs devices.)	D3 **0x00** ⇒ **0xFF** Address High Byte
						D4 **0x00** ⇒ **0xFF** Address Low Byte
						D5 **0x00** Dump all records D5 **0x01** ⇒ **0xFF** Dump one record
						D6 ⇒ D14 Unused
					D2 **0x01** ALDB Record Response [Response to ALDB Record Request] If D5 of ALDB Record Request was 0x00, return one record, else return all records until end of ALDB is reached. (Flag Byte in last record will be 0x00). Address is automatically decremented by 8 for each record returned.	D3 **0x00** ⇒ **0xFF** Address High Byte
						D4 **0x00** ⇒ **0xFF** Address Low Byte
						D5 Unused
						D6 ⇒ D13 **0x00** ⇒ **0xFF** Returned 8-byte Record
						D14 Unused
					D2 **0x02** Write ALDB Record	D3 **0x00** ⇒ **0xFF** Address High Byte
						D4 **0x00** ⇒ **0xFF** Address Low Byte

						D5 **0x01** ⇒ **0x08** Number of Bytes (**0x09** ⇒ **0xFF** is the same as **0x08**)
						D6 ⇒ D13 **0x00** ⇒ **0xFF** 8-byte Record to Write
						D14 Unused
					D2 **0x03** ⇒ **0xFF** Unused	
SA Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**	
ALL-Link Recall (Used as *ALL-Link Light ON* by legacy controllers)	**All**	**All**	**0x11**	**0x00** for initial **SA** Broadcast, **0x00** ⇒ **0xFF** (Group Number) for **SC** Cleanups	**Req-All** Responder reverts to state remembered during ALL-Linking.	
SB Commands	**Dev Cat**	**Sub Cat**	**Cmd 1**	**Cmd 2**	**Note Keys, Description**	
SET Button Pressed Responder	**All**	**All**	**0x01**	Reserved (Set to **0xFF**)	**Req-Ex** (Required for Responder-only or Controller/Responder devices) Possible Linking Mode for a Responder or Controller/Responder device. Every INSTEON device must send *either* **SB 0x01** or **SB 0x02**	
SET Button Pressed Controller	**All**	**All**	**0x02**	Reserved (Set to **0xFF**)	**Req-Ex** (Required for Controller-only devices) Possible Linking Mode for a Controller-only device. Every INSTEON device must send *either* **SB 0x01** or **SB 0x02**	

Note that the *SET Button Pressed Responder* and *SET Button Pressed Controller* **SB** Commands are included in this table because one or the other, or possibly both Commands must be supported by all INSTEON devices.

Required Commands for Some INSTEON Devices

This table shows the Commands that certain INSTEON devices must support if they meet the conditions given in red type.

SD Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description
FX Username Request	**All**	**All**	**0x03**	**0x01**	**Req-Ex** (Only required for devices that support FX Commands) , **FX** Addressee responds with an **ED 0x0301** *FX Username Response* message
ED Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description
FX Username Response [Response to **SD 0x0301** *FX Username Request*]	**All**	**All**	**0x03**	**0x01**	**Req-Ex** (Only required for devices that support FX Commands), **FX**
					D1 ⇒ D8 **0x00** ⇒ **0xFF** FX Command Username Used for *FX Commands*, which are user-specific **SD** or **ED** commands numbered **0xFF00** ⇒ **0xFFFF**
					D9 ⇒ D14 User-defined
SB Commands	Dev Cat	Sub Cat	Cmd 1	Cmd 2	Note Keys, Description
SALad Debug Report	**All**	**All**	**0x49**	**0x00** ⇒ **0xFF** Not Parsed	**Req-Ex** (Only required for SALad-enabled devices) The two low bytes of the *To Address* contain the high and low bytes of the Program Counter for a SALad program being remotely debugged.

Required Commands for a Device Category

Device Categories (DevCats) are used to qualify Direct Commands (**SD** or **ED**), as described above in the section *Required Direct Commands within a DevCat*118. See the *INSTEON Command Set Tables*124 for the current list of required **SD** and **ED** Commands for each DevCat. Required Direct Commands within a DevCat are given in the *INSTEON Command Set Tables*124 in underline type.

INSTEON Command Number Assignment

Manufacturers may contact SmartLabs Technology in order to apply for a new INSTEON Command Number (ICN).

In the future, manufacturers of INSTEON products will receive new INSTEON Commands (ICNs) for their products from a secure INSTEON Command Number Server (ICNS). SmartLabs will provide authorized manufacturers conditional access to the ICNS.

To apply for a new ICN, a manufacturer will fill in an application for a new Command. SmartLabs personnel will review the application, and if approved, the ICNS will issue a new ICN to the manufacturer via email.

INSTEON Command Database (ICDB)

ICNs are a shared public resource. INSTEON Controller devices use them to initiate actions from linked INSTEON Responder devices. ICNs also serve as a lookup key to the INSTEON Command Database (ICDB).

ICDB Lookup Keys

The main key to the ICDB will be the ICN concatenated with the message length (Standard or Extended), the message type (Direct, ALL-Link, or Broadcast), and the 1-byte Device Category (DevCat).

ICDB Records

Queries to the ICDB will return XML files. The XML schema is not yet defined, but some fields could be:

- Text name of the Command
- Button label for the Command
- INSTEON Products (given by the INSTEON Product Key) that use the Command
- Allowed Command parameters
- Reply expected

About INSTEON Peek and Poke Commands

You can use the **SD** Standard-length *Peek One Byte* and *Poke One Byte* INSTEON Commands (**0x28** ⇒ **0x2D**) to remotely read or write one byte of memory at a time in INSTEON devices. For example, if you know the relevant memory addresses, you could inspect or alter the *INSTEON ALL-Link Database*101 of an INSTEON device to determine which *INSTEON ALL-Link Groups*93 it belongs to (i.e. which other INSTEON devices it has ALL-Links to).

However, these are legacy commands whose continued use is deprecated. Higher-level software such as that described in *Chapter 10 — INSTEON Modems*217 and *Chapter 12 — SmartLabs Device Manager (SDM) Reference*336 provide functions that read and write a device's ALL-Link Database and other key memory locations without regard to absolute memory addresses.

To transfer arbitrary blocks of data from memory to memory between INSTEON devices, SmartLabs has defined a set of **ED** Extended-length *Block Data Transfer* Commands.

This section describes both *Using Peek and Poke Commands for One Byte*162 and *Using the* Block Data Transfer *Command for Multiple Bytes*163. It also gives some *Peek and Poke Command Examples*164.

Using Peek and Poke Commands for One Byte

SALad-enabled INSTEON devices, such as *The SmartLabs PowerLinc Controller*28, map all memory to one *Flat Memory Map*170, but other INSTEON devices, such as SmartLabs' ControLinc™ V2, LampLinc™ V2, and SwitchLinc™ V2, do not have a flat address space. For flat devices, use the **SD 0x2B** *Peek One Byte* and **SD 0x29** *Poke One Byte* commands to access all memory. For non-flat devices use those Commands to access external EEPROM. You can only use **SD 0x2C** *Peek One Byte Internal*, and **SD 0x2D** *Poke One Byte Internal* to access internal EEPROM of the ControLinc™ V2. Note that you cannot access a device's ROM using these Commands.

To peek or poke remote memory data, first use the **SD 0x28** *Set Address MSB* Command to set the high byte of the 16-bit address you want to read from or write to. You will set the LSB of the address differently depending on what you will be doing next. If you are going to *Poke One Byte* or *Poke One Byte Internal*, you first have to execute a *Peek One Byte* Command to set the address LSB. In the other cases, *Peek One Byte* and *Peek One Byte Internal*, you will set the LSB in the Command itself. Note that the address LSB does not auto-increment.

To peek one byte of data, use *Peek One Byte* or Peek *One Byte Internal*. The **SD** Acknowledgement message that you receive back will contain the peeked byte in the *Command 2* field.

To poke one byte of data, use *Poke One Byte* or *Poke One Byte Internal*. Remember to set the address you want to poke to by using a *Set Address MSB* Command followed by a *Peek One Byte* Command. The *Command 2* field of the **SD**

Acknowledgement message that you receive back will contain the value of the byte you are poking before it was altered by the poke.

Using the *Block Data Transfer* Command for Multiple Bytes

The **ED** Extended-length *Block Data Transfer* Command set (**0x2Axx**) is a powerful mechanism for memory-to-memory transfer of data from one INSTEON device to another. The transfer uses **ED** Extended-length Direct messages to carry the data independently of other intervening INSTEON traffic.

To initiate a transfer, send an **ED 0x2AFF** *Request Block Data Transfer* Command with the 16-byte memory *Source Address*, memory *Destination Address*, and *Block Length* fields specified in the *User Data* field. Set the *Destination ID* field to the INSTEON address of the device that will be receiving the data. If you want the sending device to peek data from the *Request Block Data Transfer* addressee, you would set the *Destination ID* to that of the sending device. If you want to poke data to the addressee, set the *Destination ID* to that of the addressee. If you want the addressee to transfer memory data to some other INSTEON device, set the *Destination ID* to that of the other device.

Once the transfer begins, some number (possibly zero) of **ED 0x2A0D** *Transfer Continues* messages will carry 12 bytes of data each from the *Request Block Data Transfer* Command addressee to the *Destination ID* device. After as many groups of 12 bytes as needed has been transferred using *Transfer Continues* messages, a final **ED 0x2Axx** *Transfer Complete, xx Bytes Remaining* message will transfer the remainder of the bytes. Here, *xx* designates how many bytes are in the remainder message, and it ranges from **0x01** to **0x0C** (1 to 12).

If the transfer is aborted for some reason, the transfer sender should send an **ED 0x2A00** *Transfer Failure* message.

Note that not all INSTEON devices implement the *Block Data Transfer* Commands. To test whether an INSTEON device can execute this Command set, try peeking one byte of data from it. If it does not respond with **ED 0x2D01** *Transfer Complete, 1 Byte in This Last Message*, then you can try *Peek One Byte* as described in *Using Peek and Poke Commands for One Byte* 162 above.

Peek and Poke Command Examples

To see some typical examples of INSTEON Commands being used in INSTEON messages, look in the sections *Example of an INSTEON ALL-Linking Session*97 and *Example of an ALL-Link Command Sequence*99.

The examples in this section are fairly sophisticated. To use them, you must know what you are doing. In particular, you should fully understand the *INSTEON ALL-Link Database*101, which is not documented until later in this Developer's Guide.

Some of the examples involve directly poking data into an INSTEON Device's ALL-Link Database. If you do this improperly you can corrupt the database and cause the device to malfunction, in which case you will need to perform a 'Factory Reset' on the INSTEON device. The User Guide for the device will explain how to do this.

Common SmartLabs INSTEON Device Memory Locations

This table gives some memory locations found in many SmartLabs INSTEON devices, such as the LampLinc™, SwitchLinc™, and KeypadLinc™.

Address	Variable Name	Description
0x0001	EESize	MSB of size of EEPROM chip in device
0x0002	EEVersion	Firmware Revision number
0x0020	EEOnLevel	Preset On-Level for lighting control
0x0021	EERampRate	Ramp Rate for lighting control
0x0030	EEX10BaseHouse	Base X10 House Code for device
0x0031	EEX10BaseUnit	Base X10 Unit Code for device

Assumptions for the Following Examples

The numbers in these examples are all hexadecimal.

We assume that you are using *The SmartLabs PowerLinc Controller*28 (PLC) with an INSTEON Address of 01.02.03 to send and receive INSTEON messages.

The INSTEON device you are talking to is a SmartLabs LampLinc V2 Dimmer with an INSTEON Address of A1.A2.A3.

All INSTEON messages have a *Max Hops* of 3 and a *Hops Left* of 3, so the low nibble of the *Flags* byte is always 0xF.

These examples involve peeking and poking data directly in the LampLinc's memory using the INSTEON *Peek One Byte* and *Poke One Byte* Commands. We assume that you have read *Using Peek and Poke Commands for One Byte*162, which explains how to use these Commands.

Peek an On-Level

PLC Sends		LampLinc Responds
Set Address MSB to 00		**OK**
From Address	**01 02 03** (PLC)	**A1 A2 A3** (LampLinc)
To Address	**A1 A2 A3** (LampLinc)	**01 02 03** (PLC)
Flags	**0F** (**SD** Message)	**2F** (**SD** ACK Message)
Command 1	**28** (*Set Address MSB*)	**28** (*Set Address MSB*)
Command 2	**00** (MSB of 0x0020)	**00** (MSB of 0x0020)
Peek One Byte at Address 0020		**OK**
From Address	**01 02 03** (PLC)	**A1 A2 A3** (LampLinc)
To Address	**A1 A2 A3** (LampLinc)	**01 02 03** (PLC)
Flags	**0F** (**SD** Message)	**2F** (**SD** ACK Message)
Command 1	**2B** (*Peek*)	**2B** (*Peek*)
Command 2	**20** (LSB of 0x0020)	**XX (**Byte at 0x0020)

Poke a New Ramp Rate

PLC Sends		LampLinc Responds
Set Address MSB to 00		**OK**
From Address	**01 02 03** (PLC)	**A1 A2 A3** (LampLinc)
To Address	**A1 A2 A3** (LampLinc)	**01 02 03** (PLC)
Flags	**0F** (Direct Message)	**2F** (Direct ACK Message)
Command 1	**28** (*Set Address MSB*)	**28** (*Set Address MSB*)
Command 2	**00** (MSB of 0x0021)	**00** (MSB of 0x0021)
Peek One Byte to Set Address LSB to 21		**OK**
From Address	**01 02 03** (PLC)	**A1 A2 A3** (LampLinc)
To Address	**A1 A2 A3** (LampLinc)	**01 02 03** (PLC)
Flags	**0F** (Direct Message)	**2F** (Direct ACK Message)
Command 1	**2B** (*Peek*)	**2B** (*Peek*)
Command 2	**21** (LSB of 0x0021)	**XX (**Byte at 0x0021)
Poke One Byte 80 at Address 0021		**OK**
From Address	**01 02 03** (PLC)	**A1 A2 A3** (LampLinc)
To Address	**A1 A2 A3** (LampLinc)	**01 02 03** (PLC)
Flags	**0F** (Direct Message)	**2F** (Direct ACK Message)
Command 1	**29** (*Poke*)	**29** (*Poke*)
Command 2	**80** (Byte to poke)	**80** (Byte to poke)

Chapter 9 — INSTEON BIOS (IBIOS)

The INSTEON Basic Input/Output System (IBIOS) implements the basic functionality of INSTEON devices like the SmartLabs PowerLinc™ V2 Controller (PLC). Developers who are using one of the INSTEON Modems described in *Chapter 10 — INSTEON Modems*217 may skip this chapter, since an IM provides high-level access to IBIOS functionality along with additional functions that the IBIOS does not support directly.

Built on a flat 16-bit address space, the IBIOS firmware includes PLC event generation, USB or RS232 serial communications, IBIOS Serial Command processing, a software realtime clock, a SALad language interpreter, an INSTEON Engine that handles low-level INSTEON messaging, and various other functions.

The SALad language interpreter runs SALad applications that can stand alone or interface serially with other computing devices. *The SmartLabs PowerLinc Controller*28 comes from the factory with a pre-installed *SALad coreApp Program*272 that provides onboard event handling, INSTEON device ALL-Linking, and many other useful functions not supported by IBIOS. But even without a SALad program like coreApp running, IBIOS offers a sophisticated serial interface to the outside world.

This chapter explains what the INSTEON Engine does, what IBIOS Events occur, how serial communication works, how to use IBIOS Serial Commands directly, and other features of IBIOS. See *Chapter 11 — SALad Language Documentation*263 for complete information on writing and running SALad programs.

In This Chapter

IBIOS Flat Memory Model

All IBIOS memory, no matter what its physical address, is accessed using a flat 16-bit address space. Microcontroller RAM, microcontroller internal (high-speed) EEPROM, external I^2C serial (low-speed) EEPROM, and any other I^2C chips are all mapped into this single 16-bit space.

In This Section

Flat Memory Addressing

The 16-bit flat address space is broken up into two regions. The bottom 32K (0x0000 through 0x7FFF) is a fixed area that always maps to the same physical memory. The top 32K (0x8000 through 0xFFFF) is a variable area that maps to the physical memory of up to four different I^{2}C chips provided in hardware, under control of the register `I2C_Addr`.

The bottom 32K fixed area always contains the microcontroller RAM registers in addresses 0x0000 through 0x01FF, and microcontroller internal EEPROM in addresses 0x200 through 0x02FF. The remainder of the bottom 32K, from 0x0300 through 0x7FFF, always maps to the physical memory of the primary external I^{2}C serial EEPROM chip from 0x0000 through 0x7CFF. (The *primary* external I^{2}C serial chip has all of its chip-select pins tied low in hardware.)

To choose which of the four possible I^{2}C chips to map to the top 32K of the flat address space, set the top 7 bits of the `I2C_Addr` register. The top 4 bits (bits 7 – 4) will match the I^{2}C address burned into the I^{2}C chip by the manufacturer. Bits 3 and 2 will match the way the chip-select pins of the I^{2}C chip are wired in hardware. Set the bit to 0 for a pin that is tied low, or to 1 for a pin that is tied high. The third chip-select pin must always be tied low in hardware so that bit 1 of the `I2C_Addr` register can be used to indicate whether the I^{2}C chip uses 8-bit or 16-bit addressing. (A chip with no more than 256 memory locations, such as a realtime clock chip, will normally use 8-bit addressing, but chips with more than 256 memory locations, such as serial EEPROMs, *must* use 16-bit addressing.)

While the top 7 bits of the `I2C_Addr` register uniquely identify which I^{2}C chip to map into the top 32K of flat memory, the least-significant bit (the LSb, bit 0), selects whether the bottom 32K or the top 32K of this chip's memory will appear in that space. If the I^{2}C chip in question does not contain more than 32K of memory, it does not actually matter how the `I2C_Addr` register's LSb is set. But if the I^{2}C chip does contain more than 32K of memory, then setting the LSb to 0 will access the bottom 32K, and setting the LSb to 1 will access any memory above 32K.

Note that I^{2}C chips that contain fewer than 32K addresses in power-of-two increments (e.g. 1K, 2K, 4K, 8K, or 16K chips) will have their available addresses repeated multiple times in the top 32K of flat memory. For example, an 8K chip will appear four times in the 32K space, and a 16K chip will appear twice. (Mathematically, if the chip contains 2^N addresses, and N is 15 or less, then the chip's contents will be repeated 2^(15 – N) times in the top 32K of flat memory.)

Overall Flat Memory Map

Address Range	Contents
0x0000⇒0x01FF	Microcontroller RAM Registers
0x0200⇒0x02FF	Microcontroller Hi speed EEPROM
0x0300⇒0x7FFF	Serial I^{2}C EEPROM physical addresses 0x0000 through 0x7CFF
0x8000⇒0xFFFF	Varies depending on contents of register I2C_Addr I2C_Addr: xxxxxxx0 = I^{2}C physical addresses 0x0000⇒0x7FFF I2C_Addr: xxxxxxx1 = I^{2}C physical addresses 0x8000⇒0xFFFF

I2C_Addr Register

I2C_Addr Register	
Bit	**Meaning**
7 (MSb)	Top nibble of I^2C Address burned into I^2C chip by manufacturer.
6	
5	
4	
3	Top 2 of 3 chip-select pins tied high or low in hardware design. There can be a total of 4 I^2C chips.
2	
1	3rd of 3 chip-select pins must be tied low in hardware design, so that IBIOS can use this bit as follows: 0 = I^2C chip uses 8-bit addressing (e.g. realtime clock chip). 1 = I^2C chip uses 16-bit addressing (e.g. serial EEPROM).
0 (LSb)	Defined by I^2C spec as Read/Write bit, but used by IBIOS as follows: 0 = bottom 32K of I^2C chip is mapped to top 32K of flat memory. 1 = top 32K of I^2C chip is mapped to top 32K of flat memory.

Flat Memory Map

The memory map for devices containing an i2 INSTEON Engine has changed. See *i2 Engine Memory Map*170 for the new memory map and *i1 Engine Memory Map*178 for the original one.

i2 Engine Memory Map

i2 Address	i2 Register and Bits	Description
0x0024	NTL_CNT	Count for SALad block mode operations
0x0026	RD_H	Remote Debugging breakpoint address MSB
0x0027	RD_L	Remote Debugging breakpoint address LSB
0x0028	PC_H	SALad Program Counter MSB
0x0029	PC_L	SALad Program Counter LSB
0x002A	DB_H	Database Pointer MSB
0x002B	DB_L	Database Pointer LSB
0x002C	NTL_SP_H	Return Stack Pointer MSB
0x002D	NTL_SP_L	Return Stack Pointer LSB
0x0033	NTL_BUFFER	Pointer to end of Timer Buffer, which begins at 0x0046. This 8-bit pointer defaults to 0x4D to allow room for 4 timers which are 2 bytes each.
0x0034	NTL_RND	Random Number Register
0x0035	NTL_REG_H	High byte of Pointer to R0
0x0036	NTL_REG_L	Low byte of Pointer to R0
0x0037	NTL_EVENT	Event used to invoke SALad
0x0038 ⇒ 0x003F	NTL_EVNT0- NTL_EVNT7	Static Event Queue
0x0040	NTL_TIME_H	Time-of-day alarm (minutes since midnight MSB)
0x0041	NTL_TIME_L	Time-of-day alarm (minutes since midnight LSB)
0x0042	NTL_TICK	Zero Crossing count down tick timer
0x0048 ⇒ NTL_BUFFER	NTL_TIMERS	Timer Buffer; Starts at 0x0046
NTL_BUFFER ⇒ NTL_SP	NTL_REGS	User Register Space
NTL_SP ⇒ 0x006F	NTL_STACK	SALad Return Stack
NOTE: The following serial receive buffer only exists for devices with serial communications		
0x011F ⇒ 0x0140	RX_Buffer	Serial receive buffer (33 bytes)

i2 Address	i2 Register and Bits		Description
0x0141	RX_PTR		Points to next open slot in serial receive buffer
	_RX_NotEmpty	5	1=Receive buffer contains data
	_RX_Full	6	1=Receive buffer full
NOTE: The following locations differ with or without an RX_Buffer			
0x013D if RX_Buffer 0x0153 if no RX_Buffer	I_Control		INSTEON result flags
	_I_SendDirect	5	0=Send INSTEON from working buffer, 1=Send INSTEON direct
	_I_Transmit	4	1=Request To Send INSTEON
	_RF_LowBatt	2	1=Battery Low
	_RF_Disable	1	1=Disables RF
	_InsteonDisable	0	1=Disables INSTEON Engine
0x013E if RX_Buffer 0x0154 if no RX_Buffer	I_Error		INSTEON error flags
	_MsgFail	4	1=Transmitted message was interrupted by incoming message
NOTE: The following locations are the same with or without an RX_Buffer			
0x016B	TAP_CNT		Counts multiple *SET Button* taps
0x016C	Control		General system control flags
	_Reset	7	1=Request system reset
	_Watchdog	6	1=Request watchdog reset
	_ForceDebug	5	1=Force Debugging to start each time SALad starts
	_PDI	2	1=Daughter card interrupt occurred and has been serviced
	_NoEventRpt	1	1=Inhibit static event report
	_TAP_LAST	0	Last state of push button
0x0174	TOKEN		Currently executing SALad instruction token
0x0175	NTL_STAT		SALad Status Register
	_I2C_E1	7	0=I2C Closed, 1=I2C Open
	_I2C_E0	6	0=!2C read, 1=I2C Write
	_NTL_16	5	0=8 bit, 1=16 bit
	_NTL_Idle	4	Idle process active
	_DB_PASS	3	1=ALL-Link Database search successful
	_NTL_DZ	2	1=Divide by Zero
	_NTL_BO	1	1=Buffer Overrun

i2 Address	i2 Register and Bits		Description		
	_NTL_CY	0	1=Carry from Math and Test operations		
0x0176	NTL_CONTROL		SALad debugging control flags		
	_RD_STEP	7	_RD_HALT	_RD_STEP	
	_RD_HALT	6	0	0	Normal execution
			0	1	Animation (Trace)
			1	0	Execution halted
			1	1	Single step requested
	_RD_BREAK	5	0=Range Check Mode, 1=Breakpoint Mode		
	_MEM_LOCK	4	1=memory is write-locked to SALad programs		
	_I2C_WD	3	1=I2C Write Delay disable		
	_I2C_EM	2	Last EE Mode, 0=read, 1=write		
	_NTL_FN	1	1=Extended function enable		
	_NTL_AI	0	1=Auto Increment enable		
0x01DB	TX_PTR		Address of next byte in TX_Buffer		
0x01DC ⇒ 0x01DF	TX_Buffer		Serial Transmit Buffer (4 bytes)		
0x01E0	Tick		Incremented from 0⇒120 every second		
0x01E0	Tick		Incremented from 0⇒120 every second		
0x01E1	RTC_TIME_H		Time since midnight in minutes (MSB, 0-1439)		
0x01E2	RTC_TIME_L		Time since midnight in minutes (LSB, 0-1439)		
0x01E3	RTC_YEAR		Year (0-99)		
0x01E4	RTC_MON		Month (1-12)		
0x01E5	RTC_DAY		Day (1-31, month specific)		
0x01E6	RTC_DOW		Day-of-Week bitmap (0SSFTWTM)		
0x01E7	RTC_HOUR		Hour (0-23)		
0x01E8	RTC_MIN		Minute (0-59)		
0x01E9	RTC_SEC		Second (0-59)		
0x01ED	X10_RX		X10 Receive Buffer		
0x01EE	X10_TX		X10 Transmit Buffer		
0x01EF	X10_FLAGS		X10 Flags		
	_X10_RTS	7	1=Request To Send		
	_X10_TX	6	1=Transmitting, 0=Receiving		

i2 Address	i2 Register and Bits		Description
	_X10_EXTENDED	5	1=Extended transfer in progress (Tx or Rx)
	_X10_COMBUF	4	1=Command, 0=Address (for internal use)
	_X10_TXCOMMAND	3	1=Command, 0=Address for transmit
	_X10_RXCOMMAND	2	1=Command, 0=Address for receive
	_X10_VALID	1	1=X10 receive valid
	_X10_ACTIVE	0	1=X10 active (for internal use)
0x0164	LED_MODE		Bitmap defines flashing pattern for LED 1=On, 0=Off
0x0165	LED_TMR		Duration of LED flashing in seconds
0x0166	LED_DLY		Period between each flash. Defaults to 5, which is 1/8 second per bit in LED_MODE.
0x0167	RS_CONTROL		Control flags for serial command interpreter
	_RS_ComReset	7	These are used for serial command time limit. This limits how long a command remains active in SALad. This prevents the serial engine from locking you out if SALad receives a corrupt command (non-native serial command). Default time limit is two seconds. To disable, clear bits 5⇒7.
	_RS_ComLimit2	6	
	_RS_ComLimit1	5	
	_I_DebugEnable	4	1=Enable debug report
	_RS_AppLock	3	1=Enable overwriting of SALad code from 0x0200 to end of SALad App given in 0x0216 and 0x0217
	_RS_ComDisable	2	1=Core command processing disabled
	_RS_ComActive	1	1=Command active for SALad processing (Non-native serial command)
	_RS_02	0	1=0x02 received for command start
0x0169	RS_ERROR		RS232 Error register
	_TX_Full	4	Transmit buffer full
	_TX_Empty	3	Transmit buffer empty
0x016F	EventMask		Mask to enable or disable events
	_EM_BtnTap	7	1=enabled
	_EM_BtnHold	6	1=enabled
	_EM_BtnRel	5	1=enabled
	_EM_TickTimer	4	1=enabled
	_EM_Alarm	3	1=enabled
	_EM_Midnight	2	1=enabled
	_EM_2AM	1	1=enabled

i2 Address	i2 Register and Bits		Description
	_EM_RX	0	1=enabled
0x017D	I2C_ADDR		Address of I^2C device; bit 0 controls 0x8000 ⇒ 0xFFFF of flat model, 1=hi region, 0=low region
0x01A0	DB_FLAGS		Database search mode bitmap
0x01A1	DB_0		Database ID_H; (INSTEON construction buffer) From Address_H; ignored for INSTEON message construction
0x01A2	DB_1		Database ID_M; (INSTEON construction buffer) From Address_M; ignored for INSTEON message construction
0x01A3	DB_2		Database ID_L; (INSTEON construction buffer) From Address_L; ignored for INSTEON message construction
0x01A4	DB_3		Database Command 1; (INSTEON construction buffer) To Address_H
0x01A5	DB_4		Database Command 2; (INSTEON construction buffer) To Address_M
0x01A6	DB_5		Database Group Number; (INSTEON construction buffer) To Address_L
0x01A7	DB_6		Database State; (INSTEON construction buffer) Message Flags
0x01A8	DB_7		Message Command 1; (INSTEON construction buffer) Command 1
0x01A9	DB_8		Message Command 2; (INSTEON construction buffer) Command 2
0x01AA	DB_9		
NOTE: The following locations may move as a group in different devices			
0X01AB	RxFrom0		Receive “From” address high byte
0x01AC	RxFrom1		Receive “From” address middle byte
0x01AD	RxFrom2		Receive “From” address low byte
0x01AE	RxTo0		Receive “To” address high byte
0x01AF	RxTo1		Receive “To” address middle byte
0x01B0	RxTo2		Receive “To” address low byte
0x01B1	RxExtRpt		Receive Control Flags
	_RxBroadcastBit	7	Broadcast Message
	_RxGroup	6	ALL-Link Message

i2 Address	i2 Register and Bits		Description
	_RxAckBit	5	Acknowledge Message
	_RxExtMsgBit	4	Extended Message
	_RxMsgRpt	3,2	Hops Left
	_RxTotalRpt	1,0	Max Hops
0x01B2	RxCmd1		Command byte 1
0x01B3	RxCmd2		Command byte 2
0x01B4	RxExtData0		Standard message CRC or Extended message User Data D1
0x01B5	RxExtData1		Extended message User Data D2
0x01B6	RxExtData2		Extended message User Data D3
0x01B7	RxExtData3		Extended message User Data D4
0x01B8	RxExtData4		Extended message User Data D5
0x01B9	RxExtData5		Extended message User Data D6
0x01BA	RxExtData6		Extended message User Data D7
0x01BB	RxExtData7		Extended message User Data D8
0x01BC	RxExtData8		Extended message User Data D9
0x01BD	RxExtData9		Extended message User Data D10
0x01BE	RxExtDataA		Extended message User Data D11
0x01BF	RxExtDataB		Extended message User Data D12
0x01C0	RxExtDataC		Extended message User Data D13
0x01C1	RxExtDataD		Extended message User Data D14
0x01C2	RxExtCrc		Extended message CRC
0x01C3	TxFrom0		Transmit “From” address high byte
0x01C4	TxFrom1		Transmit “From” address middle byte
0x01C5	TxFrom2		Transmit “From” address low byte
0x01C6	TxTo0		Transmit “To” address high byte
0x01C7	TxTo1		Transmit “To” address middle byte
0x01C8	TxTo2		Transmit “To” address low byte
0x01C9	TxExtRpt		Transmit Control Flags
	_TxBroadcastBit	7	Broadcast
	_TxGroup	6	ALL-Link
	_TxAckBit	5	Acknowledge
	_TxExtMsgBit	4	Extended

i2 Address	i2 Register and Bits		Description
	_TxMsgRpt	3,2	Hops Left
	_TxTotalRpt	1,0	Max Hops
0x01CA	TxCmd1		Command byte 1
0x01CB	TxCmd2		Command byte 2
0x01CC	TxExtData0		Standard message CRC or Extended message User Data D1
0x01CD	TxExtData1		Extended message User Data D2
0x01CE	TxExtData2		Extended message User Data D3
0x01CF	TxExtData3		Extended message User Data D4
0x01D0	TxExtData4		Extended message User Data D5
0x01D1	TxExtData5		Extended message User Data D6
0x01D2	TxExtData6		Extended message User Data D7
0x01D3	TxExtData7		Extended message User Data D8
0x01D4	TxExtData8		Extended message User Data D9
0x01D5	TxExtData9		Extended message User Data D10
0x01D6	TxExtDataA		Extended message User Data D11
0x01D7	TxExtDataB		Extended message User Data D12
0x01D8	TxExtDataC		Extended message User Data D13
0x01D9	TxExtDataD		Extended message User Data D14
0x01DA	TxExtCrc		Extended message CRC
NOTE: The following locations are the same in all devices			
0x0200	VALID		= ‘P’ if EEPROM is valid; 0x0200 is beginning of microcontroller internal EEPROM
0x0201	ID_H		High byte of ID
0x0202	ID_M		Middle byte of ID
0x0203	ID_L		Low byte of ID
0x0204	DEV_TYPE		Device Category
0x0205	SUB_TYPE		Device Subcategory
0x0206	REV		Firmware Revision (MSN=Release, LSN=Ver)
0x0207	MEM_SIZE		Mask for installed external memory: 00000000=none 00001111=4K 01111111=32K 11111111=64K
0x0210 ⇒ 0x0211	APP_ADDR_TEST		Address of range of application for verification test

i2 Address	i2 Register and Bits	Description
0x0212 ⇒ 0x0213	APP_LEN_TEST	Length of range of application for verification test
0x0214 ⇒ 0x0215	APP_CHECK_TEST	Two's complement checksum of range of application for verification test
0x0216 ⇒ 0x0217	APP_END	Top of currently loaded SALad application. EEPROM is write-protected from 0x0200 to the address contained here. Set 0x16B bit 7 to enable over-writing.
0x0230	TIMER_EVENT	Entry point of SALad application for Timer Events
0x0237	STATIC_EVENT	Entry point of SALad application for Static Events
0x0230 ⇒ 0x02FF	SALad application code for fast execution; Microcontroller Internal EEPROM	
0x0300 ⇒ 0x7FFF	Slow SALad application code; External serial EEPROM	
0x8000 ⇒ 0xFFFF	If I2C_Addr bit 0 = **0**, then 0x8000⇒0xFFFF is **low** region (0x0000⇒0x7FFF) of memory in I2C device specified by upper 7 bits of I2C_Addr. If I2C_Addr bit 0 = **1**, then 0x8000⇒0xFFFF is **high** region (0x8000⇒0xFFFF) of memory in I2C device specified by upper 7 bits of I2C_Addr.	

i1 Engine Memory Map

i1 Address	i1 Register and Bits		Description
0x0024	NTL_CNT		Count for SALad block mode operations
0x0026	RD_H		Remote Debugging breakpoint address MSB
0x0027	RD_L		Remote Debugging breakpoint address LSB
0x0028	PC_H		SALad Program Counter MSB
0x0029	PC_L		SALad Program Counter LSB
0x002A	DB_H		Database Pointer MSB
0x002B	DB_L		Database Pointer LSB
0x002C	NTL_SP_H		Return Stack Pointer MSB
0x002D	NTL_SP_L		Return Stack Pointer LSB
0x0033	NTL_BUFFER		Pointer to end of Timer Buffer, which begins at 0x0046. This 8-bit pointer defaults to 0x4D to allow room for 4 timers which are 2 bytes each.
0x0034	NTL_RND		Random Number Register
0x0035	NTL_REG_H		High byte of Pointer to R0
0x0036	NTL_REG_L		Low byte of Pointer to R0
0x0037	NTL_EVENT		Event used to invoke SALad
0x0038 ⇒ 0x003F	NTL_EVNT0- NTL_EVNT7		Static Event Queue
0x0040	NTL_TIME_H		Time-of-day alarm (minutes since midnight MSB)
0x0041	NTL_TIME_L		Time-of-day alarm (minutes since midnight LSB)
0x0042	NTL_TICK		Zero Crossing count down tick timer
0x0046 ⇒ NTL_BUFFER	NTL_TIMERS		Timer Buffer; Starts at 0x0046
NTL_BUFFER ⇒ NTL_SP	NTL_REGS		User Register Space
NTL_SP ⇒ 0x006F	NTL_STACK		SALad Return Stack
0x0074	TOKEN		Currently executing SALad instruction token
0x0075	NTL_STAT		SALad Status Register
	_DB_END	4	1=ALL-Link Database search reached end of database
	_DB_PASS	3	1=ALL-Link Database search successful
	_NTL_DZ	2	1=Divide by Zero
	_NTL_BO	1	1=Buffer Overrun

i1 Address	i1 Register and Bits		Description		
	_NTL_CY	0	1=Carry from Math and Test operations		
0x0076	NTL_CONTROL		SALad debugging control flags		
	_RD_STEP _RD_HALT	7 6	_RD_HALT	_RD_STEP	
			0	0	Normal execution
			0	1	Animation (Trace)
			1	0	Execution halted
			1	1	Single step requested
	_RD_BREAK	5	0=Range Check Mode, 1=Breakpoint Mode		
0x0142	I_Control		INSTEON result flags		
	_I_DebugRpt	6	1=Enable INSTEON Debug Report		
	_I_SendDirect	5	0=Send INSTEON from working buffer, 1=Send INSTEON direct		
	_I_Transmit	4	1=Request To Send INSTEON		
	_Repeat_On	1	1=Hops enabled		
0x0154	Control		General system control flags		
	_Reset	7	1=Request system reset		
	_Watchdog	6	1=Request watchdog reset		
	_PDI	2	1=Daughter card interrupt occurred and has been serviced		
	_NoEventRpt	1	1=Inhibit static event report		
	_TAP_LAST	0	Last state of push button		
0x0156	TAP_CNT		Counts multiple *SET Button* taps		
0x0157	Tick		Incremented from 0⇒120 every second		
0x0158	RTC_TIME_H		Time since midnight in minutes (MSB, 0-1439)		
0x0159	RTC_TIME_L		Time since midnight in minutes (LSB, 0-1439)		
0x015A	RTC_YEAR		Year (0-99)		
0x015B	RTC_MON		Month (1-12)		
0x015C	RTC_DAY		Day (1-31, month specific)		
0x015D	RTC_DOW		Day-of-Week bitmap (0SSFTWTM)		
0x015E	RTC_HOUR		Hour (0-23)		
0x015F	RTC_MIN		Minute (0-59)		
0x0160	RTC_SEC		Second (0-59)		
0x0164	X10_RX		X10 Receive Buffer		

i1 Address	i1 Register and Bits		Description
0x0165	X10_TX		X10 Transmit Buffer
0x0166	X10_FLAGS		X10 Flags
	_X10_RTS	7	1=Request To Send
	_X10_TXEX	6	1=Start extended transmit after current command (for internal use)
	_X10_EXTENDED	5	1=Extended transfer in progress (Tx or Rx)
	_X10_COMBUF	4	1=Command, 0=Address (for internal use)
	_X10_TXCOMMAND	3	1=Command, 0=Address for transmit
	_X10_RXCOMMAND	2	1=Command, 0=Address for receive
	_X10_VALID	1	1=X10 receive valid
	_X10_ENABLED	0	1=X10 active (for internal use)
0x0168	LED_MODE		Bitmap defines flashing pattern for LED 1=On, 0=Off
0x0169	LED_TMR		Duration of LED flashing in seconds
0x016A	LED_DLY		Period between each flash. Defaults to 5, which is 1/8 second per bit in LED_MODE.
0x016B	RS_CONTROL		Control flags for serial command interpreter
	_RS_ComReset	7	These are used for serial command time limit. This limits how long a command remains active in SALad. This prevents the serial engine from locking you out if SALad receives a corrupt command (non-native serial command). Default time limit is two seconds. To disable, clear bits 5⇒7.
	_RS_ComLimit2	6	
	_RS_ComLimit1	5	
	_RS_AppLock	3	1=Enable overwriting of SALad code from 0x0200 to end of SALad App given in 0x0216 and 0x0217
	_RS_ComDisable	2	1=Core command processing disabled
	_RS_ComActive	1	1=Command active for SALad processing (Non-native serial command)
	_RS_02	0	1=0x02 received for command start
0x016F	EventMask		Mask to enable or disable events
	_EM_BtnTap	7	1=enabled
	_EM_BtnHold	6	1=enabled
	_EM_BtnRel	5	1=enabled
	_EM_TickTimer	4	1=enabled
	_EM_Alarm	3	1=enabled
	_EM_Midnight	2	1=enabled
	_EM_2AM	1	1=enabled

i1 Address	i1 Register and Bits		Description
	_EM_RX	0	1=enabled
0x017D	I2C_ADDR		Address of I^2C device; bit 0 controls 0x8000 ⇒ 0xFFFF of flat model, 1=hi region, 0=low region
0x01A0	DB_FLAGS		Database search mode bitmap
0x01A1	DB_0		Database ID_H; (INSTEON construction buffer) From Address_H; ignored for INSTEON message construction
0x01A2	DB_1		Database ID_M; (INSTEON construction buffer) From Address_M; ignored for INSTEON message construction
0x01A3	DB_2		Database ID_L; (INSTEON construction buffer) From Address_L; ignored for INSTEON message construction
0x01A4	DB_3		Database Command 1; (INSTEON construction buffer) To Address_H
0x01A5	DB_4		Database Command 2; (INSTEON construction buffer) To Address_M
0x01A6	DB_5		Database Group Number; (INSTEON construction buffer) To Address_L
0x01A7	DB_6		Database State; (INSTEON construction buffer) Message Flags
0x01A8	DB_7		Message Command 1; (INSTEON construction buffer) Command 1
0x01A9	DB_8		Message Command 2; (INSTEON construction buffer) Command 2
0x01AA	DB_9		
0x01AB	DB_A		
0x01AC	RxFrom0		Receive "From" address high byte
0x01AD	RxFrom1		Receive "From" address middle byte
0x01AE	RxFrom2		Receive "From" address low byte
0x01AF	RxTo0		Receive "To" address high byte
0x01B0	RxTo1		Receive "To" address middle byte
0x01B1	RxTo2		Receive "To" address low byte
0x01B2	RxExtRpt		Receive Control Flags
	_RxBroadcastBit	7	Broadcast Message
	_RxGroup	6	ALL-Link Message

i1 Address	i1 Register and Bits		Description
	_RxAckBit	5	Acknowledge Message
	_RxExtMsgBit	4	Extended Message
0x01B3	RxCmd1		Command byte 1
0x01B4	RxCmd2		Command byte 2
0x01B5	RxExtDataD		Standard message CRC or Extended message Data D
0x01B6	RxExtDataC		Extended message Data C
0x01B7	RxExtDataB		Extended message Data B
0x01B8	RxExtDataA		Extended message Data A
0x01B9	RxExtData9		Extended message Data 9
0x01BA	RxExtData8		Extended message Data 8
0x01BB	RxExtData7		Extended message Data 7
0x01BC	RxExtData6		Extended message Data 6
0x01BD	RxExtData5		Extended message Data 5
0x01BE	RxExtData4		Extended message Data 4
0x01BF	RxExtData3		Extended message Data 3
0x01C0	RxExtData2		Extended message Data 2
0x01C1	RxExtData1		Extended message Data 1
0x01C2	RxExtData0		Extended message Data 0
0x01C3	RxExtCrc		Extended message CRC
0x01C4	TxFrom0		Transmit “From” address high byte
0x01C5	TxFrom1		Transmit “From” address middle byte
0x01C6	TxFrom2		Transmit “From” address low byte
0x01C7	TxTo0		Transmit “To” address high byte
0x01C8	TxTo1		Transmit “To” address middle byte
0x01C9	TxTo2		Transmit “To” address low byte
0x01CA	TxExtRpt		Transmit Control Flags
	_TxBroadcastBit	7	Broadcast
	_TxGroup	6	ALL-Link
	_TxAckBit	5	Acknowledge
	_TxExtMsgBit	4	Extended
0x01CB	TxCmd1		Command byte 1
0x01CC	TxCmd2		Command byte 2

i1 Address	i1 Register and Bits		Description
0x01CD	TxExtDataD		Standard message CRC or Extended message Data D
0x01CE	TxExtDataC		Extended message Data C
0x01CF	TxExtDataB		Extended message Data B
0x01D0	TxExtDataA		Extended message Data A
0x01D1	TxExtData9		Extended message Data 9
0x01D2	TxExtData8		Extended message Data 8
0x01D3	TxExtData7		Extended message Data 7
0x01D4	TxExtData6		Extended message Data 6
0x01D5	TxExtData5		Extended message Data 5
0x01D6	TxExtData4		Extended message Data 4
0x01D7	TxExtData3		Extended message Data 3
0x01D8	TxExtData2		Extended message Data 2
0x01D9	TxExtData1		Extended message Data 1
0x01DA	TxExtData0		Extended message Data 0
0x01DB	TxExtCrc		Extended message CRC
0x01DD	RS_ERROR		RS232 Error register
	_RX_Empty	7	Receive buffer empty
	_Rx_Full	6	Receive buffer full
	_RX_OF	5	Receive buffer overflow
	_RX_Busy	4	Receive busy
	_TX_Empty	3	Transmit buffer empty
	_TX_Full	2	Transmit buffer full
	_TX_OF	1	Transmit buffer overflow
	_TX_Busy	0	Transmit busy
0x01E8 ⇒ 0x01EF	RX_Buffer		RS232 receive buffer
0x01DF	RX_PTR		Points to next open slot in serial receive buffer, if it contains 0xE8, the buffer is empty
0x0200	VALID		= 'P' if EEPROM is valid; 0x0200 is beginning of microcontroller internal EEPROM
0x0201	ID_H		High byte of ID
0x0202	ID_M		Middle byte of ID
0x0203	ID_L		Low byte of ID
0x0204	DEV_TYPE		Device Category

i1 Address	i1 Register and Bits	Description
0x0205	SUB_TYPE	Device Subcategory
0x0206	REV	Firmware Revision (MSN=Release, LSN=Ver)
0x0207	MEM_SIZE	Mask for installed external memory: 00000000=none 00001111=4K 01111111=32K 11111111=64K
0x0210 ⇒ 0x0211	APP_ADDR_TEST	Address of range of application for verification test
0x0212 ⇒ 0x0213	APP_LEN_TEST	Length of range of application for verification test
0x0214 ⇒ 0x0215	APP_CHECK_TEST	Two's complement checksum of range of application for verification test
0x0216 ⇒ 0x0217	APP_END	Top of currently loaded SALad application. EEPROM is write-protected from 0x0200 to the address contained here. Set 0x16B bit 7 to enable over-writing.
0x0230	TIMER_EVENT	Entry point of SALad application for Timer Events
0x0237	STATIC_EVENT	Entry point of SALad application for Static Events
0x0230 ⇒ 0x02FF	SALad application code for fast execution; Microcontroller Internal EEPROM	
0x0300 ⇒ 0x7FFF	Slow SALad application code; External serial EEPROM	
0x8000 ⇒ 0xFFFF	If I2C_Addr bit 0 = **0**, then 0x8000⇒0xFFFF is **low** region (0x0000⇒0x7FFF) of memory in I2C device specified by upper 7 bits of I2C_Addr. If I2C_Addr bit 0 = **1**, then 0x8000⇒0xFFFF is **high** region (0x8000⇒0xFFFF) of memory in I2C device specified by upper 7 bits of I2C_Addr.	

IBIOS Events

IBIOS events are all *Static Events*. If a SALad application is present, all of these Static Events are sent to a SALad event handler, like the one in the *SALad coreApp Program* 272, which comes pre-installed in *The SmartLabs PowerLinc Controller* 28. In fact, some of these events, such as receiving INSTEON or X10 messages, *require* SALad handling in order to guarantee realtime processing. *Timer Events* also occur, but these must be handled by a SALad application, so they are documented elsewhere (see *SALad Timers* 273).

Whenever an IBIOS Event occurs (assuming you have not disabled event reporting), IBIOS will notify your PC by sending an *Event Report (0x45)* IBIOS Serial Command. See the *IBIOS Serial Command Summary Table* 197 for more information about event reporting.

You can force an event to fire under program control by sending a *Simulated Event* IBIOS Command to the PLC. See the *IBIOS Serial Command Summary Table* 197 below for more information, and the *IBIOS Simulated Event* 209 section for an example.

Eight events (**0x0A** through **0x11**) can be prevented from occurring by clearing individual bits in the *EventMask* register at 0x016F (see *Flat Memory Map* 170). Initialization code sets *EventMask* to 0xFF at power up, so all events are enabled by default. Bit 7 (the MSb) of *EventMask* controls event **0x0A**, down through bit 0 (the LSb), which corresponds to event **0x11**.

IBIOS Event Summary Table

This table lists all currently defined Static Event handles.

Handle
Gives the number used by IBIOS to report the Event. **coreApp** means that the Event is only fired by the *SALad coreApp Program* 272, revision 12 (June, 2006) or later, or an equivalent SALad application.

Name
Gives the name of the Event as used in software.

Note
See the item with the same number in *IBIOS Event Details* 187 for more information.

Description
Briefly describes what happened to fire the event.

IBIOS Static Events			
Handle	**Name**	**Note**	**Description**
0x00	EVNT_INIT	1	SALad initialization code started (automatic at power-up).
0x01	EVNT_IRX_MYMSG	2	Received the first message in a hop sequence addressed to me.
0x02	EVNT_IRX_MSG	2	Received the first message in a hop sequence not addressed to me.

