NETWORK Working Group
INTERNET-DRAFT
Category: Informational
6 June 2001
Expires in six months

Erik Guttman
Sun Microsystems

# An API for the Zeroconf Multicast Address Allocation Protocol (ZMAAP)
<draft-ietf-zeroconf-zmaap-api-00.txt>

## Status of this Memo



## Copyright Notice



## Abstract

Today, with the rapid rise of home networking, there is an increasing need for auto-configuration mechanisms. This document specifies an api to be used for applications which require multicast addresses on small networks without a multicast address allocation server.

## 1.0 Introduction

The Zeroconf Multicast Address Allocation Protocol (ZMAAP) is defined elsewhere [1]. This document specifies an application programmer interface (API) which builds upon the foundation of the Abstract API for Multicast Address Allocation [2]. Specifically, there are additional requirements posed by ZMAAP which are not considered in

RFC 2771:

- Shared ownership of allocations (renewal and defense)
- Notification of conflicts with specific allocations
- Allocations all start immediately and continue until they are released. This is a simplified API which does not allow applications to manage allocations via absolute times.

It should be transperant to the API whether the allocations are done using ZMAAP, MADCAP [3] or some other mechanism.

In this document, the key words "MAY", "MUST, "MUST NOT", "optional", "recommended", "SHOULD", and "SHOULD NOT", are to be interpreted as described in [4].

## 2.0 Abstract ZMAAP API Definition

The ZMAAP API provides the functions described in the Abstract Multicast Address Allocation API [2], with a few additions.

## 2.1 Request Enumeration of Available Scopes

Scopes available for allocation are returned.

This corresponds to the Abstract API get_multicast_addr_scopes().

Parameters: the address family.

Return value: Scope Records, each of which contains:

* the address family
* the start and end address of the range
* a suggested transmission IPv4 TTL or IPv6 Hop Count for messages multicast into the scope range.

Mini-MAAS behavior: None.

## 2.2 Allocation

The application specifies which scope to allocate from and how many addresses are needed.

This corresponds to the Abstract API alloc_multicast_addr().

Parameters:

* Scope Record:  Which scope to use.

* Integer: How many addresses to attempt to allocate.
* Integer: How many milliseconds to block before giving up, if allocation is not successful.

Return value: No result, or a Lease Descriptor containing:

* An address range
* A lease identifier (this is useful information for including in session announcements, see [1], Appendix B).

Mini-MAAS behavior:

The mini-MAAS attempts to claim the address(es). It will give up after the time allotted for allocation has expired. If it succeeds, it will enter the allocation into the allocation record. The mini-MAAS will select a lease duration. Before this lease duration expires, the mini-MAAS send an addtional AIU message - effectively renewing the lease.

## 2.3 Release

An application indicates it is no longer interested in an allocation.

This corresponds to the Abstract API deallocate_multicast_addr().

Parameters: Lease Descriptor.

Return value: Success or Failure (ie. bad parameter).

Mini-MAAS behavior:

The allocation is removed from the allocation record.

## 2.4 Defense

An application, having discovered a session (see [1], Appendix B) wishes to participate in defense of a multicast address.

This is a new interface, not present in the Abstract API.

Parameters: Lease Descriptor.

Return value: Success or Failure (bad parameter or bad descriptor).

Mini-MAAS behavior:

The mini-MAAS adds the lease descriptor to its allocation record.

## 2.5 Conflict Notification.

An application registers its desire to be notified if a conflict is discovered for a given address allocation. This address allocation could be made by the application (using the Allocation interface) or it may have be a discovered session (see [1], Appendix B).

This is a new interface, not present in the Abstract API.

Parameters:

* Lease Descriptor.
* Opaque User Parameter
* Callback function or the equivalent. The parameters to this function will be the Lease Descriptor which has a conflict and the Opaque User Parameter.

Return value: Success or Failure (bad parameter or bad descriptor).

