

IPWorks Configuration Management

DESCRIPTION

Copyright

© Ericsson AB 2017, 2018. All rights reserved. No part of this document may be reproduced in any form without the written permission of the copyright owner.

Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design and manufacturing. Ericsson shall have no liability for any error or damage of any kind resulting from the use of this document.

Trademark List

All trademarks mentioned herein are the property of their respective owners. These are shown in the document Trademark Information.



Contents

1	Introduction	1
1.1	Related Information	1
2	Basic Concepts	3
2.1	Managed Object Model (ECLI)	3
2.2	IPWorks Managed Objects (IPWCLI)	4
3	User Management	9
3.1	User	9
3.2	Profile	9
3.3	Login History	10
3.4	Password	11
4	Storage Server	13
4.1	Basic Concepts	13
4.2	Fields	13
4.3	Classes	14
4.4	Relationships	14
4.5	Operations	14
4.6	Enumerations	15
4.7	Finding Objects	15
4.8	Creating Objects	15
4.9	Modifying Objects	16
4.10	Deletion of an Object	16
4.11	Making Decisions	16
5	DNS Management	17
5.1	DnsServer	17
5.2	View	18
5.3	Zones	18
5.4	DNS Resource Records	21
5.5	ACL	24
5.6	TSIGKeys	24
5.7	Options	25



6	ActiveSelect DNS Management	27
6.1	ActiveSelect Application Scenarios	27
6.2	ActiveSelect Functions	28
6.3	ActiveSelect DNS Components	29
6.4	ActiveSelect Managed Objects	33
7	ENUM Management	35
7.1	ENUM	35
7.2	ENUM Managed Objects	36
8	AAA Management	39
8.1	Radius AAA	39
8.2	EPC AAA	43
9	DHCPv4 Management	47
9.1	DHCPv4 Overview	47
9.2	DHCPv4 Service	48
9.3	NACF Service	48
10	MySQL Database Management	51
10.1	MySQL File Locations	51
10.2	MySQL Scripts	51
10.3	MySQL Server	51
10.4	MySQL NDB Cluster	52
10.5	MySQL CLI	54
11	Service Life Cycle Management	55
12	Transaction Logging	57
12.1	Transaction Logging Periodic Maintenance Operations	57
12.2	Transaction Logging Message Format	57
12.3	Transaction Logging Events	58
13	Server Manager Configuration Properties	61
13.1	Log Files for Server Manager	61
13.2	Configuration Files	61
13.3	Server Manager Instances	62
13.4	Startup of Server Manager	62
13.5	Communication with DNS Server	63
13.6	Communication with ActiveSelect DNS Monitor	63



13.7	Communication with AAA Server	64
13.8	Communication with DHCPv4 Server	64
13.9	Server Status	64
13.10	Query Limits	65
14	Configuration Management	67
15	Appendix	69
15.1	DNS Options	69
	Reference List	77





1 Introduction

This document describes how to use the Configuration Management (CM) function in the IPWorks.

1.1 Related Information

Trademark information, typographic conventions, and definition and explanation of abbreviations and terminology can be found in the following documents:

- *Trademark Information*
- *Typographic Conventions*
- *Glossary of Terms and Acronyms*





2 Basic Concepts

This section defines the basic concepts used when working with the Configuration Management (CM) in the IPWorks product.

It describes the IPWorks schema, the data model that is used to configure the services within a network and is also the base of the IPWorks architecture.

IPWorks provides two configuration methods. One is provided by Ericsson Command-Line Interface (ECLI) to control basic functions of IPWorks components. This method is based on the Ericsson Information Model (ECIM) compliant Managed Object Model (MOM). The other method is IPWorks CLI (also called as IPWCLI), it is used to perform provisioning related configuration for DNS, ASDNS, and ENUM related objects.

For more information about how to use IPWCLI and ECLI, refer to *Managed Object Model User Guide* and *Command Line Interface User Guide for IPWorks SS*.

2.1 Managed Object Model (ECLI)

In IPWorks, Ericsson Command-Line Interface (ECLI) is used for controlling the basic functions of the IPWorks components, including:

- Configuring logging of IPWorks processes.
- Defining configuration parameters specific to the current host.
- Determining whether enable or disable functions.

The IPWorks managed objects used by ECLI are represented in the *Managed Object Model (MOM)* as follows:

```
ManagedElement
+-IpworksFunction
+-IPWorksAAARoot
  +-IPWorksAAACommonRoot
  +-IPWorksDiameterAAARoot
  +-IPWorksRadiusAAARoot
+-IpworksCommonRoot
  +-StorageServer
+-IpworksDnsRoot
  +-AsdnsServer
  +-DnsServer
  +-IpworksEnumRoot
  +-ServerType
```



For general information about the MOM, MOCs, MOs, cardinality, and related concepts, refer to *Managed Object Model User Guide*.

Table 1 describes the IPWorks specific Managed Object Classes (MOCs).

Table 1 IPWorks Managed Object Classes

Managed Object Class	Description
<i>IpworksFunction</i>	The root of the IPWorks model. Defines some common information.
<i>IPWorksAAARoot</i>	The root MOC of IPWorks AAA.
<i>IPWorksAAACommonRoot</i>	The root MOC of IPWorks AAA Common
<i>IPWorksDiameterAAARoot</i>	Defines the AAA for EPC (Diameter)
<i>IPWorksRadiusAAARoot</i>	Defines the AAA for Radius.
<i>IpworksCommonRoot</i>	Defines the IPWorks Element Manager (EM).
<i>StorageServer</i>	Defines the Storage Server.
<i>IpworksDnsRoot</i>	The root MOC of IPWorks DNS.
<i>AsdnsServer</i>	Defines the ActiveSelect DNS (ASDNS) server.
<i>DnsServer</i>	Defines the DNS server.
<i>IpworksEnumRoot</i>	The root MOC of ENUM server.
<i>ServerType</i>	Defines the type of server.

2.2 IPWorks Managed Objects (IPWCLI)

The IPWorks configuration is defined by a set of configuration objects. These objects are managed in the `IPWCLI`. Each object has a *class* which determines how the object is used.

Every object has a set of *fields* associated with it. The fields associated with an object are determined by the class of the object. Fields have values that are specific to that object. Some multi-valued fields preserve the order of the values. For object classes, there are fields that require a value, while other fields are optional.

For every object, there is a set of fields that uniquely identify that object. These fields are referred to as the *key fields* and the values of those fields are called the *key values*. The combination of key values must be unique. They are also used to identify that object.

Some classes also have special operations that are performed on the managed objects. These operations may take parameters.

For details about the IPWorks managed objects used by `IPWCLI`, refer to *IPWorks DNS, ASDNS, ENUM Parameter Description*.



2.2.1 Organizational Concepts

This section introduces classes that assist the organization of data in IPWCLI.

2.2.1.1 Partition

A partition is a **management container** that allows the user to maintain different sets of data within IPWorks. All objects are placed into a partition.

- When a new object is created, the object is assigned to a specific partition.
- If the user is working with multiple partitions, all the objects in one partition are independent of objects in another partition.