IBIOS Static Events			
Handle	**Name**	**Note**	**Description**
0x03	EVNT_IRX_PKT	2	Received a duplicate message in a hop sequence whether or not addressed to me (*may* occur after an **0x01** or **0x02** event).
0x04	EVNT_ITX_ACK	3	Received expected Acknowledgement message after Direct message sent.
0x05	EVNT_ITX_NACK	3	Did not receive expected Acknowledgement message after Direct message sent using 5 retries.
0x06	EVNT_IRX_BADID	4	Received a message with an unknown *To Address*. The message was censored by replacing its contents with 0xFFs, except for the *From Address* and *To Address* LSBs).
0x07	EVNT_IRX_ENROLL	5	Received an INSTEON message containing an enrollment-specific INSTEON Command. The INSTEON message was received *after* a SALad *Enroll* instruction was executed while the *SET Button* was being held down, and *before* a four-minute timeout expired. The received message's contents are in plaintext (i.e. not censored by masking with 0xFFs).
0x08	EVNT_XRX_MSG	6	Received an X10 byte.
0x09	EVNT_XRX_XMSG	6	Received an X10 Extended Message byte.
0x0A	EVNT_BTN_TAP	7	The *SET Button* was tapped for the first time.
0x0B	EVNT_BTN_HOLD	7	The *SET Button* is being held down.
0x0C	EVNT_BTN_REL	7	The *SET Button* is no longer being tapped or held down. The number of taps is in the *TAP_CNT* register at 0x0156.
0x0D	EVNT_TICK	8	Tick counter has expired (NTL_TICK)
0x0E	EVNT_ALARM	8	Hours and minutes equals current time
0x0F	EVNT_MIDNIGHT	8	Event occurs every midnight
0x10	EVNT_2AM	8	Event occurs every 2:00 am
0x11	EVNT_RX	9	Received a serial byte for SALad processing
0x12	EVNT_SRX_COM	9	Received an unknown IBIOS Serial Command
0x13	EVNT_DAUGHTER	10	Received interrupt from daughter card
0x14	EVNT_LOAD_ON	11	Load turned on
0x15	EVNT_LOAD_OFF	11	Load turned off
0x32 **coreApp**	EVNT_INIT_DB_CLR	12	Reinitialize RAM and clear the ALL-Link Database
0x33 **coreApp**	EVNT_INIT_DB_NO_CLR	12	Reinitialize RAM but do not clear the ALL-Link Database
0x41 **coreApp**	EVNT_LINK	13	Enter ALL-Linking mode for a single device
0x42	EVNT_MULTI_LINK	14	Enter ALL-Linking mode for multiple devices

IBIOS Static Events			
Handle	**Name**	**Note**	**Description**
coreApp			
0x43 **coreApp**	EVNT_UNLINK	15	Enter Unlinking mode for a single device
0x44 **coreApp**	EVNT_MULTI_UNLINK	16	Enter Unlinking mode for multiple devices
0x45 **coreApp**	EVNT_END_LINK	17	End ALL-Linking mode
0x46 **coreApp**	EVNT_CONTINUE_LINK	18	Continue ALL-Linking mode for 4 additional minutes

IBIOS Event Details

The numbers below refer to the **Note** column in the above *IBIOS Event Summary Table*185. **[coreApp]** means that the Event is only fired by the *SALad coreApp Program*272, revision 12 (June, 2006) or later, or an equivalent SALad application.

1. **EVNT_INIT (0x00)**

   When power is first applied, IBIOS runs its initialization code, then it checks for the existence of a valid SALad application program. If there is one, IBIOS fires this event and then starts SALad with this event in the event queue.

   You can force a power-on reset in software by setting bit 7 (*_Reset*) of the *Control* register at 0x0154 to one (see *Flat Memory Map*170).

2. **EVNT_IRX_MYMSG (0x01), EVNT_IRX_MSG (0x02), EVNT_IRX_PKT (0x03)**

   When IBIOS first receives a new INSTEON message that has not been seen before, the message may be arriving on its first hop, or it may be arriving on a subsequent hop, depending on the INSTEON environment. If the *To Address* of this new message matches the 3-byte IBIOS ID burned in at the factory, then the new message is *to me*, and an **EVNT_IRX_MYMSG (0x01)** event will fire. If the new message is *not to me*, then an **EVNT_IRX_MSG (0x02)** event will fire instead. If the new message was received before its last hop, then the same message may be received again on subsequent hops. Whenever this happens an **EVNT_IRX_PKT (0x03)** event will fire whether the duplicate message is *to me* or *not to me*.

   After any of these events fire, IBIOS will use a SALad application like the *SALad coreApp Program*272 to send an *INSTEON Received (0x4F)* IBIOS Serial Command (see *IBIOS Serial Command Summary Table*197).

3. **EVNT_IRX_ACK (0x04), EVNT_IRX_NACK (0x05)**

   After IBIOS sends a Direct INSTEON message, it expects the addressee to respond with an INSTEON Acknowledgement message. If IBIOS receives the Acknowledgement message as expected, it fires an **EVNT_IRX_ACK (0x04)** event. On the other hand, if the expected Acknowledgement message is not received, IBIOS will automatically retry sending the Direct message again. If after five retries the addressee still does not respond with an Acknowledgement message, IBIOS will fire an **EVNT_IRX_NACK (0x05)** event. One or the other (but not both) of these events is guaranteed to fire after sending a Direct message, although the **EVNT_IRX_NACK (0x05)** event may not fire for a long time due to the retries.

After either of these events fires, IBIOS will use a SALad application like the *SALad coreApp Program*272 to send an *INSTEON Received (0x4F)* IBIOS Serial Command (see *IBIOS Serial Command Summary Table*197).

4. **EVNT_IRX_BADID (0x06)**

   If the *To Address* of an incoming INSTEON message does not match the 3-byte IBIOS ID burned in at the factory (i.e. the message is *not to me*), and the *To Address* does not match any of the IDs in IBIOS's *INSTEON ALL-Link Database*101, then for security reasons, the message is censored. A censored INSTEON message will have all of its data bytes replaced with 0xFFs, except for the low bytes of the *From Address* and the *To Address*. See *Masking Non-linked Network Traffic*112 for more information on *INSTEON Security*112.

   After this event fires, IBIOS will use a SALad application like the *SALad coreApp Program*272 to send an *INSTEON Received (0x4F)* IBIOS Serial Command (see *IBIOS Serial Command Summary Table*197).

5. **EVNT_IRX_ENROLL (0x07)**

   This event supports *INSTEON Device ALL-Linking*93, also known as *enrollment*, that is being handled by a suitable SALad application. The event may only fire **after** a SALad *Enroll* instruction (see *SALad Instruction Summary Table*281) executes during the time that the *SET Button* is held down, and **before** a four-minute timer has expired. If during that time IBIOS sends or receives an INSTEON Broadcast message containing an INSTEON *SET Button Pressed Responder (0x01)* or *SET Button Pressed Controller (0x02)* Command or an INSTEON Direct message containing an INSTEON *Assign to ALL-Link Group (0x01)* or *Delete from ALL-Link Group (0x02)* Command (see *INSTEON Command Set Tables*124), the event will fire. The received message that triggered the event will *not* be censored by replacing its contents with 0xFFs, so that the SALad program can enroll the INSTEON device that sent the message in the *INSTEON ALL-Link Database*101.

   Requiring that the *SET Button* be pushed enforces *INSTEON Security*112 by requiring *Physical Possession of Devices*112.

   After this event fires, IBIOS will send an *INSTEON Received (0x4F)* IBIOS Serial Command (see *IBIOS Serial Command Summary Table*197).

6. **EVNT_XRX_MSG (0x08), EVNT_XRX_XMSG (0x09)**

   These events occur when IBIOS receives an X10 byte over the powerline. When IBIOS receives a new X10 byte, it fires an **EVNT_XRX_MSG (0x08)** event. If IBIOS determines that subsequent received X10 bytes are part of an Extended X10 message, then it will fire an **EVNT_XRX_XMSG (0x09)** event for those bytes. After IBIOS detects the end of the Extended X10 message (three 0x00 bytes in succession), or else if a timeout expires, IBIOS will revert to firing an **EVNT_XRX_MSG (0x08)** event for the next X10 byte it receives.

After either of these events occurs, IBIOS will use a SALad application like the *SALad coreApp Program*272 to send an *X10 Byte Received (0x4A)* IBIOS Serial Command (see *IBIOS Serial Command Summary Table*197).

7. **EVNT_BTN_TAP (0x0A), EVNT_BTN_HOLD (0x0B), EVNT_BTN_REL (0x0C)**

These events fire when the *SET Button* is pushed in various ways. A *Button Tap* occurs when the user pushes the *SET Button* and then lets up less than 350 milliseconds (350 ms) later. A *Button Hold* occurs when the user pushes the *SET Button* and then lets up more than 350 ms later.

The first time IBIOS detects a Button Tap, it fires an **EVNT_BTN_TAP (0x0A)** event. If there are more Button Taps following the first one, with less than 350 ms between each one, then IBIOS does not fire an event, but it does count the number of Button Taps. When more than 350 ms elapses after a Button Tap, IBIOS fires an **EVNT_BTN_REL (0x0C)** event to indicate the *SET Button* has been released. At that time, you can inspect the *TAP_CNT* register at 0x0156 (see *Flat Memory Map*170) to see how many Button Taps occurred before the release.

Whenever IBIOS detects a Button Hold it fires an **EVNT_BTN_HOLD (0x0B)** event, then when it detects that the button has been released it fires an **EVNT_BTN_REL (0x0C)** event.

Note that a Button Hold can follow some number of Button Taps, in which case events **EVNT_BTN_TAP (0x0A)**, **EVNT_BTN_HOLD (0x0B)**, and **EVNT_BTN_REL (0x0C)** would occur in that order. Inspect *TAP_CNT* after the **EVNT_BTN_REL (0x0C)** event to see how many Button Taps there were.

8. **EVNT_TICK (0x0D), EVNT_ALARM (0x0E), EVNT_MIDNIGHT (0x0F), EVNT_2AM (0x10)**

These events depend on the *IBIOS Software Realtime Clock/Calendar*212 (RTC), which counts powerline zero crossings (120 per second) for timing. The Software RTC must be set by a SALad program or by IBIOS Serial Commands upon power up for **EVNT_ALARM (0x0E), EVNT_MIDNIGHT (0x0F), EVNT_2AM (0x10)** to work properly.

You can use **EVNT_TICK (0x0D)** to measure short time intervals, ranging from 8.333 milliseconds (ms) up to 2.125 seconds. Each tick is one powerline zero crossing, which is 1/120 second, or 8.333 ms. To cause this event to fire, load a value from 1 to 255 into the *NTL_TICK* register at 0x0042 (see *Flat Memory Map*170). After that number of ticks, IBIOS will fire this event one time only. You can disable this event at any time by loading 0x00 into *NTL_TICK*.

Registers *RTC _TIME_H* and *RTC _TIME_L* at 0x0158 and 0x0159 contain a 16-bit value ranging from 0 to 1439 that counts the number of minutes that has elapsed since midnight. You can cause an **EVNT_ALARM (0x0E)** event to fire by loading a valid minutes-from-midnight value into the *NTL_TIME_H* and *NTL _TIME_H* registers at 0x0040 and 0x0041. When the value in *RTC_TIME_H,L* matches your value in *NTL_TIME_H,L*, IBIOS will fire **EVNT_ALARM (0x0E)**. The value you loaded into *NTL_TIME_H,L* is not altered when the event fires, so the event will fire every day. However, if you load an invalid value (greater than 1439) into *NTL_TIME_H,L*, then the event will never fire.

The **EVNT_MIDNIGHT (0x0F)** and **EVNT_2AM (0x10)** events fire at midnight and at 2:00 am, as you would expect.

9. **EVNT_RX (0x11), EVNT_SRX_COM (0x12)**

   These events allow the number of *IBIOS Serial Commands*196 to be extended. All IBIOS Serial Commands begin with 0x02, followed by the number of the Command. If IBIOS receives a Serial Command Number outside the range 0x40 through 0x48 (see *IBIOS Serial Command Summary Table*197), it will fire the **EVNT_SRX_COM (0x12)** event, then start a two-second timer. Thereafter, every time IBIOS receives a serial byte, it will fire the **EVNT_RX_COM (0x11)** event and restart the two-second timer. If the two-second timer expires, IBIOS will send a serial NAK (ASCII 0x15) and stop firing **EVNT_RX_COM (0x11)** events.

   When you are finished parsing the incoming IBIOS Serial Command, clear the two-second timer yourself by clearing bits 5 and 6, *_RSComLimit1* and *_RSComLimit*, in the *RS_CONTROL* register at 0x016B (see *Flat Memory Map*170). If you want more time, you can get another two seconds by setting the same two bits to one.

10. **EVNT_DAUGHTER (0x13)**

    IBIOS fires this event when it detects that port pin RB6 on the microprocessor went low. Normally this is caused by an attached daughter card requesting an interrupt.

11. **EVNT_LOAD_ON (0x14), EVNT_LOAD_OFF (0x15)**

    These events are for monitoring electrical loads connected to the INSTEON device. They will fire in response to the state of current-sensing hardware, and so are implementation-specific. Contact the INSTEON Developer's Forum or email sdk@insteon.net for more information.

12. **EVNT_INIT_DB_CLR (0x32), EVNT_ INIT_NO_DB_CLR (0x33) [coreApp]**

    These events can *only* be fired under program control, by sending the *Simulated Event* IBIOS Command to the PLC. (Send 0x02 **0x47 0x32** 0x00, or 0x02 **0x47 0x33** 0x00 respectively.) See the *IBIOS Serial Command Summary Table*197 below for more information.

    The action of these events is similar to **EVNT_INIT (0x00)**, except the realtime clock is not reset.

13. **EVNT_LINK (0x41) [coreApp]**

    This event fires when the user pushes the *SET Button* for 10 seconds to put the PLC into ALL-Linking mode.

14. **EVNT_MULTI_LINK (0x42) [coreApp]**

This event fires when the user pushes the *SET Button* for 10 seconds to put the PLC into ALL-Linking mode, and then taps the *SET Button* to enter multilinking mode. Multilinking allows the user to ALL-Link more than one device without having to push the *SET Button* again for 10 seconds.

15. **EVNT_UNLINK (0x43) [coreApp]**

    This event fires when the user pushes the *SET Button* for 10 seconds to put the PLC into ALL-Linking mode, and then pushes the *SET Button* a second time for another 10 seconds to enter unlinking mode.

16. **EVNT_MULTI_UNLINK (0x44) [coreApp]**

    This fires when the user pushes the *SET Button* for 10 seconds to put the PLC into ALL-Linking mode, pushes the *SET Button* a second time for another 10 seconds to enter unlinking mode, and then taps the *SET Button* to enter multi-unlinking mode. Multi-unlinking allows the user to unlink more than one device without having to push the *SET Button* again for 10 seconds.

17. **EVNT_END_LINK (0x45) [coreApp]**

    This event fires when the user taps the *SET Button* to end ALL-Link mode, or else when the four-minute ALL-Link mode timer automatically terminates ALL-Link mode.

18. **EVNT_CONTINUE_LINK (0x46) [coreApp]**

    This event fires when the user ALL-Links to an INSTEON device while in multilinking mode, or else when the user unlinks a device while in multi-unlinking mode.

IBIOS Serial Communication Protocol and Settings

In This Section

IBIOS Serial Communication Protocol

All IBIOS Serial Commands start with ASCII `0x02` (STX, Start-of-Text) followed by the Serial Command Number (see *IBIOS Serial Commands* 196). What data follows the Command depends on the Command syntax (see *IBIOS Serial Command Summary Table* 197).

IBIOS will respond with an echo of the `0x02` and Command Number followed by any data that the Command returns.

If IBIOS is responding to a Serial Command that it received, the last byte it sends will be ASCII 0x06 (ACK, Acknowledge).

(**S:** and **R:** denote serial data you **Send to** or **Receive from** IBIOS, respectively.)

S:	0x02 <Command Number> <parameters>
R:	0x02 <Command Number> <any returned data> 0x06 (ACK)

If IBIOS is not ready, it will respond with an echo of the `0x02` and the Serial Command Number followed by ASCII 0x15 (NAK, Negative Acknowledge).

S:	0x02 <Command Number> <parameters>
R:	0x15 (NAK)

If you receive 0x15 (NAK), resend your Serial Command.

IBIOS RS232 Port Settings

To communicate to an RS232 PLC, set your PC's COM port as follows:

Setting	**Value**
Baud Rate	4800
Data Bits	8
Parity	N
Stop Bits	1
Hardware Flow Control	None
Software Flow Control	None

IBIOS USB Serial Interface

The interface to a USB PLC is a simple USB wrapper around the *IBIOS Serial Communication Protocol*193, implemented using the Human Interface Device (HID) interface. See, for example, http://www.lvr.com/hidpage.htm for more information on using HID to implement a USB interface.

SmartLabs' VendorID and ProductID are listed at http://www.linux-usb.org/usb.ids. The ProductID for the USB PowerLinc™ V2 Controller is 0x0004.

When communicating to a USB PLC, send the same Commands as given in *IBIOS Serial Commands*196, except use 8-byte USB packets.

The PLC will set the most significant bit of the *Count* byte to indicate *Clear-to-Send*, i.e. that the PLC is ready to receive more data.

For example, to send the IBIOS Serial Command `0x48` (Get PLC Version), send the following 8-byte packet (data bytes are **bold**):

HID USB 8-byte Packet							
Count	**Data**						
0x02	**0x02**	**0x48**	0x00	0x00	0x00	0x00	0x00

The PLC will reply with data similar to the following (data bytes are **bold**; 0x80 in the count indicates that the PLC is ready to receive more data):

HID USB 8-byte Packet							
Count	**Data**						
0x80	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x01	**0x02**	0x00	0x00	0x00	0x00	0x00	0x00
0x03	**0x48**	**0xff**	**0xff**	0x00	0x00	0x00	0x00
0x03	**0xff**	**0x04**	**0x00**	0x00	0x00	0x00	0x00
0x02	**0x23**	**0x06**	0x00	0x00	0x00	0x00	0x00

IBIOS Serial Commands

The IBIOS Serial Command set is the basic interface between a computing device such as a PC or dedicated home controller and a serially connected INSTEON Bridge device such as *The SmartLabs PowerLinc Controller* 28 (PLC). For example, a PC connected to a PLC could use IBIOS Serial Commands to send and receive INSTEON or X10 messages directly, or it could download and debug a SALad program that runs on the PLC.

IBIOS Serial Commands also allow indirect access, via INSTEON messages, to other INSTEON devices on the network. For instance, you could upgrade the capabilities of SALad-enabled INSTEON devices by remotely installing and debugging new SALad applications in them.

Some of the IBIOS Serial Commands require a SALad application such as the *SALad coreApp Program* 272 to be running in order to ensure realtime execution.

In This Section

IBIOS Serial Command Table

IBIOS Serial Command Parameters

This is what the common parameters shown in the *Format* column of the *IBIOS Serial Command Summary Table*197 mean. Parameters not listed here should be understood from the context.

MSB means Most-Significant Byte, *LSB* means Least-Significant Byte, *MSb* means Most-Significant bit.

Parameter	Description
Address High (Low)	MSB (LSB) of a 16-bit address
Length High (Low)	MSB (LSB) of a 16-bit length of a data block
Checksum High (Low)	MSB (LSB) of a 16-bit value calculated by summing all the bytes in a data block specified in an IBIOS Serial Command, then taking the two's complement
Data	Block of data bytes
Event Handle	8-bit number indicating the event that fired (see *IBIOS Events*185)
Timer Value	8-bit time value: 1 to 127 seconds, if the MSb (bit 7) is 0 (clear) 1 to 127 minutes, if the MSb (bit 7) is 1 (set)

IBIOS Serial Command Summary Table

This table lists all of the IBIOS Serial Commands supported by the PLC.

Code

Gives the hexadecimal number of the IBIOS Serial Command. **SALad** means that the Command is only applicable if a suitable SALad application, such as the *SALad coreApp Program*[272], is running.

Command

Gives the name of the IBIOS Serial Command.

Note

See the item with the same number in *IBIOS Serial Command Details*[198] for more information.

Format

Gives the syntax of the IBIOS Serial Command, including any parameters.

S: and **R:** denote serial data you **Send to** or **Receive from** IBIOS, respectively. See *IBIOS Serial Communication Protocol*[193] for more information.

All IBIOS Serial Commands start with ASCII 0x02 (STX, Start-of-Text) followed by the Serial Command Number. See *IBIOS Serial Command Parameters*[196], above, for the meaning of the Command parameters.

All fields in this table contain only one byte, except for those with '…' (e.g. <Data…>, or <Message…>), which contain a variable number of bytes.

IBIOS Serial Commands			
Code	**Command**	**Note**	**Format**
0x40	**Download** (IBIOS receives data from you)	1	**S:** 0x02 **0x40** <Address High> <Address Low> <Length High> <Length Low> <Checksum High> <Checksum Low> <Data…> **R:** 0x02 **0x40** <Address High> <Address Low> <Length High> <Length Low> <0x06 (ACK) | 0x15 (NAK)>
0x41 **SALad**	**Fixed-length Message**	2	**R:** 0x02 **0x41** <Length> <Message…> **NOTE:** <Message…> can contain unrestricted data, such as embedded INSTEON or X10 messages.
0x42	**Upload** (IBIOS sends data to you)	1	**S:** 0x02 **0x42** <Address High> <Address Low> <Length High> <Length Low> **R:** 0x02 **0x42** <Address High> <Address Low> <Length High> <Length Low> <Data…> <Checksum High> <Checksum Low> 0x06
0x43 **SALad**	**Variable-length Text Message**	2	**R:** 0x02 **0x43** <Message…> 0x03 (ASCII ETX, End-of-Text) **NOTE:** <Message…> must not contain 0x03 (ETX, End-of-Text).
0x44	**Get Checksum**	3	**S:** 0x02 **0x44** <Address High> <Address Low> <Length High> <Length Low> **R:** 0x02 **0x44** <Address High> <Address Low> <Length High> <Length Low> <Checksum High> <Checksum Low> 0x06
0x45	**Event Report**	4	**R:** 0x02 **0x45** <Event Handle>

IBIOS Serial Commands			
Code	**Command**	**Note**	**Format**
0x46	**Mask**	5	**S:** 0x02 **0x46** <Address High> <Address Low> <OR mask> <AND mask> **R:** 0x02 **0x46** <Address High> <Address Low> <OR mask> <AND mask> 0x06
0x47	**Simulated Event**	6	**S:** 0x02 **0x47** <Event Handle> <Timer Value> **R:** 0x02 **0x47** <Event Handle> <Timer Value> 0x06
0x48	**Get Version**	7	**S:** 0x02 **0x48** **R:** 0x02 **0x48** <INSTEON Address High> <INSTEON Address Middle> <INSTEON Address Low> <Device Type High> <Device Type Low> <Firmware Revision> 0x06
0x49 **SALad**	**Debug Report**	8	**R:** 0x02 **0x49** <Next SALad instruction to execute Address High> <Next SALad instruction to execute Address Low>
0x4A **SALad**	**X10 Byte Received**	9	**R:** 0x02 **0x4A** <0x00 (X10 Address) | 0x01 (X10 Command)> <X10 Byte>
0x4F **SALad**	**INSTEON Message Received**	10	**R:** 0x02 **0x4F** <Event Handle (0x01-0x07)> <INSTEON message (9 or 23 bytes)>

IBIOS Serial Command Details

The numbers below refer to the **Note** column in the above *IBIOS Serial Command Summary Table*[197].

1. **Download (0x40), Upload (0x42)**

   Use these Commands to write **Download (0x40)** or read **Upload (0x42)** IBIOS's memory. The *Flat Memory Map*[170] lists all of the memory locations that you can access, and defines what the contents are.

   You can neither read nor write the firmware in IBIOS's EPROM. The microprocessor's program counter (appearing at locations 0x0002, 0x0082, 0x0102, and 0x0182) is write-protected, as is the Enrollment Timer at 0x016D. No harm will occur if you attempt to read or write address locations where there is no memory, but the results will be indeterminate.

   See item 3, **Get Checksum (0x44)**, for IBIOS's method of calculating checksums (two's complement of a 16-bit sum). The checksum covers all the bytes in the <Address High>, <Address Low>, <Length High>, <Length Low>, and <Data…> fields only.

   After downloading, even if you receive 0x06 (ACK), you should immediately issue a **Get Checksum (0x44)** Serial Command to verify that IBIOS wrote the data to memory correctly.

   If you are setting or clearing individual flag bits in a register, use the **Mask**

(0x46) Command to avoid affecting flags you are not changing.

2. **Fixed-length Message (0x41), Variable-length Text Message (0x43)**

   These Serial Commands require a SALad application such as the *SALad coreApp Program*272 to be running. A SALad program can send up to 255 bytes of unrestricted (ASCII or binary) data to the host using a **Fixed-length Message (0x41)** Serial Command containing a length byte.

   To send simple text messages without a length restriction, a SALad program can use the **Variable-length Text Message (0x43)** Serial Command. This Command does not use a length byte. Instead, it uses an ASCII 0x03 (ETX, End-of-Text) byte to delimit the end of the ASCII text message, so the text message must not contain an embedded ETX character before the actual end of the message.

3. **Get Checksum (0x44)**

   IBIOS calculates checksums by summing up all of the bytes in the given range into a 16-bit register, then taking a two's complement of the 16-bit sum. You can take a two's complement by inverting all 16 bits and then incrementing by one, or else you can just subtract the 16-bit value from 0x0000.

4. **Event Report (0x45)**

   IBIOS sends this Serial Command whenever one of the *IBIOS Events*185 given in the *IBIOS Event Summary Table*185 fires.

   You can prevent IBIOS from sending Event Reports by setting bit 1, *_NoEventRpt*, in the *Control* register at 0x0154 to one (see *Flat Memory Map*170). IBIOS clears this flag to zero at power up, so Event Reports are enabled by default.

5. **Mask (0x46)**

   Use this Serial Command to set or clear individual flag bits in a register without affecting flags you are not changing.

   To **set** one or more bits in a register to **one**, set the corresponding bits in the OR Mask to one. Bits set to zero in the OR Mask will not change the corresponding bits in the register.

   To **clear** one or more bits in a register to **zero**, set the corresponding bits in the AND Mask to zero. Bits set to one in the AND Mask will not change the corresponding bits in the register.

6. **Simulated Event (0x47)**

   You can force IBIOS to fire one of the Static Events in the *IBIOS Event Summary Table*185 with this Serial Command by sending the desired <Event Handle> number with a <Timer Value> of zero. You cannot use this Serial Command to fire an **EVNT_INIT (0x00)** initialization event, but there is an alternate method to force a power-on reset (see *IBIOS Event Details*187, Note 1).

IBIOS will fire Timer Events, but unless you are running a SALad program with Timer Event handling code, nothing will happen. See *SALad Timers*273 for an explanation of SALad Timer events. To simulate a Timer Event, set <Event Handle> to the Timer Index number of the SALad handler for the timer, and set <Timer Value> to a non-zero value denoting how much time should elapse before the Timer Event fires. If the high bit of <Timer Value> is 0, then the time will be 1 to 127 seconds; if the high bit is 1, then the time will be 1 to 127 minutes.

7. **Get Version (0x48)**

   This Serial Command retrieves the 3-byte INSTEON ID number (see *Device Addresses*41), a 1-byte DevCat, a 1-byte SubCat, and a 1-byte reserved field formerly used for a Firmware Version (see *INSTEON Device Categories*83) that were burned in at the factory. This information is read-only.

8. **Debug Report (0x49)**

   You can remotely debug a SALad program with this Serial Command. This is the underlying mechanism used by the IDE debugger described in the *SALad Integrated Development Environment User's Guide*287.

   See the *IBIOS Remote Debugging*215 section for details on how IBIOS remote debugging works. When remote debugging is enabled, IBIOS will use this Serial Command to report the location of the next SALad instruction to be executed.

9. **X10 Byte Received (0x4A)**

   This Serial Command requires a SALad application such as the *SALad coreApp Program*272 to be running. After IBIOS fires an **EVNT_XRX_MSG (0x08)** or **EVNT_XRX_XMSG (0x09)** (see *IBIOS Event Details*187), The SALad app will report the received X10 byte by sending this Serial Command.

   The byte following the **0x4A** Command Number tells whether the received X10 byte is an X10 Address (the byte is 0x00) or X10 Command (the byte is 0x01). If you are receiving an X10 Extended Message, then this byte is irrelevant.

   See *IBIOS X10 Signaling*213 for more information.

10. **INSTEON Message Received (0x4F)**

    This Serial Command requires a SALad application such as the *SALad coreApp Program*272 to be running. After IBIOS fires any of the events **0x01** through **0x07** (see *IBIOS Event Details*187), The SALad app will report the INSTEON message received by sending this Serial Command.

    If the IBIOS event is **0x05** (EVNT_ITX_NACK), then IBIOS did not receive an expected INSTEON Acknowledgement message after five retries. In that case, this IBIOS Command will not contain an <INSTEON message (9 or 23 bytes)> field—instead, you will only receive the three bytes 0x02 0x4F 0x05.

    To determine if the INSTEON message's length is 9 bytes (Standard) or 23 bytes (Extended), inspect the message's *Extended Message Flag*43.

IBIOS Serial Command Examples

This section contains examples showing how to use various IBIOS Serial Commands described in the *IBIOS Serial Command Table* 196 above. The examples assume you are serially connected to *The SmartLabs PowerLinc Controller* 28 (PLC) running the default *SALad coreApp Program* 272.

In This Section

IBIOS Get Version

Summary

In this example we will use the **Get Version (0x48)** IBIOS Serial Command to get the PLC's 3-byte INSTEON ID number (see *Device Addresses*41), a 1-byte DevCat, a 1-byte SubCat, and a 1-byte reserved field formerly used for a Firmware Version (see *INSTEON Device Categories*83) that were burned in at the factory.

Procedure

1. Send the **Get Version (0x48)** IBIOS Serial Command from the *IBIOS Serial Command Summary Table*197:

   0x02 **0x48**.

2. The response should be:

   0x02 **0x48** <INSTEON Address (3 bytes)> <Device Type (2 bytes)> <Firmware Revision (1 byte)> 0x06.

IBIOS Read and Write Memory

Summary

In this example we will use the **Upload (0x42)** and **Download (0x40)** IBIOS Serial Commands to alter data stored in the PLC's serial EEPROM chip. See the *Flat Memory Map*170 for memory address locations. First we will read 4 bytes of data starting at the beginning of external EEPROM at address 0x0300, then we will write **0x55 0xAA 0x55 0xAA** to those locations, then read the data back out to observe our changes.

Procedure

1. Use the **Upload (0x42)** IBIOS Serial Command from the *IBIOS Serial Command Summary Table*197 to read 0x0004 bytes starting at 0x0300:

   `0x02` **`0x42`** `0x03 0x00 0x00 0x04.`

   You can check the response to see what the four bytes are. See the *IBIOS Serial Command Summary Table*197 for the response syntax.

2. Now use **Download (0x40)** to write **0x55 0xAA 0x55 0xAA** into those locations:

   `0x02` **`0x40`** `0x03 0x00 0x00 0x04 0xFD 0xFB` **`0x55 0xAA 0x55 0xAA`**.

   It is not mandatory that you include a valid checksum (here 0xFDFB), but it is strongly recommended, because you can then use a **Get Checksum (0x44)** IBIOS Serial Command to be sure the data was properly written, without having to read all of the data back.

   The response should be:

   `0x02` **`0x40`** `0x03 0x00 0x00 0x04 0x06.`

   Note that the response merely echoes the first 6 bytes of the received Command, followed by an ASCII 0x06 (ACK) indicating that the Command was properly received. The ACK does *not* indicate that the Command executed properly.

3. Now read out the four bytes at 0x0300 again as in Step 1:

   `0x02` **`0x42`** `0x03 0x00 0x00 0x04.`

   Check that the response contains the **0x55 0xAA 0x55 0xAA** data. For further validation, you can also verify that the checksum in the response matches a checksum that you compute over the received data.

IBIOS Get Checksum on Region of Memory

Summary

This example will demonstrate getting the checksum over the first 10 bytes of SALad application code. The checksum is computed as the two's complement of a 16-bit sum of the bytes. You can take a two's complement by inverting all 16 bits in the sum and then incrementing by one, or else you can just subtract the 16-bit value from 0x0000.

For this example the beginning of the SALad application starts at 0x0230 and is 0x0a bytes long.

Procedure

1. Use the **Get Checksum (0x44)** IBIOS Serial Command from the *IBIOS Serial Command Summary Table*197 to get the checksum on the region starting at 0x0230 and extending 0x000a bytes:

   0x02 **0x44** 0x02 0x30 0x00 0xa0.

2. Retrieve the checksum from the return message, which should be:

   0x02 **0x44** 0x02 0x30 0x00 0xA0 <checksum MSB> <checksum LSB> 0x06.

IBIOS Send INSTEON

Summary

Before sending INSTEON messages you should familiarize yourself with Chapter 5 — INSTEON Messages38 and *Chapter 8 — INSTEON Command Set*114.

In this example we will send the INSTEON **ON** Command with **Level 0xFF** (full on) from a PLC to a LampLinc™ V2 Dimmer that has an INSTEON Address of 0x0002AC. We will first copy the INSTEON message to an area of PLC memory used as a message construction buffer. Then we will set the Request-to-Send INSTEON flag, causing the PLC to send the INSTEON message in its construction buffer to the LampLinc.

Note that for security reasons, IBIOS will always insert its own address (the 3-byte IBIOS ID burned in at the factory) in the *From Address* field no matter what you write to the construction buffer. See *Masking Non-linked Network Traffic*112 for more information on *INSTEON Security*112. Therefore, we do not need to put the *From Address* in the INSTEON message.

Procedure

1. Use the **Download (0x40)** IBIOS Serial Command from the *IBIOS Serial Command Summary Table*197 to load this 6-byte INSTEON message starting with the *To Address* field

   0x00 0x02 0xAC 0x0F 0x11 0xFF

   into the PLC's C*onstruction Buffer* starting at the *To Address* field at location 0x1A4 (see *Flat Memory Map*170). **NOTE:** Although the *Construction Buffer* starts at 0x1A1, the first three bytes are the *From Address*, which the INSTEON Engine automatically fills in during sends.

   The fully formed **Download (0x40)** IBIOS Serial Command, with the embedded INSTEON message in **bold**, is:

   0x02 0x40 0x01 0xA4 0x00 0x06 0xFD 0x88 **0x00 0x02 0xAC 0x0F 0x11 0xFF**.

   The returned serial message should be an echo of the first 6 bytes of the Serial Command followed by an ASCII 0x06 (ACK), like this:

   0x02 0x40 0x01 0xA4 0x00 0x06 0x06.

2. Now use the **Mask (0x46)** IBIOS Serial Command to set the *_I_Transmit* Request-to-Send INSTEON flag (bit 4) in the *I_Control* register at 0x142 (see *Flat Memory Map*170), by sending:

   0x02 **0x46** 0x01 0x42 0x10 0xFF.

   The returned serial message should be:

   0x02 **0x46** 0x01 0x42 0x10 0xFF 0x06.

IBIOS Receive INSTEON

Summary

Before receiving INSTEON messages you should familiarize yourself with Chapter 5 — INSTEON Messages$_{38}$ and *Chapter 8 — INSTEON Command Set*$_{114}$.

Here we assume you are using a *The SmartLabs PowerLinc Controller*$_{28}$ (PLC) running the default *SALad coreApp Program*$_{272}$.

When the PLC receives an INSTEON message, IBIOS fires an IBIOS Event that a SALad program handles. PLCs come from the factory with an open-source *SALad coreApp Program*$_{272}$ installed, and you can create your own custom applications by modifying coreApp.

When an INSTEON message arrives, coreApp's event handlers send an **INSTEON Received (0x4F)** IBIOS Serial Command containing the IBIOS Event number and the INSTEON message to your PC. Your PC can then deal with the **INSTEON Received (0x4F)** IBIOS Serial Command whenever it shows up in its serial buffer.

Procedure

1. When an INSTEON message is received, coreApp (and all applications built upon it) will send the message data to your PC using the following format:

   ```
   0x02 0x4F <Event Handle> <INSTEON message>
   ```

2. The *Event Handle* byte tells which of the *IBIOS Events*$_{185}$ (0x01 through 0x07) IBIOS fired to trigger the SALad coreApp program. See the *IBIOS Event Summary Table*$_{185}$ and *IBIOS Event Details*$_{187}$ for more information about what kinds of INSTEON message fire which events.

3. To determine if the length of the *INSTEON message* is 9 bytes (Standard) or 23 bytes (Extended), inspect the message's *Extended Message Flag*$_{43}$ (bit 4 of the message's 7^{th} byte). The *INSTEON Message Summary Table*$_{46}$ shows all possible INSTEON message types.

4. To determine the meaning of the INSTEON message, look at the *Command 1 and 2*$_{44}$ fields in the message. The *INSTEON Command Set Tables*$_{124}$ enumerate all of the possible INSTEON Commands in *Chapter 8 — INSTEON Command Set*$_{114}$.

IBIOS Send X10

Summary

In this example we will send X10 **A1 / AON** over the powerline using a PLC and IBIOS Serial Commands. First we will download the **A1** X10 address into the X10 transmit buffer. Then we will set the Request-to-Send X10 bit and clear the Command/Address bit of the X10 Flags register with a **Mask (0x46)** IBIOS Serial Command. Then we will download the **AON** X10 Command into the X10 transmit buffer, followed by setting both the Request-to-Send X10 and the Command/Address bits with another **Mask (0x46)** Command.

See *IBIOS X10 Signaling*213 for more information.

Procedure

1. Use the **Download (0x40)** IBIOS Serial Command from the *IBIOS Serial Command Summary Table*197 to load an X10 **A1** address (**0x66**) into the X10 transmit buffer *X10_TX* at 0x0165 (i1 Engine, shown here) or 0x01EE (i2 Engine) (see *Flat Memory Map*170) by sending:

   0x02 **0x40** 0x01 0x65 0x00 0x01 0xFF 0x33 **0x66**,

   then check for the ASCII 0x06 (ACK) at the end of the echoed response:

   0x02 **0x40** 0x01 0x65 0x00 0x01 0x06.

2. Use the **Mask (0x46)** IBIOS Serial Command to **set** the *_X10_RTS* flag (bit 7) and **clear** the *_X10_TXCOMMAND* flag (bit 3) in the *X10_FLAGS* register at 0x0166 (i1 Engine, shown here) or 0x01EF (i2 Engine) (see *Flat Memory Map*170) via:

   0x02 **0x46** 0x01 0x66 0x80 0xF7,

   then check for the ASCII 0x06 (ACK) at the end of the echoed response:

   0x02 **0x46** 0x01 0x66 **0x80 0xF7** 0x06.

3. The PLC will now send an **A1** X10 address over the powerline.
4. As in Step 1, load an X10 **AON** Command (**0x62**) into the X10 transmit buffer by sending:

   0x02 **0x40** 0x01 0x65 0x00 0x01 0xff 0x37 **0x62**,

   and check for an appropriate response:

   0x02 0x40 0x01 0x65 0x00 0x01 0x06.

5. As in Step 2, use the Mask Command to **set** the _X10_RTS bit and **set** the _X10_TXCOMMAND bit via:

   0x02 0x46 0x01 0x66 **0x88 0xff**

   then check for an appropriate response:

`0x02 0x46 0x01 0x66 0x88 0xFF 0x06.`

6. The PLC will now send an **AON** X10 Command over the powerline.

IBIOS Simulated Event

Summary

You can use the **Simulated Event (0x47)** IBIOS Serial Command from the *IBIOS Serial Command Summary Table*197 to cause the PLC to run its SALad application with the specified event or timer handle in the event queue. See *SALad Event Handling*268 for more information on the event processing system that SALad programs use.

This example lists a demo SALad application that you can install in the PLC using the tools documented in the *SALad Integrated Development Environment User's Guide*287.

Whenever when the PLC's *SET Button* is tapped, the **EVNT_BTN_TAP (0x0A)** IBIOS Event fires (see *IBIOS Event Summary Table*185). The demo application's event handler uses a **Variable-length Text Message (0x43)** IBIOS Serial Command to send a 'Button tap detected' ASCII message over the serial connection when this event fires.

You can use this same method for testing other event processing code in your SALad applications.

To demonstrate the **Simulated Event (0x47)** IBIOS Serial Command we will send a simulated **EVNT_BTN_TAP (0x0A)** and observe the 'Button tap detected' message.

Procedure

1. Using the SALad IDE, download the following SALad application into the PowerLinc V2 Controller (iPLC_Map.sal has the definitions and equates for the *Flat Memory Map*170, and Event.sal has equates for the *IBIOS Events*185):

```
INCLUDE "iPLC_Map.sal"
INCLUDE "Event.sal"

; API Macro Definitions
DEFINE API DATA 0x04
DEFINE SendString API 0x86   ; send a null terminated string
DEFINE SendByte API 0x44

; application header
ORG 0x210
  DATA 0x02 0x10    ; start at 0x0210
  DATA 0x00 0x01    ; length 1
  DATA 0x00 0x00    ; checksum 0 (no verification)
; entry point for timers
ORG 0x230
  END   ; just exit if timer

; entry point for static events
ORG 0x237
  COMP= #EVNT_INIT, NTL_EVENT, ButtonEvent  ; if initialization
  MOVE$ #0x00, NTL_TIMERS, 0x2D             ; clear out NTL_TIMERS
ButtonEvent
  COMP= #EVNT_BTN_TAP, NTL_EVENT, Exit      ; check for button tap
  SendString strButtonMessage
Exit
  END
strButtonMessage
  DATA "Button tap detected", 0x0d, 0x0a, 0x00
```

2. Tap the *SET Button* on the PLC and you should see the message 'Button tap detected' displayed in the IDE's *Comm Window – ASCII Window*322.

3. Now send the **Simulated Event** serial Command from the *IBIOS Serial Command Summary Table* 197 to fire the **EVNT_BTN_TAP (0x0A)** IBIOS Event:

   `0x02 0x47 0x0A 0x00.`

   You should see the same 'Button tap detected' message displayed in the IDE's ASCII Window.

IBIOS INSTEON Engine

The IBIOS INSTEON Engine handles INSTEON message transport. The format and meaning of INSTEON messages are described in detail in Chapter 5 — INSTEON Messages 38. How INSTEON messages propagate in an INSTEON network is explained in *Chapter 6 — INSTEON Signaling Details* 56.

You can use the method given in the *IBIOS Send INSTEON* 205 example in the *IBIOS Serial Commands* 196 section to send an INSTEON message with the INSTEON Engine. However, receiving INSTEON messages is time-critical, and although it is technically possible to wait for one of the 'INSTEON message received' events and then poll the INSTEON receive buffer, the buffer can easily be overwritten by new INSTEON messages if it is not read quickly enough. Therefore, the safest way to receive INSTEON messages is to use the *SALad coreApp Program* 272 pre-installed in *The SmartLabs PowerLinc Controller* 28, or else to write your own SALad application that employs the same method as coreApp. See the *IBIOS Receive INSTEON* 206 example in the *IBIOS Serial Commands* 196 section for more details.

The INSTEON Engine automatically handles *INSTEON Message Hopping* 49 and *INSTEON Message Retrying* 54. It also deals with the *Message Integrity Byte* 44 so you don't have to. Whenever you send or receive a Direct INSTEON message, the INSTEON Engine knows about the expected Acknowledgement message and handles it for you, then fires one of the EVNT_ITX_ACK or EVNT_ITX_NACK *IBIOS Events* 185 to notify you of the outcome.

The INSTEON Engine does not handle *INSTEON ALL-Link Groups* 93 and ALL-Link Cleanup messages, nor anything involving the *INSTEON ALL-Link Database* 101. Both the *SALad coreApp Program* 272 and INSTEON Modems (see *Chapter 10 — INSTEON Modems* 217) do handle these functions at a higher level, however, so you do not have to worry about coding them yourself.

IBIOS Software Realtime Clock/Calendar

IBIOS keeps time using a software realtime clock (RTC) that ticks once per second. Devices that also have a hardware RTC can use it to set the software RTC. The *SALad coreApp Program* 272 uses the hardware RTC in *The SmartLabs PowerLinc Controller* 28 to set the software RTC at power up and also every midnight.

You can set the software RTC manually using the registers shown below, excerpted from the *Flat Memory Map* 170.

i1 Addr	i2 Addr	Register and Bits	Description
0x0158	0x01E1	RTC_TIME_H	Time since midnight in minutes (MSB, 0-1439)
0x0159	0x01E2	RTC_TIME_L	Time since midnight in minutes (LSB, 0-1439)
0x015A	0x01E3	RTC_YEAR	Year (0-99)
0x015B	0x01E4	RTC_MON	Month (1-12)
0x015C	0x01E5	RTC_DAY	Day (1-31, month specific)
0x015D	0x01E6	RTC_DOW	Day-of-Week bitmap (0SSFTWTM)
0x015E	0x01E7	RTC_HOUR	Hour (0-23)
0x015F	0x01E8	RTC_MIN	Minute (0-59)
0x0160	0x01E9	RTC_SEC	Second (0-59)

When you set the software RTC, you should also set the *RTC_TIME_H,L* minutes-from-midnight value, because IBIOS will only set it by zeroing it at the next midnight.

The software RTC handles leap year, but it does not handle daylight-savings time. CoreApp, however, does handle daylight savings.

IBIOS X10 Signaling

When IBIOS receives an X10 byte over the powerline, it fires an **EVNT_XRX_MSG (0x08)** or **EVNT_XRX_XMSG (0x09)** IBIOS Event, as explained in *IBIOS Event Details*187, Note 6. If the *SALad coreApp Program*272 or another SALad application with an appropriate event handler is running, SALad will send an **X10 Byte Received (0x4A)** IBIOS Serial Command, as explained in *IBIOS Serial Command Details*198, Note 9.

The manual method for transmitting an X10 Address followed by an X10 Command is explained in the *IBIOS Send X10*207 IBIOS Serial Command example.

The following excerpt from the *Flat Memory Map*170 shows the registers and flags that IBIOS uses for sending and receiving X10 bytes.

i1 Addr	i2 Addr	Register and Bits		Description
0x0164	0x01ED	X10_RX		X10 Receive Buffer
0x0165	0x01EE	X10_TX		X10 Transmit Buffer
0x0166	0x01EF	X10_FLAGS		X10 Flags
		_X10_RTS	7	1=Request To Send
		_X10_EXTENDED	5	1=Extended transfer in progress (Tx or Rx)
		_X10_TXCOMMAND	3	1=Command, 0=Address for transmit
		_X10_RXCOMMAND	2	1=Command, 0=Address for receive
		_X10_VALID	1	1=X10 receive valid

To send an X10 byte, place it in the *X10_TX* buffer, set or clear *_X10_TXCOMMAND* to show whether it is an X10 Command or X10 Address, then set the *_X10_RTS* flag.

To see if there is a new received X10 byte in the *X10_TX* buffer, inspect the *_X10_VALID* flag, or just wait for an **EVNT_XRX_MSG (0x08)** or **EVNT_XRX_XMSG (0x09)** IBIOS Event. Look at *_X10_RXCOMMAND* to see if the received byte is an X10 Command or X10 Address, and look at *_X10_EXTENDED* to see if it is part of an X10 Extended message.

IBIOS Input/Output

IBIOS I/O drivers are limited to an *IBIOS LED Flasher* 214 and an *IBIOS SET Button Handler* 214.

IBIOS LED Flasher

You can control LED flashing using the following registers excerpted from the *Flat Memory Map* 170.

i1 Addr	i2 Addr	Register and Bits	Description
0x0168	0x0164	LED_MODE	Bitmap defines flashing pattern for LED 1=On, 0=Off
0x0169	0x0165	LED_TMR	Duration of LED flashing in seconds
0x016A	0x0166	LED_DLY	Period between each flash. Defaults to 5, which is 1/8 second per bit in LED_MODE.

LED_MODE is a bitmap that defines 8 on or off periods for the LED, and *LED_DLY* defines how fast the bits are shifted to flash the LED. The default *LED_DLY* value is 5, which is 1/8 second per bit. Larger values will slow down the flashing.

To flash the LED, load the number of seconds that you want it to flash into *LED_TMR*.

For example, to flash the LED on and off at half-second intervals for three seconds, load 0xF0 into *LED_MODE* and then load 0x03 into *LED_TMR*.