Mini-MAAS behavior:

Mini-MAASs process all incoming AIU messages. AIUs are sent initially upon allocation and periodically (in order to prevent the lifetime of the lease expiring). If an AIU received conflicts with an entry in the allocation record, the record MUST be removed (see [1] section 4.4.4).

The mini-MAAS issues the callback function associated with the lease descriptor to all applications which have requested notification of a conflict. This notification occurs once (that is, only the first time there is a conflict, not every time). The notification callback function is likely to be made in the context of a different thread than the calling application.

## 2.6 Scope Name Query

The application can request the name of a scope by specifying the scope record and the language in which to return the string. If the name cannot be returned in the requested language, the name in the default language is supplied.

This corresponds to the Abstract API named get_scope_name().

Parameters:

* Scope Record: Get the name for this scope.
* String: Language Tag [5]. This is the language of the scope name to return.

Return Value:

* a string identifier describing the address scope.
* a Language Tag [5] which specifies the language for the scope identifier.

## 2.7 Abstract API Interfaces not supported

The ZMAAP API is a simplified subset of the API presente din RFC 2771. It does not include support for:

- change_multicast_addr_start_time()
- change_multicast_addr_lifetime()

All ZMAAP API allocations are considered to be continual, until released. A mini-MAAS associates a lifetime with the registration, but this is outside of the application's control.

- get_scope_netsting_state()
- get_larger_scopes()
- get_smaller_scopes()

ZMAAP supported scopes are simple at the present time, not nested.

## 3.0 Programming Language Specific Concrete APIs

This includes utilities required for the language specific API, ie., memory management functions for the C language API.

## 4.0 ZMAAP API for C

## 4.1 Definitions

Data elements in the structures below use types defined in [20].

```
typedef struct scoperec {
   struct sockaddr  sr_start_addr;   /* Scope's first address */
   struct sockaddr  sr_end_addr;     /* Scope's final address */
   int              sr_scope;        /* Scope ID, [1] Section 4.2.2
   */
   int              sr_ttl;          /* Suggested TTL to use */
} scope_record;
```

The scope_record includes a range of addresses and a TTL which hosts SHOULD use when sending messages to addresses in that scope. For example, datagrams sent to link-local scopes should set the IPv4 TTL

(or IPv6 Hop Count field) to 1. This will reduce the chance that these datagrams will be forwarded off-link by routers, incorrectly.

```
typedef struct leasedesc {
   int             ld_addr_family; /* IPv4 = 1, IPv6 = 2 */
   struct sockaddr ld_start_addr;  /* First address allocated */
   struct sockaddr ld_end_addr;    /* Final address allocated */
   struct sockaddr ld_interface;   /* The associated interface */
   unsigned long   ld_id;          /* The lease identifier */
} lease_desc;
```

The lease_desc contains information about an individual address allocation. In some cases, these are returned by the API. In others the application forms these on the basis of session discovery. (See appendix B).

```
typedef void zmaap_cb(lease_desc ld, void *pv);
```

A function with a prototype matching zmaap_cb is registered using the zmaap_register() function below. The callback function is evoked by the API (in a distinct thread) if there is an allocation conflict detected in the address range of the registered lease descriptor. The parameters to this function are described under zmaap_register(), below.

```
typedef enum { OK=0, LEASE_CONFLICT=-1, TIMEOUT=-2, BAD_PARAM=-3 }
   ZMErrCode;
```

LEASE_CONFLICT is returned if a lease descriptor parameter conflicts with another, existing multicast allocation or fails to correspond to an entry in the allocation record.

TIMEOUT is returned if the attempt to verify the validity of the lease times out before finding determining if it corresponded or was in conflict with a prior address allocation.

## 4.2 Functions

```
scope_record * zmaap_enumerate_scopes(int family);
```

Parameters: family IPv4 = 1, IPv6 = 2. No other values are allowed.