Therefore, the partition acts as a data separation mechanism.

IPWorks predefines the *active* partition. It is used to manage data that is active on the network. Other partitions can be created for test or experiment purposes. However, only the *active* partition is used for publishing data in the PL. Using *scoping*, this partition is set up as the default partition for all actions.

Note: Unless the user wants to work within a different partition, the partition being worked on does not need to be specified. IPWorks assumes that the partition is *active*.

A partition is a **named container** that can be seen using the `list` command.

The following is a sample command for partitions.

```
IPWorks> list partitions
[Partition active]
  Name: active
```

Example 1 List Partitions

A partition object is considered as a prerequisite for all objects contained in the partition. Therefore, if any partition other than the *active* partition is used, the partition object must be created before any object in the partition is created.

Data in partitions is independent of data in other partitions, therefore partitions can be used in *experimental* containers. A configuration is created, experimented with, or examined by the resulting configuration files, and so on. This can be done without any impact on the active data.

For example, there is an external file called `gprs.cli` includes all the commands for creating the sample GPRS network. To create a partition that contains these objects to experiment with, use the following commands:

```
IPWorks>create partition gprs
1 object(s) created.
IPWorks> scope partition gprs
IPWorks> gprs.cli -trace
... <actual command output removed> ...
```

Example 2 Create a Partition with a Scope



These commands create a partition and set the scope. All the subsequent operations are performed within the partition. Then, the command file is executed with *tracing* turned on to display the output.

2.2.1.2 Area

An area is a virtual container used for organizing some of the managed objects in IPWorks. An area contains all the objects that define a namespace and address-space (such as resource records). The area acts as a boundary for many of the consistency checks made by IPWorks.

Some of the sample networks in this document use two separate areas, one for the internal network configuration and the other for the external network configuration. These are called *internalarea* and *externalarea* just to keep the naming schema simple. The following IPWorks CLI commands are examples for creating and displaying these areas:

```
IPWorks> create area internalarea -set
description="Area for internal data"
1 object(s) created.
IPWorks> create area externalarea -set
description="Area for external data"
1 object(s) created.
IPWorks> list areas
```

```
[Area default]
Partition: gprs
Name: default
[Area externalarea]
Partition: gprs
Name: externalarea
Description: Area for external data
[Area internalarea]
Partition: gprs
Name: internalarea
Description: Area for internal data
```

Example 3 Create and Display Areas

Note: IPWorks has a predefined *default* area when it is installed (as shown in the example above). In sample networks where there are no overlapping address spaces, or when the network is not deployed with a split namespace, it is best to use the *default* area to configure the data.

2.2.2 Data Management Conventions

This section describes the basic data management conventions. To have a thorough understanding of the schema, it is suggested that users learn the data management conventions for IPWorks.

It is not allowed to create objects for data that is not managed in IPWorks. Whether the data is managed depends on related objects. These related objects indicate how the data is managed, therefore, usually a set of objects can only be created in one order.

When one object must exist for a second object to be created, the first object is called a prerequisite object and the second object is called the dependent



object. This principle also requires that all dependent objects must be deleted before a prerequisite object is deleted.

As an example of these principles, consider the relationship between *resource records* and *zones*. A resource record is considered to be managed if there is at least one zone already defined that could contain the resource record. The following is a list of rules for zones and resource records that must be followed:

- A zone must be created before the resource records for that zone.
- Resource records must be within a zone, otherwise the resource record is considered *unmanaged data* and causes an error.
- If a zone is deleted, the resource records belonging to that zone must be deleted as well.





3 User Management

This section describes the concepts related to SS user management and access control.

For information on the configuration procedures, refer to *Configure User Account*.

3.1 User

The default configuration for IPWorks requires users to log on to the product to gain access to the IPWorks data. This applies to all management interfaces. To facilitate this, IPWorks has a *user* object that can be created to represent information about the users of the system.

Note: Unless the default access control has been modified, only administrators are allowed to modify user objects.

When defining a user object, the following items are included:

username	Account name used when logging in.
password	Password used to log in. The value is encrypted when it is stored, but displayed when in the management interfaces.
user id	Used to log and audit the user activities. If a value is not specified for this, it uses the username as a default
profiles	Profiles associated with the user. Used to determine the level of access control that is allowed.

When IPWorks is installed, it creates an *admin* account that has administrative privileges. The username and password for this account are based on questions answered during the installation. Since this is the only account with privileges, it is important to remember both the account name and the password specified.

To show the definition of an `admin` user, refer to *Showing User Definition in Configure User Account*.

3.2 Profile

A profile is used in IPWorks to define a set of common access control rules that are applicable to one or more users. A user can have more than one profile, in which case the access control is consisted of the rules defined in all the profiles.



Note: Unless the default access control is modified, only administrators are allowed to modify profile objects.

When defining a profile object, the following items are included:

name	The name used to identify the profile
edit rules	An ordered list of rules that define the access control that is associated with users that have this profile. The rules are checked in order until an action is allowed or denied by a rule. If the user has multiple profiles, the rules from the profiles are applied in the order in which the profiles are defined. If no rule is found to define whether an action is allowed or denied, the default access control is used.

See the following example for the syntax of edit rules:

```
<permission> <type> [ <action> ]
```

Note:

- **<permission>**: either **allow** or **deny**
- **<type>**: the type of object this rule applies to. It can be a schema class, a subclass, or the value **all**.
- **<action>**: the type of edit action the rule applies to: either **create**, **delete** or **modify**. If not specified, the rule applies to all edit actions for the specified class.

IPWorks is delivered with the following predefined profiles:

Reader	Allows objects only to be read. Any attempt to create, edit, or delete objects results in an error.
Writer	Allows objects to be written, except for objects that are only allowed to be changed by administrators.
Administrator	Allows any changes to the objects.

To show the profile definition, refer to *Showing Profile Definition* in *Configure User Account*.

3.3 Login History

IPWorks supports login management for IPWorks SS user. When an SS user logs in successfully, the login history is displayed via CLI.



The system records the login attempts between two successful logins.

- If any failed attempt is available, the system displays the number of failed attempts to SS users at the next successful login.
- If no failed attempt is available, the system displays the last successful login information (session start and stop time) to SS users at the next successful login.

3.4 Password

All passwords are stored in an encrypted form in IPWorks to ensure the highest level of security. The passwords chosen are subject to constraints as described in the following table:

Table 2 Password Constraints in IPWorks

Password Constraint	Possible/Recommended Values
Password characteristics	<p>The password must:</p> <ul style="list-style-type: none"> • Contain no more than three consecutive digits. • Not include the username. • Have minimum 8 characters (default) and maximum 63 characters. • Contain at least three character classes out of the four⁽¹⁾
Force password change on first logon	Recommended value is <code>true</code>
Force password change on password reset	Recommended value is <code>true</code>
Password should not match N previous passwords	N is always 3 and it cannot be changed.
Maximum login attempts before locking password	Default value is 3. This can be changed using <code>ipworks_ss.conf</code>
Password expiration period	The expiration period must be set to more than 30 days. The default value is 45. This is configured through Control Panel.
Password Encryption	MD5 is used as the encryption algorithm for user password encryption. It is implemented by the system automatically.