IBIOS SET Button Handler

Pushing the *SET Button* generates **EVNT_BTN_TAP (0x0A)**, **EVNT_BTN_HOLD (0x0B)**, and **EVNT_BTN_REL (0x0C)** IBIOS Events, as explained in *IBIOS Event Details* 187, Note 7. The *TAP_CNT* register in the *Flat Memory Map* 170. Lets you see how many times the *SET Button* was tapped.

i1 Addr	i2 Addr	Register and Bits	Description
0x0156	0x016B	TAP_CNT	Counts multiple *SET Button* taps

IBIOS Remote Debugging

You can remotely debug a SALad program with the **Debug Report (0x49)** IBIOS Serial Command (see *IBIOS Serial Command Details* 198). This is the underlying mechanism used by the IDE debugger described in the *SALad Integrated Development Environment User's Guide* 287.

Three flags in the *NTL_CONTROL* register at 0x0076 (see *Flat Memory Map* 170) control IBIOS remote debugging. These flags are bit 7, (*_RD_STEP*), bit 6 (*_RD_HALT*), and bit 5 (*_RD_BREAK*). A 16-bit address for breakpoints or range checking can be set in the *RD_H* and *RD_L* registers at 0x0026.

<table>
<tr><th>i1 Addr</th><th>i2 Addr</th><th colspan="2">Register and Bits</th><th colspan="3">Description</th></tr>
<tr><td>0x0026</td><td>0x0026</td><td colspan="2">RD_H</td><td colspan="3">Remote Debugging breakpoint address MSB</td></tr>
<tr><td>0x0027</td><td>0x0027</td><td colspan="2">RD_L</td><td colspan="3">Remote Debugging breakpoint address LSB</td></tr>
<tr><td rowspan="7">0x0076</td><td>0x0176</td><td colspan="2">NTL_CONTROL</td><td colspan="3">SALad debugging control flags</td></tr>
<tr><td colspan="2" rowspan="5">_RD_STEP
_RD_HALT</td><td rowspan="5">7
6</td><td>_RD_HALT</td><td>_RD_STEP</td><td></td></tr>
<tr><td>0</td><td>0</td><td>Normal execution</td></tr>
<tr><td>0</td><td>1</td><td>Animation (Trace)</td></tr>
<tr><td>1</td><td>0</td><td>Execution halted</td></tr>
<tr><td>1</td><td>1</td><td>Single step requested</td></tr>
<tr><td colspan="2">_RD_BREAK</td><td>5</td><td colspan="3">0=Range Check Mode, 1=Breakpoint Mode</td></tr>
</table>

To run a SALad program normally, clear both *_RD_HALT* and *_RD_STEP*. This is the default at power up.

To halt a SALad program as soon as possible, set *_RD_HALT* and clear *_RD_STEP*. Execution will stop after the current instruction executes and a **Debug Report (0x49)** IBIOS Serial Command will report the address of the next instruction to be executed.

To send a **Debug Report** before *every* instruction executes, clear *_RD_HALT* and set *_RD_STEP*.

To single-step through a SALad program, set both *_RD_HALT* and *_RD_STEP*. This will cause the next instruction to execute followed by an immediate halt. The halt will cause a **Debug Report** to be sent.

To do range checking or to set a breakpoint, load a comparison address into the *RD_H* and *RD_L* registers. If *RH_H* contains 0x00 (the default power-up condition), range checking and breakpoints are disabled.

Clear the *_RD_BREAK* flag to use the comparison address for range checking or else set *_RD_BREAK* to use the comparison address as a breakpoint.

If you are range checking and program execution is attempted at a location greater than the comparison address, execution will be halted, a **Debug Report** will be sent, and the *IBIOS Watchdog Timer* 216 will cause a power-on reset.

If you are using the comparison register for a breakpoint, execution will halt only if the beginning of the next instruction exactly matches the comparison address.

You can also do remote debugging over the INSTEON network alone by setting the flag *_I_DebugRpt* (bit 6) in the *I_Control* register at 0x0142. This flag is cleared at power up. When the *_I_DebugRpt* flag is set, every time a **Debug Report (0x49)** IBIOS Serial Command would be sent over a serial connection, a **Debug Report (0x49)** INSTEON **SB** Command from the table of *INSTEON Standard-length Broadcast Commands*155 will be sent in an INSTEON Broadcast message. You can use INSTEON **SD** Peek and Poke Commands from the table of *INSTEON Standard-length Direct Commands*125 to set the comparison address and the debugging control flags in the remote INSTEON device.

IBIOS Watchdog Timer

The watchdog timer in IBIOS is automatic. IBIOS sets appropriate timeout values for itself and resets the watchdog whenever it returns from a task before the timeout. If a timeout does occur, the watchdog code performs a power-on reset, which puts the device in the same state as cycling power does.

There are two ways to force the watchdog timer to cause a reset. One will occur if you are using IBIOS debugging to do program counter range checking (see *IBIOS Remote Debugging*215) and the range is exceeded. The other way is to set the *_Reset* flag (bit 7) in the *Control* register at 0x0154 (i1 Engine) or 0x016C (12 Engine) to one (see *Flat Memory Map*170). Both conditions cause IBIOS to execute an endless loop, which will eventually time out the watchdog.

You can manually reset the watchdog (buying more time) by setting the *_Watchdog* flag (bit 6) in the *Control* register at 0x0154 to one.

Chapter 10 — INSTEON Modems

INSTEON Modem (IM) chips offer developers a simple, robust interface to an INSTEON network. There are currently two kinds of IM, the *IN2680A INSTEON Direct Powerline Modem Interface*10 and the *IN2682A INSTEON Direct RF Modem Interface*10. A BiPHY™ Modem that interfaces to *both* the powerline and radio is under development.

SmartLabs offers a Powerline Modem™ (PLM) module, which uses an IN2680A Modem chip to implement an interface between a host device and an INSTEON network on the powerline. The PLM is a self-contained module that plugs into the wall and connects to the host using a serial communications daughter card that is fully isolated from the powerline. See *The SmartLabs Powerline Modem*29, above, for more information about the PLM.

INSTEON Modems provide a simple interface to many of the *IBIOS Serial Commands*196 described in *Chapter 9 — INSTEON BIOS (IBIOS)*166, but they also handle ALL-Linking, ALL-Link Database management, ALL-Link Cleanup messages, X10 powerline interfacing, and message acknowledgement. The RS232 serial interface to the host is similar to the *IBIOS Serial Communication Protocol*193, and some of the *IBIOS Serial Commands*196 are duplicated in the INSTEON Modems.

As an added bonus, the easiest way to achieve INSTEON conformance for your product is to build it around an INSTEON modem, because an IM automatically handles most of the details of the INSTEON protocol for you. See the *INSTEON Conformance Specification*9 document for the full conformance requirements.

In This Chapter

IM Serial Communication Protocol and Settings

In This Section

IM Serial Communication Protocol

All INSTEON Modem (IM) Serial Commands start with ASCII `0x02` (STX, Start-of-Text) followed by the Serial Command Number (see *IM Serial Commands*222). What data follows the Command depends on the Command syntax (see *IM Serial Command Summary Table*223 and *IM Serial Command Charts*227).

When you send a message to the IM, it will respond with an echo of the `0x02` and the IM Command Number followed by any data that the Command returns (often just an echo of what you sent to it). The last byte it sends back will be ASCII 0x06 (ACK, Acknowledge).

(**S:** and **R:** denote serial data you **Send to** or **Receive from** the IM, respectively.)

S:	0x02 <Command Number> <parameters>
R:	0x02 <Command Number> <any returned data> 0x06 (ACK)

If the IM is not ready, it will respond with an echo of the `0x02` and the IM Command Number followed by ASCII 0x15 (NAK, Negative Acknowledge).

S:	0x02 <Command Number> <parameters>
R:	0x15 (NAK)

If you receive 0x15 (NAK), resend your Serial Command.

IM RS232 Port Settings

To communicate to an RS232 IM, set your PC's COM port as follows:

Setting	**Value**
Baud Rate	19,200
Data Bits	8
Parity	N
Stop Bits	1
Hardware Flow Control	None
Software Flow Control	IM echoes bytes received from host

The IM buffers IM Commands as it receives them, so you can send a complete IM Command without pause. To maintain compatibility with earlier IM versions, the IM will echo each byte that it receives (earlier versions of the IM used byte echoing for flow control). You can now ignore the byte echos, but in order to avoid overrunning the IM's receive buffer, you must wait for the IM to send its response to your current IM Command before sending a new one.

Note that there is a *maximum* time between IM Command bytes that you send to the IM. If you do not send the next expected byte of an IM Command within 240 milliseconds after sending the previous one, the IM will reset its message parser and you will have to resend the message from the beginning. You can disable this *Deadman* feature by setting a configuration bit (see *Set IM Configuration*255 below).

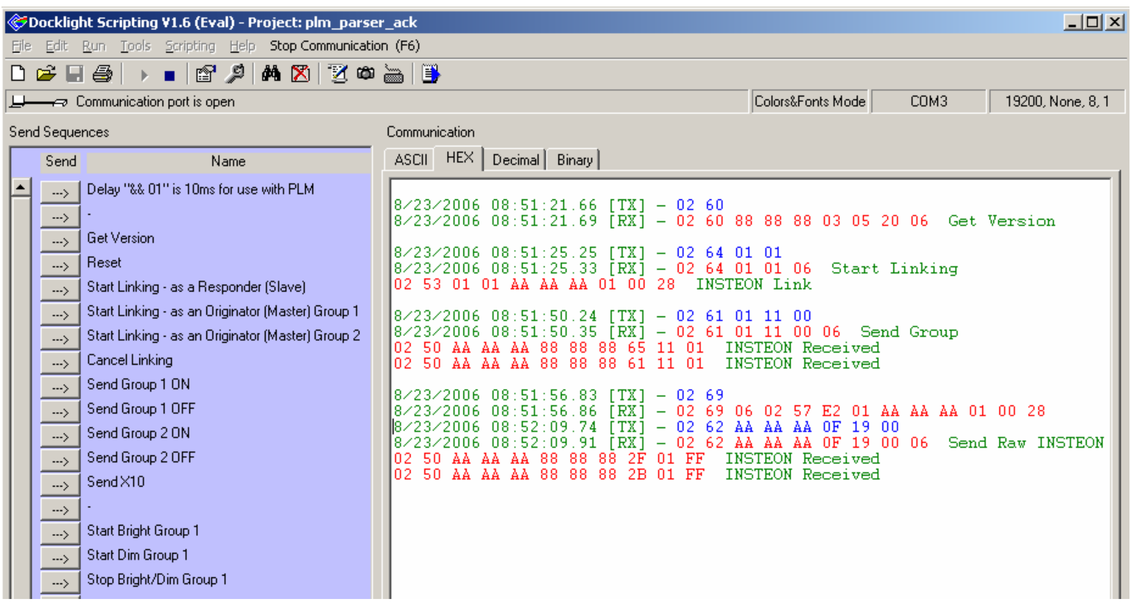
There is no flow control when the IM sends data *to* the host—the IM will transfer data to the host as fast as it can send it.

How to Quickly Start Communicating with an IM

No matter how your application intends to use the IM, it is important to gain a basic understanding of how it operates. SmartLabs suggests that developers use a terminal communications program and a serial connection to an IM to get started.

While there are many terminal programs for computers, SmartLabs has found good results with Docklight Scripting. An evaluation copy may be downloaded from http://www.docklight.de/.

Docklight Scripting allows you to set up test macros and label received *IM Serial Commands*222 for easy identification, as suggested in the following screenshot:



IM Power-up and Reset States

This section describes the *IM Power-up Behavior* 221 and the *IM Factory Reset State* 221.

IM Power-up Behavior

The table below shows the state of the IM when it powers up. Holding down the SET Button while powering up will cause a factory reset.

LED Indication	Meaning
LED on steadily	The IM detected an external EEPROM (up to 32 KB) for storage of database links.
LED blinks six times	The IM did not detect an external EEPROM, so it will use the internal EEPROM in the processor chip. A maximum of 31 ALL-Links are permitted. An attempt to add a 32nd ALL-Link will result in the 31st being erased.
LED off	The user pressed and held the IM's SET button for 10 seconds while powering up, causing the IM to perform a factory reset and go into the *IM Factory Reset State* 221. At the conclusion of the reset, the IM's LED will give one of the two indications above. You will also receive a *User Reset Detected* 253 message from the IM.

IM Factory Reset State

Resetting the IM to its factory default condition by holding down the SET Button for ten seconds while powering it up or by sending it a *Reset the IM* 252 Command puts it into the following state:

IM Resource	Factory Reset State
ALL-Link Database	Erased (set to all zeros).
Host Device Category, Device Subcategory, Firmware Version	Set to the original DevCat (0x03), SubCat (0x05), and firmware version hard-coded into the IM's firmware at the factory.
IM Configuration Flags	Cleared (set to all zeros).

IM Serial Commands

The IM Serial Command set is a simple but complete interface between a host application and an INSTEON network. For example, a microcontroller in a thermostat could use an INSTEON Powerline Modem to send and receive messages to other INSTEON or X10 devices on the home's powerline.

IM Serial Commands are similar to the *IBIOS Serial Commands*196 in both format and functionality.

In this section, the IM Serial Commands are presented twice, once in the same table format used for the *IBIOS Serial Commands*196, and again as a series of charts grouped by functionality.

In This Section

IM Serial Command Summary Table

This table lists all of the Modem Serial Commands supported by INSTEON powerline or RF modem chips.

Code

Gives the hexadecimal number of the IM Serial Command. Note that IM Commands sent by an IM to the host begin at **0x50** and IM Commands sent by the host to an IM begin at **0x60**.

Command

Gives the name of the IM Serial Command as a link to the complete explanation of the Command in the *IM Serial Command Charts*227.

Format

Gives the syntax of the IM Serial Command, including any parameters.

S: and **R:** denote serial data you **Send to** or **Receive from** the IM, respectively. See *IM Serial Communication Protocol*219 for more information.

All IM Serial Commands start with ASCII 0x02 (STX, Start-of-Text) followed by the Serial Command Number.

All fields in this table contain only one byte, except as noted.

INSTEON Modem Serial Commands		
Commands Sent from an IM to the Host		
Code	**Command**	**Format**
0x50	INSTEON Standard Message Received231	**R:** 0x02 **0x50** <INSTEON Standard message (9 bytes)>
0x51	INSTEON Extended Message Received232	**R:** 0x02 **0x51** <INSTEON Extended message (23 bytes)>
0x52	X10 Received238	**R:** 0x02 **0x52** <Raw X10> <X10 Flag>
0x53	ALL-Linking Completed245	**R:** 0x02 **0x53** <0x00 (IM is Responder) | 0x01 (IM is Controller | 0xFF Link Deleted)> <ALL-Link Group> <ID high byte> <ID middle byte> <ID low byte> <Device Category> <Device Subcategory> <0xFF | Firmware Revision>
0x54	Button Event Report260	**R:** 0x02 **0x54** <0x02> IM's SET Button tapped
		R: 0x02 **0x54** <0x03> IM's SET Button held
		R: 0x02 **0x54** <0x04> IM's SET Button released after hold
		R: 0x02 **0x54** <0x12> IM's Button 2 tapped
		R: 0x02 **0x54** <0x13> IM's Button 2 held
		R: 0x02 **0x54** <0x14> IM's Button 2 released after hold
		R: 0x02 **0x54** <0x22> IM's Button 3 tapped

INSTEON Modem Serial Commands		
Commands Sent from an IM to the Host		
Code	**Command**	**Format**
		R: 0x02 **0x54** <0x23> IM's Button 3 held
		R: 0x02 **0x54** <0x24> IM's Button 3 released after hold
0x55	User Reset Detected 253	**R:** 0x02 **0x55** User pushed and held IM's SET Button on power up
0x56	ALL-Link Cleanup Failure Report 241	**R:** 0x02 **0x56** <0x01> <ALL-Link Group> <ID high byte> <ID middle byte> <ID low byte>
0x57	ALL-Link Record Response 249	**R:** 0x02 **0x57** <ALL-Link Record Flags> <ALL-Link Group> <ID high byte> <ID middle byte> <ID low byte> <Link Data 1> <Link Data 2> <Link Data 3>
0x58	ALL-Link Cleanup Status Report 242	**R:** 0x02 **0x58** <0x06> ALL-Link Cleanup sequence completed
		R: 0x02 **0x58** <0x15> ALL-Link Cleanup sequence aborted due to INSTEON traffic
Commands Sent from the Host to an IM		
0x60	Get IM Info 257	**S:** 0x02 **0x60**
		R: 0x02 **0x60** <ID high byte> <ID middle byte> <ID low byte> <Device Category> <Device Subcategory> < Firmware Revision> <0x06>
0x61	Send ALL-Link Command 239	**S:** 0x02 **0x61** <ALL-Link Group> <ALL-Link Command> <0xFF | 0x00>
		R: 0x02 **0x61** <ALL-Link Group> <ALL-Link Command> <0xFF | 0x00> <0x06>
0x62	Send INSTEON Standard or Extended Message 228	**S:** 0x02 **0x62** <INSTEON Standard message (6 bytes, excludes *From Address*) | INSTEON Extended message (20 bytes, excludes *From Address*)>
		R: 0x02 **0x62** <INSTEON Standard message (6 bytes, excludes *From Address*) | INSTEON Extended message (20 bytes, excludes *From Address*)> <0x06>
0x63	Send X10 237	**S:** 0x02 **0x63** <Raw X10> <X10 Flag>
		R: 0x02 **0x63** <Raw X10> <X10 Flag> <0x06>
0x64	Start ALL-Linking 243	**S:** 0x02 **0x64** <0x00 (IM is Responder) | 0x01 (IM is Controller) | 0x03 (IM is either) | 0xFF (Link Deleted)> <ALL-Link Group>

INSTEON Modem Serial Commands		
Commands Sent from an IM to the Host		
Code	**Command**	**Format**
		R: 0x02 **0x64** <0x00 (IM is Responder) | 0x01 (IM is Controller) | 0x03 (IM is either) | 0xFF (Link Deleted)> <ALL-Link Group> <0x06>
0x65	Cancel ALL-Linking 244	**S:** 0x02 **0x65**
		R: 0x02 **0x65** <0x06>
0x66	Set Host Device Category 258	**S:** 0x02 **0x66** <Device Category> <Device Subcategory> <0xFF | Firmware Revision>
		R: 0x02 **0x66** <Device Category> <Device Subcategory> <0xFF | Firmware Revision> <0x06>
0x67	Reset the IM 252	**S:** 0x02 **0x67**
		R: 0x02 **0x67** <0x06>
0x68	Set INSTEON ACK Message Byte 234	**S:** 0x02 **0x68** <Command 2 Data>
		R: 0x02 **0x68** <Command 2 Data> <0x06>
0x69	Get First ALL-Link Record 246	**S:** 0x02 **0x69**
		R: 0x02 **0x69** <0x06>
0x6A	Get Next ALL-Link Record 247	**S:** 0x02 **0x6A**
		R: 0x02 **0x6A** <0x06>
0x6B	Set IM Configuration 255	**S:** 0x02 **0x6B** <IM Configuration Flags>
		R: 0x02 **0x6B** <IM Configuration Flags> <0x06>
0x6C	Get ALL-Link Record for Sender 248	**S:** 0x02 **0x6C**
		R: 0x02 **0x6C** <0x06>
0x6D	LED On 261	**S:** 0x02 **0x6D**
		R: 0x02 **0x6D** <0x06>
0x6E	LED Off 262	**S:** 0x02 **0x6E**
		R: 0x02 **0x6E** <0x06>
0x6F	Manage ALL-Link Record 250	**S:** 0x02 **0x6F** <Control Flags> <ALL-Link Record Flags> <ALL-Link Group> <ID high byte> <ID middle byte> <ID low byte> <Link Data 1> <Link Data 2> <Link Data 3>

INSTEON Modem Serial Commands		
Commands Sent from an IM to the Host		
Code	**Command**	**Format**
		R: 0x02 **0x6F** <Control Flags> <ALL-Link Record Flags> <ALL-Link Group> <ID high byte> <ID middle byte> <ID low byte> <Link Data 1> <Link Data 2> <Link Data 3> <0x06>
0x70	Set INSTEON NAK Message Byte [236]	**S:** 0x02 **0x70** <Command 2 Data>
		R: 0x02 **0x70** <Command 2 Data> <0x06>
0x71	Set INSTEON ACK Message Two Bytes [235]	**S:** 0x02 **0x71** <Command 1 Data> <Command 2 Data>
		R: 0x02 **0x71** <Command 1 Data> <Command 2 Data> <0x06>
0x72	RF Sleep [259]	**S:** 0x02 **0x72**
		R: 0x02 **0x72** <0x06>
0x73	Get IM Configuration [255]	**S:** 0x02 **0x73**
		R: 0x02 **0x73** <IM Configuration Flags> <Spare 1> <Spare 2> <0x06>

IM Serial Command Charts

The following charts describe the IM Commands individually in a chart format, grouped by functionality. These are the same IM Commands as in the *IM Serial Command Summary Table*223, which is ordered by Command Number.

Note that IM Commands sent by an IM to the host begin at **0x50** and IM Commands sent by the host to an IM begin at **0x60**. When the host sends an IM Command to an IM, the IM will respond with a message according to the *IM Serial Communication Protocol*219.

In This Section

INSTEON Message Handling

Send INSTEON Standard or Extended Message

This Command lets you send either a Standard-length or an Extended-length INSTEON message, depending only on what kind of INSTEON message you include in the body of the Command.

Send INSTEON Standard-length Message

Send INSTEON Standard-length Message (0x62)	
What it does	Allows you to send a raw Standard-length INSTEON message.
What you send	8 bytes.
What you'll get	9 bytes.
LED indication	None.
Related Commands	IM 0x50 INSTEON Standard Message Received[231] IM 0x51 INSTEON Extended Message Received[232]

Command Sent from Host to IM

Byte	**Value**	**Meaning**
1	0x02	Start of IM Command
2	**0x62**	IM Command Number
3	<To Address high>	The high byte of the INSTEON ID of the message addressee.
4	<To Address middle>	The middle byte of the INSTEON ID of the message addressee.
5	<To Address low>	The low byte of the INSTEON ID of the message addressee.
6	<Message Flags>	The INSTEON message flags indicating message type and hops. Extended Message Flag (bit 4) is 0
7	<Command 1>	INSTEON Command 1 for the addressee to execute
8	<Command 2>	INSTEON Command 2 for the addressee to execute

Message Returned by IM to Host

Byte	**Value**	**Meaning**
1	0x02	Echoed Start of IM Command
2	**0x62**	Echoed IM Command Number
3	<To Address high>	Echoed <To Address high>
4	<To Address middle>	Echoed <To Address middle>
5	<To Address low>	Echoed <To Address low>
6	<Message Flags>	Echoed <Message Flags> Extended Message Flag (bit 4) is 0
7	<Command 1>	Echoed <Command 1>
8	<Command 2>	Echoed <Command 2>
9	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly 0x15 (NAK) if an error occurred

Notes
The *From Address* is not required because the IM will automatically insert its own INSTEON ID into the message.
For more information on INSTEON Commands and the latest Command set, please download the current *INSTEON Command Tables Document*[9] from www.insteon.net.

Send INSTEON Extended-length Message

Send INSTEON Extended-length Message (0x62)	
What it does	Allows you to send a raw Extended-length INSTEON message.
What you send	22 bytes.
What you'll get	23 bytes.
LED indication	None.
Related Commands	IM 0x50 INSTEON Standard Message Received 231 IM 0x51 INSTEON Extended Message Received 232

Command Sent from Host to IM

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x62**	IM Command Number
3	<To Address high>	The high byte of the INSTEON ID of the message addressee.
4	<To Address middle>	The middle byte of the INSTEON ID of the message addressee.
5	<To Address low>	The low byte of the INSTEON ID of the message addressee.
6	<Message Flags>	The INSTEON message flags indicating message type and hops. Extended Message Flag (bit 4) is 1
7	<Command 1>	INSTEON Command 1 for the addressee to execute
8	<Command 2>	INSTEON Command 2 for the addressee to execute
9	<User Data 1>	Extended message data
10	<User Data 2>	Extended message data
11	<User Data 3>	Extended message data
12	<User Data 4>	Extended message data
13	<User Data 5>	Extended message data
14	<User Data 6>	Extended message data
15	<User Data 7>	Extended message data
16	<User Data 8>	Extended message data
17	<User Data 9>	Extended message data
18	<User Data 10>	Extended message data
19	<User Data 11>	Extended message data
20	<User Data 12>	Extended message data
21	<User Data 13>	Extended message data
22	<User Data 14>	Extended message data

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x62**	Echoed IM Command Number
3	<To Address high>	Echoed <To Address high>
4	<To Address middle>	Echoed <To Address middle>
5	<To Address low>	Echoed <To Address low>
6	<Message Flags>	Echoed <Message Flags> Extended Message Flag (bit 4) is 1
7	<Command 1>	Echoed <Command 1>
8	<Command 2>	Echoed <Command 2>
9	<User Data 1>	Echoed Extended message data
10	<User Data 2>	Echoed Extended message data
11	<User Data 3>	Echoed Extended message data
12	<User Data 4>	Echoed Extended message data
13	<User Data 5>	Echoed Extended message data
14	<User Data 6>	Echoed Extended message data

Send INSTEON Extended-length Message (0x62)		
15	<User Data 7>	Echoed Extended message data
16	<User Data 8>	Echoed Extended message data
17	<User Data 9>	Echoed Extended message data
18	<User Data 10>	Echoed Extended message data
19	<User Data 11>	Echoed Extended message data
20	<User Data 12>	Echoed Extended message data
21	<User Data 13>	Echoed Extended message data
22	<User Data 14>	Echoed Extended message data
23	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly 0x15 (NAK) if an error occurred
Notes		
The *From Address* is not required because the IM will automatically insert its own INSTEON ID into the message.		
For more information on INSTEON Commands and the latest Command set, please download the current *INSTEON Command Tables Document*[9] from www.insteon.net.		

INSTEON Standard Message Received

INSTEON Standard Message Received (0x50)	
What it does	Informs you of an incoming Standard-length INSTEON message.
When you'll get this	A Standard-length INSTEON message is received from either a Controller or Responder that you are ALL-Linked to.
What you'll get	11 bytes.
LED indication	The LED will blink during INSTEON reception.
Related Commands	IM 0x51 INSTEON Extended Message Received[232] IM 0x52 X10 Received[238]

Message Sent from IM to Host		
Byte	**Value**	**Meaning**
1	0x02	Start of IM Command
2	**0x50**	IM Command Number
3	<From Address high>	The high byte of the INSTEON ID of the message originator.
4	<From Address middle>	The middle byte of the INSTEON ID of the message originator.
5	<From Address low>	The low byte of the INSTEON ID of the message originator.
6	<To Address high>	The high byte of the INSTEON ID of the message addressee. If the message is an ALL-Link Broadcast (bits 7 and 6 of the <Message Flags> byte are set) then this will be 0.
7	<To Address middle>	The middle byte of the INSTEON ID of the message addressee. If the message is an ALL-Link Broadcast (bits 7 and 6 of the <Message Flags> byte are set) then this will be 0.
8	<To Address low>	The low byte of the INSTEON ID of the message addressee. If the message is an ALL-Link Broadcast (bits 7 and 6 of the <Message Flags> byte are set) then this will indicate the ALL-Link Group Number.
9	<Message Flags>	The INSTEON message flags indicating message type and hops.
10	<Command 1>	INSTEON *Command 1* field of the message.
11	<Command 2>	INSTEON *Command 2* field of the message. This byte contains the ALL-Link Group Number of the ALL-Link Broadcast when either bit 6 of the <Message Flags> byte is set (ALL-Link Cleanup) or bits 6 and 5 of the <Message Flags> byte are set (ALL-Link Cleanup ACK).

Notes
This is the same as IM 0x51 *INSTEON Extended Message Received*[232], except that there is no <User Data>.
Normally, the IM will only send the host INSTEON messages that are explicitly addressed to the IM or that are from devices that the IM is ALL-Linked to. This behavior can be modified—see the *About Monitor Mode*[256] note in the *Set IM Configuration*[255] chart for more information.
For more information on INSTEON Commands and the latest Command set, please download the current *INSTEON Command Tables Document*[9] from www.insteon.net.

INSTEON Extended Message Received

INSTEON Extended Message Received (0x51)	
What it does	Informs you of an incoming Extended-length INSTEON message.
When you'll get this	An Extended-length INSTEON message is received from either a Controller or Responder that you are ALL-Linked to.
What you'll get	25 bytes.
LED indication	The LED will blink during INSTEON reception.
Related Commands	IM 0x50 INSTEON Standard Message Received 231 IM 0x52 X10 Received 238

Message Sent from IM to Host

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x51**	IM Command Number
3	<From Address high>	The high byte of the INSTEON ID of the message originator.
4	<From Address middle>	The middle byte of the INSTEON ID of the message originator.
5	<From Address low>	The low byte of the INSTEON ID of the message originator.
6	<To Address high>	The high byte of the INSTEON ID of the message addressee. If the message is an ALL-Link Broadcast (bits 7 and 6 of the <Message Flags> byte are set) then this will be 0.
7	<To Address middle>	The middle byte of the INSTEON ID of the message addressee. If the message is an ALL-Link Broadcast (bits 7 and 6 of the <Message Flags> byte are set) then this will be 0.
8	<To Address low>	The low byte of the INSTEON ID of the message addressee. If the message is an ALL-Link Broadcast (bits 7 and 6 of the <Message Flags> byte are set) then this will indicate the ALL-Link Group Number.
9	<Message Flags>	The INSTEON message flags indicating message type and hops.
10	<Command 1>	INSTEON *Command 1* field of the message.
11	<Command 2>	INSTEON *Command 2* field of the message. This byte contains the ALL-Link Group Number of the ALL-Link Broadcast when either bit 6 of the <Message Flags> byte is set (ALL-Link Cleanup) or bits 6 and 5 of the <Message Flags> byte are set (ALL-Link Cleanup ACK).
12	<User Data 1>	Extended message data
13	<User Data 2>	Extended message data
14	<User Data 3>	Extended message data
15	<User Data 4>	Extended message data
16	<User Data 5>	Extended message data
17	<User Data 6>	Extended message data
18	<User Data 7>	Extended message data
19	<User Data 8>	Extended message data
20	<User Data 9>	Extended message data
21	<User Data 10>	Extended message data
22	<User Data 11>	Extended message data
23	<User Data 12>	Extended message data
24	<User Data 13>	Extended message data
25	<User Data 14>	Extended message data

INSTEON Extended Message Received (0x51)
Notes
This is the same as IM 0x50 INSTEON Standard Message Received231, except that there are 14 bytes of <User Data>.
Normally, the IM will only send the host INSTEON messages that are explicitly addressed to the IM or that are from devices that the IM is ALL-Linked to. This behavior can be modified—see the *About Monitor Mode*256 note in the *Set IM Configuration*255 chart for more information.
For more information on INSTEON Commands and the latest Command set, please download the current *INSTEON Command Tables Document*9 from www.insteon.net.

Set INSTEON ACK Message Byte

Set INSTEON ACK Message Byte (0x68)		
What it does	Allows you to put one byte of data into the *Command 2* field of the INSTEON ACK message that the INSTEON Engine automatically sends after it receives an INSTEON Direct message.	
What you send	3 bytes.	
What you'll get	4 bytes.	
LED indication	None.	
Related Commands	IM 0x50 INSTEON Standard Message Received 231 IM 0x51 INSTEON Extended Message Received 232 IM 0x71 Set INSTEON ACK Message Two Bytes 235 IM 0x70 Set INSTEON NAK Message Byte 236	
Command Sent from Host to IM		
Byte	**Value**	**Meaning**
1	0x02	Start of IM Command
2	**0x68**	IM Command Number
3	<Command 2 Data>	Data byte to place into the *Command 2* field of the ACK response.
Message Returned by IM to Host		
Byte	**Value**	**Meaning**
1	0x02	Echoed Start of IM Command
2	**0x68**	Echoed IM Command Number
3	<Command 2 Data>	Echoed <Command 2 Data>
4	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly. 0x15 (NAK) if an error occurred.
Notes		
You have only about 15 milliseconds after the receipt of an INSTEON Direct message from the IM to send this Command to the IM. The reason is that the INSTEON Engine in the IM automatically sends Acknowledgement messages in assigned timeslots.		
Use *Set INSTEON ACK Message Two Bytes* 235 when you need to return two bytes of data in an ACK message.		
Use *Set INSTEON NAK Message Byte* 236 when you need to return one byte of data in a NAK message.		
Certain INSTEON Direct Commands require returned data in the Acknowledgement message. For more information on INSTEON Commands and the latest Command set, please download the current *INSTEON Command Tables Document* 9 from www.insteon.net.		

Set INSTEON ACK Message Two Bytes

Set INSTEON ACK Message Two Bytes (0x71)	
What it does	Allows you to put two bytes of data into the combined *Command 1* and *Command 2* fields of the INSTEON ACK message that the INSTEON Engine automatically sends after it receives an INSTEON Direct message.
What you send	4 bytes.
What you'll get	5 bytes.
LED indication	None.
Related Commands	IM 0x50 INSTEON Standard Message Received[231] IM 0x51 INSTEON Extended Message Received[232] IM 0x68 Set INSTEON ACK Message Byte[234] IM 0x70 Set INSTEON NAK Message Byte[236]

Command Sent from Host to IM

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x71**	IM Command Number
3	<Command 1 Data>	Data byte to place into the *Command 1* field 2 of the ACK response.
4	<Command 2 Data>	Data byte to place into the *Command 2* field 2 of the ACK response.

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x71**	Echoed IM Command Number
3	<Command 1 Data>	Echoed <Command 1 Data>
4	<Command 2 Data>	Echoed <Command 2 Data>
5	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly. 0x15 (NAK) if an error occurred.

Notes

You have only about 15 milliseconds after the receipt of an INSTEON Direct message from the IM to send this Command to the IM. The reason is that the INSTEON Engine in the IM automatically sends Acknowledgement messages in assigned timeslots.

Use *Set INSTEON ACK Message Byte*[234] when you only need to return one byte of data in an ACK message.

Use *Set INSTEON NAK Message Byte*[236] when you need to return one byte of data in a NAK message.

Certain INSTEON Direct Commands require returned data in the Acknowledgement message. For more information on INSTEON Commands and the latest Command set, please download the current *INSTEON Command Tables Document*[9] from www.insteon.net.

Set INSTEON NAK Message Byte

Set INSTEON NAK Message Byte (0x70)	
What it does	Allows you to change the INSTEON ACK message that the INSTEON Engine automatically sends after it receives an INSTEON Direct message into a NAK message, and to put one byte of data into the *Command 2* field of that message.
What you send	3 bytes.
What you'll get	4 bytes.
LED indication	None.
Related Commands	IM 0x50 INSTEON Standard Message Received[231] IM 0x51 INSTEON Extended Message Received[232] IM 0x68 Set INSTEON ACK Message Byte[234] IM 0x70 Set INSTEON ACK Message Two Bytes[235]

Command Sent from Host to IM

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x70**	IM Command Number
3	<Command 2 Data>	Data byte to place into the *Command 2* field of the ACK response.

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x70**	Echoed IM Command Number
3	<Command 2 Data>	Echoed <Command 2 Data>
4	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly. 0x15 (NAK) if an error occurred.

Notes

You have only about 15 milliseconds after the receipt of an INSTEON Direct message from the IM to send this Command to the IM. The reason is that the INSTEON Engine in the IM automatically sends Acknowledgement messages in assigned timeslots.

Use *Set INSTEON ACK Message Byte*[234] or *Set INSTEON ACK Message Two Bytes*[235] when you need to return one or two bytes of data in an ACK message.

NAK messages report certain error conditions in a receiving device. See *NAK Error Codes*[119] for more information.

X10 Message Handling

Send X10

Send X10 (0x63)	
What it does	Allows you to send a raw X10 Address or X10 Command.
What you send	4 bytes.
What you'll get	5 bytes.
LED indication	None.
Related Commands	IM 0x52 X10 Received[238]

Command Sent from Host to IM

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x63**	IM Command Number
3	<Raw X10>	The four most significant bits contain the X10 House Code. The four least significant bits contain the X10 Key Code.
4	<X10 Flag>	0x00 indicates that the X10 Key Code is an X10 Unit Code. 0x80 indicates that the X10 Key Code is an X10 Command.

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x63**	Echoed IM Command Number
3	<Raw X10>	Echoed <Raw X10>
4	<X10 Flag>	Echoed <X10 Flag>
5	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly 0x15 (NAK) if an error occurred

X10 Translation Table

	4 MSBs of <Raw X10>	**4 LSBs of <Raw X10>**	
4-bit Code	**X10 House Code**	**X10 Unit Code <X10 Flag> = 0x00**	**X10 Command <X10 Flag> = 0x80**
0x6	A	1	All Lights Off
0xE	B	2	Status = Off
0x2	C	3	On
0xA	D	4	Preset Dim
0x1	E	5	All Lights On
0x9	F	6	Hail Acknowledge
0x5	G	7	Bright
0xD	H	8	Status = On
0x7	I	9	Extended Code
0xF	J	10	Status Request
0x3	K	11	Off
0xB	L	12	Preset Dim
0x0	M	13	All Units Off
0x8	N	14	Hail Request
0x4	O	15	Dim
0xC	P	16	Extended Data (analog)

X10 Received

X10 Received (0x52)	
What it does	Informs you of an X10 byte detected on the powerline.
When you'll get this	Any X10 traffic is detected on the powerline.
What you'll get	4 bytes.
LED indication	The LED will blink during X10 reception.
Related Commands	IM 0x63 Send X10 237 IM 0x50 INSTEON Standard Message Received 231 IM 0x51 INSTEON Extended Message Received 232

Message Sent from IM to Host

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x52**	IM Command Number
3	<Raw X10>	The four most significant bits contain the X10 House Code. The four least significant bits contain the X10 Key Code.
4	<X10 Flag>	0x00 indicates that the X10 Key Code is an X10 Unit Code. 0x80 indicates that the X10 Key Code is an X10 Command.

X10 Translation Table

	4 MSBs of <Raw X10>	4 LSBs of <Raw X10>	
4-bit Code	**X10 House Code**	**X10 Unit Code <X10 Flag> = 0x00**	**X10 Command <X10 Flag> = 0x80**
0x6	A	1	All Lights Off
0xE	B	2	Status = Off
0x2	C	3	On
0xA	D	4	Preset Dim
0x1	E	5	All Lights On
0x9	F	6	Hail Acknowledge
0x5	G	7	Bright
0xD	H	8	Status = On
0x7	I	9	Extended Code
0xF	J	10	Status Request
0x3	K	11	Off
0xB	L	12	Preset Dim
0x0	M	13	All Units Off
0x8	N	14	Hail Request
0x4	O	15	Dim
0xC	P	16	Extended Data (analog)

INSTEON ALL-Link Commands

Send ALL-Link Command

Send ALL-Link Command (0x61)	
What it does	Sends an ALL-Link Command to an ALL-Link Group of one or more Responders that the IM is ALL-Linked to.
What you send	5 bytes.
What you'll get	6 bytes for the echo of the Command and then an additional 11 bytes in an *INSTEON Standard Message Received*231 message for each device in the group that acknowledges ALL-Link Cleanup, or 7 bytes in an *ALL-Link Cleanup Failure Report*241 message for each device in the group that does not acknowledge ALL-Link Cleanup.
LED indication	None.
Related Commands	IM 0x50 INSTEON Standard Message Received231 IM 0x56 ALL-Link Cleanup Failure Report241 IM 0x58 ALL-Link Cleanup Status Report242

Command Sent from Host to IM

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x61**	IM Command Number
3	<ALL-Link Group>	ALL-Link Group Number that the ALL-Link Command is sent to
4	<ALL-Link Command>	ALL-Link Command
5	<Broadcast Command 2>	Sent in the *Command 2* field of the ALL-Link Broadcast message only. *Command 2* will always contain the ALL-Link Group Number for the ALL-Link Cleanup messages that follow.

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x61**	Echoed IM Command Number
3	<ALL-Link Group>	Echoed <ALL-Link Group>
4	<ALL-Link Command>	Echoed <ALL-Link Command>
5	<Broadcast Command 2>	Echoed <Broadcast Command 2>
6	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly 0x15 (NAK) if an error occurred or the group does not exist

Notes

The IM automatically sends ALL-Link Cleanup messages to each member of an ALL-Link Group following an ALL-Link Broadcast message. If the IM detects other INSTEON traffic during this process, it will abort the ALL-Link Cleanup sequence and send you an *ALL-Link Cleanup Status Report*242 with a *Status Byte* of 0x15 (NAK). The Cleanup sequence proceeds in the order in which the devices in the ALL-Link Group were added to the ALL-Link Database. If the IM finishes sending *all* of the Cleanup messages, it will send you an *ALL-Link Cleanup Status Report*242 with a *Status Byte of* 0x06 (ACK).

For *each* ALL-Link Cleanup message that the IM sends, you will either receive an *INSTEON Standard Message Received*231 when the Responder answers with a Cleanup acknowledgement message, or else you will receive an *ALL-Link Cleanup Failure Report*241 if the Responder fails to answer with a Cleanup acknowledgement message. The IM will send you an *ALL-Link Cleanup Status Report*242 whether or not every ALL-Link Group member acknowledges the Cleanup Command that the IM sends to it.

You can cause the IM to cancel its own Cleanup sequence by sending it a new *Send ALL-Link Command*239 or *Send INSTEON Standard or Extended Message*228 during the time that it is sending a Cleanup sequence (i.e. *after* it has finished sending an ALL-Link Broadcast message). The IM *will* send you an *ALL-Link Cleanup Status Report*242 in those cases.

The IM first sends an ALL-Link Broadcast message with *Max Hops* set to 3. When it sends the ensuing ALL-Link Cleanup messages, it sets *Max Hops* to 1. If the IM's INSTEON Engine needs to retry a Cleanup message, it will automatically increment *Max Hops* for each retry, up to a maximum of value of 3.

Send ALL-Link Command (0x61)
The IM sends the ALL-Link Broadcast message immediately if there is no other INSTEON traffic. If there is other INSTEON traffic, the IM will wait for one silent powerline zero crossing following a completed INSTEON message. The IM will send the first ALL-Link Cleanup message after a delay of 7 zero crossings. Subsequent Cleanups will go out with a delay of 2 zero crossings.
Do not use this command to control light levels with the *Light Start Manual Change* INSTEON Command **SA 0x17**. Use *Send INSTEON Standard-length Message*228 to send INSTEON Command **SD 0x17** instead.
For more information on INSTEON Commands and the latest Command set, please download the current *INSTEON Command Tables Document*9 from www.insteon.net.

ALL-Link Cleanup Failure Report

ALL-Link Cleanup Failure Report (0x56)	
What it does	Reports that an ALL-Link Group member did not acknowledge an ALL-Link Cleanup Command.
When you'll get this	An ALL-Link Group member that you are trying to control did not acknowledge the ALL-Link Cleanup Command sent by the IM.
What you'll get	7 bytes.
LED indication	None.
Related Commands	IM 0x58 ALL-Link Cleanup Status Report242

Message Sent from IM to Host

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x56**	IM Command Number
3	0x01	Indicates that this ALL-Link Group member did not acknowledge an ALL-Link Cleanup Command.
4	<ALL-Link Group>	Indicates the ALL-Link Group Number that was sent in the ALL-Link Cleanup Command.
5	<ID high byte>	The high byte of the INSTEON ID of the device that did not respond.
6	<ID middle byte>	The middle byte of the INSTEON ID of the device that did not respond.
7	<ID low byte>	The low byte of the INSTEON ID of the device that did not respond.

Notes

The IM automatically sends ALL-Link Cleanup messages to each member of an ALL-Link Group following an ALL-Link Broadcast message. If the IM detects other INSTEON traffic during this process, it will abort the ALL-Link Cleanup sequence. If the Cleanup sequence is aborted, you will not receive this message nor will you receive a Cleanup acknowldgement message for any subsequent devices in the ALL-Link Group. The Cleanup sequence proceeds in the order in which the devices in the ALL-Link Group were added to the ALL-Link Database.

For each ALL-Link Cleanup message the IM sends, you will either receive an *INSTEON Standard Message Received*231 when the Responder sends you an ACK, or you will receive this message. However, it can take awhile before you receive this message. Worst case, if the IM has to wait for a clear line and then retries the Cleanup message for the maximum of five times, the wait will be 2.150 seconds after sending the ALL-Link Broadcast message, or 1.550 seconds after receiving the first Cleanup acknowledgement or this message. If the Cleanup sequence was aborted due to other INSTEON traffic, you will not get this message even then. However, you will receive *ALL-Link Cleanup Status Report*242 with a *Status Byte* of 0x15 (NAK) indicating that the Cleanup sequence was aborted.

It is possible that this ALL-Link Group member did in fact properly receive the ALL-Link Broadcast message that preceded the ALL-Link Cleanup message.

ALL-Link Cleanup Status Report

ALL-Link Cleanup Status Report (0x58)	
What it does	Notifies you if a *Send ALL-Link Command*239 completed with all Cleanup messages sent, or else if Cleanups were interrupted due to other INSTEON traffic.
When you'll get this	After you issue a *Send ALL-Link Command*239 and the IM finishes sending Cleanups to all members of the ALL-Link Group, or else when the Cleanup sequence is aborted due to other INSTEON traffic.
What you'll get	3 bytes.
LED indication	None.
Related Commands	IM 0x61 Send ALL-Link Command239 IM 0x56 ALL-Link Cleanup Failure Report241

Message Sent from IM to Host

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x58**	IM Command Number
3	<Status Byte>	<0x06> (ASCII ACK) The ALL-Link Command sequence initiated previously using *Send ALL-Link Command*239 completed. The IM first sent an ALL-Link Broadcast message, followed by ALL-Link Cleanup messages sent to all members of the specified ALL-Link Group. If any member of the ALL-Link Group does not return a Cleanup acknowledgement, you will receive an *ALL-Link Cleanup Failure Report*241 from that member.
		<0x15> (ASCII NAK) The ALL-Link Command sequence initiated previously using *Send ALL-Link Command*239 terminated before the IM sent ALL-Link Cleanup messages to all members of the specified ALL-Link Group. This is normal behavior when the IM detects INSTEON traffic from other devices.

Notes

The IM automatically sends ALL-Link Cleanup messages to each member of an ALL-Link Group following an ALL-Link Broadcast message. If the IM detects other INSTEON traffic during this process, it will abort the ALL-Link Cleanup sequence and send you this message with a *Status Byte* of 0x15 (NAK). The Cleanup sequence proceeds in the order in which the devices in the ALL-Link Group were added to the ALL-Link Database. If the IM finishes sending *all* of the Cleanup messages, it will send you this message with a *Status Byte of* 0x06 (ACK).

For *each* ALL-Link Cleanup message that the IM sends, you will either receive an *INSTEON Standard Message Received*231 when the Responder answers with a Cleanup acknowledgement message, or else you will receive an *ALL-Link Cleanup Failure Report*241 if the Responder fails to answer with a Cleanup acknowledgement message. The IM will send you *this* message whether or not every ALL-Link Group member acknowledges the Cleanup Command that the IM sends to it.

You can cause the IM to cancel its own Cleanup sequence by sending it a new *Send ALL-Link Command*239 or *Send INSTEON Standard or Extended Message*228 during the time that it is sending a Cleanup sequence (i.e. *after* it has finished sending an ALL-Link Broadcast message). The IM *will* send you this message in those cases.

ALL-Linking Session Management

Start ALL-Linking

Start ALL-Linking (0x64)	
What it does	Puts the IM into ALL-Linking mode without using the SET Button.
What you send	4 bytes.
What you'll get	5 bytes for this Command response and then an additional 10 bytes in an *ALL-Linking Completed*245 message once a successful ALL-Link has been established.
LED indication	The LED will blink continuously at a rate of ½ second on and ½ second off until the ALL-Link is completed or canceled.
Related Commands	IM 0x53 ALL-Linking Completed245 IM 0x65 Cancel ALL-Linking244

Command Sent from Host to IM

Byte	Value	Meaning	
1	0x02	Start of IM Command	
2	**0x64**	IM Command Number	
3	<Link Code>	The type of ALL-Link to establish.	
		0x00	ALL-Links the IM as a Responder (slave).
		0x01	ALL-Links the IM as a Controller (master).
		0x03	ALL-Links the IM as a Controller when the IM initiates ALL-Linking, or as a Responder when another device initiates ALL-Linking.
		0xFF	Deletes the ALL-Link.
4	<ALL-Link Group>	The ALL-Link Group Number to be linked to or deleted.	

