Internet Engineering Task Force
INTERNET DRAFT

Yaron Y. Goland
Ting Cai
Paul Leach
Ye Gu
Microsoft Corporation
Shivaun Albright
Hewlett-Packard Company

June 21, 1999
Expires December 1999

# Simple Service Discovery Protocol/1.0
<draft-cai-ssdp-v1-02.txt>

## Status of this Memo

## Abstract

The Simple Service Discovery Protocol (SSDP) provides a mechanism where by network clients, with little or no static configuration, can discover network services. SSDP accomplishes this by providing for multicast discovery support as well as server based notification and discovery routing.

## 1.   Introduction

With the growing number of small peer-to-peer TCP/IP networks, such as home or small office networks, users need a way to discover resources in a network easily, quickly, dynamically, and without any a priori knowledge.  This document provides a discovery protocol that meets this user requirement.

The proposed protocol is called the Simple Service Discovery Protocol (SSDP).

SSDP provides support for services declaring their presence.

SSDP provides support for those seeking services to search for the desired services.

SSDP provides support for subscription arbiters as defined in [GENA] with limited SEARCH method support so allow clients to discover services and receive service notifications.

## 2.   Terminology

Client - A HTTP resource that makes use of a service.

Service - A HTTP resource that provides a service used by clients.

SSDP Proxy - A SSDP service that routes notifications from services to clients, maintains a store reflecting the current state of services in the system and allows clients to search that store in order to ascertain information about the current state of the system.

Service Type - A URI that identifies the nature of a service. Service types can be very broad, for example, all SSDP compliant services to very narrow, a particular batch of a particular model of a particular product that comes in green. Service type URIs are treated as opaque identifiers by the SSDP protocol.

Unique Service Name (USN) - An identifier that is unique across all services for all time. It is used to uniquely identify a particular service in order to allow services of identical service type to be differentiated.

In addition, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and

"OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 3. SSDP Resources

### 3.1. Services

#### 3.1.1. Overview

A non-SSDP proxy aware service is a very simple resource. When it wakes up it sends a httpmu ssdp:alive NOTIFY message to the SSDP reserved multicast channel and then it waits.

If it should receive a httpmu ssdp:discover SEARCH request over the SSDP reserved multicast channel then it will check the ST header. If

the value in the ST header matches its service type then it will send a httpmu response to the requestor.

If it should receive a httpmu ssdp:notproxyaware SEARCH request over the SSDP reserved multicast channel then it will send a httpmu response. The response is the exact same response it would have sent for a matching ssdp:discover SEARCH request.

If the service sends out caching information, which it should, then it will also send out a ssdp:alive NOTIFY request to the SSDP reserved multicast channel before the expiration given in its previous SEARCH responses and NOTIFY requests.

Finally, if the service is able, it will send out a ssdp:byebye NOTIFY request to the SSDP reserved multicast channel before it goes off-line.

#### 3.1.2. Requirements

SSDP compliant services MUST have a USN and a service type URI.

SSDP compliant services SHOULD monitor the SSDP reserved multicast channel for SEARCH requests.

SSDP compliant services that monitor the SSDP reserved multicast channel MUST support ssdp:discovery and ssdp:notproxyaware SEARCH request extensions sent over the SSDP reserved multicast channel.

SSDP compliant services SHOULD send out a ssdp:alive NOTIFY method when they first join a new network.

SSDP services MUST NOT send out more than MAXNOTE NOTIFY requests a minute to the SSDP reserved multicast channel.

[Ed. Note: The number 5 is totally arbitrary. But we need a very hard limitation to keep the channel from getting overloaded. Anyone who needs more than 5 notify requests a minute needs to be a proxy aware service.]

## 3.2. Proxy Aware Services

### 3.2.1. Overview

A proxy aware service meets all the requirements for a non-proxy aware service plus requires support for http in addition to httpmu, is required to perform proxy detection, and support ssdp:proxyawareservices SEARCHs.

See section .4.1 for information on proxy discovery.

If proxy detection fails then the SSDP aware services will act like a non-proxy aware service until it hears a ssdp:alive NOTIFY request

from a proxy over the SSDP reserved multicast channel. In which case it will act as specified below.

