**Simple Device Discovery Protocol**

**Implementation Guide**

**Copyright**

|  |
| --- |
| The information contained in this document is the intellectual property of Control4 Corporation (“Control4”) and use without a valid license from Control4 is strictly prohibited. The holder of this document shall keep all information contained herein confidential and shall protect this information in whole or in part from disclosure to any and all third parties except as specifically authorized in writing by Control4. |

**Disclaimer**

The information in this document is provided in connection with Control4 products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by or in this document. Except as provided in Control4’s terms and conditions for the license of such products, Control4 Corporation and its affiliates, (“Control4”) assume no liability whatsoever, and Control4 disclaims any express or implied warranty, relating to the sale and/or use of Control4 products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Control4 products are not intended for use in medical, life saving, or life sustaining applications.

Information regarding third-party products is provided solely for educational purposes.  Control4 is not responsible for the performance or support of third-party products and does not make any representations or warranties whatsoever regarding the quality, reliability, functionality or compatibility of these products.  The reader is advised that third parties can have intellectual property rights that can be relevant to this document and the technologies discussed herein, and is advised to seek the advice of competent legal counsel, without obligation of Control4.

Control4 retains the right to make changes to this document or related product specifications and descriptions, at any time, without notice. Control4 makes no warranty for the use of this document and assumes no responsibility for any errors that can appear in the document nor does it make a commitment to update the information contained herein.

**Trademarks**

Control4 and the Control4 logo are trademarks or registered trademarks of Control4 Corporation. Other product and company names mentioned in this document may be the trademarks or registered trademarks of their respective owners.

**Copyright**

Copyright © 2004-2016 Control4. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

Contact Us

Control4 Corporation

11734 S. Election Road

Salt Lake City, UT 84020 USA

<http://www.control4.com>

Table of Contents

Introduction to SDDP 4

Getting Started 4

SDDP Implementation Summary and Scenarios 6

Access and Extract the SDK Contents 8

Modifying the Example Configuration File 9

Modifying the SddpServer.c File 11

Using the SDDP makefile to compile sddpd 12

Testing SDDP Implementation 12

Appendix 16

## Introduction to SDDP

SDDP is a proprietary Device Discovery Protocol developed by Control4. It defines a discovery protocol similar to SSDP, which is the discovery protocol used by Universal Plug and Play (UPnP).

Significant benefits of an SDDP implementation include:

* Auto Device Discovery - SDDP enables Control4's Director to easily discover and identify devices configured with the SDDP client that reside on the network.
* Auto Driver Download - Upon device discovery, SDDP provides for automatic downloads and installations of the correct device driver in the Control4 project.
* Unique Device Identification - SDDP allows for an IP device to use a UUID string rather than a static IP address when identified in a Control4 system. This means that it will work well in a DHCP environment and be much less problematic for Control4 dealers/installers to work with and support.

Note that SDDP implementation is compatible with IP devices only. Its architecture (and this document) assumes that there is just one “controller” device that is running as the SDDP server in a system and that the system contains one or more “controlled” devices running as SDDP client(s).

For the purposes of this document, the assumption is made that a Control4 master controller running Director fulfills the role as the SDDP server. Any references to "Director" in this document refer to this SDDP server.

## Getting Started

SDDP implementation requires a functional, updated Control4 system as well as access to the Control4 Services that support system functionality. As a developer implementing SDDP, it is very important for you to have access to the services outlined below. Registration enables you to leverage all of the hardware/software functionality and support provided through the SDDP SDK.

1. Dealer and End-user Account: [www.control4.com/o/dealer](http://www.control4.com/o/dealer)

While an End-User account will be useful for testing your SDDP implementation**, it is crucial that a Dealer account be the first account created**. Using the link above, you will need to apply for dealership credentials. These credentials are important and will be used off and on throughout the driver developer process for your product. Please make sure they are stored in a secure location.

2. ComposerPro Credentials: <https://dealer.control4.com>

Once your Dealer Account has been created you can access the latest Operating System and ComposerPro releases using the link above. Once logged in, go to Support then Software Downloads. **It is critical that all of your development is performed on the latest release!**

3. Access to Control4’s External Server: <https://dealer.control4.com>

Register your Control4 Project using the link above.. Follow the "Controller Registration" link. After registering your Control4 system, update your controller to the latest version of firmware through ComposerPro/Tools/Update Manager.