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x64**	Echoed IM Command Number
3	<Code>	Echoed <Code>
4	<ALL-Link Group>	Echoed <ALL-Link Group>
5	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly 0x15 (NAK) if an error occurred

Cancel ALL-Linking

Cancel ALL-Linking (0x65)	
What it does	Cancels the ALL-Linking process that was started either by holding down the IM's SET Button or by sending a *Start ALL-Linking*[243] Command to the IM.
What you send	2 bytes.
What you'll get	3 bytes.
LED indication	The LED will stop blinking.
Related Commands	IM 0x64 Start ALL-Linking[243] IM 0x54 Button Event Report[260]

Command Sent from Host to IM

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x65**	IM Command Number

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x65**	Echoed IM Command Number
3	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly 0x15 (NAK) if an error occurred

ALL-Linking Completed

ALL-Linking Completed (0x53)	
What it does	Informs you of a successful ALL-Linking procedure.
When you'll get this	An ALL-Linking procedure has been completed between the IM and either a Controller or Responder.
What you'll get	10 bytes.
LED indication	None.
Related Commands	IM 0x64 Start ALL-Linking243 IM 0x65 Cancel ALL-Linking244

Message Sent from IM to Host

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x53**	IM Command Number
3	<Link Code>	Indicates the type of link made. 0x00 means the IM is a Responder (slave) to this device 0x01 means the IM is a Controller (master) of this device 0xFF means the ALL-Link to the device was deleted If done manually (by pushing the SET Button) the Controller / Responder relationship between the IM and the device is determined automatically. You can assign the Controller / Responder relationship unconditionally by using the *Start ALL-Linking*243 Command.
4	<ALL-Link Group>	Indicates the ALL-Link Group Number that was assigned to this link. If done manually (by pushing the SET Button) the ALL-Link Group Number is automatically assigned by the IM. You can assign ALL-Link Group Numbers unconditionally by using the *Start ALL-Linking*243 Command.
5	<ID high byte>	The high byte of the INSTEON ID of the device that was ALL-Linked.
6	<ID middle byte>	The middle byte of the INSTEON ID of the device that was ALL-Linked.
7	<ID low byte>	The low byte of the INSTEON ID of the device that was ALL-Linked.
8	<Device Category>	The Device Category (DevCat) of the Responder device that was ALL-Linked. (Only valid when the IM is a Controller)
9	<Device Subcategory>	The Device Subcategory (SubCat) of the Responder device that was ALL-Linked. (Only valid when the IM is a Controller)
10	<0xFF | Firmware Version>	0xFF for newer devices. For legacy devices this is the firmware version of the Responder device that was ALL-Linked. (Only valid when the IM is a Controller)

ALL-Link Database Management

Get First ALL-Link Record

Get First ALL-Link Record (0x69)	
What it does	Returns the first record in the IM's ALL-Link Database. The data will follow in an *ALL-Link Record Response* 249 message.
What you send	2 bytes.
What you'll get	3 bytes.
LED indication	None.
Related Commands	IM 0x57 ALL-Link Record Response 249 IM 0x6A Get Next ALL-Link Record 247 IM 0x6C Get ALL-Link Record for Sender 248

Command Sent from Host to IM

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x69**	IM Command Number

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x69**	Echoed IM Command Number
3	<ACK/NAK>	0x06 (ACK) if an *ALL-Link Record Response* 249 follows 0x15 (NAK) if the database is empty.

Note

Use this to begin scanning the IM's ALL-Link Database. Follow up with *Get Next ALL-Link Record* 247 Commands until you receive a NAK.

In the *IM Factory Reset State* 221 the ALL-Link Database will be cleared, so you will receive a NAK.

Get Next ALL-Link Record

Get Next ALL-Link Record (0x6A)		
What it does	Returns the next record in the IM's ALL-Link Database. The data will follow in an *ALL-Link Record Response*249 message.	
What you send	2 bytes.	
What you'll get	3 bytes.	
LED indication	None.	
Related Commands	IM 0x57 ALL-Link Record Response249 IM 0x69 Get First ALL-Link Record246 IM 0x6C Get ALL-Link Record for Sender248	
Command Sent from Host to IM		
Byte	**Value**	**Meaning**
1	0x02	Start of IM Command
2	**0x6A**	IM Command Number
Message Returned by IM to Host		
Byte	**Value**	**Meaning**
1	0x02	Echoed Start of IM Command
2	**0x6A**	Echoed IM Command Number
3	<ACK/NAK>	0x06 (ACK) if an *ALL-Link Record Response*249 follows 0x15 (NAK) if there are no more records.
Note		
Use this to continue scanning the IM's ALL-Link Database until you receive a NAK. Begin the scan up with a *Get First ALL-Link Record*246 Command.		
In the *IM Factory Reset State*221 the ALL-Link Database will be cleared, so you will receive a NAK.		

Get ALL-Link Record for Sender

Get ALL-Link Record for Sender (0x6C)	
What it does	This gets the record from the IM's ALL-Link Database for the last INSTEON message received from an INSTEON device that is in the IM's ALL-Link Database. The data will follow in an *ALL-Link Record Response*[249] message.
What you send	2 bytes.
What you'll get	3 bytes.
LED indication	None.
Related Commands	IM 0x57 ALL-Link Record Response[249] IM 0x69 Get First ALL-Link Record[246] IM 0x6A Get Next ALL-Link Record[247]

Command Sent from Host to IM

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x6C**	IM Command Number

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x6C**	Echoed IM Command Number
3	<ACK/NAK>	0x06 (ACK) if an *ALL-Link Record Response*[249] follows 0x15 (NAK) if the last INSTEON message received had a *From Address* not in the IM's ALL-Link Database.

Note

Note
If you send this after receiving an INSTEON message from an INSTEON device that is not in the IM's ALL-Link Database, you will receive a NAK in response.
Sending a *Get Next ALL-Link Record*[247] Command after this will return the ALL-Link Record that follows this one, but your actual position within the ALL-Link Database will be unknown (unless you are at the end).
In the *IM Factory Reset State*[221] the ALL-Link Database will be cleared, so you will receive a NAK.

ALL-Link Record Response

ALL-Link Record Response (0x57)	
What it does	Provides a record from the IM's ALL-Link Database.
When you'll get this	You get this when you have requested it, in response to a *Get First ALL-Link Record*[246] a *Get Next ALL-Link Record*[247] , or a *Get ALL-Link Record for Sender*[248] Command.
What you'll get	10 bytes.
LED indication	None.
Related Commands	IM 0x69 Get First ALL-Link Record[246] IM 0x6A Get Next ALL-Link Record[247] IM 0x6C Get ALL-Link Record for Sender[248]

Message Sent from IM to Host

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x57**	IM Command Number
3	<ALL-Link Record Flags>	ALL-Link Database control flags for this ALL-Link Record
4	<ALL-Link Group>	ALL-Link Group Number for this ALL-Link Record
5	<ID high byte>	INSTEON ID high byte for device ALL-Linked to
6	<ID middle byte>	INSTEON ID middle byte for device ALL-Linked to
7	<ID low byte>	INSTEON ID low byte for device ALL-Linked to
8	<Link Data 1>	Link Information (varies by device ALL-Linked to)
9	<Link Data 2>	Link Information (varies by device ALL-Linked to)
10	<Link Data 3>	Link Information (varies by device ALL-Linked to)

Note

See the section *INSTEON ALL-Link Database*[101] above for details about the contents of an ALL-Link Record.

Manage ALL-Link Record

Manage ALL-Link Record (0x6F)	
What it does	Updates the IM's ALL-Link Database with the ALL-Link Record information you send. Use caution with this Command—the IM does not check the validity of the data.
What you send	11 bytes.
What you'll get	12 bytes.
LED indication	None.
Related Commands	IM 0x57 ALL-Link Record Response[249]

Command Sent from Host to IM

Byte	Value		Meaning
1	0x02		Start of IM Command
2	**0x6F**		IM Command Number
3	<Control Code>		What to do with the ALL-Link Record
		0x00	Does an ALL-Link Record exist for this ID + ALL-Link Group? You will receive an ACK (0x06) at the end of the returned message if the ALL-Link Record exists, or else a NAK (0x15) if it doesn't. If the record exists, the IM will return it in an *ALL-Link Record Response*[249] message.
		0x01	Search for the next ALL-Link Record following the one found using Control Code 0x00 above. This allows you to find both Controller and Responder records for a given ID + ALL-Link Group. Be sure to use the same ID + ALL-Link Group (bytes 5 – 8) as you used for Control Code 0x00. You will receive an ACK (0x06) at the end of the returned message if the ALL-Link Record exists, or else a NAK (0x15) if it doesn't. If the record exists, the IM will return it in an *ALL-Link Record Response*[249] message.
		0x20	Update an ALL-Link Record with a matching <ALL-Link Group>, <ID high byte>, <ID middle byte>, <ID low byte>, and bit 6 (1 = Controller, 0 = Responder) of the <ALL-Link Record Flags> byte. If there is a matching ALL-Link Record, then the IM will update the <ALL-Link Record Flags>, <Link Data 1>, <Link Data 2>, and <Link Data 3> fields and return an ACK (0x06) at the end of the message it returns to you. If no ALL-Link Record matches, the IM will return a NAK (0x15) at the end of the message it returns to you.
		0x40	Add new Controller (master) ALL-Link Record. Returns a NAK (0x15) if the ALL-Link Record already exists.
		0x41	Add new Responder (slave) ALL-Link Record. Returns a NAK (0x15) if the ALL-Link Record already exists.
		0x80	Delete ALL-Link Record
4	<ALL-Link Record Flags>		ALL-Link Database control flags for this ALL-Link Record
5	<ALL-Link Group>		ALL-Link Group Number for this ALL-Link Record
6	<ID high byte>		INSTEON ID high byte for device ALL-Linked to
7	<ID middle byte>		INSTEON ID middle byte for device ALL-Linked to
8	<ID low byte>		INSTEON ID low byte for device ALL-Linked to
9	<Link Data 1>		Link Information: varies by device ALL-Linked to
10	<Link Data 2>		Link Information: varies by device ALL-Linked to
11	<Link Data 3>		Link Information: varies by device ALL-Linked to

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command

Manage ALL-Link Record (0x6F)		
2	**0x6F**	Echoed IM Command Number
3	<Control Code>	Echoed <Control Code>
4	<ALL-Link Record Flags>	Echoed <ALL-Link Record Flags>
5	<ALL-Link Group>	Echoed <ALL-Link group>
6	<ID high byte>	Echoed <ID high byte>
7	<ID middle byte>	Echoed <ID middle byte>
8	<ID low byte>	Echoed <ID low byte>
9	<Link Data 1>	Echoed <Link Data 1>
10	<Link Data 2>	Echoed <Link Data 2>
11	<Link Data 3>	Echoed <Link Data 3>
12	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly. 0x15 (NAK) if an error occurred, if a searched-for ALL-Link Record doesn't exist, or if an ALL-Link Record to be added already exists.
Notes		
See the section *INSTEON ALL-Link Database* 101 above for details about the contents of an ALL-Link Record.		
Please be aware that you can damage the IM's ALL-Link Database if you misuse this Command. For instance, if you use a <Control Code> of 0x20 to zero bit 1 of the <ALL-Link Record Flags> byte in the first ALL-Link Record, the ALL-Link Database will then appear empty.		

IM Status Management

Reset the IM

Reset the IM (0x67)	
What it does	Puts the IM into the *IM Factory Reset State* 221, which clears the entire ALL-Link Database.
What you send	2 bytes.
What you'll get	3 bytes.
LED indication	While the reset procedure is being processed, the Status LED will turn off. At the conclusion of the reset procedure, the Status LED will illuminate steadily.
Related Commands	IM 0x55 User Reset Detected 253

Command Sent from Host to IM

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x67**	IM Command Number

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x67**	Echoed IM Command Number
3	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly 0x15 (NAK) if an error occurred

Notes

The IM will send the <ACK/NAK> byte after it erases the EEPROM.
~20 seconds for models with external EEPROM
~2 seconds for models with no external EEPROM

See the *IM Factory Reset State* 221 section for complete information on the state of the IM after sending this Command.

User Reset Detected

User Reset Detected (0x55)	
What it does	Reports that the user manually put the IM into the *IM Factory Reset State* 221.
When you'll get this	The user held down the IM's SET Button for at least 10 seconds when power was first applied.
What you'll get	2 bytes (not until about 20 seconds after applying power to the IM with the SET Button held down).
LED indication	The LED will turn off for about 20 seconds. Once the LED turns back on the reset is complete.
Related Commands	IM 0x67 Reset the IM 252

Message Sent from IM to Host

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x55**	IM Command Number

Notes

The IM will send this message after it erases the EEPROM.
~20 seconds for models with external EEPROM
~2 seconds for models with no external EEPROM

See the *IM Factory Reset State* 221 section for complete information on the state of the IM after receiving this message.

Get IM Configuration

Get IM Configuration (0x73)	
What it does	Returns the IM's Configuration Flags byte. Also returns two spare bytes of data reserved for future use.
What you send	2 bytes.
What you'll get	6 bytes.
LED indication	None.
Related Commands	IM 0x6B Set IM Configuration 255

Command Sent from Host to IM

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x73**	IM Command Number

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x73**	Echoed IM Command Number
3	<IM Configuration Flags>	IM's Configuration Flags. See *Set IM Configuration* 255 for bit definitions.
4	<Spare 1>	0x00, reserved for future use
5	<Spare 2>	0x00, reserved for future use
6	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly 0x15 (NAK) if an error occurred

Note

Because *Set IM Configuration* 255 sets all of the <IM Configuration Flags> at once, to change an individual bit, first use this Command to determine the current state of all of the <IM Configuration Flags>.

Set IM Configuration

Set IM Configuration (0x6B)	
What it does	Allows you change operating parameters of the IM.
What you send	3 bytes.
What you'll get	4 bytes.
LED indication	None.
Related Commands	IM 0x73 Get IM Configuration[255] IM 0x54 Button Event Report[260] IM 0x50 INSTEON Standard Message Received[231] IM 0x51 INSTEON Extended Message Received[232] IM 0x6D LED On[261] IM 0x6E LED Off[262]

Command Sent from Host to IM

Byte	Value	Meaning	
1	0x02	Start of IM Command	
2	**0x6B**	IM Command Number	
3	<IM Configuration Flags>	Flag byte containing Configuration Flags that affect IM operation. These all default to 0.	
		Bit 7 = 1	Disables automatic linking when the user pushes and holds the SET Button (see *Button Event Report*[260]).
		Bit 6 = 1	Puts the IM into *Monitor Mode* (see *About Monitor Mode*[256] in the **Notes** below).
		Bit 5 = 1	Disables automatic LED operation by the IM. The host must now control the IM's LED using *LED On*[261] and *LED Off*[262].
		Bit 4 = 1	Disable host communications *Deadman* feature (i.e. allow host to delay more than 240 milliseconds between sending bytes to the IM). See *IM RS232 Port Settings*[219].
		Bits 3 - 0	Reserved for internal use. **Set these bits to 0.**

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x6B**	Echoed IM Command Number
3	<IM Configuration Flags>	Echoed <IM Configuration Flags>
4	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly. 0x15 (NAK) if an error occurred.

Notes

When the IM is in the *IM Factory Reset State*[221], the <IM Configuration Flags> will all be set to zero.

This Command sets all of the <IM Configuration Flags> at once. To change an individual bit, first use *Get IM Configuration*[255] to determine the current state of all of the <IM Configuration Flags>.

Set IM Configuration (0x6B)
About Monitor Mode Normally, the IM will only send the host an *INSTEON Standard Message Received*231 or *INSTEON Extended Message Received*232 notification when it receives an INSTEON messages directed specifically to the IM. There are three possibilities: 1. The IM received a Direct message with a *To Address* matching the IM's INSTEON ID, 2. The IM received an ALL-Link Broadcast message sent to an ALL-Link Group that the IM belongs to as a Responder (i.e. the message's *From Address* and ALL-Link Group Number match a Responder entry in the IM's ALL-Link Database), or 3. The IM received an ALL-Link Cleanup message with a *To Address* matching the IM's INSTEON ID and the message's *From Address* and ALL-Link Group Number match a Responder entry in the IM's ALL-Link Database. In *Monitor Mode*, the IM will also notify the host of received INSTEON messages that contain a *From Address* matching *any* INSTEON ID in the IM's ALL-Link Database, even if the *To Address* does not match the IM's INSTEON ID or the IM does not belong to an ALL-Link Group associated with the message. In other words, if the message originator is in the IM's ALL-Link Database as either a Controller or Responder, the IM will pass the message to the host even if it is not specifically directed to the IM. In this way you can monitor messages between other INSTEON devices as long as the sender is in the IM's ALL-Link Database. Please be aware that the IM may not always detect this traffic. If the message originator and addressee are close to one another and the IM is farther away, the message originator may not cause the message to hop enough times for the IM to hear it. To know for sure what an INSTEON device's status is, you can usually query it directly using an appropriate INSTEON Direct Command. For more information on INSTEON Commands and the latest Command set, please download the current *INSTEON Command Tables Document*9 from www.insteon.net.

Get IM Info

Get IM Info (0x60)	
What it does	Identifies the IM's 3 byte INSTEON ID, Device Category (DevCat), Device Subcategory (SubCat), and firmware version.
What you send	2 bytes.
What you'll get	9 bytes.
LED indication	None.
Related Commands	IM 0x66 Set Host Device Category258 IM 0x73 Get IM Configuration255 IM 0x6B Set IM Configuration255

Command Sent from Host to IM

Byte	**Value**	**Meaning**
1	0x02	Start of IM Command
2	**0x60**	IM Command Number

Message Returned by IM to Host

Byte	**Value**	**Meaning**
1	0x02	Echoed Start of IM Command
2	**0x60**	Echoed IM Command Number
3	<ID high byte>	IM's INSTEON ID high byte
4	<ID middle byte>	IM's INSTEON ID middle byte
5	<ID low byte>	IM's INSTEON ID low byte
6	<Device Category>	IM's Device Category
7	<Device Subcategory>	IM's Device Subcategory
8	<Firmware Version>	IM's Firmware Version
9	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly 0x15 (NAK) if an error occurred

Note

Using the *Set Host Device Category*258 Command to change the host's DevCat and SubCat will only affect the data transmitted by the IM to other INSTEON devices during ALL-Linking.
When the host sends this Command to the IM, the IM will return the original DevCat, SubCat and firmware version hard-coded into the IM's firmware at the factory.

Set Host Device Category

Set Host Device Category (0x66)	
What it does	Lets you set the Device Category (DevCat) and Device Subcategory (SubCat) of the host device connected to the IM.
What you send	5 bytes.
What you'll get	6 bytes.
LED indication	None.
Related Commands	IM 0x60 Get IM Info[257]

Command Sent from Host to IM

Byte	**Value**	**Meaning**
1	0x02	Start of IM Command
2	**0x66**	IM Command Number
3	<Device Category>	INSTEON Device Category (DevCat) of the host device connected to the IM.
4	<Device Subcategory>	INSTEON Device Subcategory (SubCat) of the host device connected to the IM.
5	<0xFF | Firmware Version>	0xFF In legacy devices this byte represented a BCD-encoded firmware version. The high nibble (4 bits) gave the major revision number and the low nibble gave the minor revision. In current devices use the INSTEON *Product Data Request* and *Product Data Response* Commands to retrieve the firmware version as user-defined data.

Message Returned by IM to Host

Byte	**Value**	**Meaning**
1	0x02	Echoed Start of IM Command
2	**0x66**	Echoed IM Command Number
3	<Device Category>	Echoed <Device Category>
4	<Device Subcategory>	Echoed <Device Subcategory>
5	<0xFF | Firmware Version>	Echoed <0xFF> or <Firmware Version>
6	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly 0x15 (NAK) if an error occurred

Notes

For INSTEON compliance, you must obtain an approved DevCat and SubCat assignment for your host product from SmartLabs.
The IM stores these values in EEPROM so they will not be erased if power is lost.
When the IM is in the *IM Factory Reset State*[221], these values will be set to those hard-coded into the IM's firmware at the factory.
Using this Command to change the host's DevCat and SubCat will only affect the data transmitted by the IM to other INSTEON devices during ALL-Linking. When the host sends a *Get IM Info*[257] Command to the IM, the IM will return the original DevCat, SubCat and firmware version hard-coded into the IM's firmware at the factory.
For the latest list of assigned INSTEON DevCats, please download the *INSTEON Device Categories and Product Keys Document*[9] from www.insteon.net.

RF Sleep

RF Sleep (0x72)	
What it does	Directs an RF IM to go into power saving sleep mode. To wake up the RF IM, send it one byte of serial data.
What you send	2 bytes.
What you'll get	3 bytes.
LED indication	None.
Related Commands	None.

Command Sent from Host to IM

Byte	Value	Meaning
1	0x02	Start of IM Command
2	**0x71**	IM Command Number
3	<Command 1 Data>	Data byte to place into the *Command 1* field 2 of the ACK response.
4	<Command 2 Data>	Data byte to place into the *Command 2* field 2 of the ACK response.

Message Returned by IM to Host

Byte	Value	Meaning
1	0x02	Echoed Start of IM Command
2	**0x71**	Echoed IM Command Number
3	<Command 1 Data>	Echoed <Command 1 Data>
4	<Command 2 Data>	Echoed <Command 2 Data>
5	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly. 0x15 (NAK) if an error occurred.

Notes
It does not matter what byte you send serially to wake up the RF IM.
When the RF IM wakes up, it will reinitialize, but memory will not be altered as it would be in the *IM Factory Reset State* 221. Wait a minimum of 40 milliseconds before sending any further IM Serial Commands.

IM Input/Output

Button Event Report

Button Event Report (0x54)	
What it does	Reports user SET Button events.
When you'll get this	The user operates the SET Button, or if they exist, Button 2 or Button 3.
What you'll get	3 bytes.
LED indication	If the event is *SET Button Press and Hold* the IM will automatically go into ALL-Linking mode which will cause the LED to blink continuously at a rate of ½ second on and ½ second off. Automatic linking may be turned off by setting *IM Configuration Flags* bit 7 (see *Set IM Configuration*255).
Related Commands	IM 0x53 ALL-Linking Completed245 IM 0x64 Start ALL-Linking243 IM 0x65 Cancel ALL-Linking244

Message Sent from IM to Host

Byte	Value	Meaning	
1	0x02	Start of IM Command	
2	**0x54**	IM Command Number	
3	<Button Event>	Indicates the type of SET Button event that occurred.	
		0x02	The SET Button was *Tapped*
		0x03	There was a SET Button *Press and Hold* for more than three seconds. This automatically puts the IM into ALL-Linking mode unless *IM Configuration Flags* bit 7 is set.
		0x04	The SET Button was released after a SET Button *Press and Hold* event was recorded.
		0x12	Button 2 was *Tapped*
		0x13	There was a Button 2 *Press and Hold* for more than three seconds.
		0x14	Button 2 was released after a Button 2 *Press and Hold* event was recorded.
		0x22	Button 3 was *Tapped*
		0x23	There was a Button 3 *Press and Hold* for more than three seconds.
		0x24	Button 3 was released after a Button 3 *Press and Hold* event was recorded.

LED On

LED On (0x6D)	
What it does	Turns on the IM's LED if *IM Configuration Flags* bit 5 = 1.
What you send	2 bytes.
What you'll get	3 bytes.
LED indication	The LED will go on.
Related Commands	IM 0x6B Set IM Configuration 255 IM 0x6E LED Off 262

Command Sent from Host to IM		
Byte	**Value**	**Meaning**
1	0x02	Start of IM Command
2	**0x6D**	IM Command Number

Message Returned by IM to Host		
Byte	**Value**	**Meaning**
1	0x02	Echoed Start of IM Command
2	**0x6D**	Echoed IM Command Number
3	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly. 0x15 (NAK) if an error occurred or *IM Configuration Flags* bit 5 = 0.

LED Off

LED Off (0x6E)	
What it does	Turns off the IM's LED if *IM Configuration Flags* bit 5 = 1.
What you send	2 bytes.
What you'll get	3 bytes.
LED indication	The LED will go off.
Related Commands	IM 0x6B Set IM Configuration 255 IM 0x6D LED On 261

Command Sent from Host to IM		
Byte	**Value**	**Meaning**
1	0x02	Start of IM Command
2	**0x6E**	IM Command Number

Message Returned by IM to Host		
Byte	**Value**	**Meaning**
1	0x02	Echoed Start of IM Command
2	**0x6E**	Echoed IM Command Number
3	<ACK/NAK>	0x06 (ACK) if the IM executed the Command correctly. 0x15 (NAK) if an error occurred or *IM Configuration Flags* bit 5 = 0.

Chapter 11 — SALad Language Documentation

The INSTEON PowerLinc™ V2 Controller (PLC) and other planned INSTEON devices contain an embedded language interpreter, called SALad, that allows programming of complex behavior into SALad-enabled devices. See *SALad Applications*33 in *Chapter 4 — INSTEON Application Development Overview*27 for a description of the application development process using SALad.

The SALad language is designed to make execution of INSTEON Internal Applications fast, while keeping the size of the programs small.

SALad is event driven. Examples of events that can occur in a PLC include reception of an INSTEON message or an X10 Command, expiration of a timer, or pushing the *SET Button*. As events occur, the PLC firmware posts event handles to an event queue. The firmware then starts the SALad program with the current event handle located in a specific memory location called NTL_EVENT. The SALad program inspects NTL_EVENT in order to determine what action to take based on the event that occurred. Most SALad programming is just a matter of writing event-handling routines, or modifying the routines found in sample applications.

The SALad Integrated Development Environment (IDE) makes writing and debugging SALad programs fast and easy. Besides a built-in SALad editor, compiler, and debugger, the IDE contains an integrated set of INSTEON-specific tools that give the programmer access to every aspect of the INSTEON environment.

In This Chapter

SALad Programming Guide

In This Section

Structure of a SALad Program

Application Header

All SALad applications require a program header for program verification. This header can have many pieces of information in it, but it must contain the application verification data.

Starting at address 0x0210 (see *Flat Memory Map* 170), there are 8 bytes of data that define a region of code that will not be altered during execution. This is used to test the application for possible corruption.

Address	Register and Bits	Description
0x0210 ⇒ 0x0211	APP_ADDR_TEST	Address of range of application for verification test
0x0212 ⇒ 0x0213	APP_LEN_TEST	Length of range of application for verification test
0x0214 ⇒ 0x0215	APP_CHECK_TEST	Two's complement checksum of range of application for verification test
0x0216 ⇒ 0x0217	APP_END	Top of currently loaded SALad application. EEPROM is write-protected from 0x0200 to the address contained here. Set 0x16B bit 7 to enable over-writing.

APP_ADDR_TEST is the address of the beginning of the application code, normally 0x0230. *APP_LEN_TEST* is the length of the region to be tested on device initialization, normally the length of the application. *APP_CHECK_TEST* is a two's complement checksum of that region. If you are using the SALad IDE, it will fill in this number for you. *APP_END* is the address of the last byte-plus-one in the currently loaded application, and is used to write-protect the EEPROM code segment (see *Overwriting a SALad Application* 274).

When the PLC is reset, IBIOS uses the checksum to verify the SALad program before running it. If it is corrupt, IBIOS will flash the PLC's LED at about 2 Hz. If the application is valid, the LED will be lit continuously.

An Application Header structure using literal values might look like this:

```
ORG 0x210
  DATA 0x0230 ; address of beginning of application
  DATA 0x000a ; length of application
  DATA 0x0041 ; checksum of verification region
  DATA 0x023b ; end of application
```

If you are using the SALad IDE, you can use labels and skip filling in the checksum, like this:

```
ORG 0x0210
;=======Application Header================================
        DATA    Start          ;Beginning of application
        DATA    App_End-Start  ;Length of application
        DATA    0x00, 0x00     ;Two's complement checksum
        DATA    App_End        ;End of application
```

Program Body

The general structure of a SALad program is similar to that of other low-level assembly programming languages, with varying details depending on the application. As a stand-alone language, SALad has no specific structural requirements. However, most INSTEON applications are event-driven, requiring that SALad event handlers follow a definite structural organization.

For a simple direct SALad application, start the program at 0x0237, which is the standard entry vector for Static *IBIOS Events*185. When the PLC is reset (either by power cycling or a reset Command), the SALad application will be started with an initialization **EVNT_INIT (0x00)** IBIOS Event posted to the event queue.

It is possible to write a SALad application without event handlers, but you must then poll all hardware for a change of state, and *SALad Timers*273 will not work without event processing.

By far the easiest way to write SALad applications is use the IDE as explained in the *SALad Integrated Development Environment User's Guide*287 to modify the event handlers in the open-source *SALad coreApp Program*272.

The SALad Version of Hello World

The SALad language contains general Commands that are useful for most SALad-enabled INSTEON products. Any commands that are unique to a subset of the product line are provided through the API (Application Program Interface) feature. This allows custom firmware and commands to be added to the SALad language without altering the core SALad engine.

In this example, the RS232 port is utilized to send a `Hello World!` ASCII message. Because the RS232 port is not found in all the SmartLabs products, these c are provided as API calls. In this case, we will use the `API_RS_SendStr` command which has a command code of 0x82. The format of this command is the API code (0x0A) followed by the command code (0x82) followed by the 2-byte address of the data containing the message to send.

When this example is executed, the output will be to the RS232 Port at 4800 Baud, 8 bits, 1 stop bit, and no parity. The application looks like this:

```
ORG 0x0210
;=======Application Header================================
        DATA    Start           ;Beginning of application
        DATA    App_End-Start   ;Length of application
        DATA    0x00, 0x00      ;Two’s complement checksum
        DATA    App_End         ;End of application

ORG 0x0237
;=======Application Code==================================
Start
        API     0x82, Message   ;Send Message to RS232 port
        JUMP    Start           ;Loop non-stop
Message
        DATA    "Hello "
        DATA    "World!"
        DATA    0x0D,0x0A,0x00 ;Zero terminates string
App_End
```


SALad Event Handling

As events occur, IBIOS posts messages to a message queue that the SALad application can respond to for processing. When SALad is idle, the event processor continually looks for new events to process. When one appears, IBIOS puts the Event Handle in the *NTL_EVENT* register and then calls SALad at one of two possible vectors:

0x0230 – for Timer Events (see *SALad Timers* 273)

0x0237 – for Static Events (see *IBIOS Events* 185)

Upon entry, the SALad application must process the event handle in *NTL_EVENT* in order to run the correct event handler. In the following code, taken from the *SALad coreApp Program* 272, the event handle is used as an index into a table of addresses that point to the beginning of the event handling routines.

```
;=======Register Definitions==============================
APP_TMP_H      EQU     0x006E ;These stay alive only until
APP_TMP_L      EQU     0x006F ;the first CALL.  They are
                              ;used for the Event handler.

ORG 0x0210
;=======Application Header================================
        DATA    Start          ;Beginning of application
        DATA    App_End-Start  ;Length of application
        DATA    0x00, 0x00     ;Two's complement checksum

ORG 0x0230
;=======Event Process Code================================
Start
StartTimer
        JUMP    ProcessTimer          ;jump to Timer process

StartEvent
        MOVE    NTL_EVENT, APP_TMP_L  ;setup event offset
        MUL     #0x0002, APP_TMP_H    ;multiply by 2 for
                                      ;word offset (16bit)
        ADD     #EventJmpTbl, APP_TMP_H ;add table address 0243:
                                      ;to offset (16bit)
        JUMP    ProcessEvent          ;process table entry

ProcessTimer
        MOVE    NTL_EVENT, APP_TMP_L  ;setup event offset
        MUL     #0x0002, APP_TMP_H    ;multiply by 2 for
                                      ;word offset (16bit)
        ADD     #EventJmpTbl,APP_TMP_H ;add table address 0256:
                                      ;to offset (16bit)
ProcessEvent
         MOVE$  @APP_TMP_H,APP_TMP_H,2 ;get indirect pointer
                                      ;from table
        JUMP    @APP_TMP_H            ;execute code at table
                                      ;entry
EventJmpTbl
        DATA    Event00               ;EVNT_INIT
        DATA    Event01               ;EVNT_IRX_MSG
        DATA    Event02               ;EVNT_IRX_NACK
        DATA    Event03               ;EVNT_XRX_MSG
        DATA    Event04               ;EVNT_XRX_XMSG
        DATA    Event05               ;EVNT_BTN_TAP
        DATA    Event06               ;EVNT_BTN_HOLD
        DATA    Event07               ;EVNT_BTN_REL
        DATA    Event08               ;EVNT_ALARM
        DATA    Event09               ;EVNT_MIDNIGHT
```



```
        DATA    Event0A                 ;EVNT_2AM
        DATA    Event0B                 ;EVNT_SRX_COM

TimerJmpTbl
        DATA 0x00, 0x00                 ;No Timer Events

;=======Static Events==============================
;-------EVNT_INIT
Event00
        END

;-------EVNT_IRX_MSG
Event01
        END

;-------EVNT_IRX_NACK
Event02
        END

;-------EVNT_XRX_MSG
Event03
        END

;-------EVNT_XRX_XMSG
Event04
        END

;-------EVNT_BTN_TAP
Event05
        END

;-------EVNT_BTN_HOLD
Event06
        END

;-------EVNT_BTN_REL
Event07
        END

;-------EVNT_ALARM
Event08
        END

;-------EVNT_MIDNIGHT
Event09
        END

;-------EVNT_2AM
Event0A
        END

;-------EVNT_SRX_COM  - Serial Received a command
Event0B
        END

;======Timer Events================================
;       No Timer Events
App_End
```


Hello World 2 – Event Driven Version

The following example shows the "Hello World" program implemented as an event driven process. This version prints "Hello World!" out the RS232 port each time the *SET Button* is pressed. This demonstrates the processing of the **EVNT_BTN_TAP (0x0A)** and **EVNT_BTN_HOLD (0x0B)** *IBIOS Events*185 that are generated when the button is tapped or held. If the button is pressed for more than 350 ms, the program will send "Good-Bye World!" instead.

When this example is executed, the output will be to the RS232 Port at 4800 Baud, 8 bits, 1 stop bit, and no parity. The program is a simple modification of the *SALad coreApp Program*272 event processor shown in *SALad Event Handling*268.

```
;=======Register Definitions==============================
APP_TMP_H      EQU     0x006E  ;These stay alive only until
APP_TMP_L      EQU     0x006F  ;the first CALL.  They are
                               ;used for the Event handler.


ORG 0x0210
;=======Application Header================================
       DATA    Start           ;Beginning of application
       DATA App_End-Start      ;Length of application
       DATA    0x00, 0x00      ;Two's complement checksum


ORG 0x0230

;=======Event Process Code================================
Start
StartTimer
       JUMP    ProcessTimer            ;jump to Timer process

StartEvent
       MOVE    NTL_EVENT, APP_TMP_L    ;setup event offset
        MUL    #0x0002, APP_TMP_H      ;multiply by 2 for
                                       ;word offset (16bit)
       ADD     #EventJmpTbl, APP_TMP_H ;add table address 0243:
                               ;to offset (16bit)
       JUMP    ProcessEvent            ;process table entry
ProcessTimer
       MOVE    NTL_EVENT, APP_TMP_L    ;setup event offset
       MUL     #0x0002, APP_TMP_H      ;multiply by 2 for
                                       ;word offset (16bit)
       ADD     #EventJmpTbl,APP_TMP_H ;add table address 0256:
                               ;to offset (16bit)
ProcessEvent
MOVE$   @APP_TMP_H,APP_TMP_H,2 ;get indirect pointer
                                       ;from table
       JUMP    @APP_TMP_H              ;execute code at table
                                       ;entry
EventJmpTbl
       DATA    Event00                 ;EVNT_INIT
       DATA    Event01                 ;EVNT_IRX_MSG
       DATA    Event02                 ;EVNT_IRX_NACK
       DATA    Event03                 ;EVNT_XRX_MSG
       DATA    Event04                 ;EVNT_XRX_XMSG
       DATA    Event05                 ;EVNT_BTN_TAP
       DATA    Event06                 ;EVNT_BTN_HOLD
       DATA    Event07                 ;EVNT_BTN_REL
       DATA    Event08                 ;EVNT_ALARM
       DATA    Event09                 ;EVNT_MIDNIGHT
       DATA    Event0A                 ;EVNT_2AM
       DATA    Event0B                 ;EVNT_SRX_COM
TimerJmpTbl
```



```
        DATA 0x00, 0x00              ;No Timer Events

;=======Static Events=============================
;-------EVNT_INIT
Event00
        END
;-------EVNT_IRX_MSG
Event01
        END
;-------EVNT_IRX_NACK
Event02
        END
;-------EVNT_XRX_MSG
Event03
        END
;-------EVNT_XRX_XMSG
Event04
        END
;-------EVNT_BTN_TAP
Event05
        API     0x82, Message1 ;Send Message to RS232 port
        END
;-------EVNT_BTN_HOLD
Event06
        API     0x82, Message2 ;Send Message to RS232 port
        END
;-------EVNT_BTN_REL
Event07
        END
;-------EVNT_ALARM
Event08
        END
;-------EVNT_MIDNIGHT
Event09
        END
;-------EVNT_2AM
Event0A
        END

;-------EVNT_SRX_COM  - Serial Received a command
Event0B
        END

;======Timer Events===============================
;       No Timer Events
Message1
        DATA    "Hello "
        DATA    "World!"
        DATA    0x0D,0x0A,0x00 ;Zero terminates string
Message2
        DATA    "Goodbye "
        DATA    "World!"
        DATA    0x0D,0x0A,0x00 ;Zero terminates string
App_End
```

SALad coreApp Program

As shipped by SmartLabs, the PowerLinc™ V2 Controller (PLC) contains a 1200-byte SALad program called coreApp that performs a number of useful functions:

- When coreApp receives messages from INSTEON devices, it sends them to the computing device via its serial port, and when it receives INSTEON-formatted messages from the computing device via the serial port, it sends them out over the INSTEON network.
- CoreApp handles ALL-Linking to other INSTEON devices and maintains a *Threaded ALL-Link Database (ALDB/T)*105.
- CoreApp is event-driven, meaning that it can send messages to the computing device based on *IBIOS Events*185 and *SALad Timers*273.
- CoreApp can send and receive X10 Commands.
- CoreApp sets the software realtime clock using the hardware realtime clock and handles daylight savings time.

Several of the IBIOS Events listed in the *IBIOS Event Summary Table*185 and IBIOS Serial Commands listed in the *IBIOS Serial Command Summary Table*197 *require* SALad event handlers like those in coreApp in order to ensure realtime execution.

Source code for coreApp is available to developers to modify for their own purposes. Using the tools described in the *SALad Integrated Development Environment User's Guide*287 to modify coreApp is by far the easiest way to develop your own SALad applications. Once programmed with an appropriately modified SALad App, the PLC can operate on its own without being connected to a computing device.

SALad Timers

You can set up a SALad Timer Event by loading 2 bytes into a timer buffer. The first byte, called the *Timer Index*, is the number of the timer handler routine that you want to execute when the Timer Event fires. The second byte, called the *Timer Time*, designates how much time you want to elapse from the time you set up the Timer Event until the Timer Event fires. If the high bit of *Timer Time* is 0, the other 7 bits designate 1 to 127 seconds; if the high bit is 1, the other 7 bits designate 1 to 127 minutes. A *Timer Time* of 0x00 designates that the timer is disabled.

The default number of co-pending Timer Events that you can set up is four. If you need more Timer Events, increase the pointer value stored in *NTL_BUFFER* at 0x0033 (see *Flat Memory Map* 170) by two for each additional Timer Event you want to support. *NTL_BUFFER* points to the *end* of the timer buffer, which *begins* at 0x0046.

You need to write a SALad timer handler routine for each *Timer Index* that you will be using. *Timer Index 1* will fire the first handler, *Timer Index 2* will fire the second handler, and so on.

There are four SALad instructions for setting up and removing Timer Events (see *Two-byte SALad Instructions* 283):

ONESHOT (*Timer Index*, *Timer Time*)
Set up a new Timer Event if there is no pre-existing Timer Event with the same *Timer Index*. If there is such a Timer Event, replace its *Timer Time* value. After this Timer Event fires, remove it by setting its *Timer Index* to 0x00.

TIMER (*Timer Index*, *Timer Time*)
Set up a new Timer Event if there is no pre-existing Timer Event with the same *Timer Index*. If there is such a Timer Event, replace its *Timer Time* value. After this Timer Event fires, disable it by setting its *Timer Time* to 0x00, but do not remove it.

TIMERS (*Timer Index*, *Timer Time*)
Set up a new Timer Event unconditionally. After this Timer Event fires, disable it by setting its *Timer Time* to 0x00, but do not remove it.

KILL (*Timer Index*)
If there is a pre-existing Timer Event with the same *Timer Index*, remove it by setting its *Timer Index* to 0x00.

If you try to set up a Timer Event but there is no room in the timer buffer, the Timer Event will not be set up. If this happened, the *_NTL_BO* buffer overrun flag (bit 1) of *NTL_STAT* at 0x0075 will be 1.

SALad Remote Debugging

It is possible to debug SALad applications remotely using either Serial (RS232 or USB) or INSTEON communications. IBIOS provides the necessary support. See *IBIOS Remote Debugging*215 for a full explanation of how this works at the low level.

The comprehensive debugging features built into the SALad IDE (see the *SALad Integrated Development Environment User's Guide*287) are built on this low level IBIOS mechanism. Therefore, the easiest way to take advantage of remote SALad debugging is to use the SALad IDE.

Overwriting a SALad Application

SALad code in EEPROM is write-protected from 0x0200 to the top of the SALad application pointed to by *APP_END*, as shown in the *Flat Memory Map*170 excerpt below. You must set the *_RS_AppLock* flag before you attempt to write to this area.

i1 Addr	i2 Addr	Register and Bits		Description
0x016B	0x0167	RS_CONTROL		Control flags for serial command interpreter
		_RS_AppLock	3	_RS_AppLock
0x0216 ⇒ 0x0217	0x0216 ⇒ 0x0217	APP_END		Top of currently loaded SALad application. EEPROM is write-protected from 0x0200 to the address contained here. Set 0x16B bit 7 to enable over-writing.

This mechanism is not implemented in PLC firmware versions earlier than 2.10K.

Preventing a SALad Application from Running

While developing a SALad application that runs on the PLC, you may need to manually prevent the SALad code from executing, because of faulty SALad code that prevents serial communication.

To prevent SALad execution while allowing IBIOS to run normally, remove and then re-apply power to the PLC while holding down the *SET Button* for 5 seconds. IBIOS will then write the *complement* of the SALad program's checksum to the SALad checksum verification register. Since the SALad program's checksum will not match the complemented checksum, the SALad program will not run.

To restore the SALad checksum verification register to its correct value, just complement it again by repeating the 'apply power while holding the *SET Button* for 5 seconds' procedure.

SALad Language Reference

In This Section

SALad Memory Addresses

SALad uses the same *Flat Memory Addressing*[168] as IBIOS. See the *Flat Memory Map*[170] for a table of important memory locations.

See *Structure of a SALad Program*[265] for the location of the SALad program itself and the structure of its header.

SALad program flags appear in the register *NTL_STAT* as follows:

i1 Addr	i2 Addr	Register and Bits		Description
0x0075	0x0175	NTL_STAT		SALad Status Register
		_DB_END	4	_DB_END
		_DB_PASS	3	_DB_PASS
		_NTL_DZ	2	_NTL_DZ
		_NTL_BO	1	_NTL_BO
		_NTL_CY	0	_NTL_CY

SALad Instruction Set

The SALad instructions are designed to support all the processing needs of a basic programming language. Additional device-specific functions can be defined using Application Programming Interface (API) calls to firmware.

SALad universally supports access to or from RAM or EEPROM directly or indirectly. Literal data can be provided for immediate operations. The *Compare* and *Move* instructions have a block mode to perform string compares or block moves.

Indirect addressing modes support address pointers and jump tables.

In This Section

SALad Universal Addressing Module (UAM)

The UAM is a mechanism for encoding addressing mode and parameter information in the instructions. The commands have UAM-specific information in the low nibble of the instruction. These bits define what kind of memory is being accessed and the number of parameter bytes.

SALad instructions use Full-UAM, Half-UAM, or No-UAM encodings. Full-UAM instructions take two parameters, Half-UAM instructions take one parameter, and No-UAM instructions do not use the UAM at all and have no parameters.

These tables show how SALad instructions are UAM-encoded in a byte:

Full-UAM SALad Instruction							
Command				**Source Parameter**	**Destination Parameter**	**Mode**	
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
4-bit command identifier				0=8 bit Parameter 1=16 bit Parameter	0=8 bit Parameter 1=16 bit Parameter	00 = Direct to Direct 01 = Direct to Indirect 10 = Indirect to Direct 11 = Literal to Direct	

Half-UAM SALad Instruction							
Command							**Mode**
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
7-bit command identifier							0=Direct 1=Indirect

No-UAM SALad Instruction							
Command							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
8-bit command identifier							

SALad Parameter Encoding

Parameters are of types:

- Register
- Direct
- Indirect
- 8-bit Literal Value
- 16-bit Literal Value

A Register parameter is an 8-bit location in memory that is referenced by an offset that is added to a base value stored in NTL_EVENT.

A Direct parameter is a parameter that is referenced by a 16-bit address in the *Flat Memory Map* 170.

An Indirect parameter is a pointer to a location in memory.

Literal values are constant values that are coded directly into the program.

In This Section

Parameter Reference Tables

Parameter Types

This table shows the types of parameters described above, whether they are 8 or 16-bit, whether they refer to addresses relative to the Program Counter or relative to the Absolute address in the *Flat Memory Map*170, and whether they are Indirect or Direct.

Ref	Description	8-bit or 16-bit	Relative or Absolute	Indirect or Direct
Rn	Direct Register Mode (n + NTL_REG)	8-bit	Absolute	Direct
@Rn	Indirect Register Mode (n + NTL_REG)	8-bit	Absolute	Indirect
D	Direct Mode (PC + D)	16-bit	Relative	Direct
@D	Indirect Mode (PC + D)	16-bit	Relative	Indirect
#8	8-bit Literal	8-bit	Absolute	Direct
#16	16-bit Literal	16-bit	Absolute	Direct

Full-UAM Instruction Encoding

This table shows how to encode the low nibble for full UAM instructions (i.e. instructions that take both source and destination parameters). Parameter references are from the table above.

Command and Parameters	Instruction Code	Command and Parameters	Instruction Code
Command Rn, Rn	0x?0	Command D, Rn	0x?8
Command Rn, @Rn	0x?1	Command D, @Rn	0x?9
Command @Rn, Rn	0x?2	Command @D, Rn	0x?A
Command #8, Rn	0x?3	Command #16, Rn	0x?B
Command Rn, D	0x?4	Command D, D	0x?C
Command Rn, @D	0x?5	Command D, @D	0x?D
Command @Rn, D	0x?6	Command @D, D	0x?E
Command #8, D	0x?7	Command #16, D	0x?F

Half-UAM Instruction Encoding

This table shows how to encode half UAM instructions (i.e. instructions that take only a destination parameter). Parameter references are from the table above. Instruction Code is shown as 8-bit pattern.

Command and Parameters	Instruction Code	Command and Parameters	Instruction Code
Command D	xxxxxxx0	Command @D	xxxxxxx1

SALad Instruction Summary Table

One-byte SALad Instructions

Command	Code	UAM	Parameters	Description
NOP	0x00	None		No Operation
RET	0x01	None		Return from call
END	0x02	None		Return to System
Function	0x03	None		Enable extended function mode
API	0x04	None		Execute external firmware routines
8BIT	0x05	None		Select 8 bit processing
16BIT	0x06	None		Select 16 bit processing
HALT	0x07	None		Halt SALad execution
RL	0x08⇒0x09	Half-UAM	<dest>	Rotate Left value stored in dest by 1-bit
RR	0x0A⇒0x0B	Half-UAM	<dest>	Rotate Right value stored in dest by 1-bit
JUMP	0x0C⇒0x0D	Half-UAM	<dest>	Jump to specified destination relative to PC
CALL	0x0E⇒0x0F	Half-UAM	<dest>	Call routine at specified destination relative to PC
TEST	0x10⇒0x1F	Half-UAM	<dest><jump>	Test bit in byte at destination, jump if false
CLR	0x20⇒0x2F	Half-UAM	<dest>	Clear bit in byte at destination
SET	0x30⇒0x3F	Half-UAM	<dest>	Set bit in byte at destination
ADD	0x40⇒0x4F	Full-UAM	<source><dest>	Add source to destination, store result in destination
OR	0x50⇒0x5F	Full-UAM	<source><dest>	OR source to destination, store result in destination
AND	0x60⇒0x6F	Full-UAM	<source><dest>	AND source to destination, store result in destination
XOR	0x70⇒0x7F	Full-UAM	<source><dest>	XOR source to destination, store result in destination
COMP>	0x80⇒0x8F	Full-UAM	<source><dest><jump>	Compare source greater to destination, jump if false
COMP<	0x90⇒0x9F	Full-UAM	<source><dest><jump>	Compare source less than destination, jump if false

Command	Code	UAM	Parameters	Description
LOOP-	0xA0⇒0xAF	Full-UAM	<source><dest><jump>	Decrement destination, branch while greater than source
LOOP+	0xB0⇒0xBF	Full-UAM	<source><dest><jump>	Increment destination, branch while less than source
MOVE	0xC0⇒0xCF	Full-UAM	<source><dest>	Move source to destination
COMP=	0xD0⇒0xDF	Full-UAM	<source><dest><jump>	Compare source equal to destination, jump if false
SUB	0xE0⇒0xEF	Full-UAM	<source><dest>	Subtract source from destination, store result in destination

Two-byte SALad Instructions

These are Extended Commands, preceded by 0x03.