If there is a SSDP proxy in the network then the service will send it a ssdp:alive NOTIFY request to provide the SSDP proxy with information about itself.

Any further NOTIFY methods the service has to send out will be sent directly to the SSDP proxy using http.

The service will continue to monitor the SSDP reserved multicast channel in order to hear the proxy's regular announcements that it is still functional. The server may optionally answer any SEARCH requests it receives over the SSDP reserved multicast channel.

If the proxy should fail to send out its regularly ssdp:alive NOTIFY request then the service will automatically fall back to using the

SSDP reserved multicast channel.

### 3.2.2. Requirements

Services which intend to send out a significant number of notifications MUST support SSDP proxies as specified in section .0 as well as all rules for non-proxy aware SSDP services.

A proxy aware service MUST follow all the rules for a non-proxy aware service with the exception that they MUST support ssdp:proxyawareservice SEARCH requests over the SSDP reserved multicast channel rather than ssdp:notproxyaware SEARCH requests.

If a SSDP proxy is discovered then the service MUST send all NOTIFY requests to the SSDP proxy using http. When a proxy is available the service MAY continue to answer ssdp:discover requests sent over the SSDP reserved multicast channel and MUST continue to answer ssdp:proxyawareservice SEARCH requests sent over the SSDP reserved multicast channel.

## 3.3. Clients

### 3.3.1. Overview

Upon waking up a client's first task is likely to be initializing its cache in order to track services it is interested in.

The first step in initializing the cache is to find out if there is a proxy. Therefore the client will perform proxy discovery as specified in section .4.1.

The next step is to send out ssdp:discovery SEARCH requests for each of the service types the client is interested in. If there is a proxy then the client will use http to send the requests to the

proxy. If there is no proxy then the client will use httpmu to send the requests to the SSDP reserved multicast channel.

Each response that has caching information, such as an expires header or a cache-control: max-age will be put into the client's cache.

The client will expect that services that sent out caching information will send out ssdp:alive NOTIFY requests before the cached information is set to expire. ssdp:alive NOTIFY requests contain the same information as ssdp:discover SEARCH responses. Therefore the client can use the ssdp:alive NOTIFY requests to keep the cache up to date.

If there is a proxy then the client will get its notify requests through the proxy. That is, the proxy will open a http connection to the client and send it notifications. In order to let the proxy know what notifications it is interested in the client will use the [GENA] SUBSCRIBE method setting the NT header value equal to the service types the client is interested in. The client will record the timeout value returned in the SUBSCRIBE response so that it can make sure to re-subscribe before its subscription expires.

When using a proxy the client will monitor the SSDP reserved multicast channel in order to hear the SSDP proxy's ssdp:alive NOTIFY requests. The client will ignore any other NOTIFY requests it hears over the SSDP reserved multicast channel when it is using a proxy.

If there isn't a proxy then the client will listen to the SSDP reserved multicast channel in order to hear notifications from any services that it may be interested in.

Proxy or not, the client can keep the cache up to date just by initializing it with ssdp:discovery SEARCH requests and then receiving appropriate ssdp:alive NOTIFY requests.

If the proxy should fail then the client will switch over to the SSDP reserved multicast channel until it hears a ssdp:alive NOTIFY request from a proxy. At that point it will repeat the cycle of subscribing to the proxy.

### 3.3.2. Requirements

Clients MUST provide for SSDP proxy support as specified in section . 4.1.

If a SSDP proxy is present then the client SHOULD use the SUBSCRIBE method as defined in [GENA] to subscribe to the proxy for all service types the client wishes to receive notifications for. The NT header of the SUBSCRIBE method is to be set to the service types the client is interested in.

If a SSDP proxy is present then the client SHOULD send all ssdp:discover, ssdp:notproxyaware and ssdp:proxyawareservice SEARCH requests to the proxy using http rather than to the SSDP reserved multicast channel using httpmu.

When a client is using a SSDP proxy it SHOULD NOT listen to any NOTIFY requests it receives over the SSDP reserved multicast channel other than those sent by the SSDP proxy. This prohibition is meant to allow the proxy to tailor the information a client receives based on the client's particular needs.