(1) The four character classes are digit, lower-case letters, upper-case letters, and special characters. The special characters are defined for the following 32 characters:

~!@#\$%^&*()_-=+{[]}\|:;'"<, > . ? /





4 Storage Server

This section describes the concepts related to Storage Server.

4.1 Basic Concepts

The Storage Server stores **objects**. Each object has a **class** that defines basic structural information about the object. Classes can be defined as extensions of other classes (in other words, there is an inheritance hierarchy of class definitions). The class defines a set of **fields** that may have values for the object. Some fields have only one value, some have multiple values (that may or may not be ordered).

Each class has a field, or set of fields, whose values can be used to identify an object uniquely in that class. This unique identifier is known as the **key value**. In some classes the key consists of several fields, while in other it is a single field.

Each class also defines a set of **relationships** that exist between objects of that class and objects in other classes. Relationships are established by having one or more common values in fields between objects. Often the common values are keys. Relationships can be used to navigate among the stored objects. A relationship can link an object with one other object, or with many other objects. This depends on the context and meaning of the relationship.

Each class also defines **operations** that can be performed on objects in that class. An operation represents a special action that can be invoked upon an object.

The Storage Server also manages **enumerations**. These are known sets of values or objects that are assigned a name to simplify their retrieval. The current set of values or objects, or both, are defined as a static collection, or as search criteria to be dynamically computed if necessary.

4.2 Fields

A storage **field** defines a field (or attribute) that can have a value for a stored object. Following information can be defined for a field:

- **Field Names:** For example: display name, database name, and aliases.
- **Field Type:** For example: Single Valued, Multi Valued (ordered) and Multi Valued (unordered).
- **Plugin Methods:** These can be called to customize the field behavior. For example, methods is defined to invoke when the values are read from the

field and when the values such as the data type are changed. This defines the semantics of the values that the field can have.

- **Enumeration:** The values of the field can be restricted by associating it with an enumeration.
- **Flags:** Define special characteristics for the field, for example: Read only.

4.3 Classes

A storage **class** defines the structure of a stored object. This includes the following:

- The fields that can have values for the object.
- How the object is to be stored in the repository.
- Which fields are used as keys for the object.
- The relationships that apply to the object.

A storage class is similar to both a Java class and an LDAP object class, but does not map directly into either one. Each stored object is an instance of only one storage class.

4.4 Relationships

A storage **relationship** is a link between stored objects. Relationships are 'relative' to a 'source' object. The relationship can be followed from the source object to the destination objects linked by that relationship.

The following relationship is one way of navigating through the stored objects. Some of the information that can be defined for a relationship are as follows:

- Search criteria to define how to find related objects.
- Fields that are used to link related object.
- Plug-in methods that can be called to customize the default behaviors.
- Flags to define special characteristics of a relationship.
- Automatic creation and deletion of related objects.

4.5 Operations

An **operation** represents a custom action that can be performed. The Storage Server allows the operations to be defined so that they act on a specific object, or they can be defined as global operations.



Operations can also have parameters. These are extra pieces of information (sometimes optional) that can be specified when the operation is invoked to specify how the action is performed.

4.6 Enumerations

A storage **enumeration** is a named set of values or stored objects. The named set can be defined by either of the following:

- List a predefined static set of values explicitly.
- Specify search criteria that are applied dynamically to determine that current values or objects to be included in the named set.

Named sets of objects are always dynamic, while named sets of values can be either static or dynamic. An enumeration that represents a set of objects can be used in a context where a set of values is required. In this situation, the key values for each object are used to form the set of values.

When the enumeration is defined, it is often defined as a current search to be performed, so that the current set of objects or values may change, except the name of the enumeration. This is one of the primary mechanisms for navigating through the stored objects.

4.7 Finding Objects

To perform operations on an object (or set of objects), first find the objects that the user is interested in. Use one of the following methods:

- **Finding objects by key values:** If user knows the class and keys of object, then user can retrieve the object directly.
- **Finding objects in an enumeration:** If there is an enumeration that defines the set of objects, then the user can retrieve the objects using that enumeration.
- **Finding objects through relationships:** If user uses either of the preceding methods to obtain some objects, then user can retrieve more objects by traversing the known relationships.

All searches can be customized, so that only a subset of the values is retrieved or to customize the sorting order within the returned results.

4.8 Creating Objects

To create an object, the IPWorks starts a transaction and puts the object in the transaction. A transaction guarantees consistency of the object. If creation of

an object also requires creation of other objects, the creation of other objects is added to the same transaction as well.

An object can be in only one transaction at a time. The Storage Server uses locking capabilities to ensure this. Keep transactions as small and as short as possible.

4.9 Modifying Objects

To change an object, the user must first place the object in a transaction. A transaction represents a set of changes to one or more objects that must all be done together to guarantee consistency. If a change to an object also requires changes to be made to other objects, the user needs to add them to the same transaction.

4.10 Deletion of an Object

To delete an object, IPWorks starts a transaction and put the object in the transaction. Similar to modifying objects, a transaction guarantees consistency of the object. If a deletion of an object also requires deletion of other objects, the deletion of other objects is added to the same transaction as well.

4.11 Making Decisions

There are situations where the Storage Server must choose between two (or more) behaviors - someone must decide what to be done. The Storage Server provides a flexible algorithm for decisions as follows:

- Check whether there is a property in the configuration file of server that defines what to do. If so, this represents an administrator decision about what to do.
- When there is a transaction associated with the operation, check whether the client has specified a **DecisionMaker**. This is a client application object that interacts with the user, or checks a client-side properties file to decide what to do.
- Check whether the client session has a **DecisionMaker** that defines what to do.
- Check whether the value can be determined from the property defaults of the server. . It is another configuration file that the administrator can edit to provide default settings for properties.
- Use a hard-coded default for the decision.

This algorithm provides flexibility to both client applications and administrators in how decisions get made.



5 DNS Management

This section provides concepts that are required for mapping DNS.

Table 3 DNS Objects and Object Field

Object		Reference
	DnsServer	See Section 5.1 on page 17
	View	See Section 5.2 on page 18
	Zones	See Section 5.3 on page 18
	DNS Resource Records	See Section 5.4 on page 21
	ACL	See Section 5.5 on page 23
	TSIGKeys	See Section 5.6 on page 24
Object Field		
	Options	See Section 5.7 on page 24

For information on how to configure IPWorks DNS, refer to *Configure DNS and ENUM*.

5.1 DnsServer

A *DnsServer* object represents a DNS server that is running on the network.

For more information about the object, refer to the *DnsServer* section in *IPWorks DNS, ASDNS, ENUM Parameter Description*.

5.1.1 Auto Register

If an IPWorks DNS server is installed on a system in the network, it can be configured to *auto-register* when the Server Manager is started. In the process of doing this, it creates a *DnsServer* object in the database to correspond to the active server. It is defined with the IP addresses and domain names for that system. With this feature, it is unnecessary to create the object manually. Review the automatically created object to ensure that it reflects the configuration that is required for the DNS server.