4. Accessing the SDDP SDK: <https://files.control4.com>

Once your Control4 system is registered and updated, the current SDDP SDK is stored at the link above.

If you have any issues in setting up your Control4 System for SDDP Implementation or questions about accessing the SDDP SDK, please email: [SDDP-Support@Control4.com](mailto:SDDP-Support@Control4.com).

## SDDP Implementation Summary and Scenarios

Before you begin your SDDP implementation, you may find it useful to understand the intended use of the SDK as well as the target implementation scenarios it was designed to support.

Through the SDDP SDK, Control4 provides you with the source code to create an "SDDP Daemon." This daemon is referred to as "sddpd" in this document. When the source code is modified correctly the daemon will integrate SDDP into your product. The source code can be adapted as necessary to function in most product environments. Each of the products that will communicate to Control4's Director using SDDP will have a unique configuration file (.conf) (which is delivered in the SDK and you will edit) that configures the SDDP Daemon so it can send and receive SDDP messages. There are four types of messages that send information about the device and its status: Notify Alive, Notify Offline, Notify Identify, and Search Response.

The messages or announcements are made on a multicast address when the device becomes available and immediately before it goes away. SDDP uses UDP broadcasts on the multicast address 239.255.255.250 port 1902. It also supports unicasts directly from the client (your device) to the server (Director).

An overview of the implementation process follows a few basic steps that are explained in the following sections of this document. To implement SDDP, you will:

1. Access and extract the software provided in the SDK (sddp-1.0.tar.gz).
2. Modify the example.conf files and any other files based on your implementation needs
3. Using the supplied makefile, compile the files to successfully create sddpd and integrate sddpd into your device software build.
4. Implement an SDDP announcement with some physical button press or UI
5. Test the SDDP Implementation

In most cases, an SDDP Implementation falls into three scenarios, which are described below. It is important to note that users of the SDK are expected to have a level of knowledge of not only the product they are implementing SDDP into, but the C language set as well. If your implementation scenario is not addressed below, it may fall outside the scope of this documentation. Please contact Control4 SDDP support at: [SDDP-Support@Control4.com](mailto:SDDP-Support@Control4.com) for further instructions.

Scenario 1: SDK Developer who is knowledgeable in C and running on a Linux environment. Native product knowledge is sound. The system architecture involved is similar to that of Control4. This implementation can be as simple as modification of the .conf file. The environment is close enough to Control4's that the existing make file can be used "as is.".

Scenario 2: SDK Developer who is knowledgeable in C and running on a Linux or Linux-like environment. Native product knowledge is sound. This implementation scenario might involve implementing SDDP on a large-scale system. This would involve modification of the .conf file and likely modification to the supplied makefile as well.

Scenario 3: SDK Developer who is knowledgeable in C. Native product knowledge is sound. Environment is a non-Linux-like environment such as Windows or a one off flavor of Linux. The implementation might also involverunning SDDP from a module within their own product's own executable file. This implementation will include modifying the .conf file as well as the make file. It is also likely that modification of the Sddpserver.c file will be needed to generate a unique hostname for the device.

## Access and Extract the SDK Contents

*Use of the SDDP SDK involves contractual obligations with Control4 as well as access to Control4's external servers. The pre-Implementation requirements are addressed in the "How to Get Started" section of this document. This section assumes successful completion of the items detailed there.*

The following files comprise the SDDP SDK. They are located in the SDDP SDK.tar.gz file:

**1. hc800.conf & example-receiver.conf** *Modification Required - Yes*

.conf files serve as the link between your device and the SDDP services. The purpose of the. Conf files included in the SDK are to not only serve as an example, but they can also be used as template to create your own device configuration file. Creating a unique .conf file is a requirement for SDDP Implementation. The two samples provided are real, working copies of successfully created .conf files. Detailed instruction on how to create a unique .conf file is provided later in this document.

**2. SddpServer.c** *Modification Required - Possible*

Contains the SddpSetHost function. This is used to set your device's unique hostname. You will need to modify the SddpServer.c file to call SddpSetHost with a globally unique string. This string could be a UUID, MAC address or some other globally unique identifier. As a suggestion, Control4 products concatenate the name using the following format: device type-MAC address. For example: home-controller-800-000fff123456

**3. makefile** *Modification Required - Probable*

Control4 recommends using the makefile supplied in the SDK to compile and test the your version of the sddpd. After successful compilation, you will be ready to integrate the sddpd and the modified .conf file into your build system.