#### 3.3.3. SUBSCRIBE example

```
SUBSCRIBE dude HTTP/1.1
Host: iamthedude:203
NT: <upnp:toaster>
Callback: <http://blah/bar:923>
Scope: <http://iamthedude/dude:203>
Timeout: Infinite

HTTP/1.1 200 O.K.
Subscription-ID: uuid:kj9d4fae-7dec-11d0-a765-00a0c91e6bf6
Timeout: Second-604800
```

### 3.4. ssdp:aproxy Service

#### 3.4.1. Overview

When a SSDP proxy first wakes up or if it has lost a proxy election then it is an un-elected proxy of service type ssdp:aproxy.

The ssdp:aproxy service's first action will be to perform a proxy election. If the ssdp:aproxy service doesn't win the proxy election then it will remain a ssdp:aproxy.

Other than performing proxy election a ssdp:aproxy is just like any other proxy aware service, including the need to send out a ssdp:alive NOTIFY request when it wakes up and answering ssdp:discover and ssdp:proxyawareservices SEARCH requests.

#### 3.4.2. Requirements

SSDP proxies are SSDP services of type ssdp:aproxy and MUST comply with all requirements for SSDP proxy aware services.

All SSDP proxies MUST support the proxy election algorithm as specified in .4.3.

## 3.5. ssdp:proxy Service

### 3.5.1. Overview

If a ssdp:aproxy wins the proxy election then it still remains a ssdp:aproxy service. However it also takes on a new service type, ssdp:proxy.

The main job of a ssdp:proxy service is to cache information regarding the state of services in its assigned network. It does this by sending out a ssdp:notproxyaware SEARCH request in order to obtain information about all the non-proxy aware services. Then it sends out a ssdp:alive NOTIFY request of notification type ssdp:proxy. This will cause all the SSDP proxy aware services to automatically send NOTIFY requests to the proxy informing the proxy of their state.

Services that are not proxy aware will continue to send NOTIFY requests to the SSDP reserved multicast channel. The ssdp:proxy is responsible for listening to those NOTIFY requests and recording the information as specified in section .4.2.

Services which are SSDP proxy aware will start to send all of their NOTIFY requests directly to the proxy using http. As with services that aren't SSDP proxy aware, the ssdp:proxy will be responsible for recording the information the NOTIFY requests provide.

In addition to caching NOTIFY information the ssdp:proxy is also a subscription arbiter, as defined in [GENA]. This means that the ssdp:proxy will accept SUBSCRIBE requests from clients and will route NOTIFY requests that it receives accordingly. When the proxy announces itself with a ssdp:alive request all clients on the network will automatically send their SUBSCRIBE requests to the proxy.

The ssdp:proxy provides very basic SEARCH services, specifically it supports ssdp:discover, ssdp:notproxyaware and ssdp:proxyawareservice over http. Unlike a normal service the proxy is able to answer these SEARCH requests with information for services other than itself. This information is taken from the cache it built up from the NOTIFY requests it has received. By mapping the notification type of the NOTIFY request to service types the ssdp:proxy is able to answer SEARCH requests for particular service types. Unlike SEARCH requests over the SSDP multicast channel, which only have one answer, the ssdp:proxy will often need to return multiple responses to a single search request. As such some sort of compound format is needed. The property search result format first specified in [WEBDAV] and later adopted for SEARCH use in [DASL] is used for this purpose.

### 3.5.2. Requirements

A ssdp:proxy service MUST also be a ssdp:aproxy service and follow all the requirements for ssdp:aproxy services.

If the ssdp:aproxy service wins the proxy election then it MUST take on the service type ssdp:proxy and all the requirements thereof as specified in this document.

ssdp:proxy services are proxy aware services and MUST follow all the requirements thereof. Note, however, that a ssdp:proxy does not have to send out a proxy discovery request because the ssdp:proxy knows through out of band means who the proxy is, i.e. itself.

Upon becoming a ssdp:proxy service type the ssdp:proxy MUST perform a ssdp:notproxyaware SEARCH on the SSDP reserved multicast channel as well as announce itself through a ssdp:alive NOTIFY request on the SSDP reserved multicast channel.