5.1.2 DNS Server IPv6 Support

IPWorks DNS server supports queries on both IPv4 and IPv6 transport as specified by *listen-on* option and *listen-on-v6* option. When creating a *DnsServer* object, it is configured with IPv4 support by default. If IPv6 supported is needed, the option *listen-on-v6* must be specified. When



the option is used to configure IPWorks DNS server in order for it to listen for incoming queries using different version of Internet Protocol, the option is configured as follows:

- Queries using IPv4: `listen-on`
- Queries using IPv6: `listen-on-v6`
- Queries using both IPv4 and IPv6: both `listen-on` and `listen-on-v6`

For more information about how to configure IPv6 for DNS, refer to the Section *DNS Server IPv6 Support* in *Configure DNS and ENUM*.

5.2 View

A *View* object is a DNS management concept used to present a name-server configuration to a community of hosts and a different configuration to another community, based on the IP addresses of the hosts. Views permit the support of multiple (possibly conflicting) definitions of zones within a DNS server. Many DNS configurations do not need to use any views. The reason for using views in the DNS configuration is to implement a *split namespace* (having a zone with different internal and external definitions) within a single DNS server.

Within the configuration of a DNS server, all the zones are considered to be within a view. When a DNS server is created, a default view is automatically created for each server. All zones in that server that are not added to a view are automatically associated with the default view.

For more information about the object, refer to the *View* section in *IPWorks DNS, ASDNS, ENUM Parameter Description*.

Default View

In addition to views that are created, every DNS server also has a default view. When the zones are created, they are placed in the default view if a view is not specified. When a view is created, the default area for zones that are in the view must be specified. This is the area that is used as the source for resource records for zones in that view if the zone does not specify an area.

5.3 Zones

A Zone is an area of authority within the DNS name space. It is important to understand what zones are managing the DNS. For details on DNS, refer to *DNS and BIND 4th Edition*.

Every zone in the DNS has a server that is designated as the *primary* or *master* authority for that zone. In addition, zones can have other DNS servers that are designated as *secondary* or *slave* authorities for the zone. These are other servers that are considered as authoritative sources for the zone, but they obtain



their authoritative data from the master server for that zone. The slave servers provide redundancy in the network in case the master server goes down.

The most important aspects in configuring the DNS within the network are planning the following details:

- Where the servers are run?
- Which servers are masters?
- Which servers are slaves for each zone?

In IPWorks, zones are not created explicitly. Instead, declarations are created within each server for the *role* that the server has for the zone. For example, if a DNS server is a *master* for a zone, then this is indicated by creating a *MasterZone* object within that DNS server.

IPWorks DNS has the following types of zones (see Table 4):

Table 4 Zones

Zone	Description
<i>MasterZone</i>	A <i>MasterZone</i> is an object defined within a server which is considered as the master (or primary) authority for the zone.
<i>SlaveZone</i>	A <i>SlaveZone</i> is a replica of a <i>MasterZone</i> . The master list specifies one or more IP addresses of master servers that the slave contacts to update its copy of the zone.
Dynamic Zone	<p>A dynamic zone is a zone that has dynamic updates (DDNS) enabled.</p> <p>The resource records are stored in the DNS server (or more accurately — in a set of files on the DNS server system) and managed directly by the DNS server. The DNS server is the ultimate authority on the data that is defined in that zone.</p> <p>Dynamic zone is introduced because of an issue that comes up when dynamic DNS (DDNS) updates are used in a network.⁽²⁾</p> <p>Issue: When a DDNS update is received by the DNS server, it does not immediately update the zone file for that zone. Rather, the changes to the zone are stored in an <i>intermediate</i> state and the zone file is updated later. This process is highly optimized to minimize the performance impact that the DDNS updates can have on the server. As a rule, once the server begins to manage the zone using this process, the zone file cannot be changed by any external process or user, except for the DNS server. Otherwise, it can interfere the processing of that zone and result in a possible loss of data.</p> <p>This implicates that if DNS data is to be managed manually, then ensure that the data is in a separate zone from any dynamic zone. In this way, data cannot lose.</p>
Static Zone	<p>Zones that do not have DDNS enabled are called static zones.</p> <p>The resource records are stored in the central IPWorks database. When the changes are made, the changed records are transferred to the DNS server system during the <i>server update</i> process. The central database is the ultimate authority on the data that is defined in a zone.⁽³⁾</p> <p>In many ways IPWorks hides the differences between the dynamic zones and static zones, a few minor differences are available though:</p> <ul style="list-style-type: none"> • If the dynamic zones are used, then the data is stored on the DNS server. • If the user wants to keep a backup of the data, then procedures of backing up the appropriate files on the server system must be established. • If the central database is backed up, then it is not necessary for static zones.



Zone	Description
<i>FixedZone</i>	<p>A <code>FixedZone</code> object represents a special zone whose contents are not directly managed by IPWorks. A fixed zone can be associated with more than one server, or with more than one view within a server.</p> <p>The contents of a fixed zone are the same for all servers. Since the contents never change, only a few special zones that can be correctly configured across multiple servers. These special zones are hint zones, zones for the loopback address and localhost domain name (<code>HintZone</code>, <code>LocalhostZone</code>, and <code>LoopbackZone</code>), and possibly some SDB-based zones.</p> <p>A difference between a fixed zone and a regular zone is how IPWorks manages the contents of the zone file. Unlike a regular zone, where the user is forced to work with individual resource records. With a fixed zone, the entire contents of the zone file is edited as a single text string. In this case, this behavior provides more control over the zone file as a whole.</p> <p>Fixed zones with the standard definitions are included in IPWorks. However, to edit the contents, the user must understand the syntax of a zone file.</p>
<i>HintZone</i>	<p>A <code>HintZone</code> object defines the location of the root DNS servers. These root servers are used to resolve queries for zones that are not included in the servers configuration. IPWorks includes the standard definition of a hint zone for the internet root servers (called <code>Internet</code>). This is an example as it is not needed. The IPWorks DNS server has built-in support for the standard internet root servers that are used if there is no hint zone. This means that if the DNS server is managing a namespace that is part of the standard internet namespace, a hint zone is not needed.</p>
<i>LocalhostZone</i>	<p>An <code>LocalhostZone</code> object defines the standard forward lookup interface for the localhost name (normally 127.0.0.1 for IPv4). IPWorks includes a standard definition for this zone (called <code>localhost</code>) for both IPv4 and IPv6.</p>
<i>LoopbackZone</i>	<p>A <code>LoopbackZone</code> object defines the reverse lookup interfaces for the loopback address. IPWorks includes standard definitions for both IPv4 (called <code>loopback</code>) and IPv6 (called <code>v6loopback</code>).</p>
<i>ForwardZone</i>	<p>A <code>ForwardZone</code> is a special zone declaration that is used to configure domain-specific forwarding (for unresolved queries). Although the forward zone is created within the context of a DNS server (and view), the zone is not considered to be managed by that server. Instead, it is a portion of the namespace that forwards the requests to other DNS servers. The default forwarding can be configured at the server or view level. In fact, there is no requirement that the name specified in a forward zone must correspond to a zone in some other server. If no forwarders statement is present or an empty list for forwarders is given, no forwarding is done for the domain. Therefore canceling the effects of any forwarders is defined for the server.</p>
Root Zone	<p>A root zone represents the configuration of the <code>.</code> zone that is considered to be the root of the namespace. It can be configured the same way as any other zone. Both <code>MasterZone</code> and <code>SlaveZone</code> can be created for the root zone.</p> <p>A server that includes the root zone in its configuration is frequently referred to as a root server. However, there is nothing special about the configuration of either the server, or the zone itself.</p>
<i>StubZone</i>	<p>An <code>StubZone</code> object that is similar to a <code>SlaveZone</code>, except that it replicates only the NS records and SOA records of a master zone instead of the entire zone. Stub Zones are not a standard part of the DNS; they are a feature specific to the BIND implementation.</p>