Command	Code	UAM	Parameters	Description
ENROLL	0x03 0x00	None		Enable Enrollment for 2 minutes Starts 2-minute enrollment timer and enables Enrollment command pass-through. a. Button must be pushed when this command is executed b. Timer is restarted when valid enrollment command is received c. Enrollment command generates a unique event: EVNT_IRX_ENROLL or 0x07
NEXT	0x03 0x01	None		Find next ALL-Link Group in ALL-Link Database
SEND	0x03 0x02	None		Send INSTEON
ENDPROC	0x03 0x03	None		Skip parameter on stack and return
KILL	0x03 0x04	None	<index>	Delete pending timer event specified by index from event queue
PAUSE	0x03 0x05	None	<time>	Pause for time in 25ms increments

<table>
<tr><th>Command</th><th>Code</th><th>UAM</th><th>Parameters</th><th>Description</th></tr>
<tr><td>FIND</td><td>0x03 0x06</td><td>None</td><td><flags></td><td>Find record in database. These bits define the field(s) of the record that you are searching for in the Enrollment database.

Flags is defined as:

DB_FLAGS
76543210
XXXXXXXX
ID0-------------+|||||||
ID1--------------+||||||
ID2---------------+|||||
Group--------------+||||
Linear Search-------+|||
Mode-----------------++|
INSTEON Master/Slave---+

Search Mode (Mode bits):

00 Deleted
01 Other
10 INSTEON Slave
11 INSTEON Master

DB_Flags configuration examples:

EMPTY EQU 0x00 ; look for empty slot
; index is in DB_H

SLAVE EQU 0xE4 ; match ID, look for
; SLAVE, ID is in
; RxFrom0-RxFrom2

MASTER EQU 0xE6 ; match ID, look for
; MASTER, ID is in
; RxFrom0-RxFrom2

INSTEON EQU 0xE5; match ID, look for
; MASTER or SLAVE,
; ID is in RxFrom0-
; RxFrom2

MEMBER EQU 0x1C ; match group, look
; for SLAVE, index
; is in DB_0

GROUP EQU 0xF6 ; match ID and group,
; look for MASTER, ID
; is in RxFrom0-
; RxFrom2, group is
; in RxTo2

To Find a member of a local group, set group number in DB_3 and execute:

FIND MEMBER

To Find either a Master or Slave INSTEON record that matches the received message, execute:

FIND INSTEON</td></tr>
<tr><td>X10</td><td>0x03 0x08</td><td>None</td><td><HC/UC><HC/com></td><td>Send X10 message</td></tr>
</table>

Command	Code	UAM	Parameters	Description
LED	0x03 0x09	None	<pattern><time>	Flash LED; Pattern defines the blinking pattern, and time specifies the duration of the blinking from 0 to 255 seconds. For example: LED 0x55 0x0A would make the LED blink at 8Hz for 10 seconds; The delay between blinks can be controlled by writing to LED_DLY
RANDOM	0x03 0x0A	None	<limit><register>	Generates random number between 0 and *limit* and stores in *register*
RANDOMIZE	0x03 0x0B	None	<dest>	Get next random number from generator using 16 bit absolute address to an 8-bit seed value and stores in NTL_RND
ONESHOT	0x03 0x0C	None	<index><time>	Set One-Shot timer: Index is the event number that the firmware stores in NTL_EVENT when the timer expires; Time is 0⇒127 seconds, or 2 minutes 8 seconds ⇒ 130 minutes 8 seconds if MSb is set.
TIMER	0x03 0x0D	None	<index><time>	Set or Reset timer
TIMERS	0x03 0x0E	None	<index><time>	Set multiple timers
Undefined	0x03 0x0F	None		
X10EXT	0x03 0x10⇒ 0x03 0x11	Half-UAM	<dest>	Additional data for extended X10 message
SEND$	0x03 0x20 ⇒ 0x03 0x21	Half-UAM	<dest>	Send 9 byte INSTEON message located at destination (from ID is inserted automatically)
SENDEXT$	0x03 0x30 ⇒ 0x03 0x31	Half-UAM	<dest>	Send 23 byte INSTEON message located at destination (from ID is inserted automatically)
MUL	0x03 0x50⇒ 0x03 0x5F	Full-UAM	<source><dest>	Multiply source to destination, store result in destination and destination +1
DIV	0x03 0x60⇒ 0x03 0x6F	Full-UAM	<source><dest>	Divide source into destination, store result in destination
MOD	0x03 0x70⇒ 0x03 0x7F	Full-UAM	<source><dest>	Divide source into destination, store remainder in destination
PROC	0x03 0x80⇒ 0x03 0x8F	Full-UAM	<source><dest><jump>	Place source on stack and call destination, jump if _NTL_CY=0 upon return
MASK	0x03 0x90⇒ 0x03 0x9F	Full-UAM	<source><dest><jump>	AND source to destination, jump if zero
COMP$>	0x03 0xA0⇒ 0x03 0xAF	Full-UAM	<source><dest><len> <jump>	Compare source string greater to destination string, jump if false
COMP$<	0x03 0xB0⇒ 0x03 0xBF	Full-UAM	<source><dest><len> <jump>	Compare source string less than destination string, jump if false
TJUMP	0x03 0xC0⇒ 0x03 0xCF	Full-UAM	<source><dest><limit> source: <index> dest: <base address> limit: <end of table>	Increment destination, branch while less than source

Command	Code	UAM	Parameters	Description
TCALL	0x03 0xD0⇒ 0x03 0xDF	Full-UAM	<source><dest><limit> source: <index> dest: <base address> limit: <end of table>	Decrement destination, branch while greater than source

SALad Integrated Development Environment User's Guide

The SALad Integrated Development Environment (IDE) is a powerful tool for developing applications to run on SALad-enabled INSTEON devices. For information about the SALad Language, refer to the *SALad Programming Guide*264 and *SALad Language Reference*275 sections above.

The SALad IDE's source code editor handles multiple files, using color to indicate code context. In debug mode, programmers can use single-stepping, breakpoints, tracing, and watches to find and fix coding errors quickly.

At the heart of the IDE is The SALad Compiler, which reads SALad language source files and creates SALad object code, along with an error listing and symbol map. Compiled object code can either be serially downloaded to a *real* SmartLabs PowerLinc™ V2 Controller (PLC) plugged into the powerline, or else it can be run on a *virtual* PLC simulated in software. Besides the virtual PLC, the IDE can also simulate a virtual powerline environment with any number of virtual LampLinc™ devices plugged into it, so that developers can create and test complex SALad applications on a standalone PC before validating them in a real INSTEON environment.

Using an integrated set of INSTEON-specific tools, programmers can compose and monitor INSTEON, X10, ASCII, or raw data messages, simulate PLC or realtime clock/calendar events, and directly manipulate the PLC's ALL-Link Database, all without ever leaving the IDE.

In This Section

SALad IDE Quickstart

Follow these steps to get familiar with the IDE quickly. The IDE works with a SmartLabs PowerLinc™ V2 Controller, called a PLC for short. You can either use a real PLC plugged into the powerline and connected to your PC via a USB or RS232 serial port, or else the IDE can simulate a virtual PLC in software. This quick tutorial will get you connected to the PLC, then guide you through compiling, downloading, testing and debugging a sample 'Hello World' SALad program

1. **Connect the PowerLinc Controller.**

   If you are using a real PLC, connect a USB or RS232 serial cable from it to your Windows PC, and then plug the PLC into an electrical outlet.

   If you wish to use the PLC Simulator instead, simply turn it on by selecting *PLC Simulator* from the IDE's *Mode* menu.

2. **Run the SALad IDE.**

   After installing the SALad IDE on your Windows PC, go to *Start->Programs->SALad IDE->SALad IDE* to launch the program.

3. **Test the serial connection to the PLC.**

   When you run the IDE for the first time, a Startup Wizard will automatically help you test the serial connection to the PLC. You can also launch the Startup Wizard from the *View* menu.

   If you would rather test the serial connection to the PLC manually, follow these steps:

   a. Click the *Edit->Application Options* menu item. An *Options* dialog box will appear.
   b. Under the *Communications* tab, select either *USB* or the *COM* (RS232) port you are using to connect to the PLC. If you are using the PLC Simulator, it does not matter what you select here.
   c. Click the *Connect Now* button, which will establish the serial connection and then automatically perform the same test as the *Test Connection* button. After a short delay, a `Successfully connected` message should appear next to the *Test Connection* button.
   d. You do not have to press the *Download Core Application* button because you will be downloading a different SALad application below.
   e. Click *OK* to close the *Options* window.

4. **Load the sample 'Hello World' SALad program.**

   In the IDE, press the *Open* toolbutton (or else click the *File->Open…* menu item), then find and open *HelloWorld1.sal* in the *Samples* directory. A collection of files will appear in the editor window, with each file under a different tab. The tab labeled *HelloWorld1* should automatically be selected after the files finish loading into the editor.

5. **Mark *HelloWorld1* as the Main Code Module.**

   Right-click on the *HelloWorld1* tab and select *Mark as Main Code Module*. This tells the compiler which file contains the beginning of the SALad application program.

6. **View the ASCII Communications Window.**

   Select the *Comm Window* tab at the bottom of the IDE in the *Windows and Inspectors* section.

   Under the *Comm Window* tab, select the *ASCII Window* tab to see text messages sent from the PLC to the PC.

7. **Compile and Download *HelloWorld1.sal*.**

   In the IDE, press the *Compile/Download* toolbutton, or else click the *Project->Compile/Download* menu item.

   The IDE's LED will turn yellow and a progress bar will indicate that *HelloWorld1.sal* is being compiled and downloaded to the PLC. The IDE's LED will turn green when the download is completed.

   After the PLC receives the code, it will automatically reset and run the downloaded *HelloWorld1.sal* program from the beginning.

   The IDE's *ASCII Window* will display the text:

   ```
   Database Initialized
   Core App Running
   Core App Running
   ```

   NOTE: If the PLC's LED continues to flash on and off once per second after the download, the SALad program failed its checksum test. Recheck the serial connection and try the *Compile/Download* again.

8. **Test the *HelloWorld1.sal* program.**

   Tap the PLC's *SET Button* (located above the LED) to fire the *Button Tap* SALad event. The *HelloWorld1* SALad program will respond to the event by sending an ASCII message to the PC.

   The IDE's *ASCII Window* will display the text:

   ```
   Core App Running-Button Pressed
   ```

9. **Debug the *HelloWorld1.sal* program.**

   To the right of the toolbar, click the *Debug Mode* checkbox to begin a debugging session. A watch window will appear to the left of the editor pane. In debug mode, the PLC will report the address of its program counter to the IDE and optionally stop on each line, allowing you to debug the code.

   To try out the debugger, press the *Fast Step* toolbutton, and then tap the PLC's *SET Button* again. This time, the IDE will step rapidly through the program in the editor and highlight each line of code that it executes.

You can cause the debugger to execute one line of code at a time by single stepping. Try pressing the *Single Step* toolbutton, and then tap the PLC's *SET Button*. The editor will highlight the first line of SALad code to be executed. Thereafter, each time you press the *Single Step* toolbutton, the highlighted line of code will execute, then the editor will jump to and highlight the next line to be executed after that one.

You can set (soft) breakpoints in the code by clicking on the green dots in the margin at the left of the code. Active breakpoints are indicated by a red dot. To clear a breakpoint, just click it again.

Try using a breakpoint by single-stepping a few lines into the code and setting a breakpoint there. Press the *Fast Step* toolbutton to let the code finish executing, and then press the PLC's *SET Button* to fire a new event. The debugger will stop at the breakpoint you set.

To run the debugger as fast as it will go, press the *Run* toolbutton. Soft breakpoints will be ignored but code highlighting will still work. If you don't want the editor to jump around in the code as it executes, uncheck the *Show CP* check box (CP means Code Point).

To exit the debugging session and allow the PLC to run normally, simply uncheck *Debug Mode*.

10. **Congratulations, you have compiled, downloaded, tested and debugged a sample SALad program running on a PowerLinc Controller.**

    HelloWorld1.sal is built on a *SALad coreApp Program*272 called *coreApp.sal* that provides basic INSTEON and X10 communication along with other essential features such as initialization, serial communication, and timers. You can find other sample SALad programs and templates in the *SALad IDE\Samples* and *SALad IDE\Code Templates* directories.

11. **Further steps.**

    You can become more familiar with INSTEON and X10 SALad programming by reading this Developer's Guide, trying out other sample SALad programs, and using the debugger. Since SALad programs mostly consist of event handlers, the easiest and safest way to write SALad code is to modify existing code, such as *coreUser.sal* that already contains the necessary infrastructure.

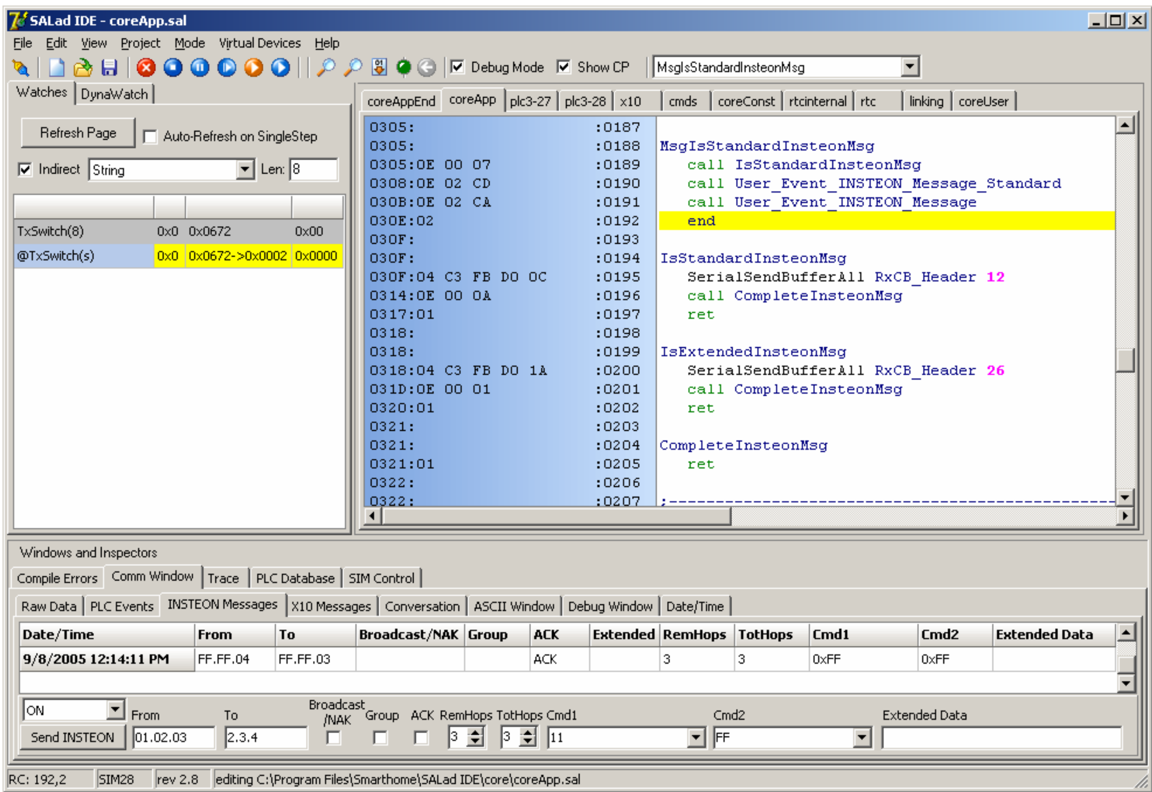
    SmartLabs and other leading developers are continuously creating new sample and real-world SALad applications. Be sure to check the INSTEON Forum frequently at http://www.insteon.net/sdk/forum/ for the latest information.

IDE Main Window

This is what the SALad IDE's main window looks like. This section explains the *IDE Menus*292 and *IDE Toolbar*301 at the top.

The *IDE Editor*303 is below the toolbar on the right. During debugging, the *IDE Watch Panel*306 appears to the left of the Editor. You can access the various *IDE Windows and Inspectors*315 at the bottom by clicking on the tabs.

You can drag the borders at the left and bottom of the Editor to resize the panes, and you can instantly collapse or expand the Windows and Inspectors pane by clicking at the top of it.



IDE Menus

These are the main menus in the SALad IDE.

In This Section

Menu – File

The *File* menu is for opening and saving source and output files.

or **Toolbutton** Creates a new source code file in the editor. A new blank file will appear under a new tab in the editor, labeled *unnamedx*, where *x* will increment as necessary to provide a unique filename. To save the new file under the name you really want it to have, use *Save As…*.

or **Toolbutton** Opens an existing source code file in the editor.

This is the same as *Open…* except the submenu lets you choose from a list of the last files you opened.

or **Toolbutton** **(Keyboard Shortcut: Ctrl-S)** Saves the file currently displayed in the editor using the same name as shown in the active tab (with an extension of *sal*), in the same directory from which it was opened.

Saves the file currently displayed in the editor, using whatever name you choose, in whatever directory you choose.

Saves all of the files under all tabs in the editor in the same way that an individual file would be saved.

This compiles your SALad program and saves it as a binary file (with an extension of *slb*) in whatever directory and with whatever name you specify.

This compiles your SALad program and saves it as a hexadecimal file (with an extension of *hex*) in whatever directory and with whatever name you specify.

This is a very convenient option for saving your work and optionally sending it to someone else (or yourself). It will create a Zip file containing your source code, then launch your emailer with the Zip file already attached to a default message. If you don't want to email the Zip file, just close the emailer.

(Keyboard Shortcut: Ctrl-F4) Closes the file currently displayed in the editor. This will not delete the file saved on disk (if there is one).

Closes all of the files currently open in the editor. This will not delete any of the files saved on disk (if they exist). After all of the files are closed, a new blank file will appear under a new tab in the editor, labeled *unnamedx*, where *x* will increment as necessary to provide a unique filename.

Launches the *Options* dialog box, which lets you change various settings for the IDE. See *IDE Options Dialog Box*307 for details.

This compiles your SALad program and saves it as a binary file (with an extension of *slb*) in whatever directory and with whatever name you specify.

This saves the currently displayed source file in the editor as a listing file (with the extension *txt*). The listing includes the information in the 'gutter' at the left of the editor, which includes hex addresses, bytecode and line numbers.

This ends your IDE session and closes the program. Be sure to save your work if you did not specify the *Save all files on close* option in the *Options – Saving*313 dialog box.

Menu – Edit

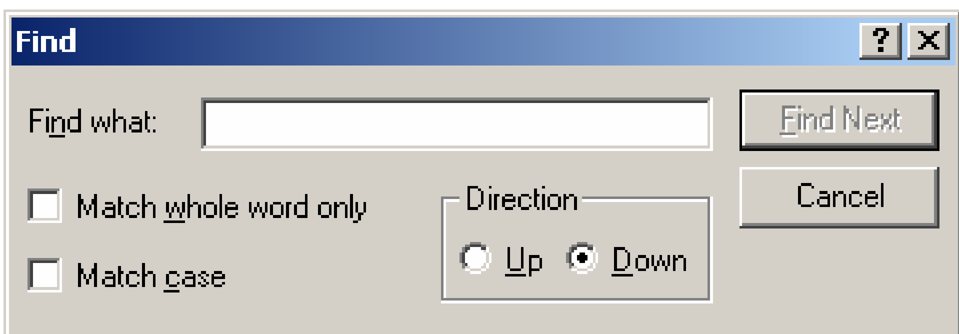
The *Edit* menu helps you navigate within your source code, find text, replace found text, and modify IDE options.

(Keyboard Shortcut: Alt-G) This lets you jump to a specified line number in the source file currently displayed in the editor. The editor will display the line highlighted for a few seconds after it jumps to it. If you specify a line number beyond the end of the file, the editor will jump to the last line in the file.

If this option is checked, you can edit source files. To disable editing, uncheck this option.

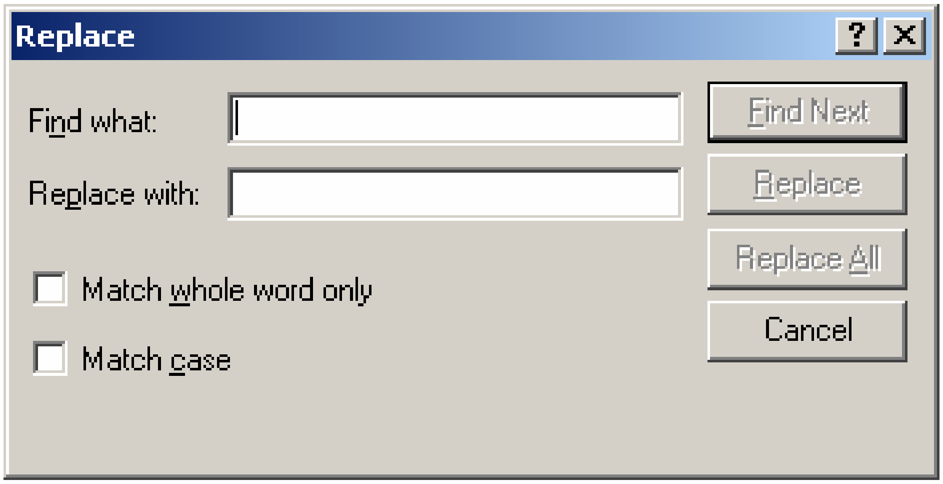
Launches the *Options* dialog box, which lets you change various settings for the IDE. See *IDE Options Dialog Box*307 for details.

or **Toolbutton**  **(Keyboard Shortcut: Alt-F)** Launches the *Find* dialog box, which looks like this:



You use **Ctrl-Down** to find the next occurrence of the string you are finding, and **Ctrl-Up** to find the previous occurrence.

or **Toolbutton**  **(Keyboard Shortcut: Alt-R)** Launches the *Find and Replace* dialog box, which looks like this:



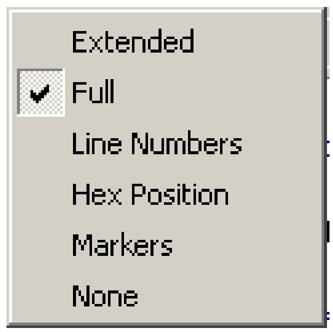
You can use **Ctrl-Down** to find the next occurrence of the string you are finding, and **Ctrl-Up** to find the previous occurrence.

Menu – View

The *View* menu changes the appearance of the IDE by displaying and hiding various parts of it.

Check or uncheck this option to display or hide the *Windows and Inspectors* tabs at the bottom of the IDE window. You can achieve the same result by clicking on the *Windows and Inspectors* label bar.

This gives a submenu that changes the appearance of the information to the left of the source code in the editor. The submenu looks like this:



You can check only one of the options. *Full*, the default, displays (from left to right) hex addresses, hex bytecode, source code line numbers, and breakpoint markers, as shown in the section *IDE Editor*303, below. If you check *Extended*, the gutter pane is widened so you can see more of the bytecode. If you check any of the other options, so will only what you selected. During debugging, you will not be able to set breakpoints unless the *Markers* column is displayed.

This launches the *Startup Wizard* that guides you through setting up communications with your PLC. You can achieve the same results manually by going to the *Options – Communications*310 menu.

Menu – Project

The *Project* menu controls compilation and loading of new firmware into the PLC.

(Keyboard Shortcut: F9 while not debugging)
This compiles your SALad program and downloads the compiled bytecode to your PLC (which can be a real PLC or the PLC Simulator). The Compile/Download  toolbar button does the same thing.

This lets you download a selected version of firmware code to the PLC. Firmware object code has the file extension *ump*.

Menu – Mode

The *Mode* menu switches between using a real PLC, a simulated PLC, or no PLC. You can select only one of the options.

Select this option if you are connected to a real PLC. This option is the default.

Select this option if you are not connected to a PLC. You will not be able to download or debug code while you are offline.

Select this option if you wish to use the software PLC Simulator. You can download code to the PLC Simulator and debug it just as you would for a real PLC. See the *PLC Simulator*332 section below for more information. While you are using the PLC Simulator you can also simulate a *Virtual Powerline*333 environment along with any number of *Virtual LampLinc*334 devices plugged into it.

Menu – Virtual Devices

The *Virtual Devices* menu launches the virtual powerline and virtual LampLinc devices for use with the PLC Simulator.

This will launch a virtual LampLinc device simulated in software. The virtual LampLinc will appear in a separate window. See *Virtual LampLinc*334 for details.

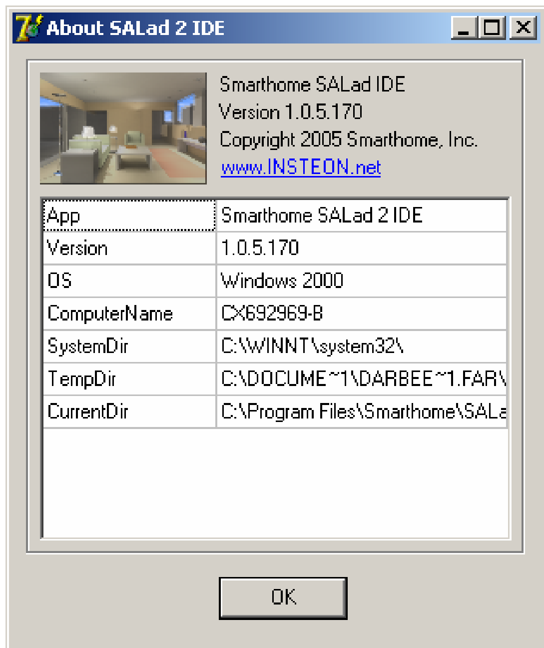
This will launch a virtual powerline environment simulated in software. The virtual powerline will appear in a separate window. If you are going to use the Virtual LampLinc, launch a Virtual Powerline first. See *Virtual Powerline*333 for details.

Menu – Help

The *Help* menu launches a Developer's Guide in compiled help form, gives version information about the IDE, lets you check for IDE updates, and links you to the INSTEON Developer's Forum.

This launches a Developer's Guide in compiled help format.

This displays a screen like this one:



This will automatically check for newer versions of the IDE and update it if appropriate.

This will open your web browser and take you to www.insteon.net.

IDE Toolbar

This is what the main Toolbar looks like:

This is what the individual toolbuttons do:

 **Connect or Reconnect** serially to the PLC.

 **New File**. Create a new SALad file (*.sal).

 **Open File**. Open a SALad file (*.sal) or SALad Template (*.salt).

 **Save File**. Save the current SALad file (*.sal) or SALad Template (*.salt).

 **Reset**. Reset the PLC by forcing an **EVNT_INIT (0x00)** IBIOS Event (see *IBIOS Event Details*187, Note 1). This will start the SALad program with **EVNT_INIT (0x00)** in its event queue.

 **Stop**. Stop execution of the SALad program.

 **Pause**. During debugging only, pause the SALad program at the next line of code.

 **Single Step**. During debugging only, execute the next line of code in the SALad program. This line is normally highlighted in the IDE.

 **Fast Step to Next Breakpoint**. During debugging only, execute one line of code at a time, reporting each line executed, and stop at the next soft or hard breakpoint.

 **Run in Animate Mode**. During debugging, run at top debugging speed, reporting each line executed, and stop at the next soft or hard breakpoint or at the end of the program. During normal execution, run at full speed and stop at the next hard breakpoint or at the end of the program.

 **Find**. Find text in the currently displayed SALad program.

 **Find and Replace**. Find and replace text in the currently displayed SALad program.

 **Compile and Download**. Compile the SALad program and download the bytecode into the PLC over the serial port. If you right-click this button, the following options appear:

1. 

Display differences between the code to download and the code read back from the PLC.

2. 

Download all files on the next download instead of just incrementally downloading files that have changed.

3. 

Always download all files. This is the default setting.

4. 

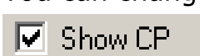
Read back the code image from the PLC after the next code download.

 **LED**. This is an indicator only, not a button. After pressing the  *Connect/Reconnect* button, green means you are successfully connected to the PLC and red means you are not connected. After pressing the  *Compile/Download* button, green means the program downloaded to the PLC successfully, red means the program did not download successfully, and yellow means the program is currently downloading.

 **Go Back**. Return to the previous SALad program location in the editor. Use this after hot-jumping to another SALad program by right-clicking on a label.

There are two checkboxes, used for debugging:

Enable / Disable Debugging Mode. When checked, you will be in debugging mode and the Watch window will appear to the left of the edit window. You can change the size of the Watch window by dragging the separator bar.



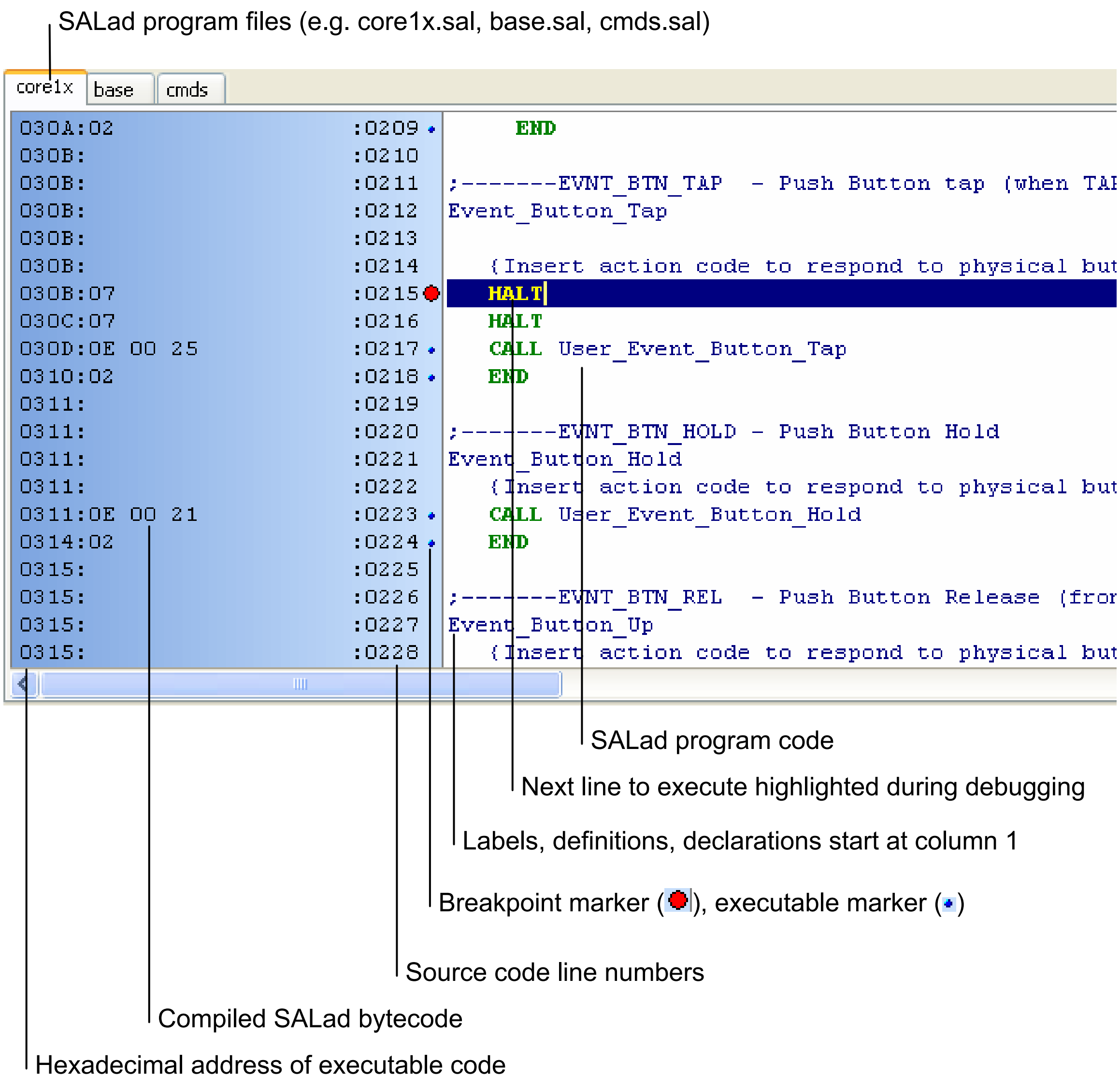
Show Code Point. When you are debugging and *Show CP* is checked, the editor will jump to whatever code line is executing and highlight that line, even if the line is not on the currently displayed page. When *Show CP* is unchecked, the currently displayed page will remain visible during execution.

The pulldown box works with the editor:

Jump To Label. This pulldown box contains all labels used in the SALad program. When you select a label from the pulldown box, the editor will jump to the program location where the label first appears. During editing and debugging, the label displayed in the pulldown box will be the one that fits the current context.

IDE Editor

The IDE editor handles multiple source files and displays SALad source code in context-sensitive color. The screenshot below shows how an editing session would typically appear.



You can change the appearance of the editor by right-clicking anywhere in the editor pane and selecting *Editor Properties…*. This displays the *Editor* tab in the *Options* dialog box. See the *Options – Editor*312 section for details.

You can change which columns of information are displayed in the 'gutter' using the *View->Gutter* menu item (see *Menu – View*297 above).

Right-clicking anywhere in the editor displays the following menu options:

(Keyboard Shortcut: Ctrl-C) This copies selected text to the clipboard.

(Keyboard Shortcut: Ctrl-X) This cuts selected text to the clipboard.

(Keyboard Shortcut: Ctrl-V) This Pastes text in the clipboard at the cursor location.

(Keyboard Shortcut: Ctrl-A) This selects all of the text in the current file.

This sets the program counter so that the next line to be executed will be where the cursor is, i.e. where you right-clicked to get this popup menu.

This sets the program counter so that the next line to be executed will be where the caret is. The caret (a vertical bar) is where you last left-clicked clicked with the cursor.

Run the program from the Code Point (the current address of the program counter) to the line where the cursor is, i.e. where you right-clicked to get this popup menu.

Run the program from the Code Point (the current address of the program counter) to the line where the caret is. The caret (a vertical bar) is where you last left-clicked clicked with the cursor.

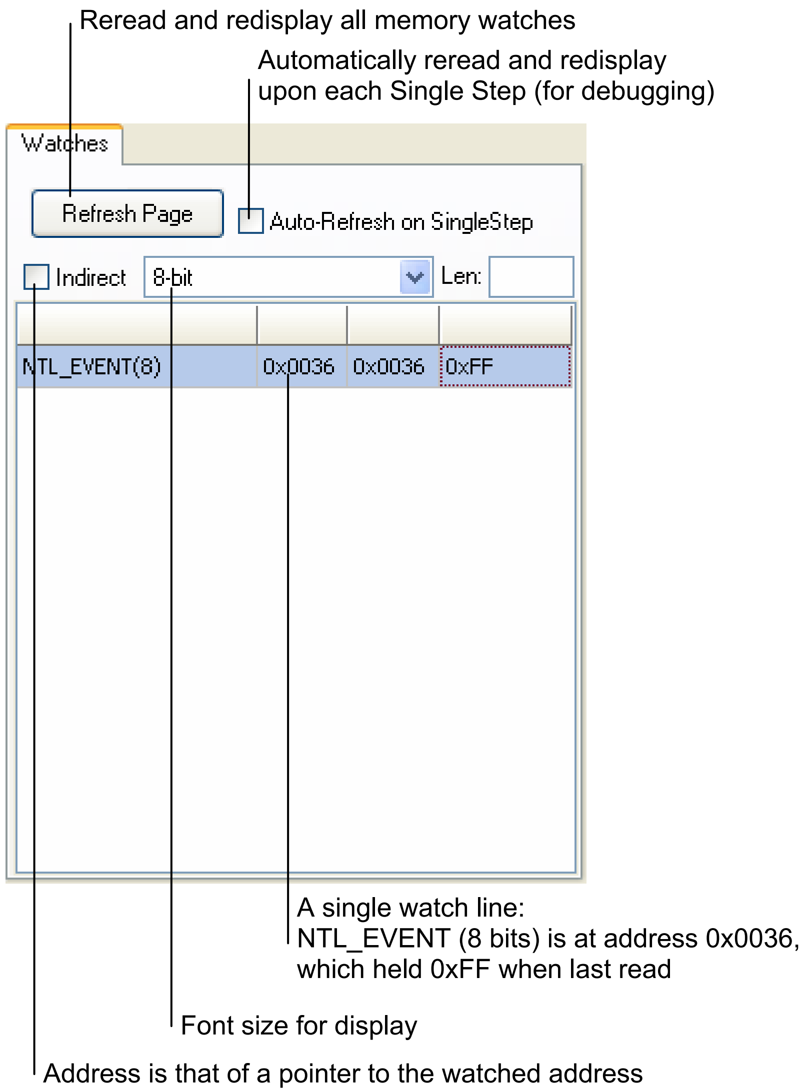
Add the variable under the caret to the Watch Window. The caret (a vertical bar) is where you last left-clicked clicked with the cursor. See *IDE Watch Panel* 306 for details.

If there are any undefined labels in your SALad program, this will insert them at the cursor position so you can define them.

This displays the *Editor* tab in the *Options* dialog box, so you can change the way the editor looks and behaves. See *Options – Editor* 312 for details.

IDE Watch Panel

The watch panel appears to the left of the editor during debugging sessions. You can change its size by dragging the border between it and the editor.



You can add variables that you want to watch by clicking on a variable in the editor, then right-clicking and choosing *Add Watch* in the popup menu.

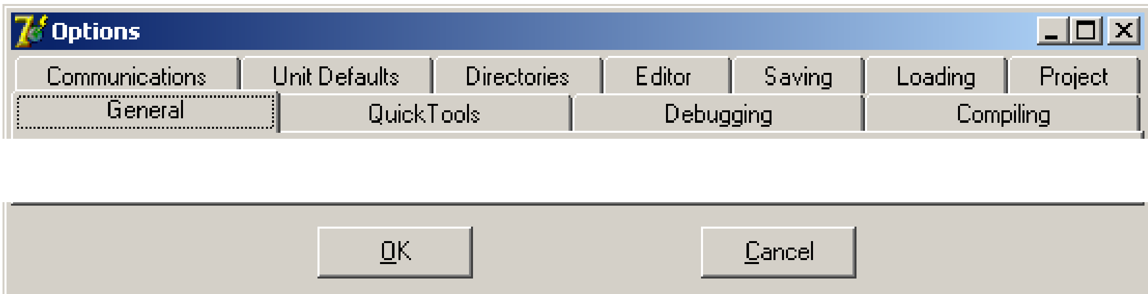
Click the *Refresh Page* button to update all watches. If the *Auto-Refresh on SingleStep* check box is checked, then single stepping during debugging (keyboard shortcut F8) will update the watches after each step.

Check *Indirect* if the variable that you are watching contains a pointer to another variable. The Watch Window will display then also display the contents of the variable being pointed to.

The pulldown box lets you choose whether the watched address is the beginning of an 8-bit variable, a 16-bit variable, an ASCII string, or a hex string. If it is an ASCII or hex string, set the length of the string to display in the *Len:* box.

IDE Options Dialog Box

The Options Dialog Box has a number of tabs that look like this. The panels that the various tabs display are explained below. You can launch the *Options* Dialog Box by choosing the *Edit->Applications Options…* menu item.



The *OK* and *Cancel* buttons appear at the bottom of each panel, but they are not shown in the figures below. Settings take effect when you push *OK*. If you push *Cancel* the previous settings will remain in effect.

In This Section

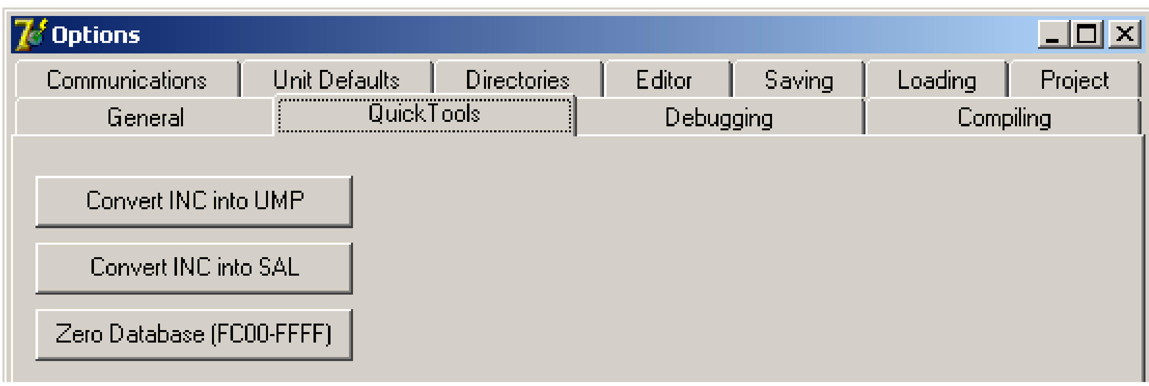
Options – General

The *General* tab controls overall IDE behavior.

Currently the only option is to show the Splash Screen when the program starts. This defaults to off after the first time the program is run.

Options – Quick Tools

Quick Tools are for working with the PLC's firmware and ALL-Link Database. These are here for convenience and others may be added in the future.



Convert INC into UMP and *Convert INC into SAL* are utilities for converting assembler include (INC) files into Unit Map files (UMP) or SALad source code files (SAL).

Zero Database (FC00-FFFF) clears the PLC's *Threaded ALL-Link Database (ALDB/T)*105 from hex address 0xFC00 to 0xFFFF by writing zeros into it. You can also zero the ALL-Link Database from the *PLC Database*326 tab under *Windows and Inspectors*.

Options – Debugging

The *Debugging* tab contains controls for debugging sessions.

Auto-disconnect device after 5 consecutive access violations stops serial data flow if faulty SALad code is generating an access violation storm that prevents you from seeing where the code went awry.

Log all raw data into <specified log file> lets you designate a text file that will log the data that appears in the *Comm Window – Raw Data*318 panel.

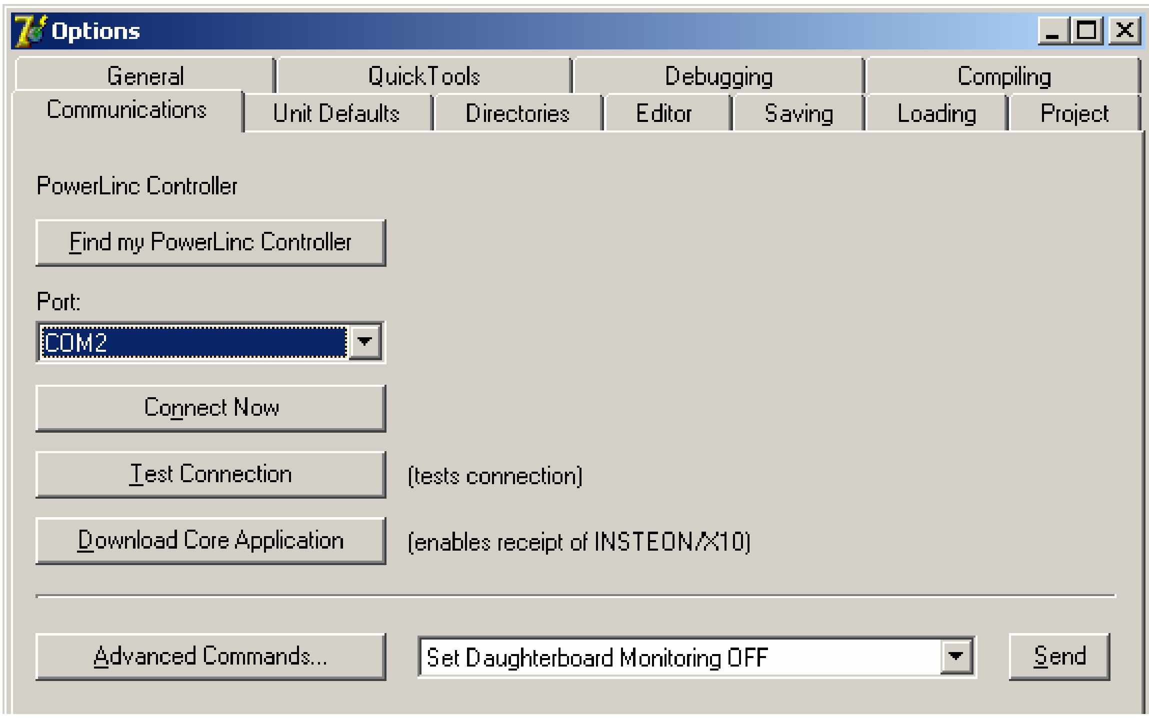
Options – Compiling

The *Compiling* tab contains controls for the compiler.

Prevent download when errors exist, when checked, will not download bytecode to the PLC if the compiler generates errors. This option is checked by default.

Options – Communications

The *Communications* tab has setup options and tools to control communication with the PLC.



Press *Find my PowerLinc Controller* to check your serial ports for an attached PLC. If a PLC is found, the port to which it is attached will appear in the *Port:* pulldown box.

If you want to manually designate the serial port that your PLC is attached to, you can use the pulldown box directly. The pulldown box contains options for your available COM (RS232) and USB ports. If you are using the PLC Simulator it does not matter which option you choose.

Press the *Connect Now* button to establish a connection with your PLC over the port designated in the *Port:* pulldown box. Pushing the *Connect Now* button will automatically "push" the *Test Connection* button.

If you are already connected to your PLC, you can test the connection at any time by pressing the *Test Connection* button.

Press *Download Core Application* to download a precompiled version of the *coreApp.sal* SALad coreApp Program 272 to your PLC. This file to be downloaded, named *coreApp.slb*, is the one in the *Program Files\Smarthome\Salad IDE\core* directory.

Press *Advanced Commands…* to toggle the appearance of a pulldown box containing PLC command options that only apply to particular versions of the PLC. The command shown, *Set Daughterboard Monitoring OFF* is for a Hardware Development Kit. Press the *Send* button to actually send the chosen command to the PLC

Options – Unit Defaults

The *Unit Defaults* tab controls settings for the PLC firmware.

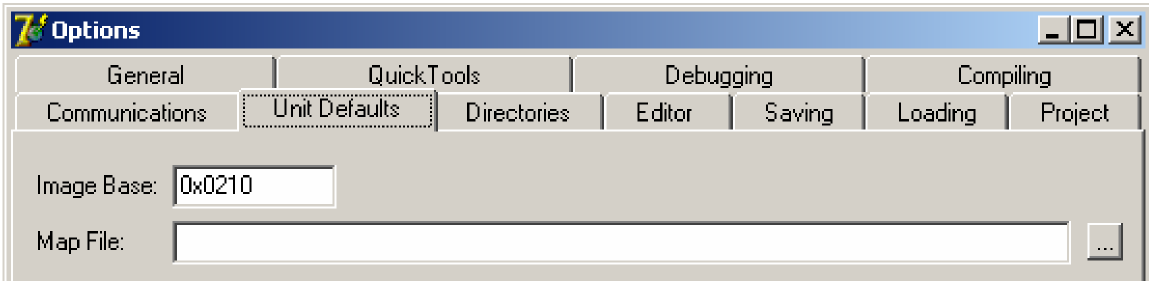
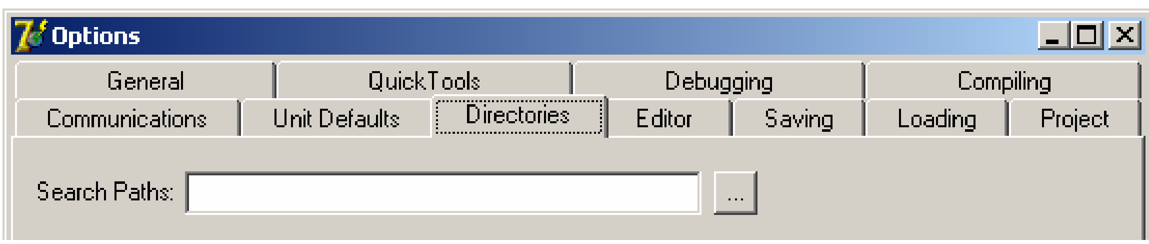


Image Base: is the base address for the downloaded compiled SALad bytecode. This should match the 'org' of the first executable code in the file you designate as 'main' in the *Options – Project*314 tab. The default is 0x0210.

Map File: lets you designate a unit map (UMP) file other than the one that the IDE defaults to by auto-sensing which firmware version is running in your PLC.