All ssdp:alive NOTIFY requests of service type ssdp:proxy MUST include a cache-control: max-age directive and a Location and/or AL header with at least one http URL.

ssdp:proxy services MUST NOT provide answers to SEARCH requests received through the SSDP reserved multicast channel with information about any service but themselves.

All ssdp:proxy services MUST comply with all requirements for subscription arbiters as defined in [GENA].

ssdp:proxy services MUST record all service information gained from NOTIFY requests. If the ssdp:proxy does not have sufficient resources to support the current load of services then it MUST send out a ssdp:byebye announcement and cease to act as a ssdp:proxy. Any SEARCH requests of type ssdp:discover, ssdp:notproxyaware or ssdp:proxyawareservice which are sent to the location in the ssdp:alive NOTIFY request MUST be answered based on the complete dataset collected from the NOTIFY requests in addition to any extra service information the ssdp:proxy may have available.

ssdp:proxy services MUST be able to map the value in the NT header on a SUBSCRIBE request to the ST header value in both ssdp:discover SEARCH and ssdp:alive/ssdp:byebye NOTIFY requests.

## 4. SSDP Algorithms

### 4.1. SSDP Proxy Support

Resources that support SSDP proxies MAY be configured using some unspecified mechanism with the address of the SSDP proxy they are to use. Resources that support SSDP proxies SHOULD be configured to

perform SSDP proxy discovery if the SSDP proxy they are configured to use is not available.

If a resource that supports SSDP proxies has not been configured with a SSDP proxy to support then it MUST perform discovery on service type ssdp:proxy in order to determine if a SSDP proxy is available.

Resources that support SSDP proxies MAY begin to make requests directly to the reserved multicast channel after the first SSDP proxy discovery request has failed without completing the full UDP HTTP request cycle. This is often necessary because SSDP proxy discovery can take some time and users are notoriously impatient.

Any time a SSDP proxy supporting resource receives a notification that a SSDP proxy is available the resource MUST switch over to using the SSDP proxy.

Resources that support SSDP proxies MUST monitor the reserved multicast channel in order to listen for the SSDP proxy's notifications. Resources that support SSDP proxies MUST support both ssdp:alive and ssdp:byebye notification subtypes when sent by SSDP proxies. Resources that support SSDP proxies MAY ignore any other notification subtypes sent by the SSDP proxy.

If the time between required ssdp:alive notifications should pass without receiving a notification then the resource MUST assume that the proxy is dead and begin using the SSDP reserved multicast channel.

Once a proxy has been declared dead the resource MUST assume that the proxy has lost all subscription information for the resource. So if the proxy comes back to life the resource will still have to resubscribe.

The ssdp:alive notifications sent out by the SSDP proxy will include all the addresses that the SSDP proxy can be accessed through. If the current address the SSDP proxy supporting resource is using doesn't match any of the listed addresses then the resource MUST change to one of the listed addresses. So long as the USN in the ssdp:alive notification is the same as the USN of the SSDP proxy the resource is currently using there is no need to resubscribe.

If the USN of a ssdp:alive NOTIFY request from a ssdp:proxy service is not the same as the USN of the ssdp:proxy the resource is currently using then the resource MUST assume that the new proxy has no knowledge of their subscriptions, notifications, etc.

## 4.2. Notification Caching

ssdp:alive NOTIFY requests contain at minimum a NT header, a NTS header, a USN header, a Location and/or AL header and potentially a HTTP cache control directive such as expires or cache-control: max-age.

The NTS header, set to ssdp:alive, lets the receiver know that this is a notification from a SSDP service.

The NT header specifies the type of service.

The USN header specifies the USN for the service.

The Location and/or AL header specifies locations at which the service can be found.

Using the NT, USN and Location/AL values it is possible to create a cache entry for the service keyed to its USN. The entry will automatically be purged when the HTTP caching directive directs. If no cache lifetime information is provided then the NOTIFY information is to be discarded. Note, however, the ssdp:proxy services will still be required to route the NOTIFY method as specified for a subscription arbiter by [GENA].