**4. SddpNetwork.c & SddpPacket.h** *Modification Required - Possible*

This file provides a module that serves as a network shim to the SDDP module. It isolates the system's network functionality to make it easier to port to various platforms. This file may need to be modified if no Operating System is used.

***The following files are delivered in a state that should require no modification on the part of the SDDP developer:***

**Sddp.h & Sddp.c**

These files contain the headers and functions that comprise a protocol used during SDDP processes such as starting/stopping services, managing identification packets and joining/leaving networks.

**Sddp.h**

This file contains functions that manage SDDP-specific device information, start and stop functions as well as functions that communicate the state of a device on the network.

**SddpStatus.h & SddpPacket.c**

These files contain the headers and functions, which comprise an SDDP Packet parser. They include Helper functions that provide status codes used by SDDP.c module.

## Modifying the Example Configuration File

The SDDP SDK includes two sample configuration files called Example-Receiver.conf and HC800.conf. As stated earlier, these files serve as the link between your device and the SDDP services. The purpose of the. Conf files included in the SDK are to not only serve as an example, but they can also be used as template to create your own device configuration file. Creating a unique .conf file for your devices is a requirement for SDDP Implementation. The two samples provided real, working copies of successfully created .conf files.

This section will detail each line of the example configuration file and define the values they require using the Example-receiver.conf file. The first step to implementing SDDP is to replace the example content provided in the sample conf file with your device's information. Note that you will need to specify the complete path to location of the .config file when compiling your code. Otherwise, the SDK will display an error that resembles: "Incomplete .config file 'abc'".

The Example-Receiver.conf file contains the following lines, which are explained below the example:

1. Type = acme:receiver:ACME-100
2. PrimaryProxy = receiver
3. Proxies = receiver,tuner,tunerXM,dvd
4. Manufacturer = ACME
5. Model = ACME-100
6. Driver = acme\_receiver\_ac100.c4i
7. Config-URL = http://<host>/netconf/
8. MaxAge = 1800

1. Type = acme:receiver:ACME-100

The Type string uniquely identifies your device’s type or model. This must identically match the <search type> value found in the device’s .c4i/.c4z file.

2. PrimaryProxy = receiver

This is your device’s primary proxy type. This must match the <Online Category> value found in your device’s .c4i/.c4z file. Note the Primary proxy value should be entered in lower case.

3. Proxies = receiver,tuner,tunerXM,dvd

The Proxies string lists all proxies implemented by your device’s driver. If the driver supports only one proxy, this value will be the same as the PrimaryProxy value. This must match the <proxies> value found in your device’s .c4i file.

4. Manufacturer = ACME

This is the manufacturer of the device. This must match the <manufacturer> value found in your device’s .c4i file.

5. Model = ACME-100

This is the model of the device. This must match the <model> value found in the device’s .c4i file.

6. Driver = acme\_receiver\_ac100.c4i

This is the actual file name of your device’s .c4i file as it is stored in the Control4 online driver database.

7. Config-URL = http://<host>/netconf/

Optional. This is the URL to the device’s http accessible management/configuration UI. If the device does not have an http accessible management UI, this can be omitted. For more information regarding Config-URL use see the section below: *Launching the Config-URL from within ComposerPro.*

8. MaxAge = 1800

This value is the announcement interval in seconds. In the example above, the time interval is 30 minutes. This value represents the amount of time between sending NOTIFY messages. Control4 recommends that you do not modify this value.

## Launching the Config-URL from within ComposerPro

The URL value defined for Config-URL for your SDDP device can be launched by dealers from within the ComposerPro environment. This feature is available with operating system 2.9.0 and later and is available in two places:

1. While in System Design, click on the Discovered tab. While holding the keyboard Control (CTRL) button down click on the SDDP device from the list. This will launch the URL defined in the Config\_URL value of the Configuration File for the device.

2. The URL can also be launched from within the Connections area of ComposerPro. Under the Available Devices area, holding the keyboard Control (CTRL) button down and clicking on the SDDP device from the list will also launch the URL defined in the Config\_URL value of the Configuration File for the device.

If no Config\_URL value is defined, the browser will open the IP address of the SDDP device by default.