Zone	Description
Zone with Hidden Masters	<p>The DNS protocol allows a zone to be configured so that some of the authoritative servers for that zone are not advertised in DNS. Doing this minimizes the number of queries that are directed to that server, because the only clients that direct queries to that server are those which are specifically configured to do so.</p> <p>Although this technique can be applied to any authoritative server, it is most commonly applied to the master (primary) server for a zone. This allows the master server to focus on zone transfers, processing dynamic updates, and so on, while the advertised slave servers process queries from resolver. This can be an effective configuration for zones that are large and are frequently queried. When the master server is configured for a zone, it is called a hidden master or a hidden primary.</p> <p>IPWorks supports the configuration of hidden masters. There are two levels of <i>hiding</i> that can be applied to the zone. In a normal zone, the master server is advertised in both the SOARecord and in two NSRecords (one in the zone itself and one in the parent zone to delegate authority for the zone). To configure a hidden master, the most important advertisements to delete are the two NSRecords, because those records are used for the delegation and for redirecting queries during the name resolution process. The servers identity can be hidden in the SOARecord, but this is not important.</p>
Zone with Multiple Masters	<p>The DNS protocol allows a zone to be configured so that multiple servers act as masters for that zone. This is not a typical configuration:</p> <ul style="list-style-type: none"> • If the servers are not configured using the same data, the configuration can result in inconsistent query results. • If a shared database is available behind the servers, this configuration can be done without any problem and in some situations can produce better overall DNS performance because the number of zone transfers is minimized. <p>IPWorks supports the configuration of zones with multiple masters (for static zones only). This can be done without inconsistencies because the central IPWorks database can be used to ensure that the zone definitions are consistent in all the servers. It is important to take precaution when working with zones that are configured in this way, but it is possible to configure a zone with multiple masters if desired.</p>

(1) If the DNS Server is down, or is unavailable, then the records in that zone are also unavailable (not just on the network, but also unavailable in IPWorks).

(2) If DDNS is not used, then there is no need for dynamic zones.

(3) If data is changed on the DNS server system, but not in the central database, it will be overwritten the next time the server is updated. When a MasterZone is created, it is configured as a static zone by default.

For more information about the parameters of zones, refer to the *Zones* section in *IPWorks DNS, ASDNS, ENUM Parameter Description*.

5.4 DNS Resource Records

Resource records are the basic elements used in DNS to represent data to be distributed by the DNS protocol. It is good to consult an external reference if not familiar with the concepts. Many types of resource records are available in the DNS. Some of them are more important to the overall DNS configuration than others, so IPWorks provides extra support, data-checking, and custom behaviors for some of these resource records to ensure that the overall DNS configuration is correct.

In the DNS protocol, resource records are organized into zones based on the *dnsname* that they are associated with. This *dnsname* is also called the *owner dnsname* and must be specified for all resource records.



Another important field on all resource records is the *container*. All resource records in IPWorks are either *contained* in a specific zone, or they are contained in an *area*. Resource records that are contained in an area can be *shared* in multiple zones. The value of the *container* field must be set as follows:

- If the resource record is contained in a specific zone, the *container* field must be set to the *Zone* for the MasterZone that contains the record. Only three types of records are to be defined in this way: SOARecords, NSRecords, and ARecords that are needed as *glue* for the NS and SOARecords.
- If the resource record is contained in an area, the *container* field must be set to the name of the area. The record is then compared with all the zones that are associated with the area to determine (based on the owner dnsname) which zones include this resource record.
- If the container field is not specified, IPWorks checks all known MasterZones to determine which zone (or zones) can include the record. The record is then automatically added to the area associated with these MasterZones if only one possible area is available. If multiple possible areas are available, then there is a prompt to identify the area that can include the resource record.

Table 5 lists the types of records that have customized support.

Table 5 DNS Resource Records

Resource Record	Description
<i>ARecord</i>	An <i>ARecord</i> object is used in the DNS to define an address binding for a dnsname. For a given dnsname, it is possible to have multiple <i>ARecords</i> if the dnsname in question has multiple address bindings.
<i>AAAARecord</i>	An <i>AAAARecord</i> object is used to define an IPv6 address binding for a dnsname.
<i>CNAMERecord</i>	<p>A <i>CNAMERecord</i> object is used in the reference on DNS to define a <i>canonical name</i> for a dnsname. In other words this allows the definitions of aliases for a given dnsname. A number of restrictions about the use of <i>CNAMERecord</i> objects (notably that a DNS name is considered as an alias are available and must not have any other resource records).</p> <p>The <i>CNAMERecord</i> objects can be used to provide an alternative configuration for the APNs in the network. The limitations associated with the <i>CNAMERecord</i> objects sometimes make this approach impractical for some APNs.</p>
<i>HINFOrecord</i>	An <i>HINFOrecord</i> object is used to acquire general information about a host.
<i>MXRecord</i>	An <i>MXRecord</i> object defines the "mail exchange" (a system that handles mail) for the specified domain. The use of MX resource record is explained in detail in RFC 974. Each MX matches a domain name with two pieces of data, a preference value (an unsigned 16-bit integer), and the name of a host. The preference number is used to indicate in what order the mailer should attempt to deliver to the MX hosts with the lowest numbered MX being first. Multiple MXs with the same preference are permitted and have the same priority.
<i>NAPTRRecord</i>	A <i>NAPTRRecord</i> is a resource record that defines a Naming Authority Pointer for a DNS name, as defined in RFC 2915.