Services keep caches which contain information about them up to date by regularly sending out ssdp:alive NOTIFY requests before the expiration of their previous ssdp:alive NOTIFY request.

Many services are best served by having extremely long expiration periods. For example, a printer or scanner that is unlikely to be moved very often will probably want to have an expiration period of a week or more.

## 4.3. Proxy Election

### 4.3.1. Proxy Election Algorithm

The proxy election algorithm is based on the use of the proxy number to determine which ssdp:aproxy should be the ssdp:proxy for the local network. Proxy numbers are decimal numbers between 0.0 and 1. The number 1 MUST NOT be configured by default into any device. It is reserved for use by administrators to force a particular ssdp:aproxy to always win election.

In the case of a tie the USN will break the tie. USNs are compared by treating them as strings of bytes and comparing each byte in network byte order. The first USN that has a larger byte value than the other wins the comparison.

Proxy elections are carried out by sending out a SSDPC httpmu method to the SSDP reserved multicast channel. As soon as the method is sent a timer is set to PROXYWAIT seconds. If no counter challenge is made before the timer expires then the challenger has won and MUST send out a ssdp:alive NOTIFY request declaring themselves ssdp:proxy with all the responsibilities that entails.

If a SSDPC request is received over the SSDP reserved multicast channel before the time expires then the timer must be reset.

When a SSDPC request is received the receiver MUST compare the PN value in the SSDPC request to its own. If the PN in the request is higher then the receiver has lost the challenge. If the PN in the request is lower then the receiver's then the receiver MUST send out another challenge. The reason being that the only way that a challenge with a lower PN could have been received is if there is another challenge underway which could indicate network connectivity problems. To be the safe side another set of challenges are issued. Note that the re-broadcast could cause a cascade of new challenges in the worst case. Finally, if the two PNs are equal then the USNs are to be compared. If the challenger's USN is larger as previously defined then the receiver has lost the challenge. If the USN is smaller then the challenge should be repeated for the same reasons as previously stated. If the two USNs are the same panic is probably called for, as this should be statistically impossible.

On the off chance that the previous was somehow comprehensible the following pseudo-code is offered to completely confuse the reader and hopefully directly contradict the previous text.

```
Challenge(PN, USN) {
   If (Challenge is not already underway)
      Send a SSDPC method with PN & USN headers;
   Proxy = Self;
   Set Timer to ProxyWait Seconds;
   While (Timer Isn't Up & Not Received a Message) {
      If Received ssdp:alive NOTIFY from ssdp:aproxy {
         If (Message.PN > PN) Proxy = Message;
         If (Message.PN == PN) {
            If (Message.USN > USN) {
               Send a SSDPC method with PN & USN headers;
               Proxy = Message;
            }
            ElseIf (Message.USN < USN)
               Send a SSDPC method with PN & USN headers;
            ElseIf (Message.USN == USN) PANIC;
         }
         If (message.PN < PN)
            Send a SSDPC method with PN & USN headers;
         Set Timer to ProxyWait Seconds;
         Continue;
      }
      If (Time is Up) {
         If (Proxy == Self) Declare Self Proxy;
```

```
            Exit;
         }
      }
   }
```

## 4.3.2. When Proxy Elections Occur

When a ssdp:aproxy service first connects to a new network, since it is by definition a proxy aware service, it will perform proxy discovery. If no response is received within the retry/time out interval defined by [HTTPUDP] then the ssdp:aproxy MUST issue a challenge.

Anytime a ssdp:aproxy receives a ssdp:alive NOTIFY request from a ssdp:proxy with a PN/USN lower than its own, a challenge MUST be issued.

Anytime a ssdp:proxy receives a ssdp:alive NOTIFY request from another ssdp:proxy, indicating a network communication failure or a buggy proxy, regardless of the PN/USN pair the receiving ssdp:proxy MUST issue a challenge. This will hopefully allow buggy proxies to recover and/or increase the chance of communicating over the apparent network problem.