Options – Directories

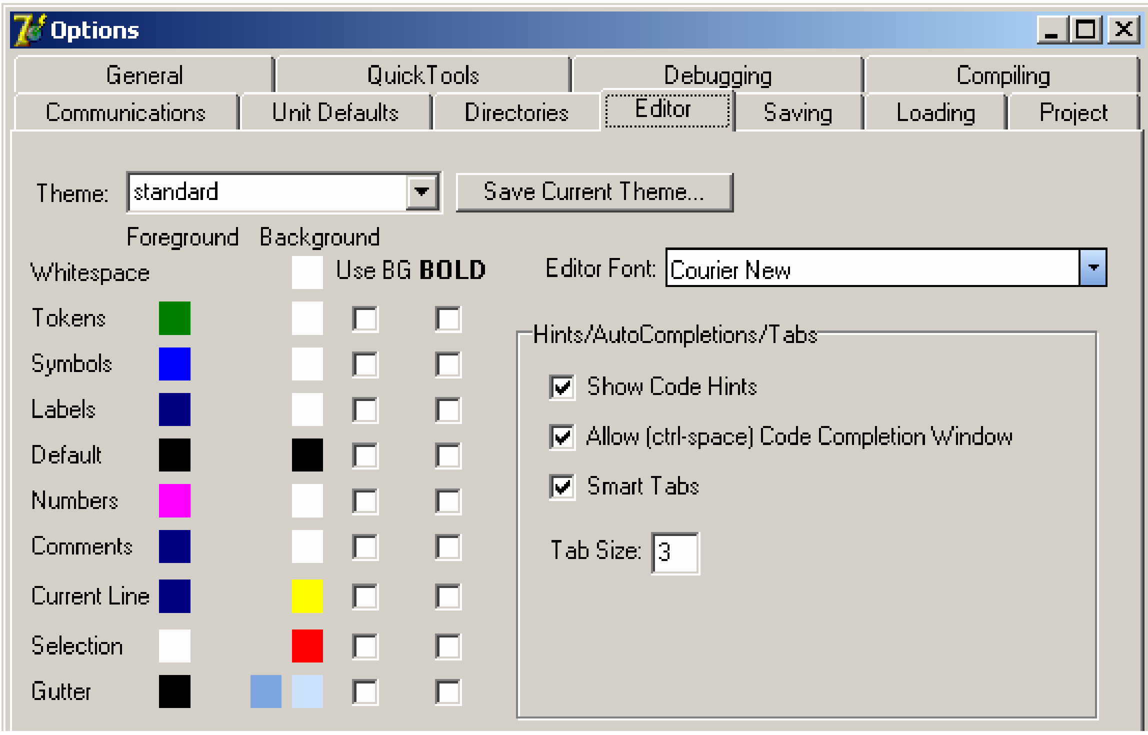
The *Directories* tab sets file search paths.



Enter the path to files that you would like the IDE to search for to be included in your project. Press the button to open a standard dialog box to find a directory. If you want multiple search paths, separate them with a semicolon. The default search paths are your current project's directory (usually under *Smarthome\SALad IDE\Projects*), then the *Smarthome\SALad IDE\Include* directory, then the *Smarthome\SALad IDE\Core* directory.

Options – Editor

The *Editor* tab controls how the editor looks and behaves. You can also launch this page directly from the editor by right-clicking anywhere in it and choosing *Editor Properties…*.



Use the *Theme:* pulldown box to select a previously saved collection of the color and font settings on this page. The default theme is called *standard*. You can create custom themes by choosing the editor options you want on this page and then pressing *Save Current Theme…* to store the settings under a name you choose. Whenever you launch the IDE, the editor will use the theme that last appeared in the pulldown box.

To change the foreground or background color for text of a given type, click on the color box and choose the color you want.

Check the *BOLD* box if you want text of the given type to appear bold, and check the *Use BG* box if you want the specified background color to actually be used.

Use the *Editor Font:* pulldown box to choose the font for the editor.

Check *Show Code Hints* if you want tooltips to appear as you type SALad instructions. The tooltips show the parameters for the instruction and a brief description of what it does.

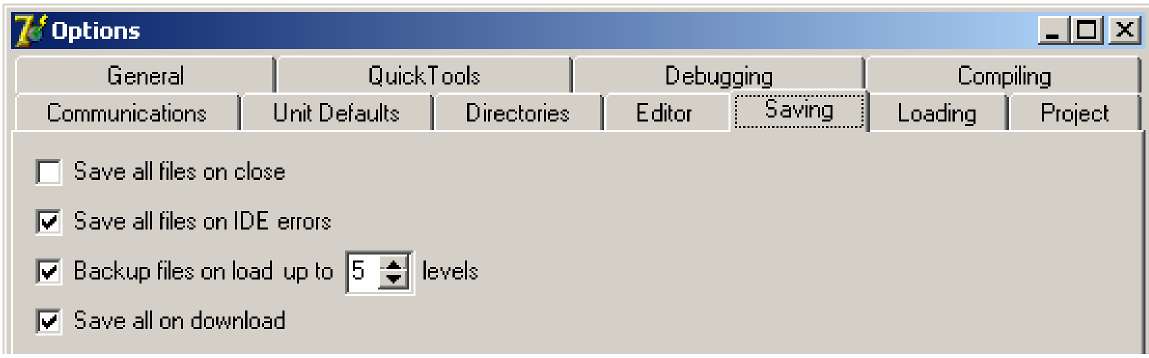
Check *Allow (ctrl-space) Code Completion Window* if you want to see possible SALad word completions by pressing Control-Space after you've typed a partial word.

Check *Smart Tabs* if you want the editor to automatically indent or outdent the next line, depending on context, when you press Enter at the end of a code line.

Set the number of spaces for a tab in the *Tab Size:* box. The default is 3. Tabs are converted to spaces in saved files.

Options – Saving

The *Saving* tab sets file saving and backup options.



Check *Save all files on close* if you want the IDE to automatically save all open source code files when you close the IDE. The files will be saved in whatever condition they are in at the time, so be careful when using this option. The default is unchecked.

Check *Save all files on IDE errors* if you want the IDE to automatically save all open source code files whenever an error occurs in the IDE program. Then, if the IDE crashes, when you restart it you will get a dialog box informing you that your files were auto-recovered from backups, and giving you the option of keeping the auto-recovered files or not. The default for this option is checked.

Check *Backup files on load up to N levels* if you want the IDE to automatically save source code files with a *bak* extension whenever they are loaded. If you set the number of levels greater than 1 (the default is 5), then *bak* files are first renamed with an extension *bk1*, *bk1* files are renamed *bk2*, and so forth. This option is checked by default. **NOTE:** This option has not been implemented as of version 1.0.5.170 of the IDE.

Check *Save all on download* if you want the IDE to automatically save all open source code files whenever you download compiled code to your PLC program. The default is checked.

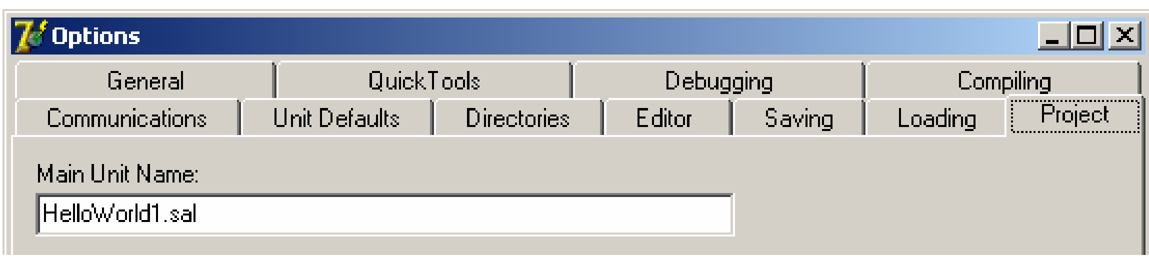
Options – Loading

The *Loading* tab sets file loading options.

Check *Automatically load all included files* if you want the IDE to find and load any files that you have named in a source file using an `include "<filespec>"` statement. The default is checked.

Options – Project

The *Project* tab lets you designate the file containing the beginning of your SALad program. If you do not do this, the compiler will not know where program execution begins. This setting is persistent, meaning that the IDE will remember it between sessions.



You can either type the name of the file in the text box, or you can right-click on the file’s tab in the editor and choose *Mark as main unit*.

IDE Windows and Inspectors

Windows and Inspectors is a collection of tools that you will find very useful when using the IDE to create INSTEON applications using SALad. The main tools appear under a set of tabs that look like this:

Choosing *Comm Window*317 will display a series of subtabs with more tools.

You can make the *Windows and Inspectors* pane collapse or expand quickly by clicking anywhere in its title bar. You can resize the pane by dragging its top border.

In This Section

Compile Errors316
: Lists errors the compiler finds and lets you jump to them in the editor for debugging.

Comm Window317
: Has a collection of subtabs that help you to see what is going on in the INSTEON environment.

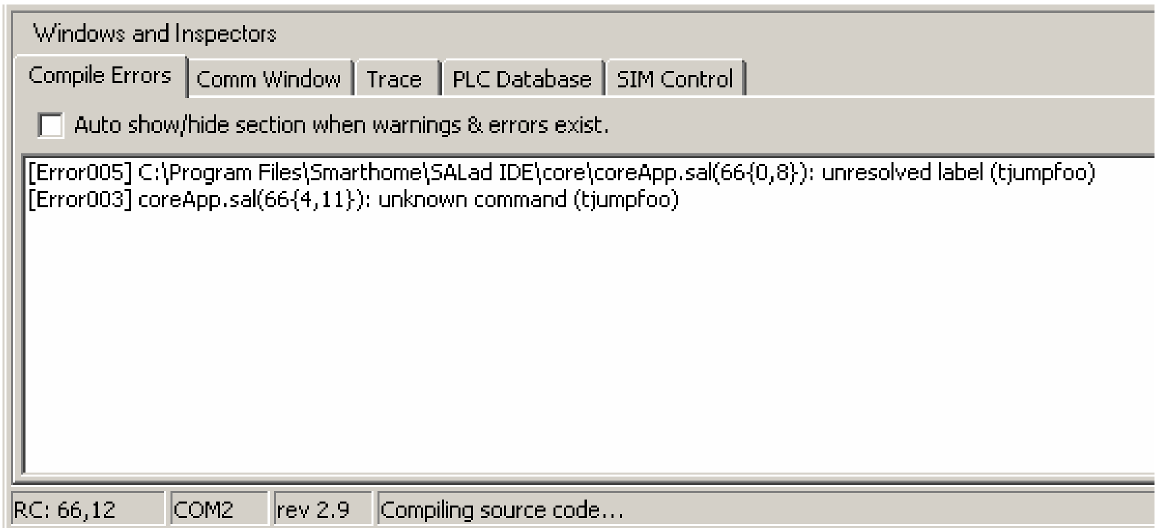
Trace325
: Lets you inspect the execution history of your program to help you with debugging.

PLC Database326
: A tool for manipulating the ALL-Link Database in your PLC.

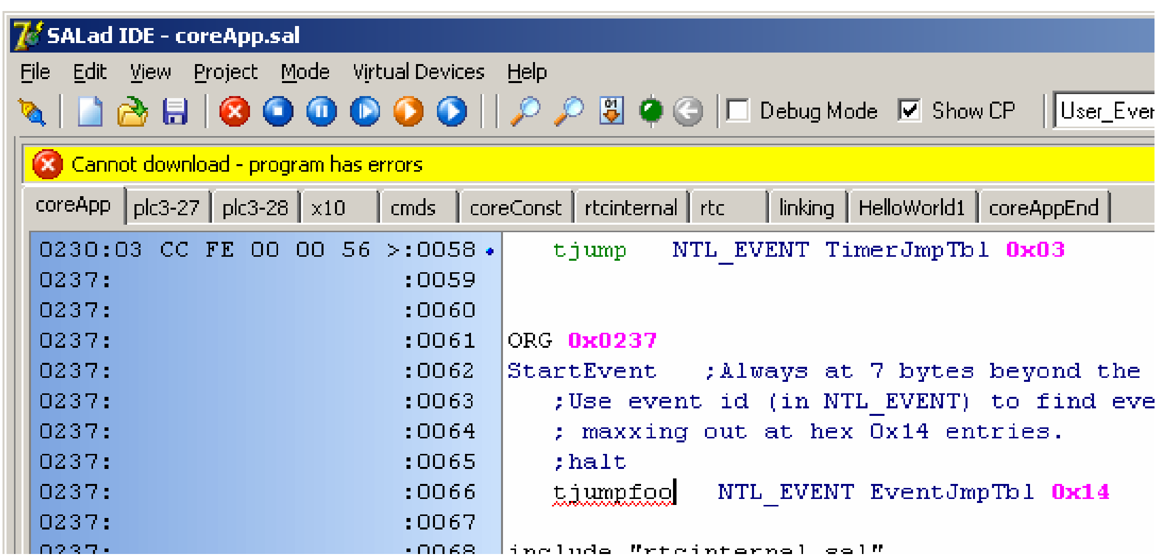
SIM Control327
: Operates the PLC Simulator.

Compile Errors

This page shows errors that the compiler finds. Each line displays an error number, the filespec of the file containing the error, the line number and column positions of the error within the file, and an error message. Double-clicking an error places you on the offending line in the editor.



In the example above, an error was deliberately induced by changing a valid label (`tjump`) to an invalid one (`tjumpfoo`). Double-clicking on one of the lines above shows the error in the editor like this:



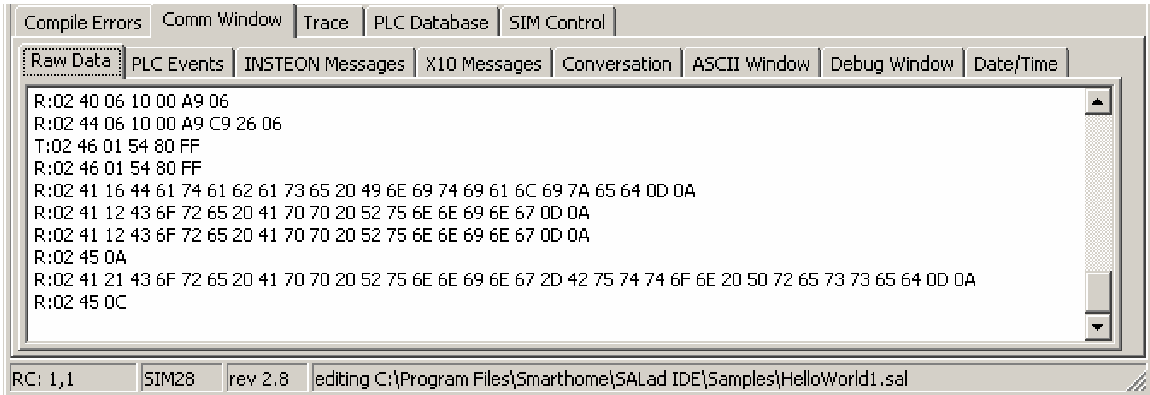
Comm Window

The Comm Window has a number of sub-tabs that look like this:

In This Section

Comm Window – Raw Data

The *Raw Data* page shows the serial data exchanged between your PLC and your PC.



The data is displayed as hexadecimal bytes.

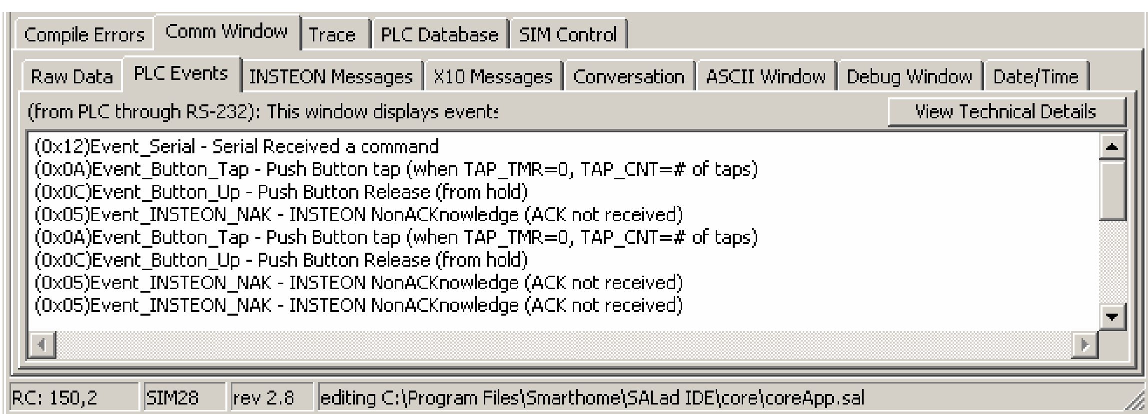
A **T:** precedes data transmitted from the PC to the PLC.

An **S:** precedes data transmitted from the PC to the PLC, but with any packet formatting that may have been applied also displayed. In particular, USB communications adds formatting as described in the section *IBIOS USB Serial Interface*194 above.

An **R:** precedes data received by the PC from the PLC.

Comm Window – PLC Events

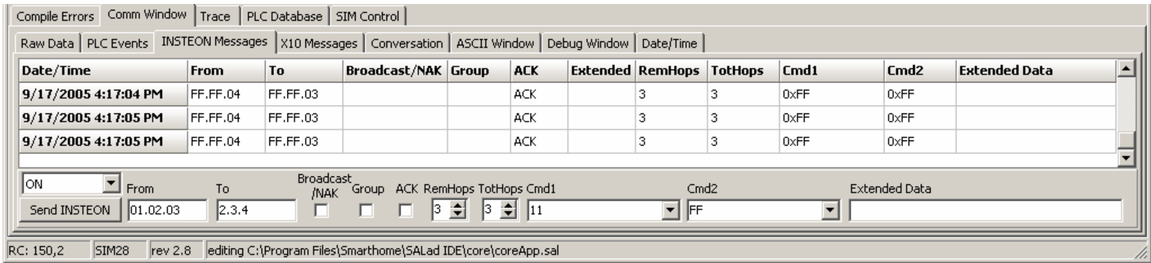
SALad is event-driven, meaning that SALad programs respond to events that occur in the host environment. This page displays IBIOS Events that have occurred in the PLC. See *IBIOS Events*185 for a list of events that SALad handles.



The left column displays the event number in hexadecimal, followed by the event name and a description of the conditions that caused the event.

Comm Window – INSTEON Messages

The *INSTEON Messages* page displays received INSTEON messages, and it lets you compose and send INSTEON messages of your choosing.



The top section of this page displays INSTEON messages received by the PLC. See Message Fields41 above for a description of the column headings.

The bottom section lets you compose and send an INSTEON message. You can choose from a number of pre-composed commands using the pulldown box at the left.

You can enter *From* and *To* Device Addresses41 in the text boxes by typing the 3-byte address as three decimal numbers separated by periods. This is similar to the way an IP address is specified on the Internet.

You can set the Message Type Flags42 the message using the check boxes.

Use the *RemHops* and *TotHops* boxes to set the Message Retransmission Flags43.

Enter the Command 1 and 244 that you want to send as hexadecimal bytes in the respective boxes.

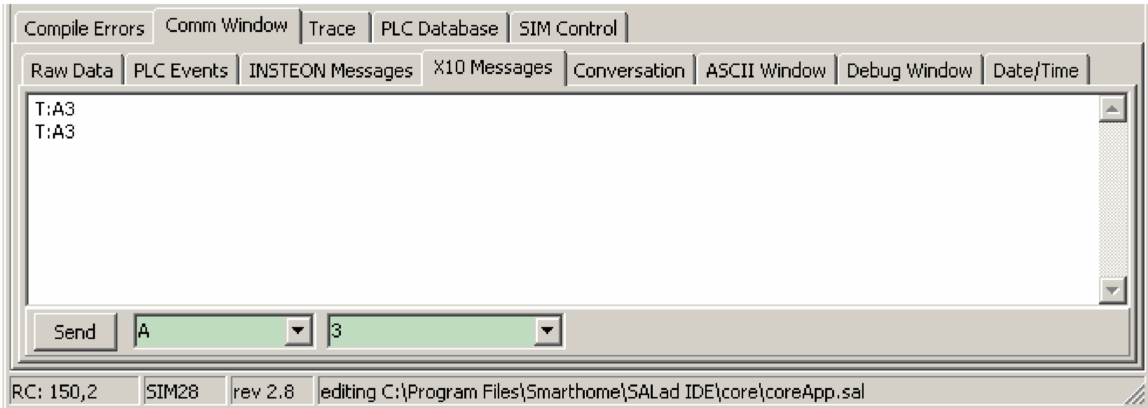
If you are sending an Extended-length message, enter the User Data44 in the *Extended Data* text box. If you enter *any* data in this box, an Extended-length message will be sent. If you enter more than 14 bytes, only the first 14 bytes will be sent. If you enter fewer than 14 bytes, the remaining bytes will be sent as 0x00. If you want to send a Standard-length message, clear this box.

You do not have to deal with the Message Integrity Byte44 (CRC) because the INSTEON Engine handles this for you.

When you have composed the INSTEON message that you want, press the *Send INSTEON* button to transmit it.

Comm Window – X10 Messages

The *X10 Messages* page displays received X10 Commands, and it lets you compose and send X10 Commands of your choosing.



The text box displays X10 traffic.

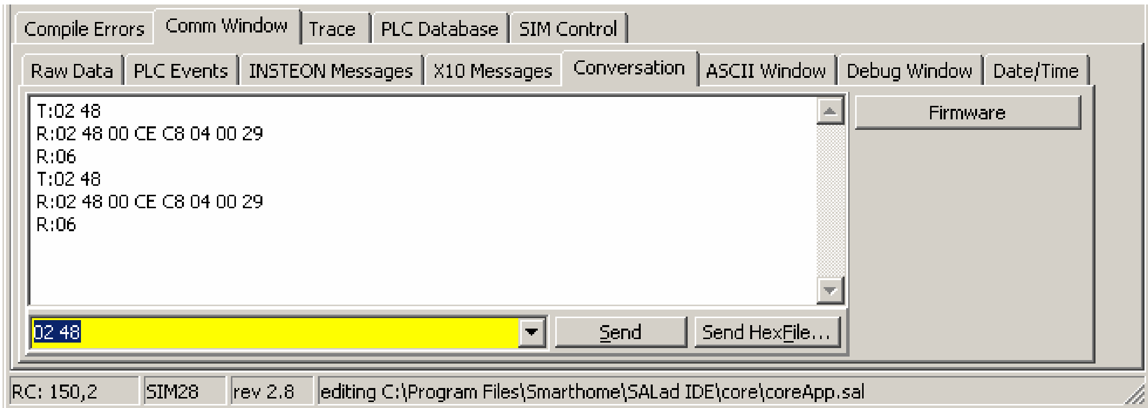
A **T:** precedes X10 Commands transmitted by the PLC.

An **R:** precedes X10 Commands received by the PLC.

To compose an X10 Command, choose its two bytes from the pulldown boxes, then press the *Send* button.

Comm Window – Conversation

The *Conversation* page allows you to have a two-way serial dialog with the PLC.



The text box displays the conversation in hexadecimal bytes.

A **T:** precedes hex bytes transmitted to the PLC.

An **R:** precedes hex bytes received from the PLC.

The yellow text box at the bottom allows you to type in hex bytes to send to the PLC. The pulldown holds the history of your sent bytes.

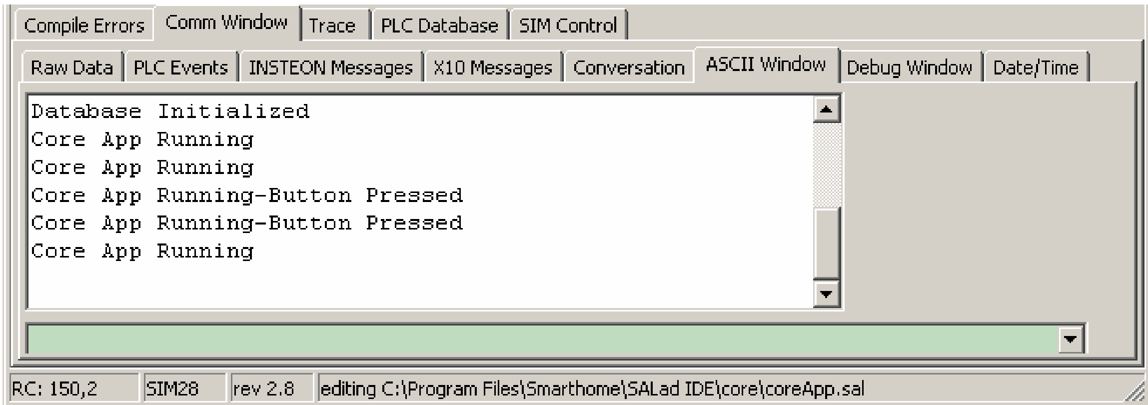
Press the *Send* button to transmit the bytes displayed in the yellow text box.

If you would like to send an entire hex file to the PLC, press the *Send Hex File…* button to choose and send the file. The hex file must be a text file containing 2-character ASCII hex bytes separated by spaces or newlines, such as the files produced by the compiler under the *Files->Save Hex Values As…* menu item.

The *Firmware* button sends 0x02 0x48, which requests the PLC's firmware revision. This is a convenient way to determine if a PLC is connected and listening.

Comm Window – ASCII Window

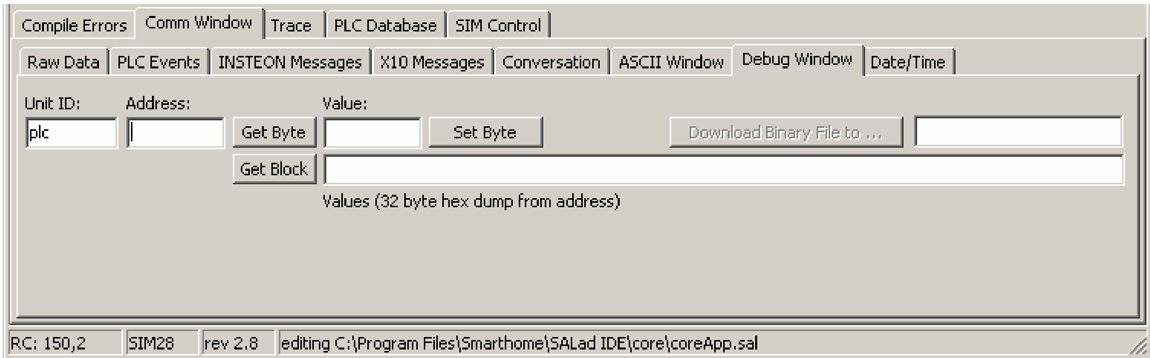
The *ASCII Window* displays text explicitly sent from a SALad 2 program running on your PLC.



Use the `SerialSendBuffer` command within a SALad program to generate messages that will be displayed in this text box.

Comm Window – Debug Window

The *Debug Window* lets you directly inspect and alter bytes within your PLC.



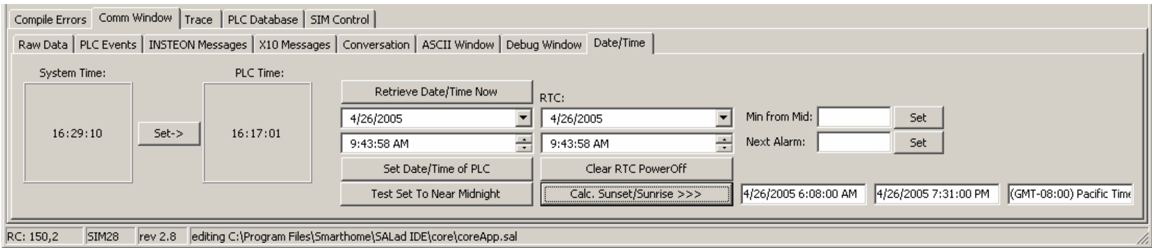
Enter *plc* in the *Unit ID:* box to inspect (peek) and write to (poke) bytes in your PLC. If you want to peek and poke bytes in a remote INSTEON device, enter that device's INSTEON Address as, for example, *1.2.3*.

Type the hex address you wish to inspect or write to in the *Address:* box. Press *Get Byte* to fetch a single hex byte at that address and display it in the *Value:* box, or else press *Get Block* to fetch a string of 32 hex bytes and display them all in the lower box.

To poke a byte, type the hex byte you wish to poke at the given *Address:* in the *Value:* box, then press *Set Byte*.

Comm Window – Date/Time

The *Date/Time* page gives you control over the realtime clock (RTC) in the PLC and lets you test realtime events. All times are in 24-hour 'military' format.



The *System Time* box displays the time according to your PC, and the *PLC Time* box displays the time according to your PLC.

To synchronize your PLC to your PC, press the *Set->* button. The time in the *PLC Time* box will update to the system time.

To set the PLC's RTC to an arbitrary date/time, enter the date and time you wish to set in the boxes above the *Set Date/Time of PLC* button, then press that button. The time in the *PLC Time* box will update within the next minute. To quickly restore the date and time boxes to the current PLC time, press the *Restore Date/Time Now* button.

Press the *Test Set to Near Midnight* button to set the PLC's time to 23:59:45 so you can test functions that occur at midnight.

Press the *Clear RTC PowerOff* button if you do not want the PLC's RTC to save the current time if the PLC loses power.

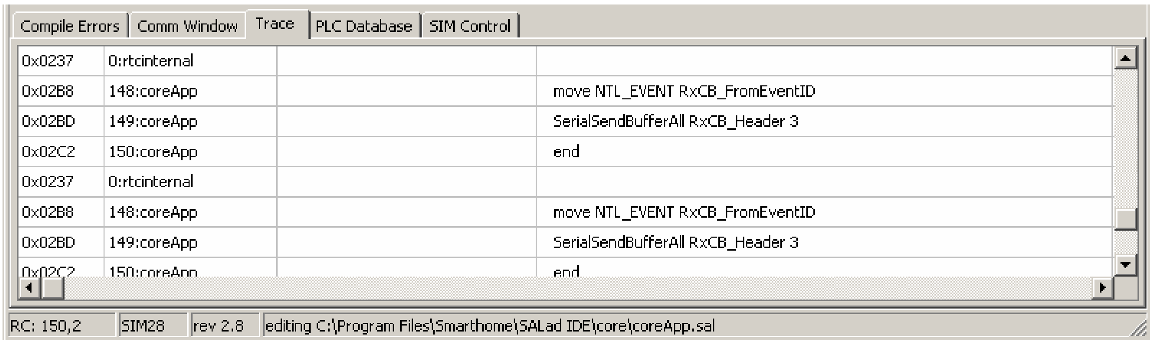
If you want to test timers that depend on the PLC's minutes-from-midnight counter (see *IBIOS Event Details* 187 Note 8), you can set the counter by entering a value from 0 to 1439 in the *Min from Mid* (*Minutes from Midnight*) box and pressing the *Set* button to the right.

Enter a minutes-from-midnight value from 0 to 1439 in the *Next Alarm* box and press the *Set* button to the right to cause an **EVNT_ALARM (0x0E)** IBIOS Event to fire when the PLC's minutes-from-midnight value matches the value you entered. See *IBIOS Event Details* 187 Note 8 for more information.

Press the *Calc Sunset/Sunrise >>>* button to fill in the boxes to the right with the sunrise and sunset times for the PLC date. The box to the right gives the hour-difference between the local time zone and GMT (Greenwich Mean Time).

Trace

The *Trace* page lets you inspect the execution history of your program to help you with debugging.



Tracing is active whenever you are in Debug mode (i.e. the *Debug Mode* checkbox is checked).

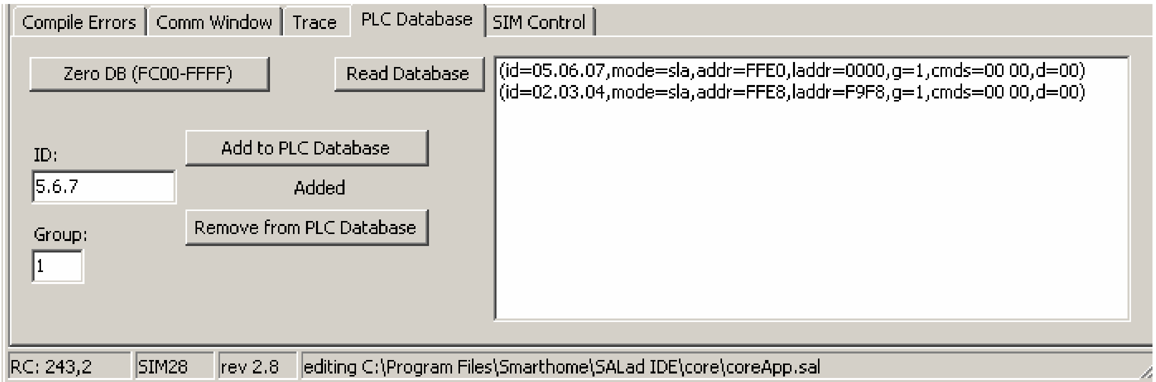
The first column gives the hexadecimal address of the object code that was executed.

The second column gives the line number and the source code filename of the line of code that was executed.

The rightmost column shows the program statement that was executed.

PLC Database

The *PLC Database* page is a tool for manipulating the ALL-Link Database in your PLC. See *INSTEON ALL-Link Database*101 above for more information.



To examine the contents of the PLC's ALL-Link Database, press *Read Database*. The contents will appear in the text box at the right. If the text box is blank, the ALL-Link Database is zeroed out.

To erase the ALL-Link Database by zeroing it out, press the *Zero DB (FC00-FFFF)* button. Zeros will be written to addresses 0xfc00 to 0xffff in the PLC's EEPROM (nonvolatile memory).

To add an entry to the ALL-Link Database, type the INSTEON Address of the device you wish to add in the *ID:* box, and the number of the ALL-Link Group you would like the device to belong to in the *Group:* box, and then press the *Add to PLC Database* button. When you enter the 3-byte INSTEON Address, type it as three decimal numbers, ranging from 0 to 255, separated by periods (for example: `126.23.4`).

To remove an entry from the ALL-Link Database, put the entry's ID and ALL-Link Group Number in the appropriate boxes, and then press *Remove from PLC Database*.

After zeroing the ALL-Link Database or adding or removing an ALL-Link Database entry, you can press the *Read Database* button to see the effect of the change.

If you double-click on a line in the text box, the IDE will jump to the *Comm Window – Debug Window*323 and display a hex dump of the bytes in the ALL-Link Database starting at the address of the entry you double-clicked on.

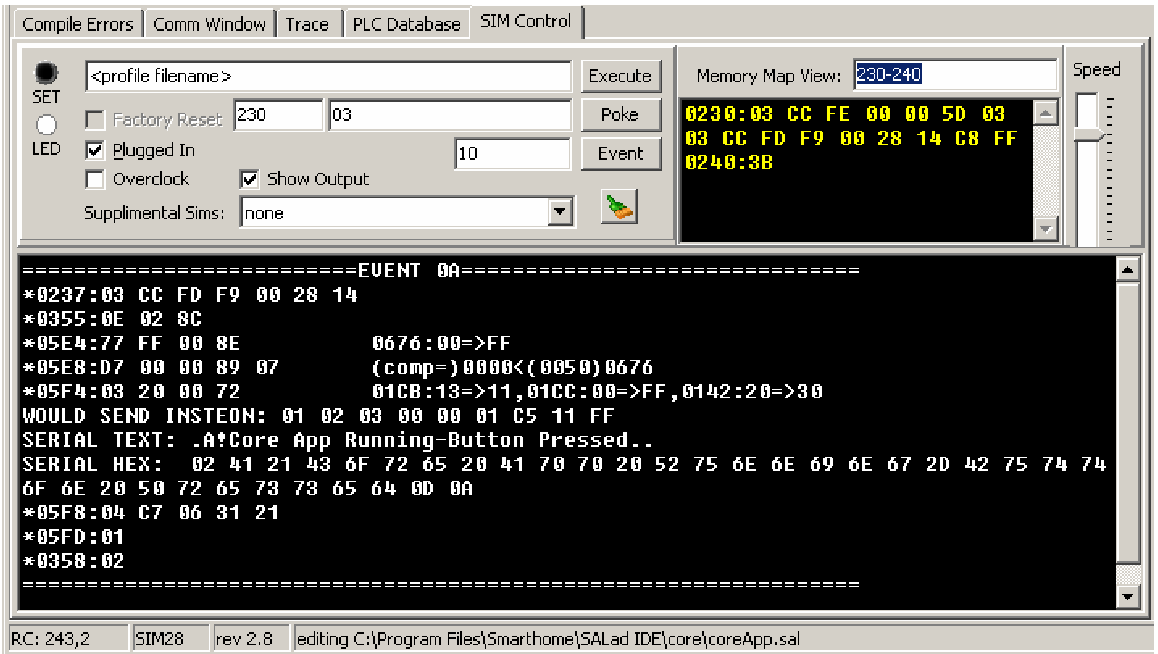
As an example, if you press the *Zero DB (FC00-FFFF)* button, enter an ID of `2.3.4` and an ALL-Link Group of `1`, press the *Add to PLC Database* button, and finally press the *Read Database* button, the following line will appear in the text box:

```
(id=02.03.04,mode=sla,addr=FFE8,laddr=0000,g=1,cmds=00 00,d=00)
```

See *Threaded ALL-Link Database (ALDB/T)*105 above for the meaning of these fields.

SIM Control

The SIM Control window operates the PLC Simulator. To use the PLC Simulator in place of a real PLC, turn it on by choosing the *Mode->Simulator* menu item (see *Menu – Mode*298).



The *PLC Simulator Control Panel*328 is at the upper left. The text box at the upper right is a hex *PLC Simulator Memory Dump*329, and the text box at the bottom is a *PLC Simulator Trace*330 of code execution. You can vary the size of the Trace box by dragging its top border.

In This Section

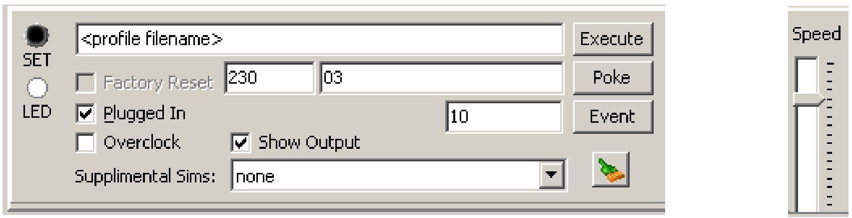
PLC Simulator Control Panel328
Explains the controls at the top of the window.

PLC Simulator Memory Dump329
Shows how to view memory in the PLC Simulator.

PLC Simulator Trace330
Describes the trace information in the bottom text box.

PLC Simulator Control Panel

This is what the PLC Simulator Control Panel section looks like:



A simulated *SET Button* (which you can press) and a simulated white *Status LED* (which you can view) appear at the left.

You can simulate unplugging and plugging in the PLC using the *Plugged In* checkbox. If you would like to simulate a factory reset, uncheck *Plugged In*, check *Factory Reset*, then check *Plugged In*.

Check *Overclock* to run the simulated PLC's realtime clock at very high speed between events. This lets you easily test code that depends on realtime events without waiting for the actual time to elapse or manually resetting the PLC's clock to fire an event.

Use the *Speed* control at the far right to slow down simulated execution speed. This can be useful for debugging.

If you are simulating a PLC with a Hardware Development Kit (HDK) added, choose the HDK from the *Supplemental Sims:* pulldown box.

You can cause the PLC Simulator to execute a string of ASCII text commands from a macro file by typing the filespec for the macro file in the text box to the left of the *Execute* button, then pressing the button. Contact info@insteon.net for the format for the macro file.

To poke a byte or bytes into the PLC Simulator's memory, enter the hex address to poke to, and the hex byte(s) you want to poke, in the text boxes to the left of the *Poke* button, and then press the button. If you are poking multiple bytes, separate them with spaces.

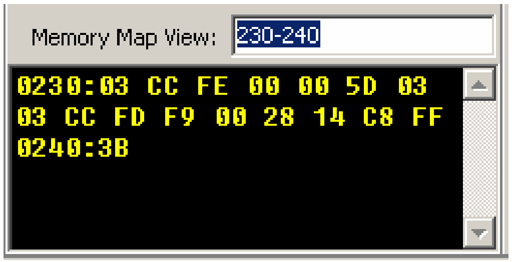
To simulate the occurrence of an IBIOS Event, enter the Event Handle number (see *IBIOS Events*185) in the text box to the left of the *Event* button, then press the button.

Check the *Show Output* checkbox if you want the *PLC Simulator Trace*330 to display code execution information.

Push the *Erase* button  to blank the *PLC Simulator Trace*330 text box.

PLC Simulator Memory Dump

This is what the PLC Simulator Memory Dump section looks like:



To view memory contents in the PLC Simulator, enter a hex address, a range of addresses, or some combination of addresses and ranges in the *Memory Map View:* text box. The text box will immediately display the memory contents you specified.

Indicate an address range by typing a beginning and ending address separated by a hyphen. Use a comma to separate multiple addresses or address ranges.

The memory dump normally refreshes automatically every few milliseconds. If for some reason refreshing stops (if you switch serial ports, for instance), you can restart it by typing anything in the *Memory Map View:* box.

PLC Simulator Trace

This is what the PLC Simulator Trace section looks like:

```
==========================EVENT 0A==============================
*0237:03 CC FD F9 00 28 14
*0355:0E 02 8C
*05E4:77 FF 00 8E          0676:00=>FF
*05E8:D7 00 00 89 07       (comp=)0000<(0050)0676
*05F4:03 20 00 72          01CB:13=>11,01CC:00=>FF,0142:20=>30
WOULD SEND INSTEON: 01 02 03 00 00 01 C5 11 FF
SERIAL TEXT: .A!Core App Running-Button Pressed..
SERIAL HEX:  02 41 21 43 6F 72 65 20 41 70 70 20 52 75 6E 6E 69 6E 67 2D 42 75 74 74
6F 6E 20 50 72 65 73 73 65 64 0D 0A
*05F8:04 C7 06 31 21
*05FD:01
*0358:02
================================================================
```

Drag the border at the top to resize the text box.

Push the *Erase* button  in the *PLC Simulator Control Panel* 328 to blank the text box.

Check the *Show Output* checkbox in the *PLC Simulator Control Panel* 328 if you want the text box to display code execution information.

When *Show Output* is checked, trace information like that shown above with an asterisk at the left will appear. Immediately following the asterisk is the hex address of the code line that was executed, followed by the object code itself. Effects that the code may have are shown to the right. For example,

`0676:00->FF`

means that memory location `0676` was changed from containing `00` to containing `FF`; and

`(comp=)0000<(0050)0676`

means that a compare was done between the literal value `0000` and the contents of the 16-bit value beginning at memory location `0050`, namely `0676`, and that `0000` is less than `0676`.

Whether or not the *Show Output* checkbox is checked, the text window will show INSTEON messages and X10 Commands sent over the powerline by the PLC, as well as both ASCII and hex data sent serially to your PC.

IDE Virtual Devices

The *Virtual Devices* included in the SALad IDE allow you to develop SALad applications without being connected to any external hardware—in other words, you don't need a connection to a real PowerLinc™ V2 Controller (PLC).

The key tool is the *PLC Simulator*332, which is a pure-software version of a real PLC running on your PC. The PLC Simulator can do everything that a real PLC can do, but it is more transparent, thanks to the special tools in the *SIM Control*327 window.

When you are using the PLC Simulator, you can also simulate a *Virtual Powerline*333 environment in software. Then, you can plug in any number of *Virtual LampLinc*334 devices to the Virtual Powerline to debug and test your SALad application.

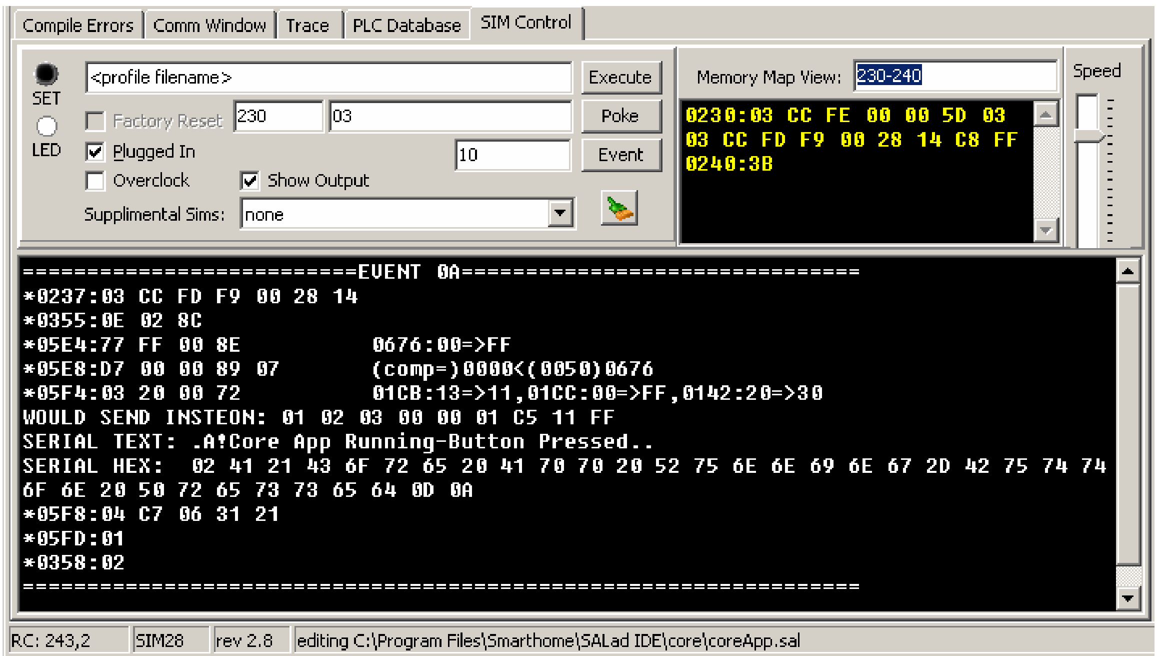
Although very convenient for code development, simulation is no substitute for real-world testing. Be sure to thoroughly shake out your application using real hardware before pronouncing it finished!

In This Section

PLC Simulator

The *PLC Simulator* is a pure-software version of a real PLC running on your PC. To use the PLC Simulator in place of a real PLC, turn it on by choosing the *Mode->Simulator* menu item (see *Menu – Mode*298).

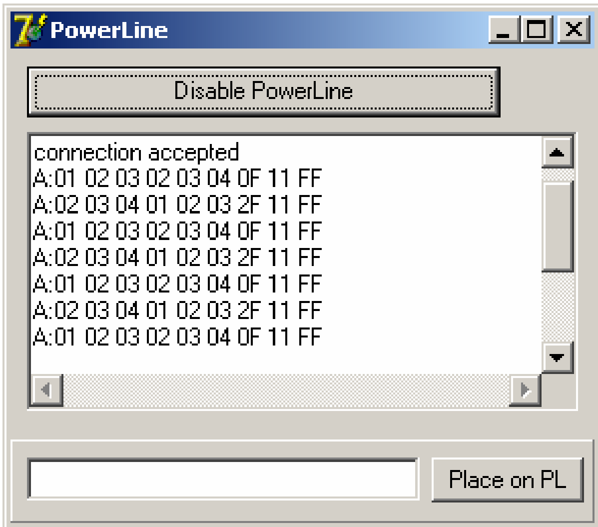
The PLC Simulator can do everything that a real PLC can do, but it is more transparent, thanks to a set of special software tools. These tools appear in the *SIM Control*327 window, which looks like this:



For a complete explanation of how to use these tools, see the *SIM Control*327 section.

Virtual Powerline

The *Virtual Powerline* is a software-simulated powerline environment that works in conjunction with the *PLC Simulator*[332]. To turn it on, choose the *Virtual Devices->Powerline* menu item (see *Menu – Virtual Devices*[299]). The Virtual Powerline will appear as a separate window that looks like this:



The button at the top will be labeled *Enable Powerline* if the Virtual Powerline is currently disabled, or else it will say *Disable Powerline* while the Virtual Powerline is enabled. To use the Virtual Powerline, enable it. If the PLC Simulator is not running, turn it on, or if it is running, you may have to 'Unplug' it then 'Plug' it back in. When the PLC Simulator is using the Virtual Powerline properly, the message

`connection accepted`

will appear in the Virtual Powerline's text box. The same message will appear every time you connect a virtual device, such as a *Virtual LampLinc*[334]. When you remove a virtual device you will get the message

`connection removed`

You can place a message on the Virtual Powerline by typing the ASCII string you want to send in the box to the left of the *Place on PL* button, then pressing the button. The string you sent will appear in the text box.

INSTEON messages that appear in the Virtual Powerline will show up as hex bytes preceded by `A:` in the text box.

To stop using the Virtual Powerline, close its window.

Virtual LampLinc

Virtual LampLinc devices are software simulations of real SmartLabs LampLinc™ V2 Dimmers. Virtual LampLincs are connected to a *Virtual Powerline*333, which in turn connects to a *PLC Simulator*332. To launch a Virtual LampLinc, choose the *Virtual Devices->LampLinc* menu item (see *Menu – Virtual Devices*299). The Virtual LampLinc will appear as a separate window that looks like this:

The text box labeled *INSTEON Address (A.B.C)* at the top of the window contains the 3-byte ID of the Virtual LampLinc. This will be a unique number for each Virtual LampLinc that you launch. If you wish to assign a different ID, type it in the text box as three decimal digits separated by periods.