Resource Record	Description
<i>NSRecord</i>	<p>The NS resource records for a DNS name, as specified in RFC 1035. A <i>NSRecord</i> object states that the named host is expected to be an authoritative server for the zone starting at the specified 'owner' DNS name.</p> <p>The <i>NSRecord</i> objects are used within DNS to define the relationships between different servers for each of the zones that they manage. These records do not need to be managed as they are automatically created and defined for the zones when the <i>MasterZone</i> objects are created. Although not necessary, these records can be manually overridden or created.</p>
<i>SOARecord</i>	<p>An <i>SOARecord</i> object is used within DNS to define the relationship between different server for each of the zone that it manages. This record does not need to be managed as it is automatically created and defined for the zone when the <i>MasterZone</i> object is created. Although not necessary, this record can be manually overridden or created.</p>
<i>PTRRecord</i>	<p>The PTR resource records for a DNS name. The <i>PTRRecord</i> objects are used in special domains (such as the INADDR.ARPA domain) to point to some other location in the domain space.</p>
<i>SRVRecord</i>	<p>An <i>SRVRecord</i> object defines the owner name, where the owner name has the format. <code><service>.<pronto>.Name</code>.</p>
<i>GenRecord</i>	<p>The <i>GenRecord</i> object is used to define resource records that are generated in a zone when it is loaded by the DNS server. The generated records are not managed by IPWorks, but rather they are created dynamically in the DNS server.</p> <p>This feature can be used to generate the PTR Records (as well as A, AAAA, CNAME, DNAME, and NS records), although it is only useful when there is some degree of regularity in the domain name assignments. This feature is based on the \$GENERATE macro supported in BIND 9. To enable this feature, the <i>GenRecord</i> object must be created.</p> <p>With <i>GenRecords</i>, the generated resource records only exist in the DNS server. They do not exist in the IPWorks database.</p>
<i>IncludeRecord</i>	<p>IPWorks supports the \$INCLUDE directive. \$INCLUDE allows inclusion in-situ of an external file containing extra directives. It is typically used to assist in maintenance of larger DNS files, for example, the user can distribute the maintenance of individual zone files to clients without exposing the global parameters or other client zones to inspection. This functionality is based on the \$INCLUDE directive supported in BIND 9 and only supports masterzone in IPWorks.</p> <p>To use this functionality, the <i>IncludeRecord</i> and <i>IncludeFile</i> objects must be created. <i>IncludeRecord</i> is used to define which masterzone uses \$INCLUDE directive; <i>IncludeFile</i> is used to define which file in local disk (persistence storage device, such as HD, CD, or SD) is used for included file. Each <i>IncludeFile</i> describes the content of included file and the alias of the file. The relation key between <i>IncludeRecord</i> and <i>IncludeFile</i> is <i>FileAlias</i>. This field in <i>IncludeRecord</i> represents which file masterzone uses.</p> <p>With <i>IncludeRecords</i>, the generated resource records only exist in the DNS server, which means, they do not exist in the IPWorks database.</p>
<i>IncludeFile</i>	<p>An <i>IncludeFile</i> object is used to define which file in disk is used for included file. IPWorks supports the following Resource Records (RR) and directive:</p> <ul style="list-style-type: none"> • RR: A, AAAA, CNAME, HINFO, MX, NAPTR, NS, PTR, SRV • Directive: \$GENERATE <p>Make sure that the correctness of the included file. Each <i>IncludeFile</i> describes the content of included file and the alias of the file.</p>

(1) Although a *DnsServer* has fields for both its *dnsname* and *Address*, IPWorks does not automatically create *ARecords* or *PTRRecords* for the server. If the *ARecords* or *PTRRecords* are wanted for the *DnsServer*, the user needs to create them manually.

For more information about the parameters of the DNS resource records, refer to the *DNS Resource Records* section in *IPWorks DNS, ASDNS, ENUM Parameter Description*.



5.5 ACL

An *ACL* is a named access control list, or address match list that is used to specify sets of incoming clients in the configuration options that are used for controlling the server security.

A named ACL in DNS can be used to associate a name with a list of *address-matching* criteria. A number of options in DNS that have values that are *address-match-lists*. These lists are used to specify criteria to compare with incoming client requests to determine how they are handled. In many instances, the same *address-match-list* is used in multiple places. Named ACLs provide a mechanism that allows the definition of the list in one place and then use a name to refer to that list. This simplifies the management of these lists because it is assured that the list is propagated correctly without having to review every option in the configuration.

In IPWorks, the namespace for ACLs is shared among all the DNS servers. Hence, when an ACL is defined it can be used anywhere — even in different servers.

For more information about the object, refer to the *ACL* section in *IPWorks DNS*, *ASDNS*, *ENUM Parameter Description*.

5.6 TSIGKeys

TSIGKeys are used in the DNS for securing or authenticating packets, or both. A TSIGKey is shared by the DNS client and the DNS server. When a DNS packet is sent to the server, the name of the key and a digital signature are included in the packet. The server checks the signature for authenticity and then performs the appropriate action.

In the configuration of DNS these keys can be used in address match lists (discussed earlier). This means they can be used for identifying certain clients and for controlling access to any functions that are managed by address match lists. In IPWorks, TSIGKeys can be used for the following items:

- Securing dynamic updates sent from a DHCP server to the DNS server.
- Securing the communications between two DNS servers.
- Securing the communications between a DNS server and an ActiveSelect Monitor.
- Securing the communications between DIG, nsupdate, and a DNS server.

For more information about the object, refer to the *TSIGKeys* section in *IPWorks DNS*, *ASDNS*, *ENUM Parameter Description*.



5.7 Options

All the DNS management concepts described here (DnsServers, Views, Zones) can be further customized by specifying configuration options. This is done by adding values on a special field on the objects.

Each of these objects has a field called *Option*. This field can have multiple values and each value is used to specify a single configuration option and the setting for that option. The syntax of the value is as follows:

```
<option-name> <value 1> <value 2> ...
```

Many configuration options can be specified. Some options can only be applied to certain types of objects. Many of these options have complicated values when they are specified. It is important to understand both, the meaning of the option and the syntax that must be used to specify values for that option.

Note: When the user tries to define a value of data type `AddrList` as some ipnet values, IPWorks supports an abbreviated notation so that the zero at the end of the subnet address can be abbreviated. For example, subnet `192.168.0.0/16` could be written as `192.168/16`.

If unsure how to define a configuration option for an object, use this command to review the option before setting it. IPWorks checks the syntax of any values that has been set for configuration options.

5.7.1 DNS Options

The user can use the command `show` in CLI to get more information about these options. For details, refer to the *Show dns Option Commands* section in *Command Line Interface User Guide for IPWorks SS*.

For more information about all the BIND 9 DNS options that are supported by the IPWorks DNS server, see Section 15.1 on page 69.





6 ActiveSelect DNS Management

ActiveSelect DNS (ASDNS) is an IPWorks-specific feature that allows redundancy to be defined in the network. It also allows better performance of complex load balancing than the DNS protocol.

ActiveSelect DNS is an extension to the IPWorks DNS server. This extension makes DNS more dynamic when responding to queries. The IPWorks DNS server with ASDNS uses information sent to it from ASDNS Monitors so that it can make more intelligent decisions about what information to include in a response.

Use of ASDNS is optional. ASDNS can be used with standard DNS servers, though query to these standard DNS servers is not benefit from the ASDNS feature.

For information about the ASDNS managed objects about provisioning management, refer to the *ActiveSelect DNS Objects* section in *IPWorks DNS, ASDNS, ENUM Parameter Description*.