Anytime a ssdp:aproxy service, remember that ssdp:proxy services are also ssdp:aproxy, receives a SSDPC request then the ssdp:aproxy MUST enter challenge mode if it isn't already in it.

ssdp:proxy resources who loose an election MUST NOT issue a ssdp:byebye NOTIFY request. This will cause needless proxy detection requests by proxy aware services and clients. The proxy election winner will issue a ssdp:alive NOTIFY request that will cause all of the loosing proxy's clients and services to switch over.

## 5. SSDPC httpmu Method

The SSDPC httpmu method is sent to the SSDP reserved multicast channel and MUST have two headers, PN and USN. There is no response to the SSDPC method.

## 5.1. Example

```
SSDPC * HTTP/1.1
Host: SSDPreservedmulticastchannel
PN: 0.001
USN: someunique:idscheme3
```

## 6. SSDP SEARCH Extensions

### 6.1. ssdp:discover SEARCH Extension

The purpose of the ssdp:discover SEARCH method is to find out about services available on a network.

The ssdp:discover SEARCH method is a SEARCH method which has been extended using the mechanism defined in [MAN].

The ssdp:discover SEARCH methods MUST have a ST header. They MAY have a body but the body MAY be ignored if not understood.

#### 6.1.1. ssdp:discover over httpmu

Only services whose service type matches the value in the ST header of a ssdp:discover SEARCH method MAY respond to a ssdp:discover SEARCH method received of httpmu. All other resources MUST NOT respond.

Services MUST only respond for themselves to ssdp:discover SEARCH methods received over httpmu.

The response to a ssdp:discover SEARCH method received over the SSDP reserved multicast channel is to be sent to the IP address and port of the requestor. The response MUST include a ST header set to the same service type as the request and a USN header containing the service's USN. The response SHOULD include a Location and/or AL header.

#### 6.1.1.1. Example

```
M-SEARCH * HTTP/1.1
Host: reservedSSDPmulticastaddress
Man: ssdp:discovery
ST: ge:fridge
MM: 0
MX: 3

HTTP/1.1 200 OK
ST: upnp:blender
USN: uuid:abcdefgh-7dec-11d0-a765-00a0c91e6bf6
AL: <blender:ixl><http://foo/bar>
```

### 6.1.2. ssdp:discover over http

Services MAY respond for other services to ssdp:discover SEARCH methods received over http. For example, this is how ssdp:proxies behave.

The response to the ssdp:discover SEARCH method when sent over http MUST comply with the response to the SEARCH method as specified in [DASL].

The location provided for each service response MUST be the service's USN. Each response MUST include the servicetype property and SHOULD include the location property.

Support for ssdp:discover does not imply nor require support for [WEBDAV] or [DASL].

Implementer's Note: DASL supports a very extensible response format so it would be expected that additional responses of different types might be mixed in with the service responses. As such it is important to check the servicetype property for each response to ensure that it is the service type that was requested.

### 6.1.2.1. Example

```
   M-SEARCH /I/AM HTTP/1.1
   Host: http://the/proxy
   Man: ssdp:discovery
   ST: upnp:blender

   HTTP/1.1 207 Multi-Status
   Content-Type: text/xml
   Content-Length: xxx

   <?xml version="1.0"?>
   <D:multistatus xmlns:D="DAV:" xmlns:S="ssdp:">
      <D:response>
         <D:href>uuid:kj9d4fae-7dec-11d0-a765-00a0c91e6bf6</D:href>
         <D:propstat>
            <D:prop>
               <S:location><D:href>http://foo/bar</D:href></S:location>
               <S:servicetype>
                  <D:href>upnp:blender</D:href>
               </S:servicetype>
            </D:prop>
         </D:propstat>
      </D:response>
      <D:response>
         <D:href>uuid:abcdefgh-7dec-11d0-a765-00a0c91e6bf6</D:href>
         <D:propstat>
            <D:prop>
               <S:location>
                  <D:href>blender:ixl</D:href>
                  <D:href>http://foo/bar</D:href>
               </S:location>
               <S:servicetype>
                  <D:href>ge:fridge</D:href>
               </S:servicetype>
```

```
            </D:prop>
         </D:propstat>
      </D:response>
   </D:multistatus>
```

## 6.2. ssdp:notproxyaware SEARCH Extension

The ssdp:notproxyaware SEARCH acts in the same manner as ssdp:discover except that the ST header is not submitted and only non-proxy aware SSDP services match the query.