Returns: An array of scope_records. The caller frees them using zmaap_free(). An improper value for family results in a NULL return value.

```
lease_desc * zmaap_allocate(scope_record *sr, struct sockaddr ifa,
   int num, int msec);
```

```
     Parameters:
       sr    A scope record returned using zmaap_enumerate_scopes();
```

ifa The interface on which to make the allocation.
num The number of addresses requested.
msec The maximum number of milliseconds to attempt allocation.

Returns: NULL if no addresses can be obtained before the allotted time expires of if the parameters are bad. The lease_desc returned must be freed using zmaap_free.

```
int zmaap_release(lease_desc ld);
```

Parameters:
ld The lease descriptor to remove from the allocation record.

Returns: ZMErrCode.

```
int zmaap_defense(lease_desc ld, int msec);
```

Parameters:
ld The lease descriptor of the multicast allocation to defend.
msec The number of seconds to block while verifying the lease to defend is valid.

Returns: ZMErrCode

```
int zmaap_register(lease_desc ld, void *pv, zmaap_cb *pf, int msec);
```

Parameters:
ld The lease descriptor of the allocation app wants to be notified of if a conflict occurs.
pv The opaque user parameter present in the callback function.
pf The user supplied callback function.
msec The maximum number of milliseconds to attempt to verify ld.

Returns: ZMErrCode

```
void zmaap_scope_name(scope_record sr, const char * tagq, char
    **name, char **taga);
```

Parameters:
sr The scope to get the name of.
tagq The language tag desired. If NULL use default.
name Will point to an allocated buffer with the scope's name.
taga Will point to an allocated buffer with the name's language tag.

Returns: This routine always succeeds. The strings allocated must

be freed with free().

```
void zmaap_free(void *mem);
```

Parameters:
mem Either an array of scope_records or a lease_desc allocated by the zmaap API.

## 5. ZMAAP API for Java

The Java API uses definitions from JDK 1.4 [6].

```
package org.zeroconf.zmaap;

import java.net.*;
import java.util.*;

public class ScopeRecord {

  public SocketAddress iaStart; // The start address of the scope
    range.
  public SocketAddress iaEnd;   // The end address of the scope
    range.
  public int ttl;               // The recommended TTL to use in
    scope.
  public int scopeid;           // See [1], Section 4.2.2

}

public BadLeaseException extends Exception {

  public BadLeaseException(String msg);
  public BadLeaseException();

}

public ScopeName {

  public Locale locale;       // The Locale of the scope name
  public String scopename;

}

public class LeaseDesc {

  public SocketAddress start;     // The allocation's start address.
  public SocketAddress end;       // The allocation's end address.
  public NetworkInterface interf; // The interface for the
```