For information on how to configure ASDNS, refer to *Configure DNS and ENUM*.

6.1 ActiveSelect Application Scenarios

ActiveSelect DNS is useful in the following cases:

- Only addresses that are reachable are returned for queries of domain names.
- Addresses that are close to each other are returned for queries of domain names.
- Addresses that are close to the source of the query are returned.
- Load balancing of resources is required.

ActiveSelect DNS is used in situations where there are multiple systems, either in a single location or multiple locations, that can provide similar services. An example is a group of web servers at two locations where each provides the same pages. In this example, ActiveSelect DNS can be used to direct traffic to a web server that is up. The addresses of the servers that are down are not included in the answers to DNS queries and the addresses of the servers that are up can be ordered based on the source of the DNS query.

Following figure shows the topology diagram of ASDNS.

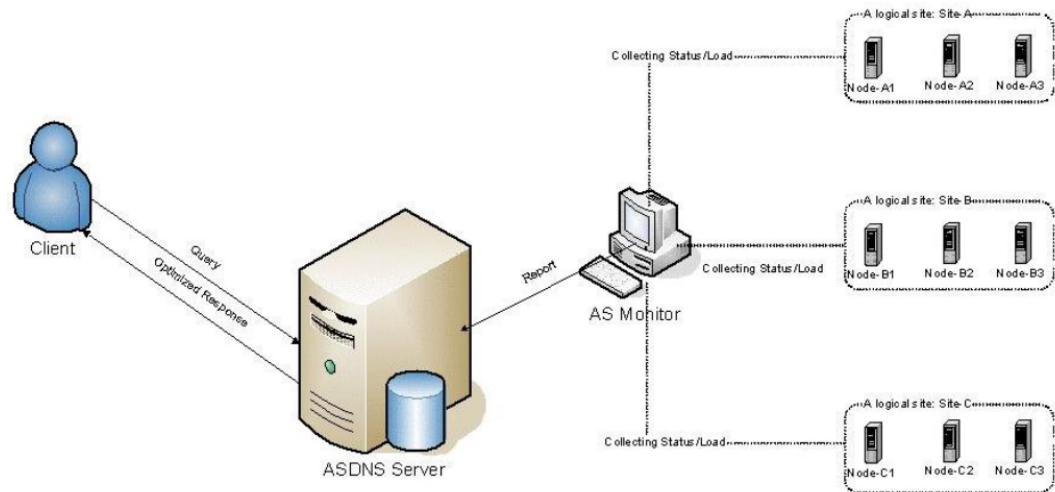


Figure 1 ASDNS Topology Diagram

6.2 ActiveSelect Functions

ActiveSelect DNS has the following four functions:

- **Monitoring** is a mechanism for an ActiveSelect DNS server to receive information about entries in its database. The ActiveSelect DNS Monitor is independent on the DNS server. One or more monitors can be deployed almost anywhere in the network. The monitor uses an external script to determine if an address (or service at an address) is responding. Information collected from this monitoring activity is sent to the DNS server, where it is stored and used to filter and order responses to future queries. Once set up, the process is automated. Information in the DNS zone files is unchanged.
- **Filtering** is done at the DNS server using information sent by ActiveSelect DNS Monitors. Addresses for systems that are down are deleted from the responses for ActiveSelect DNS enabled names.
- **Grouping** is done at the DNS server after filtering. Addresses can be grouped based on, for example, location or system capacity. Each group can also limit the number of addresses that are included from that group.
- **Ordering** is the final step used by the DNS server and allows the server to structure the answers in a way that is specific to the source of the query, such as providing local addresses before more remote addresses. Ordering can also limit the number of groups whose addresses are included.

Filtering, grouping, and, ordering can be used separately or together to form rules for answering queries.

ActiveSelect DNS does not use zone transfers or incremental zone transfers to communicate monitoring state changes. Instead, all the servers (master and slave) for a zone are configured to receive monitoring information.



6.3 ActiveSelect DNS Components

This section describes the following ActiveSelect DNS components:

- ActiveSelect DNS Enabled DNS Server
- ActiveSelect DNS Monitor

6.3.1 ActiveSelect DNS Enabled DNS Server

The ActiveSelect DNS enabled DNS server provides the following functions:

- Receiving monitoring reports from ActiveSelect DNS Monitors. The last reported status on each address is stored and used when queries for ActiveSelect DNS enabled DNS names are received. In general, when ActiveSelect DNS is being used, both the master and slave servers for the ActiveSelect DNS enabled DNS names must receive monitoring reports.
- Responding to queries for ActiveSelect DNS enabled DNS names when the answer includes A or AAAA resource records by first filtering and ordering the addresses. See the following steps for the responding process:
 - 1 Obtain all the addresses for the DNSName.
 - 2 Delete the addresses that are not reported as `up` (based on the stored monitoring reports).
 - 3 Sort the remaining addresses based on the configured sites for that DNSName. For each site, the addresses are ordered in a BIND round-robin mechanism in a fashion starting at a random point. This randomizes the order of the `up` addresses.
 - 4 Based on the site rules (which can depend on the query's source address), place the addresses from each site on the list of addresses to return. Enforce any limit placed on the site for the number of addresses from that site to include. Enforce the limit on the total number of sites (with at least one `up` address) to include.
 - 5 If all the addresses associated with the sites in a preferred list are reported as `down`, alternative actions occur. One of three optional keywords (`site`, `none`, or `default`) might be specified at the end of a preferred statement to specify what behavior is followed in this case. The default behavior is followed if the keyword is omitted.
 - When all addresses in sites listed in the preferred statement are down and if the keyword `site` is specified, ActiveSelect DNS returns all addresses in sites listed in the preferred statement.
 - When all addresses in sites listed in the preferred statement are down and if the keyword `none` is specified, ActiveSelect DNS returns no addresses.



- When all addresses in sites listed in the preferred statement are down and if the keyword `default` is specified, ActiveSelect DNS returns all addresses associated with the name.

This can be explained with the following example:

```
resource "Melbourne_GGSN"
{
  dnsname "ggsn1.mnc001.mcc001.gprs";
  key "a_dns_key";
  site "Melbourne" {10.1.0.1/24;};
  site "Stockholm" {10.2.0.1/24;};
    site "NewYork" {10.3.0.1/24;};
  //SGSNs in the 10.1.0.0 net are directed to Melbourne
  10.1.0.0/16 prefer "Melbourne" "NewYork" "Stockholm";
  //SGSNs in the 10.2.0.0 net are directed to NewYork
  10.2.0.0/16 prefer "NewYork" "Stockholm" "Melbourne";
  //SGSNs in the 10.3.0.0 net are directed to Stockholm
  10.3.0.0/16 prefer "Stockholm" "NewYork" "Melbourne";
  //Default to NewYork
  prefer "NewYork" "Stockholm" "Melbourne";
}
```

Return statement

Syntax:

```
return <integer>;
```

Description:

The return statement limits the number of IP addresses returned in a query. By default (no return statement) addresses that are available in all sites are returned for query on the domain name (unless a prefer statement is specified). If a return *n* is specified within a site statement, the number of addresses returned for that site is limited to *n*.