## 6.3. ssdp:proxyawareservices SEARCH Extension

[Ed. Note: This was originally introduced for use with the proxy free shut off algorithm which has since been taken out of the spec and moved to the appendix. Its use is still mandated as I suspect it is still useful. If this suspicion turns out to be erroneous then it will be removed.]

The ssdp:proxyawareservices SEARCH acts in the same manner as ssdp:discover except that the ST header is not submitted and only proxy aware SSDP services match the query.

## 7. GENA Notification Subtypes

### 7.1.1. ssdp:alive

ssdp:alive notification subtype MUST NOT be used for notifications from non-SSDP compliant services.

The NT header of a ssdp:alive NOTIFY method MUST be set to the service type of the SSDP compliant service sending the request.

ssdp:alive notifications MUST include a USN header set to the value of the service's USN.

ssdp:alive notifications SHOULD contain a Location and/or AL header. If there is no DNS support available on the local network then at least one location SHOULD be provided using an IP address for the host.

ssdp:alive notifications SHOULD contain a cache-control: max-age directive. Please refer to section .4.2 for details on caching ssdp:alive notifications based on cache directives.

ssdp:alive notifications sent by ssdp:proxy services MUST include the PN header.

#### 7.1.1.1. Example

```
   NOTIFY * HTTP/1.1
   Host: reservedSSDPmulticastaddress
   NT: blenderassociation:blender
   NTS: ssdp:alive
   USN: someunique:idscheme3
   AL: <blender:ixl><http://foo/bar>
   Cache-Control: max-age = 7393
```

This is a notification sent to the reserved SSDP multicast channel announcing that a service of service type blenderassociation:blender is alive and available for use at the locations blender:ixl and http://foo/bar. Because the NOTIFY request was sent using httpmu there is no response.

### 7.1.2. ssdp:byebye

ssdp:byebye notifications allow arbitrary services to inform clients that the service is about to go off-line. Clients and SSDP Proxies SHOULD remove the service's entry from their cache upon receiving a ssdp:byebye notification.

ssdp:byebye notifications MUST set the NT header to their service type and MUST include a USN header.

#### 7.1.2.1. Example

```
   NOTIFY /proxy/location HTTP/1.1
   Host: ssdpproxy
   NT: blenderassociation:blender
   NTS: ssdp:byebye
   USN: someunique:idscheme3

   HTTP/1.1 200 O.K.
```

In this case the blender is SSDP proxy aware and has switched over to using a SSDP proxy. Because the message was sent using http there is a response.

## 8. SSDP properties

The following properties are returned in SSDP enhance SEARCH responses sent across http.

### 8.1. servicetype Property

Name: servicetype
Namespace: SSDP:
Purpose: Specifies the service type of the associated service.

```
<!ELEMENT servicetype href>
```

The HREF XML element is defined in [WEBDAV].

## 8.2. location Property

Name: location
Namespace: SSDP:
Purpose: Specifies the location(s) at which the service can be contacted.

```
<!ELEMENT location (href)*>
```

## 9.   HTTP Headers

## 9.1. USN Header

USN = "USN" ":" AbsoluteURI; defined in section 3.2.1 of [RFC2068]

Contains a USN.

## 9.2. ST Header

ST = "ST" ":" AbsoluteURI

Contains a service type.

## 9.3. PN Header

PN = "PN" ":" ("1" | "0." 1*digit)

Contains a proxy number.

## 10.  SSDP Reserved Multicast Channel

The SSDP reserved multicast channel will be a locally scoped multicast address as defined in [RFC2365]. The actual address will be issued by IANA.

## 11. Security Considerations

TBD.

## 12. IANA Considerations

To ensure correct interoperation based on this specification, IANA must reserve the URI namespace starting with "ssdp:" for use by this specification, its revisions, and related SSDP specifications.

## 13. Appendix - Constants

MAXPROX - Maximum number of SSDP proxy aware services allowed on a proxy free network before SSDP services must be disabled.