## Modifying the SddpServer.c File

Modification of the SddpServer.c file is only required if your device **does not** already use a Unique Host Name that uniquely and absolutely names that device. This file may also need to be modified if no Operating System is used.

The SddpServer.c provides a module that serves as a network shim to the SDDP module. It isolates the system's network functionality to make it easier to port to various platforms. It also uses the SDDPSetHost function to set the hostname.

Generally speaking, devices that have UPnP already implemented or, for that matter almost any other device discovery protocol will already have a unique host name. A simple way to test this is to call hostname from the command line on any Linux device. This will report a hostname. You must ensure that the hostname reported is a unique hostname for your device. If your device does not use a unique host name, the section below described the required steps to modify the SddpServer.c file to

In the SddpServer.c file (delivered in the SDK) the SddpSetHost function is called. You will use this function to set your device’s unique host name.

You will need to modify the SddpServer.c file to call SddpSetHost with a globally unique string. This string could be a UUID, MAC address or some other globally unique identifier. As a suggestion, Control4 products concatenate the name using the following format: device type-MAC address. For example: home-controller-800-000fff123456.

If your device does notalready use a Unique Host Name, modification of the file that makes sense for your device will be necessary. Below is the section of code from the SddpServer.c file that calls SDDPSetHost function:

// Initialize the SDDP host name

if(gethostname(host, MAX\_HOST\_SIZE) == 0)

SDDPSetHost(handle, host);

else

SDDPSetHost(handle, "unknown");

## Using the SDDP makefile to compile sddpd

For purposes of review, at this point you should have a modified configuration file for your device. It is possible that you have also modified the SddpServer.c file so your device’s unique host name is called by the SddpSetHost function.

Control4 now recommends using the makefile supplied in the SDK to compile the files delivered in the SDK to build the SDDP Daemon or sddpd. After compiling it is recommended that you test the compilation. After successful compilation, you are now ready to integrate sddpd and the modified .conf file into your build system.

## Testing SDDP Implementation

SDDP implementations vary greatly in both is scope and unique device requirements. For this reason, it is difficult to provide a detailed testing approach to validate your specific implementation. However, this section does offer some universal guidelines that can be used as a starting point to verify that SDDP is working correctly in your environment as well as on your devices.

Testing the SDDP implementation in a Control4 system consists of the following steps:

1. Set up a Control4 system and SDDP device on the same network.
2. Access the SDDP Trace window in System Manager.
3. Validate that the four types of SDDP messages are sent after the correct intervals and device events
4. Validate that SDDP messages contain the correct headers.
5. Validate that the functionality remains consistent under various failure scenarios (stress testing).

**1. Set up a Control4 system and SDDP device on the same network.**

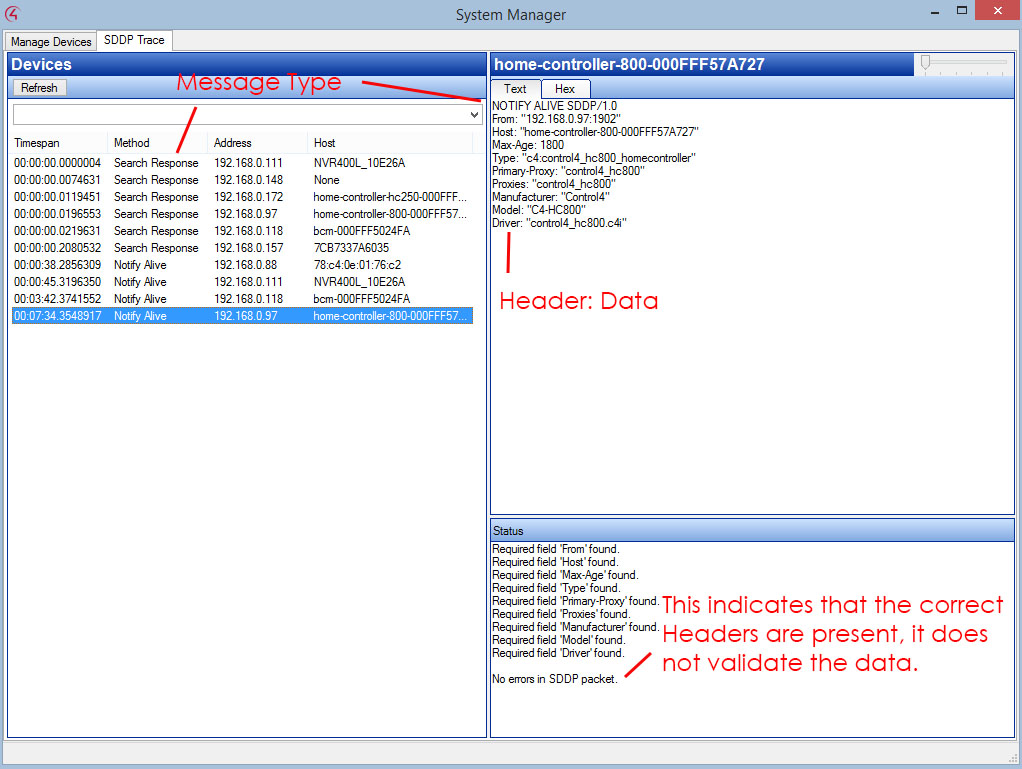
In order to validate your SDDP implementation you will need to have a Control4 system successfully configured. The system requires that a project be created or loaded in ComposerPro. The project will need to contain the driver file for the device you wish to implement with SDDP.