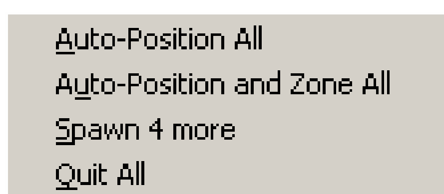
The *Zone:* pulldown box lets you place Virtual LampLincs on the Virtual Powerline at varying simulated distances from one another. The greater the difference between Zone numbers, the greater the simulated distance between LampLincs. This lets you simulate powerline environments that require multiple hops for INSTEON messages to get through (see *INSTEON Message Hopping*49, above).

You can simulate plugging in or unplugging a Virtual LampLinc with the button that will be labeled *(IN) Click to Unplug* or else *Plug In*.

The black circle at the bottom right of the picture of the LampLinc is the simulated *SET Button*, which you can push with the mouse. The white circle below that is the simulated white *Status LED*, which you can view.

The square at the bottom shows the current dimming state of the Virtual LampLinc. Its color ranges from black for off, through various shades of gray, to white for full on.

You can launch multiple Virtual LampLincs by right-clicking anywhere in a Virtual LampLinc window to display a popup menu that looks like this:



Choose *Spawn 4 more* to create four more Virtual LampLincs, each in a separate window, and each with a different INSTEON Address. Choose *Auto-Position All* to neatly tile all open Virtual LampLinc windows. If you choose *Auto-Position and Zone All*, groups of Virtual LampLincs will appear in different Virtual Powerline zones. Choose *Quit All* to close all of the Virtual LampLinc devices at once.

IDE Keyboard Shortcuts

The table below shows how you can use the keyboard to perform common actions in the IDE.

Key	Action
Ctrl-A	Select all text in the currently displayed editor file
Ctrl-X	Cut selected text in the currently displayed editor file
Ctrl-C	Copy selected text in the currently displayed editor file
Ctrl-V	Paste selected text in the currently displayed editor file
Ctrl-S	Save the currently displayed editor file
Alt-G	Go to line number
Ctrl-F	Find a search string
Ctrl-R	Find a search string and replace it with another string
Ctrl-Down	Find next occurrence of search string
Ctrl-Up	Find previous occurrence of search string
Ctrl-F4	Close currently displayed source file
Ctrl-(left mouse click)	If cursor is on an 'Include' file, open the file
F2 (in debug mode)	Stop the program
F8 (in debug mode)	Single Step the program
F9 (not in debug mode)	Compile and Download the program to the PLC
F9 (in debug mode)	Run the program

Chapter 12 — SmartLabs Device Manager (SDM) Reference

This chapter documents the SmartLabs Device Manager (SDM). SDM is in the class of *Manager Applications*31, which are programs that run on computing devices external to an INSTEON network and expose a high-level interface to the outside world.

In This Chapter

SDM Introduction

The SmartLabs Device Manager™ (SDM) is a communication and translation gateway to *The SmartLabs PowerLinc Controller* 28 (PLC). Developers use simple text commands (Home Networking Language™) through ActiveX or HTTP calls to communicate over INSTEON or X10 without the hassle of dealing with USB packets or RS232 resource issues.

These commands will also be extended to PowerLinc/IP and other Internet transports.

Using the SmartLabs Device Manager, developers can focus on their application, whether in Macromedia Flash®, Java®, .NET® or even in a Word® document, Excel® spreadsheet, or a PowerPoint® presentation.

Commands such as "SetOnLevelText=01.02.03,ON" perform the action and return a response, providing developers with simple and reliable control. Additional commands such as "speak," inet," and "mailto" further provide the developer with powerful capabilities.

SDM System Requirements

SDM Platforms

Windows XP, 2000, NT (serial only), Me, 98, and 95 (serial only).

SDM Programming Platforms

any language that provides either ActiveX, HTTP, or shell calls including (Java, VB, .NET, C#, C++, Delphi, Flash, Perl, ASP, PHP, VB/W/JScript, Tcl, Python and more).

SDM Resources

HTTP Server uses port 9020 and offers a server-pull method to facilitate firewalls. Connects via COM1~COM255 or USB.

PowerLinc Controller Firmware Requirement

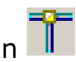
2.8 or better

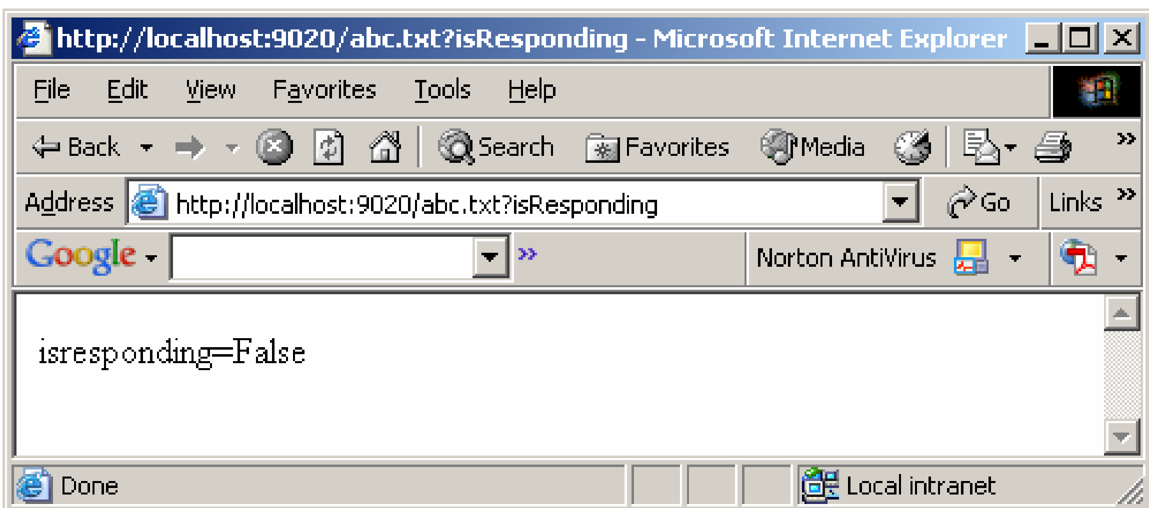
SDM Quick Test

You can get the SDM up and running quickly using either a browser or SDM's Main Window.

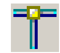
In This Section

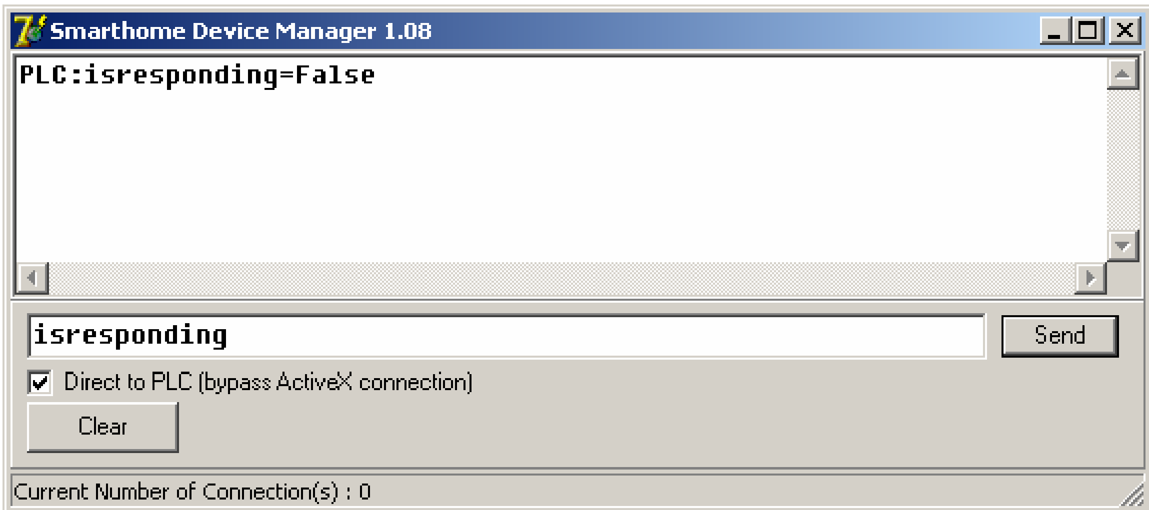
SDM Test Using a Browser

1. Run SmartLabs Device Manager (SDM2Server.exe). An icon  will appear in your systray (normally at the lower right of your screen).
2. Run a browser (such as Internet Explorer or FireFox).
3. Type **http://localhost:9020/abc.txt?isResponding** into the browser as the URL.
4. You should see a textual response isresponding=False or isresponding=True with other cached responses.



SDM Test Using SDM's Main Window

1. Run SmartLabs Device Manager (SDM2Server.exe). An icon  will appear in your systray (normally at the lower right of your screen).
2. Single-click on the icon and SDM's Main Window will appear.



3. Type **isresponding** into the bottom text box and press *Send*.
4. You should see a textual response isresponding=False or isresponding=True with other cached responses.

SDM Commands

This section lists and explains the available SDM Commands.

SDM Commands are not case-sensitive.

In This Section

SDM Commands – Getting Started

To turn on a lamp, plug in a PLC and a LampLinc V2 Dimmer, connect the PLC to your PC, then send these commands in the order given.

port=<PLCport>

Sets the sticky, global PLC port (COM#|USB4|SIM28|?). Sending a question mark '?' causes the port to be searched for. Use **getport=** to read the found port. The port is saved in the registry and reused upon restart of the SDM.

Examples: `port=COM1`

or: `port=USB4`

or: `port=?`

or: `port=SIM28`

isResponding

Asks the SDM if the port is responding (sends 0x02 0x48) and responds true or false. This also reads the map for proper name-based downloading. This is the ultimate heartbeat method to determine if the PLC is connected and talking.

Example: `isResponding`

Returns: `isResponding=true`

downloadCoreApp[=clear]

Downloads the included SALad coreApp to the PLC so that it can communicate to the SDM. The optional =clear parameter also instructs the coreApp to clear the ALL-Link Database after downloading. Automatically resets the PLC after downloading.

Example: `downloadCoreApp`

addID=<remoteINSTEONid>[<group>][,<isMaster=true>]

Adds a device's ID to the PLC's ALL-Link database, optionally specifying the ALL-Link Group number and whether the device is a Controller (master) or Responder (slave).

Example: `addID=04.05.06`

setOnLevelText=<INSTEONid>, <onLevelCmdOrValue>[,<hops>]

Sets the On-Level status (ON|OFF|dec%|dec|0xHex) of an ID, optionally specifying the number of hops. Default is 3 hops.

Examples: `setOnLevelText=04.05.06,ON`

or: `setOnLevelText=04.05.06,49%`

or: `setOnLevelText=04.05.06,127`

or: `setOnLevelText=04.05.06,0x7F`

SDM Commands – Home Control

setOnLevelText=<INSTEONid>, <onLevelCmdOrValue>[,<hops>]

Sets the On-Level status (ON|OFF|dec%|dec|0xHex) of an ID, optionally specifying the number of hops. Default is 3 hops.

Examples: `setOnLevelText=04.05.06,ON`

or: `setOnLevelText=04.05.06,49%`

or: `setOnLevelText=04.05.06,127`

or: `setOnLevelText=04.05.06,0x7F`

getOnLevelText=<INSTEONid>[,<hops>]

Gets the On-Level status from an ID as a text representation, optionally specifying the number of hops. Defaults to 3 hops. Returns ON, OFF, OUT, or percentage of on (1-99%).

Example: `getOnLevelText=04.05.06`

Returns: `getOnLevelText=04.05.06,ON`

or: `getOnLevelText=04.05.06,49%`

sendX10=<addressOrCommand>[,<addressOrCommand>]

Sends X10 addresses and commands.

Examples: `sendX10=A1,AON`

or: `sendX10=A1,A2`

or: `sendX10=AON`

sendGroupBroadcast=<groupID>,<cmd1>[,<cmd2>][,<hops>]

Sends an ALL-Link Broadcast from the PowerLinc Controller with the included ALL-Link Group number, Command 1 (ON|OFF|hex), optional Command 2 (defaults to 0), and optional Max Hops (defaults to 3).

sendInsteonDirect | SID=<id>,<attribute=value>

Sends an INSTEON message.

Example: `SID=02.03.04,OnLevel=ON`

sendInsteonDirectResponse | SIDR=<id>,<attribute=value>

Same as SendInsteonDirect, but waits for an Acknowledge message.

Example: `SIDR=02.03.04,OnLevel=ON`

SDM Commands – Notification Responses

These are for uninitiated messages.

eventRaw=<eventID>

Notification of an event in hexadecimal.

Example: `eventRaw=0A`

receiveX10=<AddressOrCommand>

Notification of an X10 address or command received, in text.

Examples: `receiveX10=A1`

or: `receiveX10=A On`

receiveX10Raw=<x10type> <AddressOrCommandByte>

Notification of an X10 address or command received, in hexadecimal form. x10type is 0 for an address, or 1 for a command.

Example: `receiveX10Raw=00 66`

receiveINSTEONRaw=<eventID> <messageBytes…>

Notification of an INSTEON message received. The number of bytes varies depending upon the eventID and whether the message is Standard-length or Extended-length.

Example: `receiveINSTEONRaw=02 04 05 06 00 00 11 8F 01 00`

enrolled=<INSTEONid>,<deviceInfoBytes>,<deviceName>

Notification that an INSTEON device was enrolled in the PLC's ALL-Link Database.

usbArrival=<VendorID>,<ProductID>,<Version>,<Company>,<ProductName>

Notification that the PLC was connected. Specific to PLC from SmartLabs.

Example: `usbArrival=4287,4,1024,SmartHome,SmartHome PowerLinc USB E,`

usbUnplugged=4287,4,1024,,,

Notification that the PLC was disconnected (once connected). Specific to PLC from SmartLabs)

Example: `usbUnplugged=4287,4,1024,,,`

SDM Commands – Direct Communications

These are for raw, low-level communication.

sendINSTEONRaw=<9 or 23 hexadecimal bytes>

Sends the 9 (Standard-length) or 23 (Extended-length) INSTEON bytes from the PLC.

Example: sendINSTEONRaw=01 02 03 04 05 06 0F 11 FF

The example sends from unit 01.02.03 (which is overwritten with the PLC's actual ID) to unit 04.05.06, an **SD** message with 3 hops (0x0F), a *Light ON* INSTEON Command (0x11), at full brightness (0xFF).

sendRecINSTEONRaw | SRIR=<9 or 23 hexadecimal bytes>

Sends the 9 (Standard-length) or 23 (Extended-length) INSTEON bytes from the PLC, and waits for the response (ACK/NAK message).

Examples: `sendRecINSTEONRaw=01 02 03 04 05 06 0F 11 FF`

or: `SRIR=01 02 03 07 08 09 0F 13 00`

Returns: `SRIR=04,07 08 09 01 02 03 2F 13 00` (the returned ACK message response).

getOnLevelRaw=<INSTEONid>[,<hops>]

Gets the On-Level status from an ID as a hexbyte representation, optionally specifying the number of hops. (Defaults to 3 hops). Returns 00-FF.

Example: `getOnLevelRaw=04.05.06`

Returns: `getOnLevelRaw=04.05.06,7F`

setOnLevelRaw=<INSTEONid>,<onLevelCmdOrValue>[,<hops>]

Sets the On-Level status of an ID as a hexbyte representation, optionally specifying the number of hops. (Defaults to 3 hops). Returns 00-FF.

Example: `setOnLevelRaw=04.05.06,7F`

Returns: `setOnLevelRaw=04.05.06,7F`

sendX10Raw=<x10type>,<AddressOrCommandByte>

Sends an X10 address or command byte. x10type is zero (0) for an address, or one (1) for a command.

Example: `sendX10Raw=00, 66` (sends X10 address A1).

sendPLC=<data...>

Sends direct raw hexadecimal bytes to the PLC, as if through a direct connection such as serial USB or RS232.

Example: `sendPLC=02 40 01 65 00 01 FF 33 66`

sendEventRaw=<eventID>

Sends an event (00-FF) for the PLC to execute (see *IBIOS Event Summary Table*[185]).

SDM Commands – Memory

setImage=<address>,<hexdata...>

Downloads data to PLC at thegiven address (map-friendly).

Examples: `setImage=0x0040,02 FF`

or: `setImage=NTL_TIMERS,02 FF`

getImage=<address>,<length>

Uploads a block of memory from the PLC to the PC. Returns hex bytes (map-friendly).

Example: `getImage=0x0040,2`

Returns: `getImage=0x0040,00 00`

saveImage=<address>,<length>,<filename>

Gets a block of memory from the PLC and saves it to a file.

Example: `saveImage=0x0210,2,image.txt`

setBit=<address>,<bit>[,<setTo0or1>]

Sets the bit (0-7) at the address (map-friendly) with an optional value of 1 (set) or 0 (clear); defaults to 1.

Examples `setBit=0x0040,4`

or: `setBit=0x0040,4,0`

clearBit=<address>,<bit>

Clears the bit (0-7) at the given address (map-friendly).

Example: `clearBit=0x0040,4`

getWord

Gets a two-byte word from the PLC. Returns the address specified, the two bytes at this address (msb, lsb), the decimal value of the two bytes, and the equivalent of the two bytes in time format (HH:MM).

Example: `getWord=0x0210`

Returns: `getWord=0x0210,02 30,560,09:20`

repeatGetByte=<address>,<bytecount>

Repeatedly gets two bytes for load testing the SDM.

Example: `repeatGetByte=0x0210,5`.

This example queries two bytes at location 0x0210 five times.

downloadCoreApp[=clear]

Downloads the included SALad coreApp to the PLC so that it can communicate to the SDM. The optional =clear parameter also instructs the coreApp to clear the ALL-Link Database after downloading. Automatically resets the PLC after downloading.

Example: `downloadCoreApp`

downloadSALadFile=<filename>

Downloads the SALad program with the given filename (as a binary/compiled file, not hex). Automatically resets the PLC after downloading.

Example: `downloadSALadFile=coreUser.slb`

downloadBinaryFile=<address>,<filename>

Downloads the binary file with the given filename to the given address. This command does *not* automatically reset the PLC after download.

Example: `downloadBinaryFile=<0x2000>,myTable.bin`

setPath

Sets the search path for expected files.

Example: `setPath=c:\mysaladfiles`

getPath

Gets the search path for expected files.

Example: `getPath`

Returns: `getPath=c:\mysaladfiles`

verifyCoreApp

Returns true if the currently downloaded SALad application matches the expected SALad coreApp.

Example: `verifyCoreApp`

Returns: `verifyCoreApp=true`

salad

Returns true for a valid SALad application loaded in the PLC. Returned vid and pid are specific to the manufacturer, and rev is the SALad application's revision number.

Example: `salad`

Returns: `salad=true,vid=0001,pid=0003,rev=000C`.

unlockSALad

Disables SALad code protection to allow downloads of SALad code.

Example: `unlockSALad`

lockSALad

Enables SALad code protection to prevent overwriting SALad code (default on powerup).

Example: `lockSALad`

isSALadLocked

Returns the status of the SALad code protection lock.

Example: `isSALadLocked`

SDM Commands – PLC Control

port=<PLCport>

Sets the sticky, global PLC port (COM#|USB4|SIM28|?). Sending a question mark '?' causes the port to be searched for. Use **getport=** to read the found port. The port is saved in the registry and reused upon restart of the SDM.

Examples: `port=COM1`

or: `port=USB4`

or: `port=?`

or: `port=SIM28`

getPort

Returns the current sticky, global PLC port.

Example: `getPort`

Returns: `getport=USB4`

getFirmware

Returns the PLC firmware revision.

Example: `getFirmware`

Returns: `getFirmware=2.9`

sendHardReset

Resets the SALad application, reinitializes (fires the IBIOS Event EVNT_ INIT).

getClock

Gets the PLC's Running Clock, which is reset from the PLC's Realtime Clock on powerup or initialization of the SALad coreApp.

Example: `getClock`

Returns: `getClock=true,8/12/2005 5:03:39 PM`

getRTClock

Gets the current Realtime Clock.

Example: `getRTClock`

Returns: `getRTClock=true,8/12/2005 5:03:39 PM`

setClock

Sets *both* the Realtime Clock and the Running Clock.

Example: `setClock=8/12/2004 5:03:39 PM`

setRunningClock

Sets only the Running Clock, not the Realtime Clock. Next time the PLC is plugged in or reset, coreApp will reset the Running Clock with the time from the Realtime Clock.

Example: `setRunningClock=8/12/2004 5:03:39 PM`

setRTClock

Sets only the Realtime Clock, not the Running Clock. Next time the PLC is plugged in or reset, coreApp will reset the Running Clock with the time from the Realtime Clock.

Example: `setRTClock=8/12/2004 5:03:39 PM`

getDST

Returns the PLC's current daylight savings time settings.

Example: `getDST`

connect

Connects to the PLC after a disconnection. Connecting is automatic at startup.

Example: `connect`

disconnect

Disconnects the PLC's port connection.

Example: `disconnect`

isResponding

Asks the SDM if the port is responding (sends 0x02 0x48) and responds true or false. This also reads the map for proper name-based downloading. This is the ultimate heartbeat method to determine if the PLC is connected and talking.

Example: `isResponding`

Returns: `isResponding=true`

GetPLCStatus

Returns a set of statuses for PowerLinc Controller (port, connection, etc).

Example: `getPLCStatus`

help

Provides a quick visual list of commands.

Example: `help`

getMyID

Returns the PLC's ID (INSTEON Address).

Example: `getMyID`

Returns: `getMyID=01.02.03`

diag1

Does a first-level diagnosis of the PLC. Returns true or false.

Example: `diag1`

Returns: `diag1=true`

debugAnimate=<map|<mapfile>|true>

Causes PLC to return its code pointer, optionally with SALad mapping.

Example: `debugAnimate=true`.

debugOff

Turns off code pointer information from debugAnimate.

Example: `debugOff`.

SDM Commands – Device Manager Control

setTextMode=<textmode>

Sets the global communication format that the SDM uses to receive and respond. Currently 'text' (default) and 'flash' are supported. 'flash' mode adds ampersands (&) around each response in order for the loadVariables() function to receive values from Macromedia Flash or SwishMax-type clients.

nop

Does nothing, but allows an HTTP connection to return collected data to the client.

echo

Echoes the text simply to see if the ActiveX or HTTP communication is working.

_localBytes=<true|false>

Turns local (window) SDM byte-level debugging on or off. Opposite of remoteBytes.

remoteBytes=<true|false>

Turns local (window) SDM byte-level debugging on or off. Opposite of _localBytes.

setBlocked=<true|false>

Turns global blocking of commands on or off. Default is off (false). When blocked, the action is executed before receiving a response and returning to the client. When not blocked, the client is returned a true response that the command was accepted for execution by the SDM. When the actual response is received by the SDM, it is sent to the client via ActiveX, or queued for return via HTTP. (Use NOP to get results when no execution action is necessary).

setPageSizeThrottle=<true|false>

Allows global throttling of the downloads in case of errors. Default is true. The download page size shrinks when multiple consecutive errors are received. The page size grows when multiple consecutive successes are received.

setPageErrors=<maxErrorCount>

Sets the global maximum number of consecutive retries before a download actually fails. Default is 7.

setPageSize=<pageSize>

Sets the global packet size for downloading. Default is 32. When throttling is true, this value automatically shrinks and grows.

if=<Command>,<matchResult>,<trueText>[,<falseText>]

Executes the command, then matches against the matchResult. If true, returns the trueText. If false (and the falseText is present), returns the falseText.

Example: `if=getFirmware,2.9,good,bad`

ifExec=<Command>,<matchResult>,<trueCommand>[,<falseCommand>]

Executes the command, then matches against the matchResult. If true, executes the trueCommand. If false (and the falseCommand is present), executes the falseCommand.

Example: `ifExec=getFirmware,2.9,"setOnLevelText=00.02.BA,ON",nop`

setAuthUsername=<username>

Allows the user to set an authorization username for the http/web connection to require. Shipped default is empty - no authorization required.

Example: `setAuthUsername=me`

setAuthPassword=<password>

Allows the user to set an authorization password for the http/web connection to require. Shipped default is empty - no authorization required. To clear, send setAuthPassword with no password.

Example: `setAuthPassword=12345`

icon=<hide|show>

Hides or shows the SDM icon.

Example: `icon=hide`

repeat=<times>,<command>

Repeats a command for testing the SDM.

Example: `repeat=5,setOnLevelText=00.57.75,on`

dm

Opens the edit field with focus on white space.

cls

Closes the log.

closeDM or **haltDM** or **exitDM**

These are different ways to force a shutdown of the SDM.

SDM Commands – ALL-Link Database Management

addID=<remoteINSTEONid>[<group>][,<isMaster=true>]

Adds a device's ID to the PLC's ALL-Link database, optionally specifying the ALL-Link Group number and whether the device is a Controller (master) or Responder (slave).

Example: `addid=04.05.06`

removeID=<remoteINSTEONid>

Deletes a device's ID from the PLC's ALL-Link database.

Example: `removeID=04.05.06`

getRemoteRecord=<INSTEONid>, <record number>[,<end-range record number>]

Gets and block-returns remote ALL-Link Database records.

Examples: `getRemoteRecord=04.05.06,2`

or: `getRemoteRecord=04.05.06,2,3`

Returns:
```
getremoterecord=
=====RECORDS BEGIN=====
remoterec(04.05.06):#2:A2 01 00 D0 80 FE 1F 00
remoterec(04.05.06):#3:A2 02 00 6B C2 FE 1F 00
=====RECORDS END=====
```

getLinks

Returns a block-list of ALL-Link Database records in the PLC.

Example: getLinks

getRemoteGroupRecord=[<recno>:] <remoteINSTEONid>, <groupID> [,<sourceINSTEONid>] [,<hops>]

Scans the remoteINSTEONid's Linear (non-PLC) ALL-Link Database for the sourceID (defaults to PLC's ID) and groupID, or uses the recno provided. Returns full record information as getRemoteGroupRecord=<remoteINSTEONid>, <sourceINSTEONid>, <groupID>, <onLevelText>, <rampRate>

Example: `getRemoteGroupRecord=04.05.06, 1`

Returns: `getRemoteGroupRecord=04.05.06, 01.02.03,1,50%,31`

setRemoteGroupRecord=[<recno>:] <remoteINSTEONid>, <groupID>,<sourceINSTEONid>,<hops>, <newPresetDim>, <newRampRate>

Scans the remoteINSTEONid's Linear (non-PLC) ALL-Link Database for the sourceID (defaults to PLC's ID), and groupID, or uses the recno provided. Sets full ALL-Link Database record information.

Examples: `setRemoteGroupRecord=1:04.05.06, 1, 01.02.03,3,50%,31`

or: `setRemoteGroupRecord=04.05.06, 1, 01.02.03,3,50%,31`

setPresetDim=<remoteINSTEONid>, <groupID>, <newPresetDim> [,<sourceINSTEONid>] [,<hops>]

Sets the preset On-Level value for the PLC or sourceINSTEONid (defaults to PLC's ID) in the remoteINSTEONid's database.

Example: `setPresetDim=04.05.06, 1, 25%`

setRampRate=<remoteINSTEONid>, <groupID>, <newRampRate> [,<sourceINSTEONid>] [,<hops>]

Sets the Ramp Rate value for the PLC or sourceINSTEONid (defaults to PLC's ID) in the remoteINSTEONid's ALL-Link Database. Ramp rate values are 0x00 (slow) to 0x1F (fast).

Example: `setPresetDim=04.05.06, 1, 0x1F`

getPresetDim=<remoteINSTEONid>, <groupID>, [,<sourceINSTEONid>] [,<hops>]

Gets the preset On-Level value for the PLC or sourceINSTEONid (defaults to PLC's ID) in the remoteINSTEONid's ALL-Link Database.

Example: `getPresetDim=04.05.06, 1`

getRampRate=<remoteINSTEONid>, <groupID>, [,<sourceINSTEONid>] [,<hops>]

Gets the Ramp Rate value for the PLC or sourceINSTEONid (defaults to PLC's ID) in the remoteINSTEONid's ALL-Link Database. Ramp rate values are 0x00 (slow) to 0x1F (fast).

Example: `setPresetDim=04.05.06, 1`

exportLinks=<filename>

Saves the PLC's ALL-Link Database to a file.

Example: `exportLinks=links.txt`

importLinks=<filename>

Loads the PLC's ALL-Link Database from a file.

Example: `importLinks=links.txt`

SDM Commands – Timers

Before adding or using timers, you must download the timerCoreApp.slb file and use the **clearTimers** command once. This resets all internal tables before you can add timers. Also, for using sunset/sunrise, you need to set your state and city (**setStateCity**) or lat/long (**setLatLong**) and download the sunset table (**downloadSunTable**).

downloadSALadFile=timerCoreApp.slb

Downloads the timer core application, required for timer usage. Downloading *will* prevent you from listing timers until clearTimers is used.

ClearTimers

Clears the timer tables. Eliminates all timers and resets timer variables.

setTimersXML

Sets all Timers using XML. Send the XML string without any newlines embedded.

Example:
```
setTimersXML=<Timers><ItemCount>1</ItemCount><Timer><TID
Name="Some &amp; Timer">2</TID><DeviceList><Device DID="1"
Address="00.02.C2" HouseCode="" UnitCode="" OnLevel="60%">1</Device>
</DeviceList><TOD>19:00</TOD><PlusMinusMin>+3</PlusMinusMin><DOW>M</DOW
><Security>N</Security></Timer></Timers>
```

Returns formatted XML:

```
setTimersXML=
<Timers>
  <ItemCount>1</ItemCount>
  <Timer>
    <TID Name="Some &amp; Timer">2</TID>
    <DeviceList>
      <Device DID="1" Address="00.02.C2" HouseCode="" UnitCode=""
OnLevel="60%">1</Device>
    </DeviceList>
    <TOD>19:00</TOD>
    <PlusMinusMin>+3</PlusMinusMin>
    <DOW>M</DOW>
    <Security>N</Security>
  </Timer>
</Timers>
```

getTimersXML

Gets all Timers in an XML string. See **getTimersXML** for format.

listTimers

Block-lists the currently existing timers. Returns <recordnumber>, <active|inactive>, <time|sunrise±mins|sunset±mins>, <INSTEON-ID:onlevel | INSTEON: 6bytes>, <DOW>, <SEC|NOSEC>. See **addTimer** for DOW and SEC|NOSEC information.

Example: `listTimers`

Returns:
```
listtimers=beginlist
timer=1,active,19:53,04.05.06:OFF,SuSaFThWTuM,NOSEC
timer=2,active,20:00,04.05.06:50%,SuSaFThWTuM,NOSEC
```

```
timer=3,active,19:59,INSTEON:04 05 06 4F 11 00,M,SEC
timer=4,active,sunset+20,04.05.06:OFF,SuSaFThWTuM,NOSEC
endlist
```

getTimer=<recordnumber>

Same as **listTimers**, except lists a single timer (not block-listed) specified by <recordnumber>.

Example: `getTimer=2`

Returns: `timer=2,active,20:00,04.05.06:50%,SuSaFThWTuM,NOSEC`.

diagTimers

Checks over the timer format for format errors and reports them.

Example: `diagTimers`.

setStateCity=<state>,<city>

Sets the latitude and longitude and sunrise/sunset information based on a state and city name. The user interface can enumerate the information located in places.csv and places.idx. You can query **getLatLong** or **getSunrise** or **getSunset** after setting the state and city.

Examples: `setStateCity=CA,IRVINE`

or: `setStateCity=California,Tustin`

setLatLong=<latitude>,<longitude>

Sets the latitude and longitude and sunrise/sunset information. You can query **getLatLong** or **getSunrise** or **getSunset** after setting this.

Example: `setLatLong=33.684065,-117.792581`

getLatLong

Returns the currently set latitude and longitude used for sunrise and sunset calculations.

Example: `getLatLong`

Returns `33.684065,-117.792581`

downloadSunTable

Verifies that the TimerCoreApp is installed and downloads the sunrise/sunset table (based on latitude and longitude) to the PLC for use within the TimerCoreApp. This is required once to enable the sunrise/sunset time specifications.

Example: `downloadSunTable`

getSunrise

Returns today's sunrise time as calculated using **setStateCity** or **setLatLong**.

Example: `getSunrise`

Returns `8/30/2005 6:24:00 AM`

getSunset

Returns today's sunset time as calculated using **setStateCity** or **setLatLong**.

Example: `getSunset`

Returns: `8/30/2005 7:20:00 PM`

interpretSunTable

Returns the sunrise and sunset times for the whole year

Example: `interpretSunTable`.

getNextAlarmTime

Returns the PLC's next scheduled alarm time. For timers that have <SEC|NOSEC> set to SEC, the returned time will be the *actual* time that the alarm will fire, which will occur randomly in the interval from 15 minutes before to 15 minutes after the alarm setting time. When the minutes-from-midnight (**getMinutes**) matches the **getNextAlarmTime**, the alarm or alarms that match will fire.

Example: `getNextAlarmTime`

Returns: `HH:MM` such as `14:25`

setNextAlarmTime=HH:MM

Sets the PLC's next alarm time. Useful to skip alarms, if desired, while they remain set for the next day.

Example: `setNextAlarmtime=18:00`

This example skips all timers until 6pm today.

getMinutes

Returns the PLC's Running Clock, set on initialization and auto-incremented each minute.

Example: `getMinutes`

Returns `HH:MM` such as `15:25`.

setMinutes=HH:MM

Sets the PLC's Running Clock, normally set on initialization and auto-incremented each minute. Useful to debug timers by setting the PLC's match for alarms without actually changing the PLC's Realtime Clock.

Example: `setMinutes=18:00`

SDM Windows Registry Settings

The SDM's root location is:

HKEY_CURRENT_USER\Software\Smarthome\ SmarthomeDeviceManager

Valuename: **port** = <port> - sticky global port for SDM to connect - USB4 or COM1 to COM255. Set from the port= command.

Valuename: **servername** = <exefilename> - SDM's executable for clients to autorun.

Valuename: **usehttp** = <true|false> - allow the http server to accept connections. (Defaults to true)

Valuename: **httpport** = <portnumber> - when the http server activates, the server uses this port (Defaults to 9020).

Chapter 13 — INSTEON Hardware Documentation

In This Chapter

INSTEON Hardware Development Kit (HDK) Reference

In This Section

Hardware Development Kit Overview

The HDK consists of a Main Board and a Daughter Board.

HDK Main Board

There are two basic HDK Main Board designs: **Isolated** (for interfacing to the powerline without a direct electrical connection), and **Non-Isolated** (for building into insulated devices like light switches that have direct access to 120 VAC). The HDK available for purchase from SmartLabs contains an **Isolated** Main Board only. The **Non-Isolated** design presented here is for reference only, and SmartLabs assumes no liability for its use.

The Main Board consists of:

- INSTEON Powerline Interface
- TTL-level Serial Communications Interface
- INSTEON Micro Controller Unit (MCU)
- INSTEON ALL-Linking User Interface (Button / LED)
- Power Supply

HDK Daughter Board

The Daughter Board brings out the signals from the Main Board, and it has an experimental area for you to develop your circuitry.

The Daughter Board consists of:

- Experimental Design Area
- Button for connecting Daughter Board Interrupt line to ground
- LED connected to one General Purpose I/O line

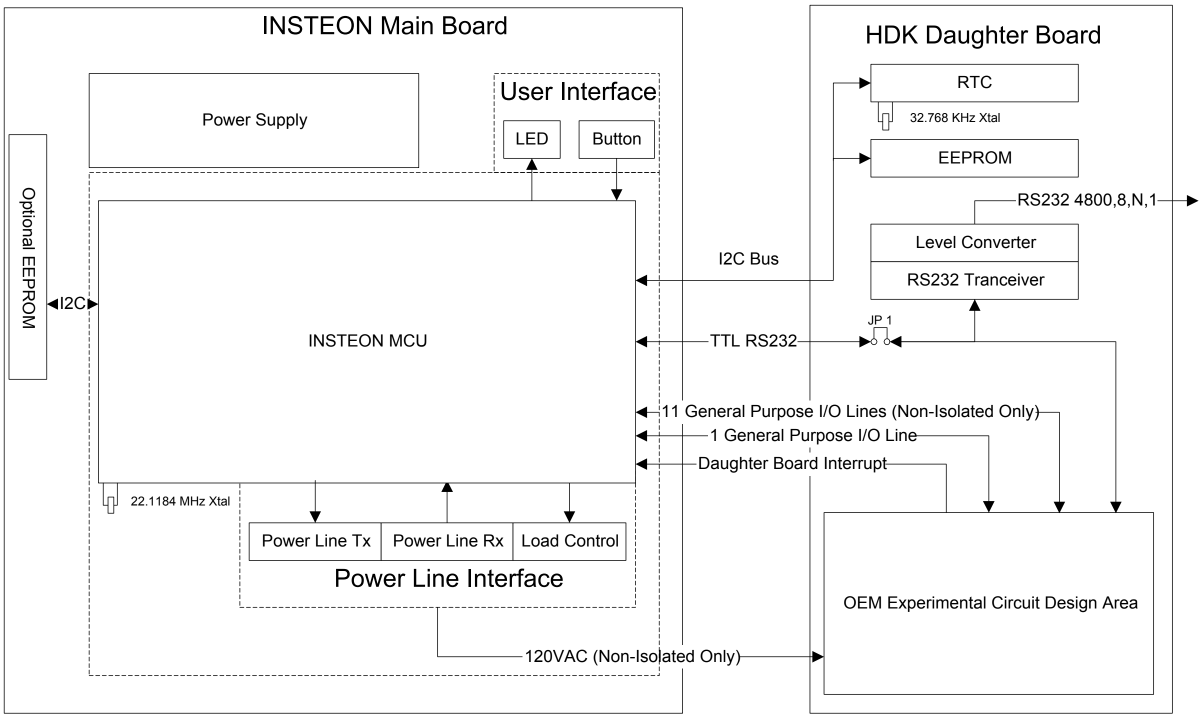
When connected to the **Non-Isolated** Main Board, the Daughter Board can make use of the following hardware:

- LED to simulate Load Control
- 11 Extra General Purpose I/O lines
- 120 VAC

Functional Block Diagram

This diagram shows the functional blocks on each board. The Main Board serves as the INSTEON modem. The INSTEON chip executes SALad application code. The Daughter Board functions as a place to develop OEM circuits.

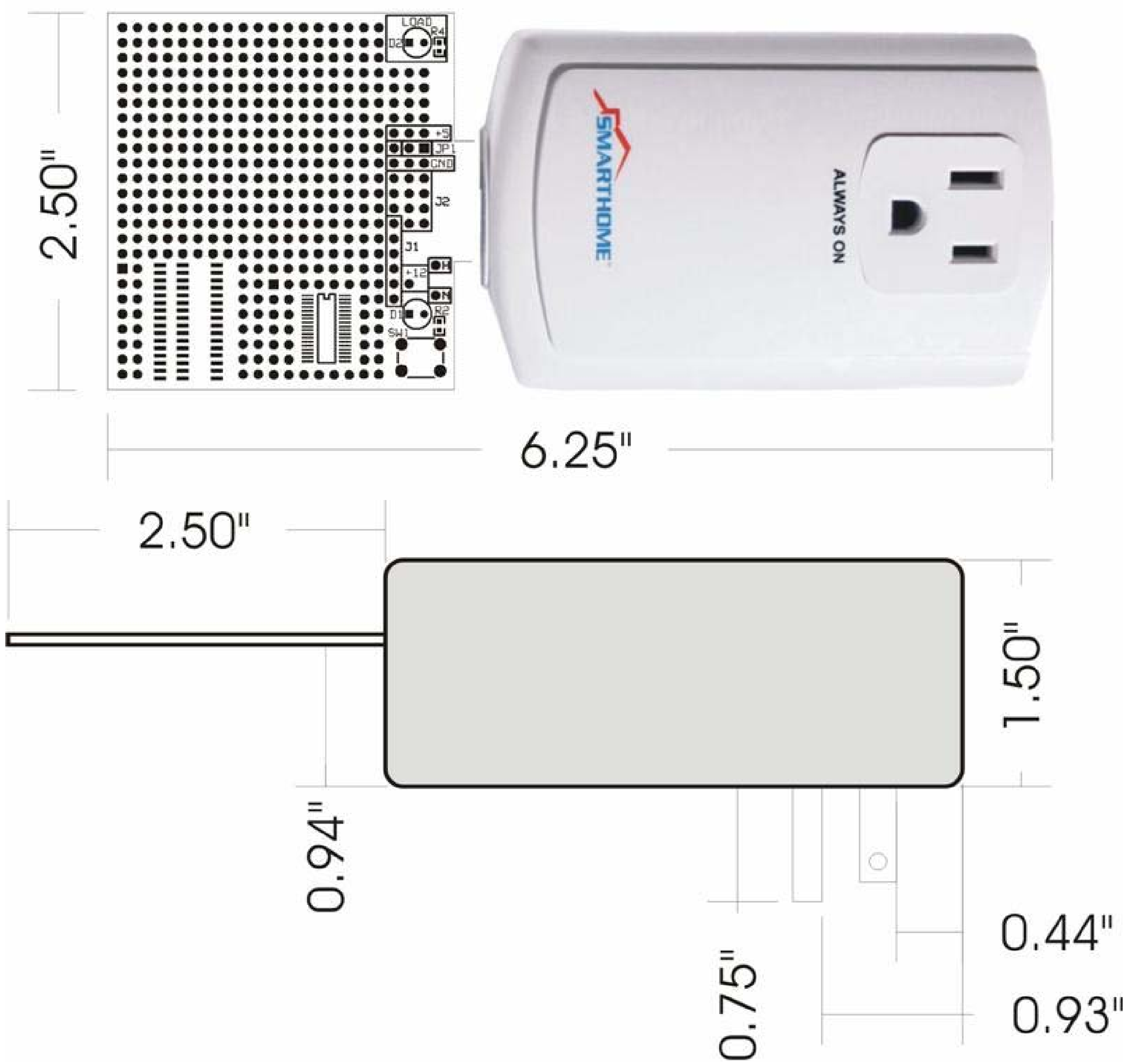
INSTEON HDK Hardware Block Diagram



HDK Physical Diagrams

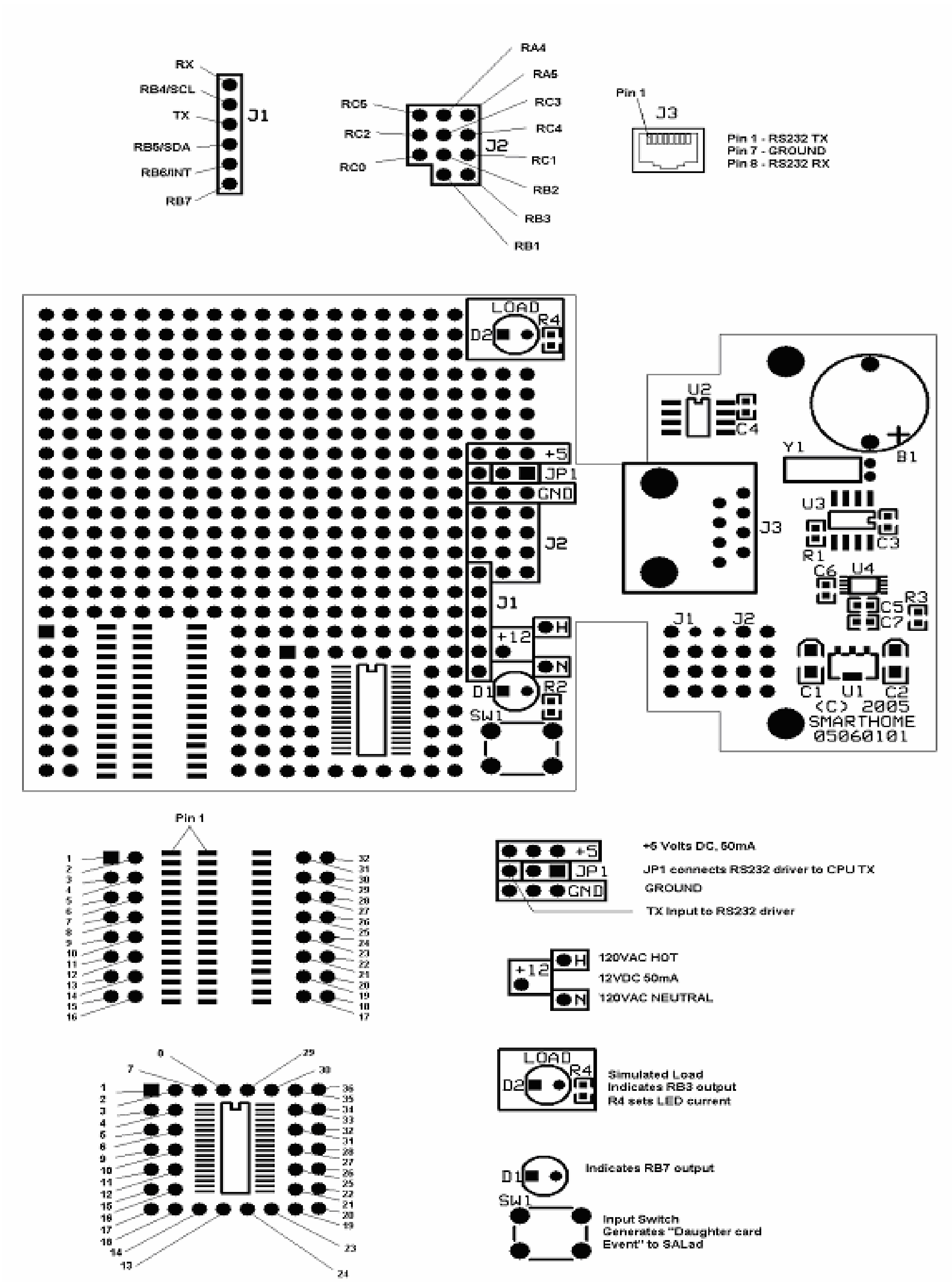
This diagram shows the physical dimensions of the HDK unit.

HDK Physical Dimensions



HDK Daughter Board

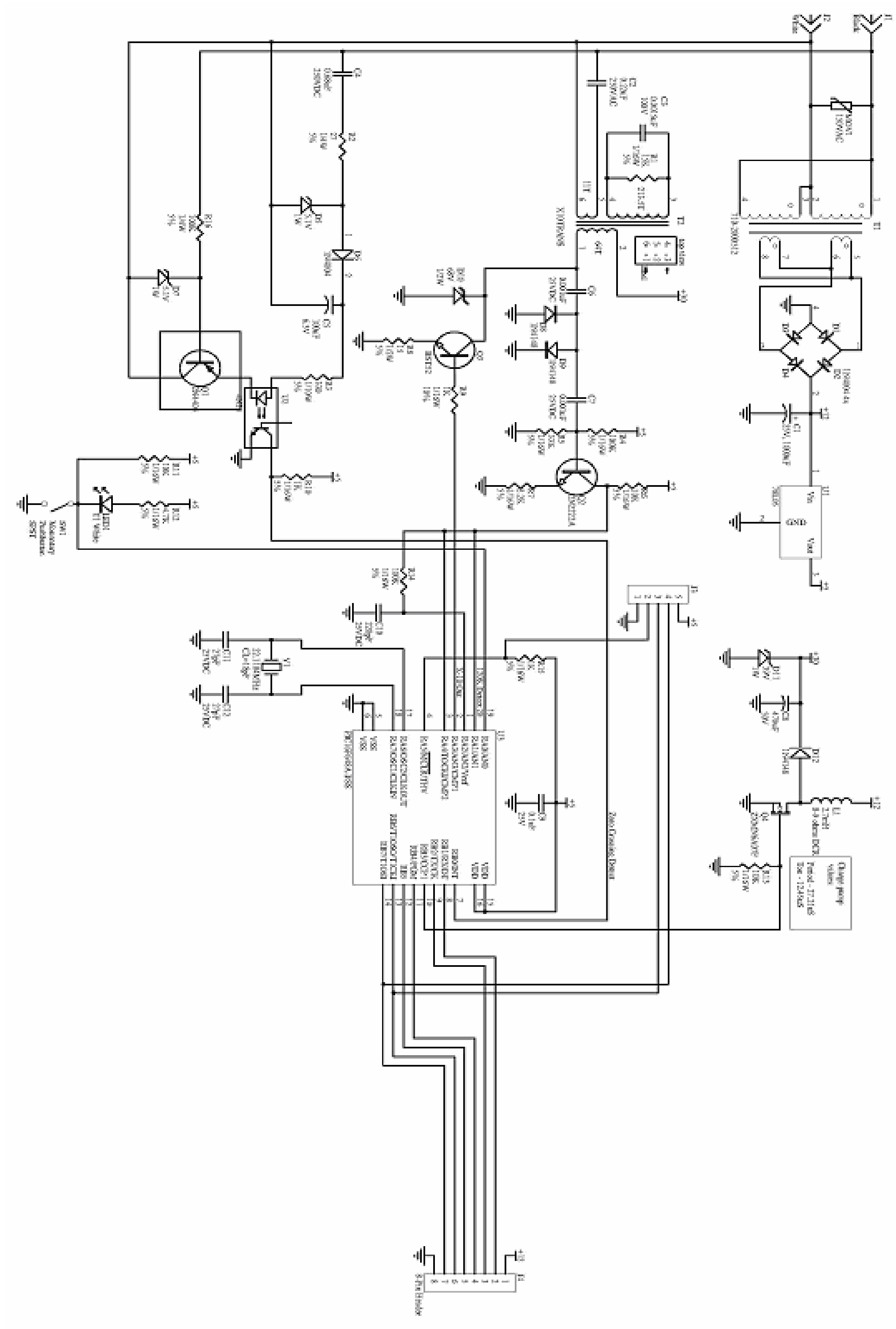
This diagram shows the HDK Daughter Board, and where signals and parts are located on the board.