PROXYWAIT - Number of seconds to wait before assuming one has won a challenge.

## 14. Appendix - Proxy Free Shut Off

[Ed. Note: This is a really wacky algorithm that doesn't even work all that well that was introduced to deal with concerns about overload from SSDP proxy aware services on large networks with no SSDP proxy support. I have removed it from the spec until we have a better idea of the need for this algorithm. To make it really work we would have to change it to have one of the services be elected as a "shut off leader" who would then answer search requests for the number of services on the network. This is just ugly. Let's hope we don't need it.]

In certain circumstances a large number of SSDP proxy aware services may be present on a network. However the network itself was never meant to run SSDP.

For example, an administrator may purchase a large number of printers that support many different access mechanisms including SSDP. The administrator never intended for the printers to use SSDP and certainly doesn't want to be bothered with have to explicitly configure them to not use SSDP.

In such a case the desired behavior is for the SSDP proxy aware services to de-active their SSDP support rather than flood the network with large numbers of notifications.

The issue is of less relevance to non-SSDP proxy aware services as these services are required to produce very few notifications and so do not pose much of a threat to the network.

In order to detect when this situation has come to pass SSDP proxy aware services that perform proxy discovery but fail to find a proxy MUST perform a SEARCH for resources of type ssdp:proxyawareservice. If the speed of the network divided by the number of responses exceeds MAXPROX then the service MUST enter the wait state. In the wait state the service MUST disable all SSDP functionality except listening for a ssdp:alive notification from a SSDP proxy and answering ssdp:proxyawareservice SEARCH requests sent across the SSDP reserved multicast channel.

If there is a SSDP proxy but it subsequently dies without replacement then services SHOULD wait for the amount of time specified in the cache-control: max-age directive before assuming

that no proxy is coming back and executing the proxy free shut off algorithm from the beginning.

[Ed. Note: There is a built in assumption here that non-proxy aware services will have such a low rate of NOTIFY activity that it would be almost impossible to put enough of them on a single local multicast loop to cause a traffic overload. Such assumptions are what administrative nightmares are made of so we may want to also put in detection of non-proxy aware services and assign them an even higher threshold, but a threshold none the less, to determine cut off. The other possibility is to have a bandwidth based cut off, but this requires knowing what the bandwidth of the underlying link is.]

## 15. Acknowledgements

This document is the result of enormous effort by a large number of people including but not limited to: Munil Shah, Holly Knight, Peter

Ford, Mike Zintel, Pradeep Bahl, Paul Moore, Babak Jahromi, Brandon Watson, Michel Guittet, Todd Fisher, and Craig White.

## 16. References

[GENA] J. Cohen, S. Aggarwal, Y. Y. Goland. General Event Notification Architecture Base: Client to Arbiter. Internet Draft - a work in progress, draft-cohen-gena-client-00.txt.

[MAN] H. Nielsen, P. Leach, S. Lawrence. Mandatory Extensions in HTTP. Internet Draft - a work in progress, draft-frystyk-http-mandatory-00.txt.

[HTTPUDP] Y. Y. Goland. Multicast and Unicast UDP HTTP Requests. Internet Draft - a work in progress, draft-goland-http-udp-00.txt.

[RFC2365] D. Meyer.  Administratively Scoped IP Multicast.  RFC 2365, July 1998.

[RFC2396] T. Berners-Lee, R. Fielding and L. Masinter.  Uniform Resource Identifiers (URI): Generic Syntax.  RFC 2396, August 1998.

[RFC2119] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels.  RFC 2119, March 1997.

[RFC2068] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol -- HTTP/1.1.  RFC 2068, January 1997.

[RFC2518] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring û WEBDAV. RFC 2518, February 1999.

## 17. Author's Addresses

Yaron Y. Goland, Ting Cai, Paul Leach, Ye Gu
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Email: {yarong, tingcai, paulle}@microsoft.com

Shivaun Albright
Hewlett-Packard Company
Roseville, CA

Email: SHIVAUN_ALBRIGHT@HP-Roseville-om2.om.hp.com

This document will expire in December 1999.