**2. Access the SDDP Trace window in System Manager**

ComposerPro has a built in tool for capturing SDDP messages on the network. It will capture the SDDP messages and display the headers and data contained in them. The tool can be accessed by navigating in ComposerPro to Tools > System Manager > SDDP Trace.

**3. Validate that the four types of SDDP messages are sent after the correct intervals and device events**

With the SDDP Trace window open, we will validate that the SDDP messages are sent after the correct events and intervals, and that they contain the appropriate data. The SDDP trace window will validate that the correct headers are present, but it will not validate the data in the headers:



There are four types of SDDP messages. The following is a list of the message types, and when they should be sent:

**Notify Alive**

This message should be sent periodically. It should be sent before the “Max Age” field expires. By default, the Max Age timer is 1800 seconds. This should also be sent any time the device is or becomes available for control, such as right after turning the device on, or right after setting up the network connection. If a device stays in network standby, it should continue sending this message periodically; if it is not sent before the Max Age timer expires, the device may not be controllable.

**Notify Offline**

This message should *only* be sent when the device is *no longer available to control.* Some devices do not need to use this message at all. An example of a device which would commonly use this is one which supports Wake-On-Lan, when the device is turned off it should send the Notify Offline message, and when it receives the Wake-On-Lan command it should send a Notify Alive message so that it can be controlled again.

**Notify Identify**

This is an option message we recommend implementing. This is not required to identify a device using SDDP, but if it is implemented it provides additional options which are useful for an installer, primarily when they are installing multiples of the same device. This message should be linked to a button press or a shortcut in the device’s GUI. When this message is sent, if the option “Auto add identified devices” is checked in ComposerPro > System Design > Discovered, the driver will automatically be added to the project and identified.   
Alternatively, if the driver is already in the project, this will allow the device to be identified in ComposerPro > Connections > Network > (Driver’s identify window).

**Search Response**

When Director sends a search request, all SDDP devices should respond with a Search Response message. This can be tested by pressing the “refresh” button in the SDDP Trace window.

Note: Some devices may not respond if they have received many search requests in a short time period. This is acceptable, but they will still need to respond to the search requests under general circumstances.  **4. Validate that SDDP messages contain the correct headers.**  
The notify messages each contain a number of headers. The table below lists each of the headers and the respective messages in which they are found:  
**Header Messages**

From: Search Response, N. Identify N. Alive N. Offline

Host: **X** Response, N. Identify N. Alive N. Offline

Tran: Search **X**  **X**  **X**  **X**

Max-Age: **X** Response **X** N. Alive, **X**

Type: **X** Response, N. Identify N. Alive N. Offline

Primary-Proxy: **X**  Response, N. Identify N. Alive, **X**

Proxies: **X** Response, N. Identify N. Alive, **X**

Manufacturer: **X** Response, N. Identify N. Alive, **X**

Model: **X**  Response, N. Identify N. Alive, **X**

Driver: **X**  Response, N. Identify N. Alive, **X**

Config-URL: **X** Response, N. Identify N. Alive **X**

Timeout: Search **X**  **X**  **X**  **X**

Note that while validation of the correct Headers within the Messages is a recommended for m of SDDP validation, the SDDP Trace screen **does not** validate the actual data within the messages. Manual data validation of the data is suggested in conjunction with this step.