Hardware Development Kit Schematics

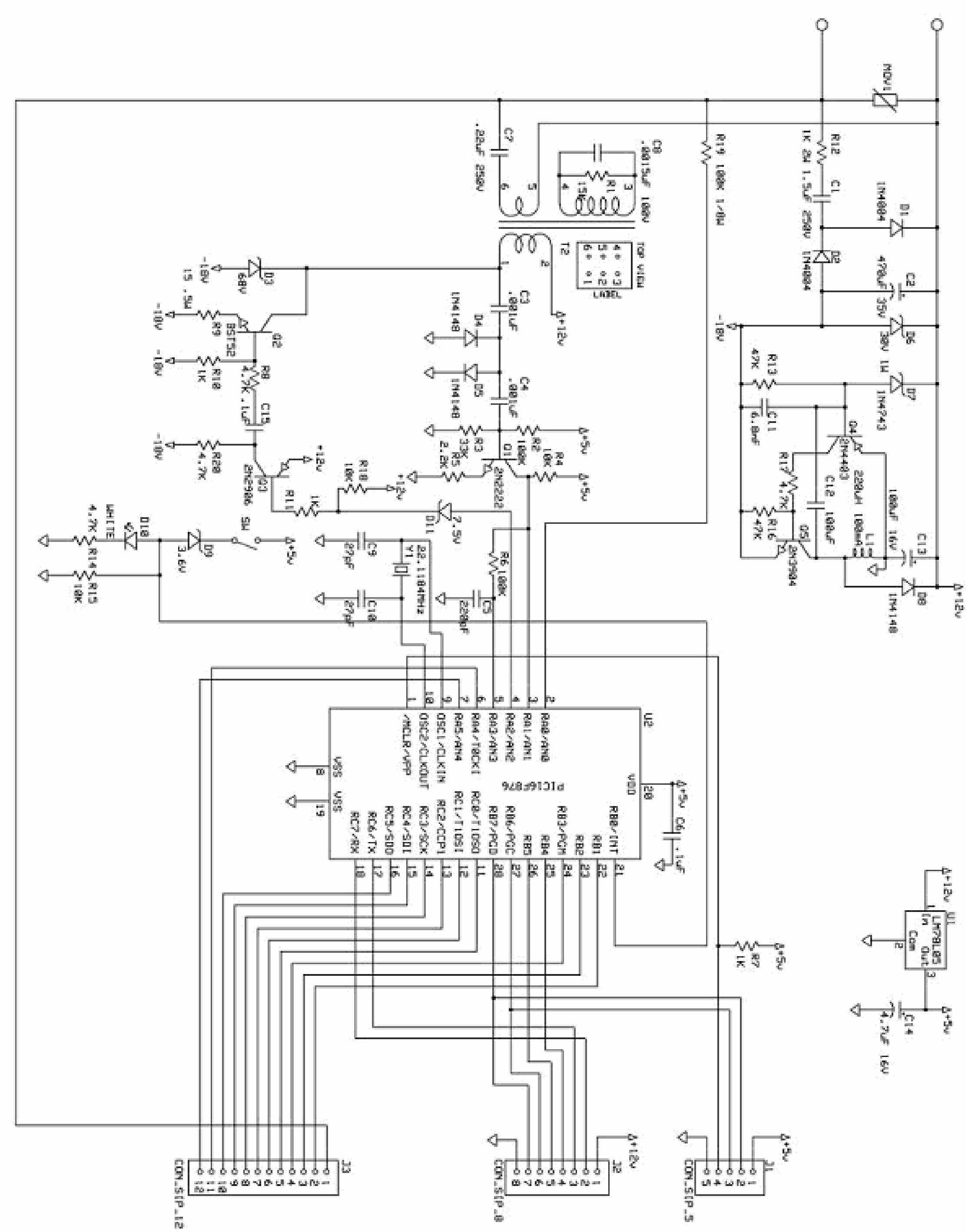
In This Section

HDK Isolated Main Board Schematic

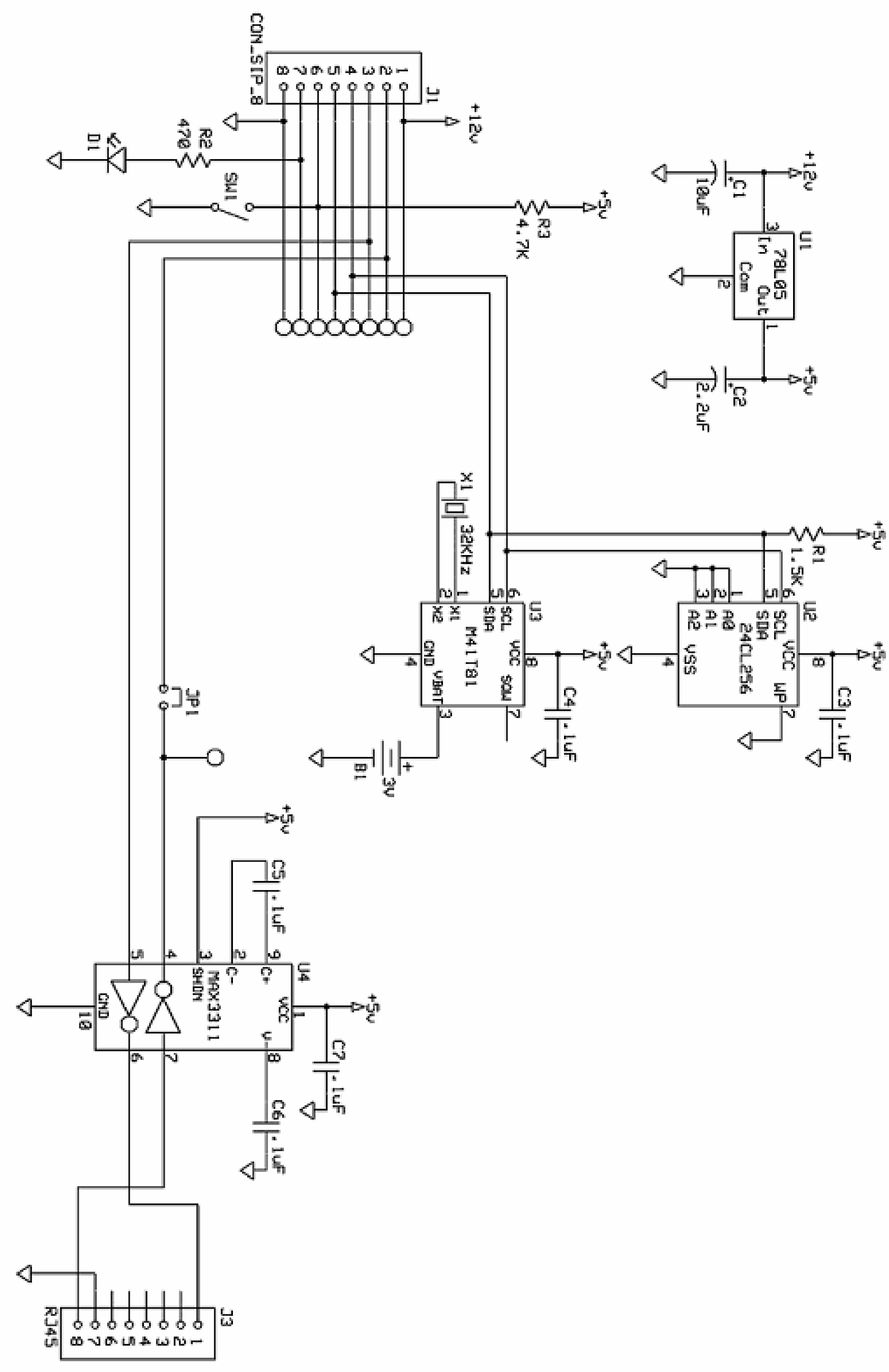


HDK Non-Isolated Main Board Schematic

This non-isolated design is only intended for those experts who are developing products that must achieve the lowest possible cost while still communicating over the powerline. To reduce the part count, the power supply connects directly to the 110-volt mains, so potentially lethal voltages are exposed. SmartLabs assumes no liability for use of this design.



HDK Daughter Board Schematic



SmartLabs Powerline Modem (PLM) Hardware Reference

This section gives a reference design for using the IN2680A Powerline Modem chip in a module connected both to the powerline and to a host device. The design uses a main board for the modem chip, power supply, INSTEON powerline interface, and TTL-level serial communications, and a daughter board for interfacing to a host.

Two different daughter board designs are included. One is for an RS232 interface, and the other is for an IP (Ethernet) interface. A USB interface is under development. Developers may create their own daughter cards to implement custom interfaces.

The reference design presented here is the same one that SmartLabs uses for its Powerline Modem™ (PLM) module.

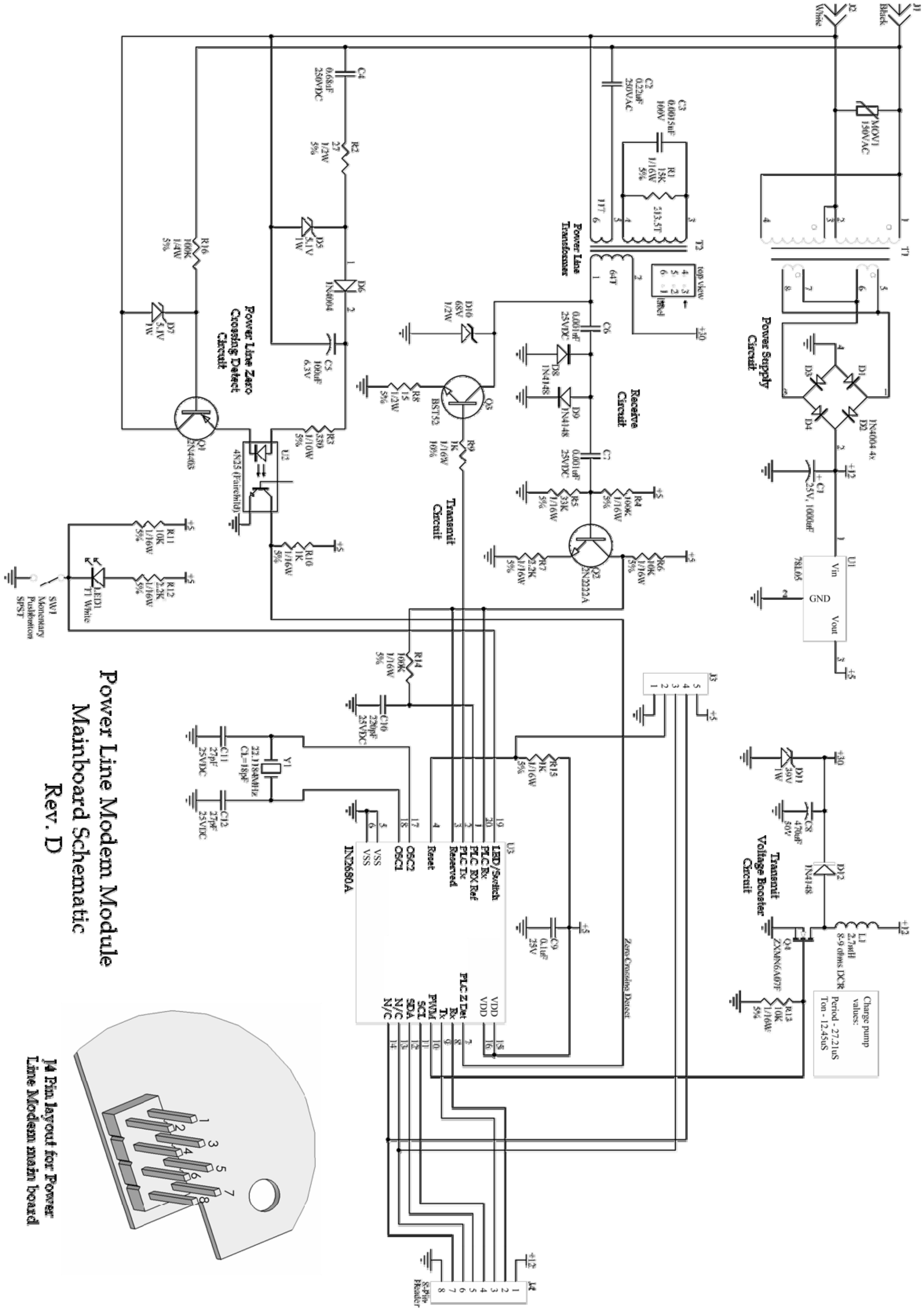
In This Section

SmartLabs Powerline Modem (PLM) Main Board

The SmartLabs Powerline Modem™ (PLM) main board includes the IN2680A Powerline Modem chip, a transformer-isolated power supply with a 30-volt charge pump booster, a transformer-coupled powerline signal transponder, an optically-isolated zero crossing detector, and an 8-pin daughter board connector for TTL-level host communications.

In This Section

SmartLabs PLM Main Board Schematic



SmartLabs PLM Main Board Bill of Materials

Description	Part Type	Desig-nator	Footprint	Remark
Capacitor	Electrolytic, 1000uF, 25V	C1	Through-hole, 0.2"	
Capacitor	Metal Polyester, 0.22uF, 250VDC	C2	Through-hole, 0.3"	
Capacitor	Ceramic, 0.0015uF, 100V	C3	SMT, 0805	
Capacitor	Metal Polyester, 0.68uF, 250VDC	C4	Through-hole, 0.4"	
Capacitor	Electrolytic, 100uF, 6.3V	C5	Through-hole, 0.1"	
Capacitor	Ceramic, 0.001uF, 25V	C6	SMT, 0603	
Capacitor	Ceramic, 0.001uF, 25V	C7	SMT, 0603	
Capacitor	Electrolytic, 470uF, 50V	C8	Through-hole, 0.2"	
Capacitor	Ceramic, 0.1uF,25V	C9	SMT, 0603	
Capacitor	Ceramic, 220pF, 25V	C10	SMT, 0603	
Capacitor	Ceramic, 27pF, 25V	C11	SMT, 0603	
Capacitor	Ceramic, 27pF, 25V	C12	SMT, 0603	
Crystal	22.1184MHz, 18pF Load	Y1	Through-hole	Recommended: Citizen model CMR309T22.1184MABJTR
Diode	DL4004	D1	SMT, MELF	
Diode	DL4004	D2	SMT, MELF	
Diode	DL4004	D3	SMT, MELF	
Diode	DL4004	D4	SMT, MELF	
Diode	Zener, 5.1V, 1W	D5	SMT, MELF	
Diode	DL4004	D6	SMT, MELF	
Diode	Zener, 5.1V, 1W	D7	SMT, MELF	
Diode	1N4148	D8	SMT, Mini-MELF	
Diode	1N4148	D9	SMT, Mini-MELF	
Diode	Zener, 68V, 1/2W	D10	SMT, Mini-MELF	
Diode	Zener, 39V, 1W	D11	SMT, MELF	
Diode	1N4148	D12	SMT, Mini-MELF	
Header	5-Pin male	J3	Through-hole, 0.1" ctr	For in-circuit programming
Header	2X4 male	J4	Through-hole, 0.1" ctr	Used to connect to daughter boards
Inductor	2.7mH, 8-9 ohms DCR, 100mA DCI	L1	Through-hole, 0.2"	
LED	Any single color is acceptable	LED1	Through-hole, T1	
MCU	INSTEON IN2680A	U3	SMT, SSOP20	
MOSFET	N-Channel, Zetex ZXMN6A07F	Q4	SMT, SOT-23	
Optocoupler	100% Transfer ratio @ 8mA I_F and 5mA I_C	U2	SMT	Recommended: Fairchild 4N25SM or 4N25S
Regulator	78L05 Positive 5V regulator	U1	Through-hole, TO-92	
Resistor	15KW, 1/16W, 5%	R1	SMT, 0603	
Resistor	27W, 1/2W, 5%	R2	SMT, 1210	Recommended: Panasonic ERJ-P14J27OU Anti-Surge

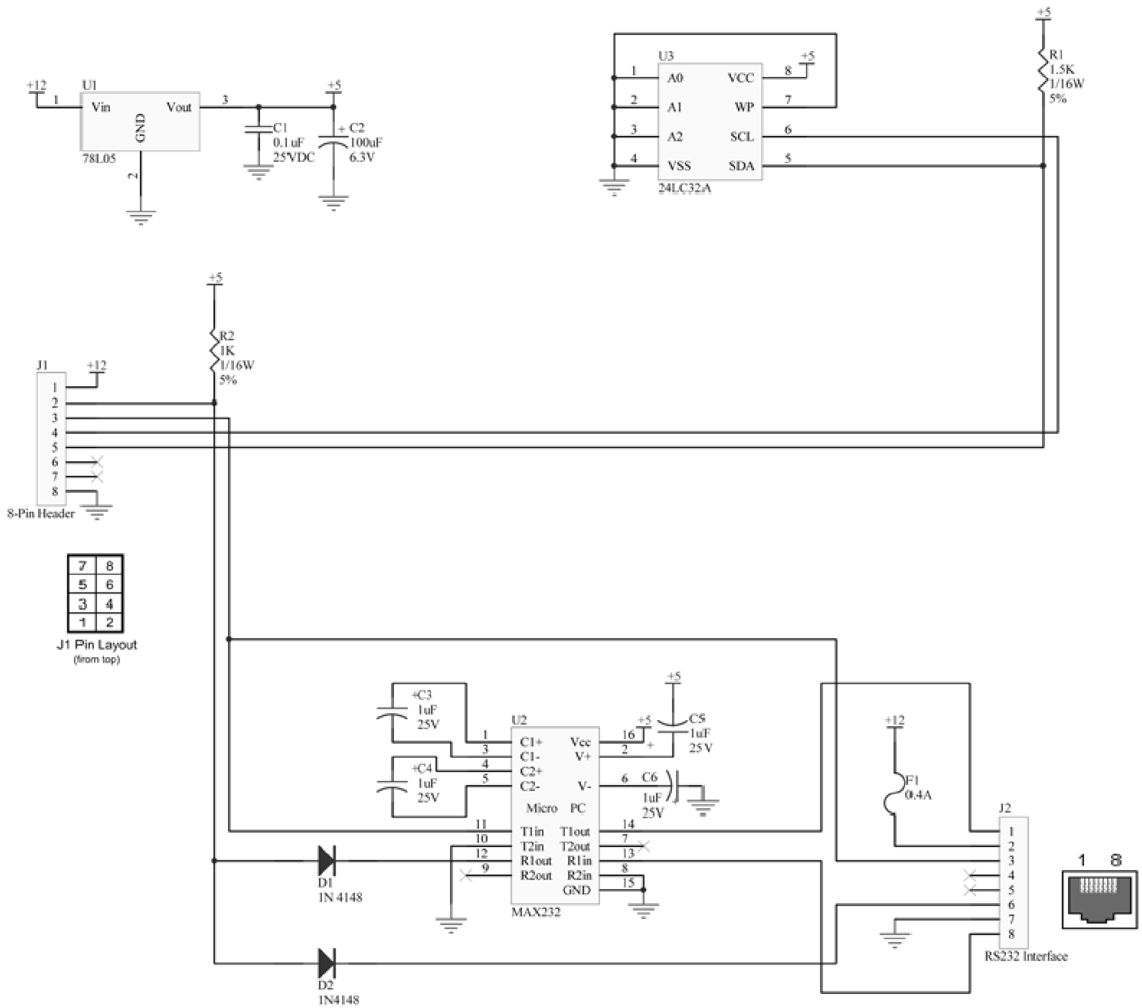
Description	Part Type	Desig-nator	Footprint	Remark
Resistor	330W, 1/10W, 5%	R3	SMT, 0805	
Resistor	100KW, 1/16W, 5%	R4	SMT, 0603	
Resistor	33KW, 1/16W, 5%	R5	SMT, 0603	
Resistor	10KW, 1/16W, 5%	R6	SMT, 0603	
Resistor	2.2KW, 1/16W, 5%	R7	SMT, 0603	
Resistor	15W, 1/2W, 5%	R8	SMT, 2010	
Resistor	1KW, 1/16W, 5%	R9	SMT, 0603	
Resistor	1K, 1/16W, 5%	R10	SMT, 0603	
Resistor	10KW, 1/16W, 5%	R11	SMT, 0603	
Resistor	2.2KW, 1/16W, 5%	R12	SMT, 0603	May be changed to control LED brightness
Resistor	10KW, 1/16W, 5%	R13	SMT, 0603	
Resistor	100KW, 1/16W, 5%	R14	SMT, 0603	
Resistor	1KW, 1/16W, 5%	R15	SMT, 0603	
Resistor	100KW, 1/4W, 5%	R16	SMT, 1206	
Switch	Tact Switch	SW1	Through-hole	
Transformer	Power Transformer, model 710-2000512	T1	Through-hole	Custom made, available from SmartLabs
Transformer	Power line transformer coil	T2	Through-hole	Abracon AIRV-111 PLC
Transistor	2N4403 PNP	Q1	SMT, SOT-23	
Transistor	2N2222A NPN	Q2	SMT, SOT-23	
Transistor	BST-52 Darlington NPN	Q3	SMT, SOT-89	Recommended brand: Zetex
Varistor	150VAC Metal Oxide Varistor	MOV1	Through-hole, 0.2"	
Wire	Hot wire, black, 16AWG, 300V, 105°C, VW-1	J1	Through-hole	In from power prong
Wire	Neutral wire, white, 16AWG, 300V, 105°C, VW-1	J2	Through-hole	In from power prong

SmartLabs PLM Serial (RS232) Daughter Board

The Serial Daughter Board attaches to the Powerline Modem™ (PLM) Main Board using an 8-pin connector, and to a host device using an RJ-45 jack. Host communications uses the RS232 protocol at TTL signal levels.

In This Section

SmartLabs PLM Serial Daughter Board Schematic



Serial & TTL Daughter
Card Schematic
Rev. A

SmartLabs PLM Serial Daughter Board Bill of Materials

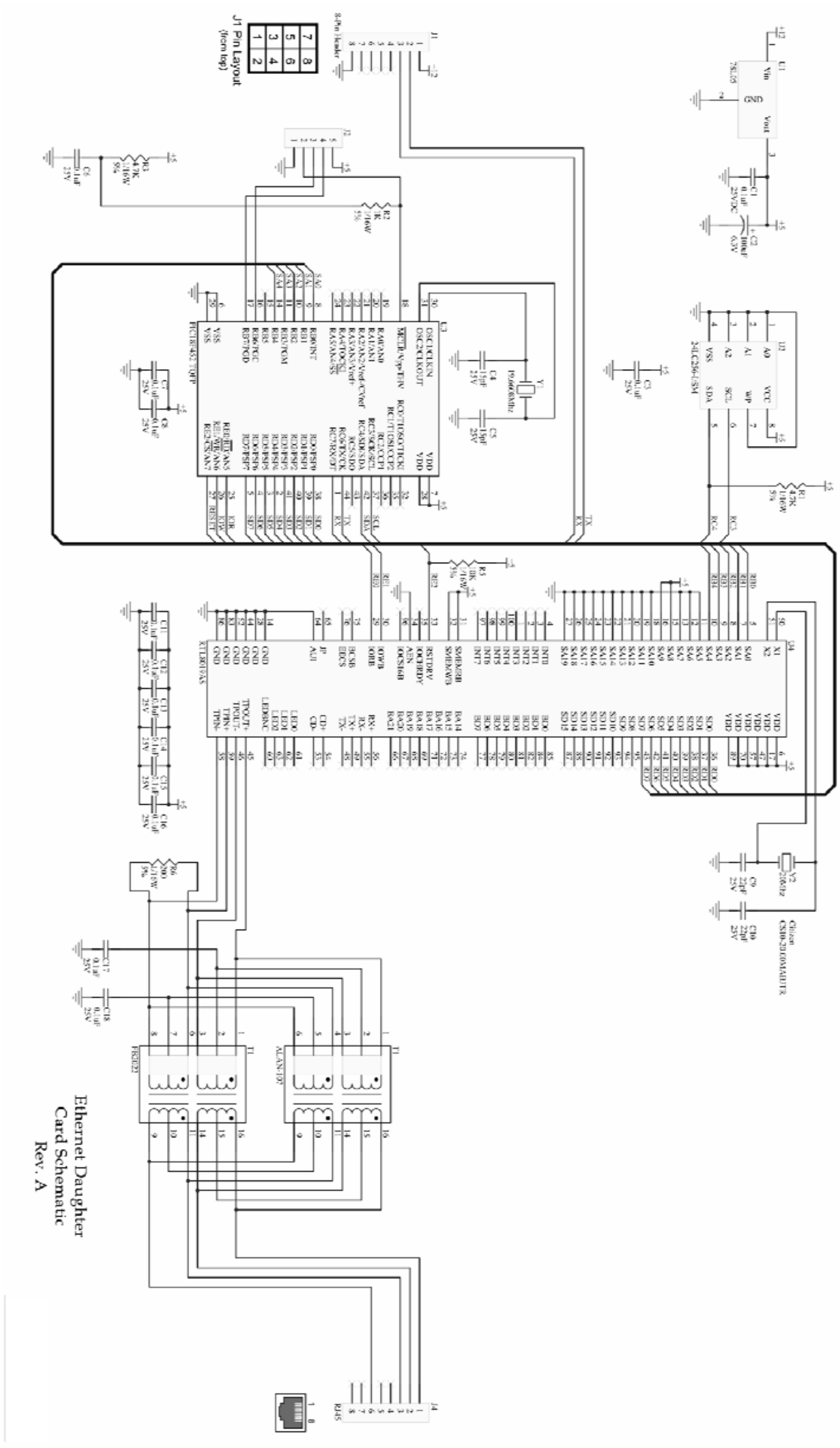
Description	Part Type	Desig-nator	Footprint	Remark
Capacitor	Ceramic, 0.1uF, 25V	C1	SMT, 0603	
Capacitor	Electrolytic, 100uF, 6.3V	C2	Through-hole	
Capacitor	Electrolytic, 1uF, 25V	C3	Through-hole	
Capacitor	Electrolytic, 1uF, 25V	C4	Through-hole	
Capacitor	Electrolytic, 1uF, 25V	C5	Through-hole	
Capacitor	Electrolytic, 1uF, 25V	C6	Through-hole	
Diode	1N4148	D1	SMT, Mini-MELF	
Diode	1N4148	D2	SMT, Mini-MELF	
Driver / Receiver	MAX232 Multichannel RS-232 ST232BDR	U2	SMT, SOIC16	
EEPROM	24LC32A	U3	SMT, SOIC8	
Fuse	250V, 0.4A	F1	Through-hole	
Header	Female 2x4, 2x4PIN, 2.54mm, 2185-20	J1	Through-hole, 0.1” ctr	
Jack	Female RJ45	J2	SMT	
Resistor	1.5KΩ, 1/16W, 5%	R1	SMT, 0603	
Resistor	1KΩ, 1/16W, 5%	R2	SMT, 0603	
Voltage Regulator	5V Zetex ZSR500G	U1	SMT, SOT223	

SmartLabs PLM Ethernet (IP) Daughter Board

The IP (Ethernet) Daughter Board attaches to the SmartLabs Powerline Modem™ (PLM) Main Board using an 8-pin connector, and to an Ethernet LAN using an RJ-45 jack.

In This Section

SmartLabs PLM Ethernet (IP) Daughter Board Schematic



SmartLabs PLM Ethernet (IP) Daughter Board Bill of Materials

Description	Part Type	Desig-nator	Footprint	Remark
Capacitor	Ceramic, 0.1uF, 25V	C1	SMT, 0603	
Capacitor	Electrolytic, 100uF, 6.3V	C2	Through-hole, 0.1"	
Capacitor	Ceramic, 0.1uF, 25V	C3	SMT, 0603	
Capacitor	Ceramic, 15pF, 25V	C4	SMT, 0603	
Capacitor	Ceramic, 15pF, 25V	C5	SMT, 0603	
Capacitor	Ceramic, 0.1uF, 25V	C6	SMT, 0603	
Capacitor	Ceramic, 0.1uF, 25V	C7	SMT, 0603	
Capacitor	Ceramic, 0.1uF, 25V	C8	SMT, 0603	
Capacitor	Ceramic, 22pF, 25V	C9	SMT, 0603	
Capacitor	Ceramic, 22pF, 25V	C10	SMT, 0603	
Capacitor	Ceramic, 0.1uF, 25V	C11	SMT, 0603	
Capacitor	Ceramic, 0.1uF, 25V	C12	SMT, 0603	
Capacitor	Ceramic, 0.1uF, 25V	C13	SMT, 0603	
Capacitor	Ceramic, 0.1uF, 25V	C14	SMT, 0603	
Capacitor	Ceramic, 0.1uF, 25V	C15	SMT, 0603	
Capacitor	Ceramic, 0.1uF, 25V	C16	SMT, 0603	
Capacitor	Ceramic, 0.1uF, 25V	C17	SMT, 0603	
Capacitor	Ceramic, 0.1uF, 25V	C18	SMT, 0603	
Controller	Ethernet controller, Realtek RTL8019AS	U4	SMT, QFP-100	
Crystal	19.6608MHz Crystal, 18pF Load	Y1	Through-hole	
Crystal	20MHz Crystal, 18pF Load	Y2	SMT	
Header	5-Pin Male	J2	Through-hole, 0.1" ctr	For in-circuit programming
Jack	RJ45 Female jack	J4	SMT	
MCU	PIC18F452-I/PT	U3	SMT, TQFP-44	
Memory	24LC256-I/SN	U2	SMT, SOP-8	
Regulator	78L05 5V Voltage regulator	U1	SMT, SOT-223	
Resistor	4.7KW, 1/16W, 5%	R1	SMT, 0603	
Resistor	1KW, 1/16W, 5%	R2	SMT, 0603	
Resistor	4.7KW, 1/16W, 5%	R3	SMT, 0603	
Resistor	10KW, 1/16W, 5%	R5	SMT, 0603	
Resistor	200W, 1/16W, 5%	R6	SMT, 0603	
Transformer	Ethernet transformer, Abracon ALAN-107	T1	SMT	

CONCLUSION

"Everything should be made as simple as possible, but not simpler."

Albert Einstein (1879-1955)

Electronic Home Improvement™ is poised to become a major industry of the twenty-first century. Two-thirds of homes in the U.S. have computers and 87% of those are connected to the Internet[4]. WiFi wireless networking is in 20% of broadband-connected homes[5]. High-def TV is falling in price and gaining momentum dramatically. But light switches, door locks, thermostats, smoke detectors, and security sensors cannot talk to one another. Without an infrastructure networking technology, there can be no hope for greater comfort, safety, convenience, and value brought about through interactivity. Homes will remain unaware that people live in them.

For a technology to be adopted as infrastructure, it must be simple, affordable, and reliable. Not all technology that gets developed gets used. Sadly, a common pitfall for new technology is overdesign—engineers just can't resist putting in all the latest wizardry. But with added performance, cost goes up and ease-of-use goes down.

Simplicity is the principal asset of INSTEON. Installation is simple—INSTEON uses existing house wiring or the airwaves to carry messages. INSTEON needs no network controller—all devices are peers. Messages are not routed—they are simulcast. Device addresses are assigned at the factory—users don't have to deal with network enrollment. Device linking is easy—just press a button on each device and they're linked.

Simplicity ensures reliability and low-cost. INSTEON is not intended to transport lots of data at high speed—reliable command and control is what it excels at. INSTEON firmware, because it is simple, can run on the smallest microcontrollers using very little memory—and that means the lowest-possible cost.

Developing applications for INSTEON-networked devices is also simple. Designers do not have to worry about the details of sending and receiving INSTEON messages, because those functions are handled in firmware. Application developers can use a simple scripting interface and SmartLabs' Device Manager to further simplify the interface to a network of INSTEON devices. Designers who wish to create new kinds of INSTEON devices can write the software for them using their favorite development tools and an INSTEON Modem (IM), or they can use SmartLabs' Integrated Development Environment and the SALad embedded language.

Although INSTEON is simple, that simplicity is never a limiting factor, because INSTEON Bridge devices can connect to outside resources such as computers, the Internet, and other networks whenever needed. SALad-enabled INSTEON devices can be upgraded at any time by downloading new SALad programs. Networks of INSTEON devices can evolve as the marketplace does.

SmartLabs' mission is to make life more convenient, safe and fun. INSTEON provides the infrastructure that can make that dream come true. Anyone can now create products that interact with each other, and with us, in remarkable new ways. The future is now!

GLOSSARY

This glossary is in alphabetical order. **Bold** terms in a definition refer to other glossary entries.

ACK Message. See **INSTEON Acknowledgement Message**.

ALDB, ALDB/T, ALDB/L. See **ALL-Link Database (ALDB)**.

ALL-Linking™. A method for associating **INSTEON Controller** buttons with groups of (one or more) **INSTEON Responders** such that the Responders instantly revert to a memorized state when the button is pushed. Users can manually ALL-Link INSTEON devices by pressing and holding a Controller Button, and then pressing and holding a Responder's *SET Button*, or they can use software.

ALL-Link Broadcast. An **INSTEON Message** containing an **ALL-Link Command** sent by an **INSTEON Controller** to all members of an **ALL-Link Group** at once. ALL-Link Broadcasts allow all members of an ALL-Link Group to respond instantly to an ALL-Link Command. Controllers follow up an ALL-Link Broadcast by sending **ALL-Link Cleanup** messages individually to each member of the ALL-Link Group.

ALL-Link Cleanup. An **INSTEON Message** containing an **ALL-Link Command** sent by an **INSTEON Controller** individually to each member of an **ALL-Link Group**. A sequence of ALL-Link Cleanups will be aborted by new INSTEON traffic on the **INSTEON Network**. ALL-Link Cleanups follow an **ALL-Link Broadcast** sent to all members of the ALL-Link Group at once.

ALL-Link Command. An **INSTEON Command** that causes an **ALL-Link Group** of **INSTEON Responder** devices to revert to the state they were in at the time they were ALL-Linked to the **INSTEON Controller** issuing the ALL-Link Command. Controllers first send an **ALL-Link Broadcast Command** to all members of an ALL-Link Group, and then follow up by sending **ALL-Link Cleanup Commands** individually to each member of the ALL-Link Group.

ALL-Link Database (ALDB). In an **INSTEON Controller**, a set of records in nonvolatile memory, each of which associates an **ALL-Link Group** established by the Controller with an **INSTEON Responder**. In an INSTEON Responder, a set of records in nonvolatile memory, each of which associates a state of the Responder with an ALL-Link Group established by a Controller. A Threaded ALL-Link Database (ALDB/T) is much faster to search than a Linear ALL-Link Database (ALDB/L).

ALL-Link Group. An association between a button or function on an **INSTEON Controller** and one or more **INSTEON Responder** devices.

BiPHY™. An **INSTEON Device** that communicates using both the **powerline** and **radio**.

coreApp. A **SALad** program running in the **SmartLabs PowerLinc Controller (PLC)** that interfaces serially to a computing device and handles

INSTEON Messages, **X10** commands, **IBIOS** events, **ALL-Linking**, and other functions.

DevCat. See **INSTEON Device Category**.

Dual Mesh™ Network. A network whose nodes may communicate by **Simulcasting** over the **powerline**, via **radio**, or both (**BiPHY**).

ED. See **INSTEON Message Types**.

Hops. See **INSTEON Message Hopping**.

i1/RF. The original INSTEON **radio** frequency signaling protocol employed by the i1 **INSTEON Engine**. **i2/RF** replaces i1/RF, although the two protocols may coexist without mutual interference.

i2/RF. The INSTEON **radio** frequency signaling protocol employed by the i2 **INSTEON Engine**. i2/RF replaces **i1/RF**, although the two protocols may coexist without mutual interference.

IBIOS. The INSTEON Basic Input/Output System that implements the basic functionality of **INSTEON Devices** like the **SmartLabs PowerLinc Controller (PLC)**.

IID. See **INSTEON ID**.

IM. See **INSTEON Modem**.

INSTEON™. A **Dual Mesh** (**powerline** and **radio**) networking technology for home control and sensing that uses simulcasting to propagate messages simply, affordably, and reliably among **INSTEON Devices**.

INSTEON Acknowledgement Message. A Direct (**SD** or **SC**) INSTEON Message returned to the sender when a message recipient receives a Direct (**SD**, **ED**, or **SC**) INSTEON Message from the sender. Acknowledgement messages are always Standard-length, and they normally echo the received *Command 1* and *Command 2* bytes unless the received INSTEON Command is a specific request for data. Depending on the *Message Flags* bits, Acknowledgement Messages may return an ACK (**SDK** or **SCK**) or a NAK (**SDN** or **SCN**) to the sender.

INSTEON Address. See **INSTEON ID**.

INSTEON Command. A one- or two-byte code occupying the *Command 1* field or both the *Command 1* and *Command 2* fields of an INSTEON Message. The *INSTEON Command Tables* document defines all valid INSTEON Commands. The meaning of an INSTEON Command depends on the type of INSTEON Message that contains it, namely **SD** (Standard-length Direct), **ED** (Extended-length Direct), **SB** (Standard-length Broadcast), **SA** (Standard-length ALL-Link Broadcast), or **SC** (Standard-length ALL-Link Cleanup).

INSTEON Controller. An **INSTEON Device** that sends **INSTEON Commands** to **INSTEON Responders**.

INSTEON Device. A module attached to an **INSTEON Network** adhering to the *INSTEON Conformance Specification*. INSTEON Devices may act as **INSTEON Controllers**, **INSTEON Responders**, or both. INSTEON devices may contain their own user interfaces or control circuitry, or they may interact with other devices via dedicated communication channels such as USB or Ethernet.

INSTEON Device Category (DevCat). A one-byte hexadecimal number stored in an **INSTEON Device's** nonvolatile memory, broadly indicating what function the device performs. The interpretation of Direct (**SD** and **ED**) **INSTEON Commands** depends on a device's DevCat. It is the responsibility of **INSTEON Controllers** to determine the DevCat of an **INSTEON Responder** before sending it an **SD** or **ED** INSTEON Command.

INSTEON Device Subcategory (SubCat). A one-byte hexadecimal number stored in an **INSTEON Device's** nonvolatile memory, further differentiating the device within its **DevCat**. Legacy INSTEON Devices may be uniquely identified by their DevCat and SubCat numbers, but new devices should use the **INSTEON Product Key (IPK)** instead.

INSTEON Engine. Firmware in an **INSTEON Device** that handles the low-level **INSTEON Message** protocol, including **Hops** and **Retries.** The current i2 INSTEON Engine replaces the original i1 INSTEON Engine.

INSTEON ID (IID). Also known as INSTEON Address, a unique 3-byte number assigned to each **INSTEON Device** at the factory and stored in nonvolatile memory. A device's INSTEON ID serves as its permanent address on an **INSTEON Network**. Because INSTEON IDs are preassigned, users do not have to enroll INSTEON Devices in an INSTEON Network.

INSTEON Message. Formatted data sent by one **INSTEON Device** to other INSTEON Devices over an INSTEON network. Standard-length INSTEON Messages contain a 3-byte *From Address*, a 3-byte *To Address*, a *Message Flags* byte, a 2-byte *INSTEON Command*, and a *Message Integrity* (CRC) byte. In addition, Extended-length INSTEON Messages contain 14 bytes of *User Data* preceding the CRC byte.

INSTEON Message Hopping. A method for repeating **INSTEON Messages** by simulcasting. When an **INSTEON Device** hears an INSTEON Message, it inspects the 2-bit *Max Hops* and 2-bit *Hops Remaining* fields of the *Message Flags* byte. If *Hops Remaining* is not zero, the device will decrement the *Hops Remaining* field in the message and then retransmit the message at a precise time based on the **powerline** zero crossing interval. Because multiple devices may hear and retransmit the message simultaneously, the energy in the retransmitted message grows, much like the sound of many voices in a choir singing at once. See **Message Simulcasting**.

INSTEON Message Retrying. Additional attempts to send a Direct INSTEON Message if the message addressee fails to respond with an **INSTEON Acknowledgement Message**. The **INSTEON Engine** automatically retries messages up to five times, each time incrementing the *Max Hops* field up to the maximum of three.

INSTEON Message Timeslot. The time interval required to send an **INSTEON Message**, including all outgoing message **Hops**, and to receive an **INSTEON Acknowledgement Message** (if expected), including all acknowledgement message hops. **Powerline** message timeslots are synchronized to the powerline zero crossing. **RF** message timeslots have the same duration but are not necessarily synchronized to the powerline.

INSTEON Message Types. There are 16 logically possible INSTEON Message Types denoted by four bits in the *Message Flags* byte within an INSTEON Message. A three-letter mnemonic designates the INSTEON Message Type. The first letter is **S** for Standard-length or **E** for Extended-length messages. The second letter is **D** for Direct, **B** for Broadcast, **A** for ALL-Link Broadcast, or **C** for ALL-Link Cleanup messages. The third letter, which is optional, is **O** for Outgoing, **K** for ACK, or **N** for NAK messages. Valid Outgoing INSTEON Message Types are **SD**, **ED**, **SA**, **SC**, and **SB**. Valid INSTEON Acknowledgement Message Types are **SDK**, **SDN**, **SCK**, and **SCN**.

INSTEON Modem (IM). INSTEON chips or modules that provide a simple serial interface to **INSTEON Engine** functions, **ALL-Linking**, **ALL-Link Database (ALDB)** management, **ALL-Link Cleanup** messages, **X10** powerline interfacing, and **INSTEON Acknowledgement Messages**.

INSTEON Network. A collection of **INSTEON Devices** using the INSTEON networking protocol to communicate with each other via **powerline**, **radio**, or both.

INSTEON Product Database (IPDB). An online or local database containing records accessible using an **INSTEON Product Key (IPK)** stored in an **INSTEON Device**. Each record contains detailed XML-formatted information about the device's capabilities.

INSTEON Product Key (IPK). A three-byte number stored in an **INSTEON Device's** nonvolatile memory that serves as a unique lookup key to the online INSTEON Product Database (IPDB).

INSTEON Responder. An **INSTEON Device** that executes the **INSTEON Commands** received within **INSTEON Messages** sent by **INSTEON Controllers**.

IPDB. See **INSTEON Product Database**.

IPK. See **INSTEON Product Key**.

Message Routing. A common networking protocol that involves finding optimum paths for messages to travel from node to node over a network. With message routing, only one node in the network transmits a message at any given time. Routed networks must contain nodes capable of computing routing tables for messages, and must provide methods for nodes to join, leave, and move around the network. Compare with **Message Simulcasting**.

Message Simulcasting. A method for increasing the reliability of message delivery in a network. When a node in a network sends a message, every other node that hears the message retransmits it at precisely the same time based on a global clock, provided that the message has not already been retransmitted some maximum number of times. Message propagation is more robust because each node adds its energy to the signal, much like voices in a choir. Simulcasting is much simpler than **Message Routing**, because there are no routing tables to maintain and nodes can join the network without any installation procedure.

NAK Message. See **INSTEON Acknowledgement Message**.

Note Key. An abbreviation used within the INSTEON Command Tables designating special properties of INSTEON Commands.

PLC. See **SmartLabs PowerLinc Controller**.

PLM. See **Powerline Modem**.

Powerline. The electrical wiring that delivers power within a building.

Powerline Modem (PLM). An **INSTEON Modem** that only communicates over the **powerline**.

Radio. For the purposes of INSTEON wireless signaling, the unlicensed band from 902 to 924 MHz.

Retries See **INSTEON Message Retrying**.

RF. Radio frequency (see **Radio**).

RFM. See **RF Modem**.

RF Modem (RFM). An **INSTEON Modem** that only communicates using **radio**.

Routing. See **Message Routing**.

SA, SB, SC, SD. See **INSTEON Message Types**.

SALad. An event-driven embedded language interpreter built into some **INSTEON Devices**, such as the **SmartLabs PowerLinc Controller (PLC)**. SALad programs typically handle events generated by an **IBIOS** in firmware, but they can do much more. SALad programs are downloadable to SALad-enabled devices over the **INSTEON Network**.

SDM. See **SmartLabs Device Manager**.

Simulcasting. See **Message Simulcasting**.

SmartLabs Device Manager (SDM). A communication and translation gateway to the **SmartLabs PowerLinc Controller** (PLC). Developers use simple text commands through ActiveX or HTTP calls to interface with an **INSTEON Network**.

SmartLabs PowerLinc™ Controller (PLC). The SmartLabs PowerLinc™ V2 Controller is an INSTEON-to-Serial (USB or RS232) Bridge for connecting an INSTEON network to a computing device. Using the PLC, application developers can create high-level user interfaces to **INSTEON Devices** on an **INSTEON Network**. The PLC runs a **SALad** program called **coreApp** that handles **IBIOS** events.

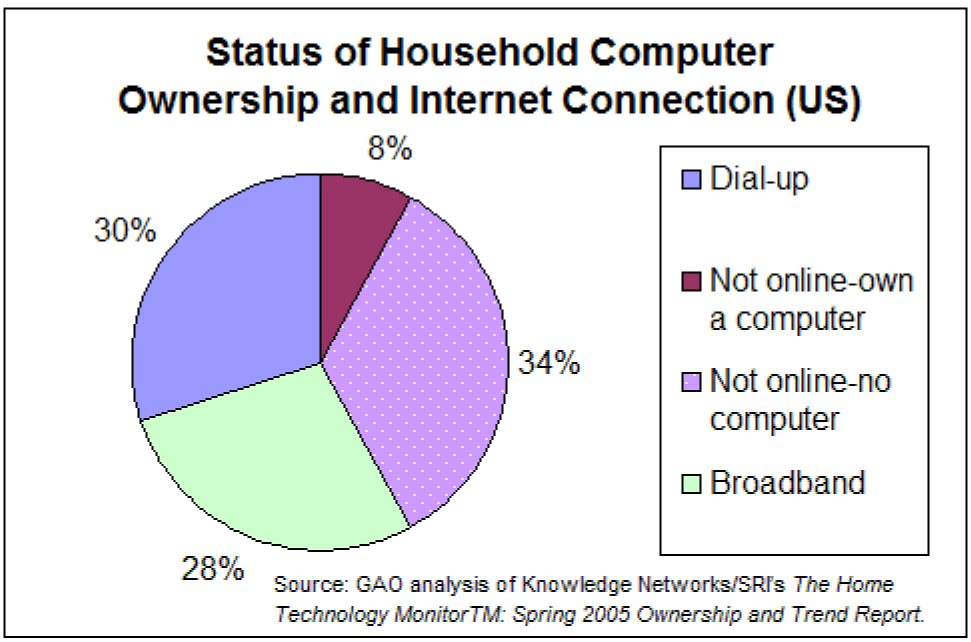
SubCat. See **INSTEON Device Subcategory**.

Timeslot. See **INSTEON Message Timeslot**.

X10. A legacy **powerline** signaling method used by many devices already deployed around the world. **INSTEON** and X10 are compatible on the powerline, and many **INSTEON Devices** can send and receive X10 signals. The **SmartLabs PowerLinc™ Controller (PLC)** and **INSTEON Modems** support X10.

NOTES

1. Battery operated INSTEON RF devices, such as security sensors and handheld remote controls, must conserve power. Accordingly, they may optionally be configured so that they do not retransmit INSTEON messages from other INSTEON devices, but act as message originators only. Such devices can nevertheless both transmit and receive INSTEON messages, in order to allow simple setup procedures and to ensure network reliability.
2. At a minimum, X10 compatibility means that INSTEON and X10 signals can coexist with each other on the powerline without mutual interference. INSTEON-only powerline devices do not retransmit or amplify X10 signals. But X10 compatibility also means that designers are free to create hybrid INSTEON/X10 devices that operate equally well in both environments. By purchasing such hybrid devices, current users of legacy X10 products can easily upgrade to INSTEON without making their X10 investment obsolete.
3. Firmware in the INSTEON Engine handles the CRC byte automatically, appending it to messages that it sends, and comparing it within messages that it receives. Applications post messages to and receive messages from the INSTEON Engine without the CRC byte being appended. See *Message Integrity Byte*44 for more information.
4. See GAO Telecommunications Report: May 2006 (GAO-06-426.



5. See http://www.wirelessweek.com/article/CA6334955.html?spacedesc=Departments.