allocation.

```
    public int id;                  // The allocation's lease ID.

  }

  public class ConflictListener implements EventListener {

    /**
     * This event occurs when a conflict arises with an allocation.
     * The ConflictListener is registered with ZMAAP.register().
     *
     * @param alloc This is the allocation which has a conflict.
     * @param o     This is an opaque object registered with ZMAAP.
     */
    public void conflictEvent(LeaseDesc alloc, Object o);

  }

  public class ZMAAP {

    /**
     * @return All scopes available for allocation.
     */
    static ScopeRecord[] availableScopes();

    /**
     * Return the name associated with a scope, in requested language.
     * If the requested language is not supported, the default language
     * will be used.
     *
     * @param locale  The locale requested for the scope name.
     * @param scope   The scope whose name is sought.
     *
     * @return The name associated with the scope.
     */
    static ScopeName queryScopeName(Locale locale, ScopeRecord scope);

    /**
     * Allocate a range of multicast addresses.  This method will
     * only return after it is successful or times out.
     *
     * @param sr    A ScopeRecord obtained using availableScopes().
     * @param num   The number of addresses requested.
     * @param msec  Maximum number of milliseconds to attempt to
     *              allocate the addresses.
     * @return      An allocation.
     * @exception java.io.InterruptedIOException
     *              If ZMAAP.allocate() runs out of time.
     * @exception java.lang.IllegalArgumentException
     *              If the parameters are unacceptable.
```

```
 */
static LeaseDesc allocate(ScopeRecord sr, int num, int msec)
```

```
    throws InterruptedIOException, IllegalArgumentException;

    /**
     * Stop defending an address allocation.  This will happen
     * anyway if this JVM exits.
     *
     * @param ld This must represent an allocation which has either
     *           been created with ZMAAP.allocate() or is being
     *           defended, using ZMAAP.defend().
     * @exception org.zeroconf.BadLeaseException
     *           If the allocation in the parameter does not
     *           correspond to a LeaseDesc this ZMAAP instance
     *           is currently defending.
     */
    static void release(LeaseDesc ld)
    throws BadLeaseException;

    /**
     * Participate in defense of an allocation.  This method
     * will only return when successful or it times out.
     *
     * @param ld This allocation must correspond to one which
     *           has been made by calling ZMAAP.allocate() or
     *           which has been discovered by some other means,
     *           such as the Session Announcement Protocol,
     *           RFC 2974.  If the allocation is unknown to the
     *           ZMAAP object, it will attempt to validate its
     *           existence.
     *
     * @param ms The maximum time in milliseconds to attempt to
     *           validate the lease descriptor before giving up.
     * @exception java.io.InterruptedIOException
     *           If validating the allocation runs out of time.
     * @exception org.zeroconf.BadLeaseException
     *           If the allocation does not correspond to an
     *           existing, defended address.
     * @exception java.lang.IllegalArgumentException
     *           If the ms argument is <= 0.
     */
    static void defend(LeaseDesc ld, int ms)
    throws InterruptedIOException, BadLeaseException,
           IllegalArgumentException;
    /**
     * Request notification if an allocation conflict occurs.
     * Note that a nonexistent allocation may be passed to this
     * method in the LeaseDesc parameter - its validity will
     * not necessarily be checked - though conflicts with it
     * will (eventually) cause a notification to occur.
```

```
     *
     * @param cl The conflictEvent method of this object will
     *           be called on a separate thread if a conflict
```

```
     *           is detected.
     * @param ld The allocation for which the ZMAAP object
     *           must detect conflicts.
     * @param o  An opaque user parameter which will be passed
     *           to the ConflictListener.conflictEvent method.
     * @param ms The maximum time in milliseconds to attempt to
     *           validate the lease descriptor before giving up.
     * @exception java.lang.IllegalArguement
     *           If cl is null.
     * @exception org.zeroconf.BadLeaseException
     *           If the allocation does not correspond to an
     *           existing, defended address.
     * @exception java.io.InterruptedIOException
     *           If validating the allocation runs out of time.
     */
    static void register(ConflictListener cl, LeaseDesc ld, Object o,
                         int ms)
    throws IllegalArgumentException, BadLeaseException,
           InterruptedIOException;
  }
```

## References


[1] Catrina, O., et. al., "Zeroconf Multicast Address Allocation Protocol (ZMAAP)", draft-ietf-zeroconf-zmaap-01.txt. A work in progress.

[2] Finlayson, R., "An Abstract API for Multicast Address Allocation", RFC 2771, February 2000.

[3] Hanna, S., Patel, B., and M. Shah, "Multicast Address Dynamic Client Allocation Protocol (MADCAP)", RFC 2730, December 1999.

[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[5] Alvestrand, H. "Tags for the Identification of Languages", RFC 3066, January 2001.

[6] Java Development Kit, version 1.4.0, Beta, Documentation, http://www.javasoft.com/j2se/1.4/#documentation


## Acknowledgments

Dave Thaler's input assisted in preparing this specification.

## Authors' Addresses

Erik Guttman
Sun Microsystems
Eichhoelzelstr. 7
74915 Waibstadt Germany

Phone: +49 172 865 5497
Email: erik.guttman@sun.com

## Full Copyright Statement



## Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.