**5. Validate that the functionality remains consistent under various failure scenarios (stress testing).**  
Some devices perform according to the specs above, but fail after going through certain events. The most common stress test events which should be tested are:

**Power Outage:** Power cycle all of the equipment, including the SDDP device, the Control4 controller, and the network equipment. Ensure that SDDP behaviors described above remain consistent.

**Device Power Failure:** Power cycle just the SDDP device. Ensure that SDDP continues to function after the device finishes booting.

**Power Cycle Network:** Power cycle just the network. Ensure that SDDP continues to function when the network comes back online.

**Remove Network Connection:** Remove the Ethernet cable from the device, or if using wireless disable the wireless connection, and then re-connect. Ensure that SDDP continues to function after the network connection has be re-established.

After successfully completing all of the validation steps in this section

## Appendix

**SDDP Messages**

SDDP messages include searches (server side), responses and notifies (client side). Search messages require responses, while responses and notifies will not receive confirmation from the server.

When the SDDP server sends out a search request, it can be global or device type specific. This method is used to discover SDDP devices. If \* is used as the argument, all devices must respond to the request. Limiting the argument to a device type or a value within the namespace of type will only return responses from those devices.

Clients use response messages to confirm to the server that they are truly online and ready for communication from director. Responses are only sent to the server, not over multicast. Responses include status codes, see the appendix for the full list.

Clients use notifies as a type of polling to notify ~~D~~irector of their current state. There are three different types of notifies: **Notify Identify, Notify Alive** and **Notify Offline**. Notifies are broadcast on the multicast address to ensure delivery.

I think it’s worth providing examples/use-cases of the events that are expected (and not expected) to trigger these notifications.

Notify Identify is sent when the device identification process is called.

Notify Alive is sent before the Max-Age expires, usually around 1200 seconds or 20 minutes. It is also sent after a network reconnect or wake-on-lan announcement.

Notify Offline is sent as part of the process of an IP address change, followed by a Notify Alive. Or with wake-on-lan devices just before they turn off. Notify Offline can also be queued when a network connection is lost, to be sent after the reconnect, the Notify Alive is then sent immediately following.

**SDDP Headers**

There are multiple headers within SDDP, not all headers are used in all messages.

Header Messages

From: Search Response, N. Identify N. Alive N. Offline

Host: **X** Response, N. Identify N. Alive N. Offline

Tran: Search **X**  **X**  **X**  **X**

Max-Age: **X** Response **X** N. Alive, **X**

Type: **X** Response, N. Identify N. Alive N. Offline

Primary-Proxy: **X**  Response, N. Identify N. Alive, **X**

Proxies: **X** Response, N. Identify N. Alive, **X**

Manufacturer: **X** Response, N. Identify N. Alive, **X**

Model: **X**  Response, N. Identify N. Alive, **X**

Driver: **X**  Response, N. Identify N. Alive, **X**

Config-URL: **X** Response, N. Identify N. Alive **X**

Timeout: Search **X**  **X**  **X**  **X**

**From:**

This value represents the address of the device that is the “sender” of the message. This value is dynamically created if using DHCP as the network protocol, or static if an IP address is configured by a user. This must be in the format of: IP:Port. i.e. “192.168.0.123:1902” This value must be quoted.

Note: The syntax used contains no spaces.

**Host:**

If you have implemented UPnP, this is the same as the Host Value used by the SSDP protocol. If you don’t have a Host Value, Control4 recommends that you use the MAC Address of the device in this value. i.e. “HomeController-000fff001234” This value must be quoted.

Note: This header is only used in messages from clients.

**Tran:**

This is a transaction number. This is a random number that is only used in search messages. The same value will be included in the response message. This allows responses to be matched against searches.

Note: The Tran header is required, but it can be sent with no value.

**Max-Age:**

This value is the announcement interval in seconds. The time interval is 30 minutes (1800 Seconds). This value represents the amount of time between sending Notify messages. This value shouldn’t be modified. It is important to send Alive messages more often than the Max-Age value. This will avoid the Max-Age time frame from expiring before an Alive message is sent.

**Type:**

This is the value of the search type in your .c4i file <search\_type> tag. Device types are organized in name spaces separated by a ( : ). i.e. “c4:HC1000HomeController” This value must be quoted.

It is important to validate that your SDDP enabled driver contains a “Type” section. If the driver currently does not have a defined <search\_type> element, it will need to be added in order for SDDP to work correctly. The value declared in the SDDP data packet for “Type” must match identically the value provided in the <search\_type> section of your driver.

When adding a new search type entry into a driver, the naming convention used can be free form. However, Control4 recommends using a format that reflects the following: manufacturer:device:simplemodelnumber. i.e. “c4:HC1000HomeController” or “ACME:Receiver:ACME-100” This value must be quoted.

Note: This line looks like this in your .c4i file: <search\_type>ACME:Receiver:ACME-100</search\_type>

**Primary-Proxy:**

This header indicates the primary proxy type defined in the driver. This line looks like this in your .c4i file: <proxy>tv</proxy>

Proxies:

This header contains a comma-separated list of proxies that are included in the driver. This list should only contain proxies as defined in the .c4i or .c4z. However, this list can be a subset of what is in the driver. i.e. Proxies: “receiver,tuner,dvd”

Note: When listing multiple proxies, the primary proxy must be listed first.

See the appendix for the full list of proxies.

**Manufacturer:**

This header contains the name of the manufacturer of the device as listed in the .c4i. This value must be in quotes.

**Model:**

This header contains the model of the device as listed in the .c4i. And yep, it must be in quotes.

**Driver:**

This header contains the file name of the .c4i driver. This file name must match the driver file name that is used in the Control4 online driver database. This value must be in quotes. The structure is “device\_Manufacturer\_ModelName.c4i” or “tv\_Control4\_c4tv.c4z”.

Note: No special characters are supported in the Driver Name. i.e. &, $, # etc.

**Config-URL:**

If this device has an http configuration page, we recommend providing the address here. This will allow the configuration page to be accessed from System Manager. If there is no http accessible management UI, this line can be omitted.

**Timeout:**

This last header is completely optional. It delays the response from the client to a random limit less than the set value in seconds. This is helpful when there are multiple devices of the same type in one project. This will space out the responses, allowing the server time to register all of the responses. It is important that delaying responses should not block the client from responding to other requests.

**SDDP Packet Examples:**

**Search Response**

SDDP/1.0 200 OK

From: “192.168.1.54:1902”

Host: “home-controller-hc250-000fff160559”

Tran:

Max-Age: 1800

Type: “c4:control4\_hc250\_homecontroller”

Primary-Proxy: “control4\_hc250”

Proxies: “control4\_hc250”

Manufacturer: “Control4”

Model: “C4-HC250”

Driver: “control4\_hc250.c4i”

**Notify Identify**

NOTIFY IDENTIFY SDDP/1.0

From: “192.168.1.54:1902”

Host: “home-controller-hc250-000fff160559”

Type: “c4:control4\_hc250\_homecontroller”

Primary-Proxy: “control4\_hc250”

Proxies: “control4\_hc250”

Manufacturer: “Control4”

Model: “C4-HC250”

Driver: “control4\_hc250.c4i”

**Notify Alive**

NOTIFY ALIVE SDDP/1.0

From: “192.168.1.54:1902”

Host: “home-controller-hc250-000fff160559”

Max-Age: 1800

Type: “c4:control4\_hc250\_homecontroller”

Primary-Proxy: “control4\_hc250”

Proxies: “control4\_hc250”

Manufacturer: “Control4”

Model: “C4-HC250”

Driver: “control4\_hc250.c4i”

Notify Offline

NOTIFY OFFLINE SDDP/1.0

From: “192.168.1.54:1902”

Host: “home-controller-hc250-000fff160559”

Type: “c4:control4\_hc250\_homecontroller”

**Proxies**

|  |  |  |  |
| --- | --- | --- | --- |
| amplifier | dvd | pool | tuner |
| avswitch | dvr | projector | tunerXM |
| blind | ipod | receiver | Tv |
| blindv2 | light | satellite | Vcr |
| cable | media\_player | security |  |
| cd | others | thermostat |  |
| discchanger | plasma | thermostatv2 |  |

**Response Status Codes**

|  |  |
| --- | --- |
| 200 | OK |
| 400 | Bad Request |
| 401 | Unauthorized |
| 500 | Internal Server Error |
| 501 | SDDP Version Not Supported |
| 502 | Not Implemented |

2xx are success codes, 4xx are client error codes and 5xx are server error codes.