Example:

```
resource "Internet_apn
dnsname " internet.mnc001.mcc001.gprs ";
key "a_dns_key";signature "a_dns_key";
site "Melbourne"
{
  return 2; // return 2 IPs from this site
  10.1.0.1;
  10.1.0.2;
  10.1.0.3;
};
site " NewYork "
{
  return 3; // return 3 IPs from this site
```



```

10.2.0.1;
10.2.0.2;
10.2.0.3;
10.2.0.4;
};
site " Stockholm "
{ //By default all IPs from this site
10.3.0.1;
10.3.0.2;
10.3.0.3;
};
};

```

- BIND round-robin is applied to any addresses returned to distribute the load. (This is not the case if Active Select is implemented).

Note: Repeated queries for a single ActiveSelect DNS enabled dnsname do not always return the exact list of addresses. Addresses that are reported up are returned and as restrictions can be placed on the number of addresses in a site that can be returned, even the addresses in the returned lists might change (not just the order) with each query.

ActiveSelect DNS also supports load balancing using load information reported by the systems, not just whether each system is up or down. In this case, the loading information is used to statistically order the list of returned addresses to place those addresses (and sites) that are least loaded first more often. This avoids redirecting all traffic to the least loaded system or site, which would quickly overload that one system or site.

When ActiveSelect DNS is being used, both the master and slave servers for the ActiveSelect DNS enabled DNS names should receive monitoring reports and should be configured for ActiveSelect DNS operation.

6.3.2 ActiveSelect DNS Monitor

The ActiveSelect DNS Monitor is responsible for:

- Monitoring various systems.
- Reporting system status to ActiveSelect DNS enabled DNS servers.

Each monitor:

- Probes a system periodically to determine whether it is responding and then reports that information, through a DNS message, to one or more ActiveSelect DNS enabled DNS servers.
- Can probe multiple systems and report the status of each monitored system to multiple servers.

- Can probe and report not just whether a system is up or down, but also loading information.

While an ActiveSelect DNS Monitor can monitor any system and report to all DNS servers, consider where to place these monitors and how many to use in monitoring a system. Particularly in cases such as:

- The monitor needs to communicate with the system being monitored. Typically this is done through *ICMP Echo/Echo Reply* messages. This means that the two systems need to be able to send packets to each other. It also means that the network connectivity between these systems affects the up or down reports — not just whether the monitored system itself is up or down.
- The monitor needs to report the status to the DNS servers. This typically means that the monitor must send reports to all the servers (primary and secondary) for a particular zone. And, as with the system being monitored, network connectivity between the monitor and DNS servers affect the ability of the monitor to report status (if a monitor report fails to reach a DNS server, the DNS server eventually assumes that the monitored systems are down). These reports are sent to the DNS server using the DNS protocol (UDP with default port 53) and therefore the network must be configured to allow this traffic between the monitors and servers.
- Multiple monitors can be used to monitor a system. And each of these monitors can report to a subset (or all) of the DNS servers. However, the impact of this on the monitored system must be considered since it receives and needs to respond to more monitor requests. Having multiple monitors reporting to a DNS server means that if one monitor (or the network connectivity between it and the DNS server) fails, the other monitor's reports suffice to provide an accurate view of the status of the monitored systems. But, this increases the processing overhead at the DNS server system.

The ActiveSelect DNS Monitor needs the following configuration information to operate:

- The DNS servers to which to send monitoring reports. This can be a primary DNS server or a secondary DNS server.
- The systems to monitor for each DNS server.
- The method and interval used to monitor each system.
- The TSIGKey needed to communicate with the DNS servers, if any.

Custom scripts (programs) can be written to perform complex monitoring of systems. These scripts return an indication of whether the monitored system is up or down and if up, can also report load information.



6.4 ActiveSelect Managed Objects

Table 6 outlines the ActiveSelect managed objects. For more information about the parameters of the objects, refer to the *ActiveSelect DNS Objects* section in *IPWorks DNS, ASDNS, ENUM Parameter Description*.

Table 6 *ActiveSelect Managed Objects*

Managed Object	Description
<i>ASDNSSite</i>	An <code>ASDNSSite</code> object is used to define a collection of nodes on the network (based on their addresses) that are configured together with ActiveSelect.
<i>ASDNSPolicy</i>	An <code>ASDNSPolicy</code> object defines a rule for how the DNS Server resolves a query for a domain name that has ActiveSelect enabled.
<i>ASDNSRecord</i>	An <code>ASDNSRecord</code> object is used to enable ActiveSelect DNS for a particular DNS name and specify the policy that is used to resolve queries for that dnsname.
<i>Monitor</i>	A <code>Monitor</code> object represents an ActiveSelect monitor that is running on the network.
<i>MonitorMethod</i>	A <code>MonitorMethod</code> object defines information about how a <code>MonitorScript</code> is invoked.
<i>MonitorResource</i>	A <code>MonitorResource</code> object defines a resource on the network that is monitored by an ActiveSelect Monitor.
<i>MonitorScript</i>	A <code>MonitorScript</code> object represents the shell script run by a <code>Monitor</code> when it is trying to determine the status or load or both the information for a resource being monitored.





7 ENUM Management

This section provides concepts that are required for mapping ENUM. It mainly focuses on the managed objects used by the ENUM Server.

For more information on how to configure ENUM service and ENUM related functions, refer to the Section *Configuring ENUM* in *Configure DNS and ENUM*.

7.1 ENUM

ENUM is an application of the DNS protocol used for mapping telephone numbers to Uniform Resource Identifiers (URIs). Specifically, ENUM is an application of the Dynamic Delegation Discovery System (DDDS), which is built on DNS. IPWorks provides a single solution for DNS and ENUM services. It is required to provide many ENUM records with high ENUM transaction rates and low latency. To realize this, the ENUM Server stores and manages its data using MySQL database.

The ENUM Server supports applications, such as IMS Networks. IMS Networks have endpoints with direct IP connectivity to the network through a Session Border Controller (SBC). Endpoints typically use the SIP protocol but some use the H.323 protocol. Users are identified by URIs, but also require an E.164 alias to enable calls from the legacy networks. Users require multiple URIs to identify different endpoint terminals.

The ENUM Server provides standard E.164 number (NAPTRRecord) query function, and also implements a view concept and an access control function based on the client IP address. Unlike the DNS server, the ENUM Server currently does not support full-split namespace and TSIG/DNSSEC functions.

Figure 2 shows the relationship between the ENUM Server and selected IPWorks components. After receiving ENUM and DNS requests from the clients, the ENUM Server handles ENUM-related requests itself and forwards all other requests to the DNS server. Responses from the DNS server are returned without change through the ENUM Server. MySQL database stores the ENUM Server configuration data. The Storage Server replicates the ENUM Server configuration data on the MySQL NDB Clusters. NDB is an in-memory clustered database from MySQL. Each MySQL NDB cluster is independent of the other clusters, that is, they can be managed, maintained, and enhanced separately.

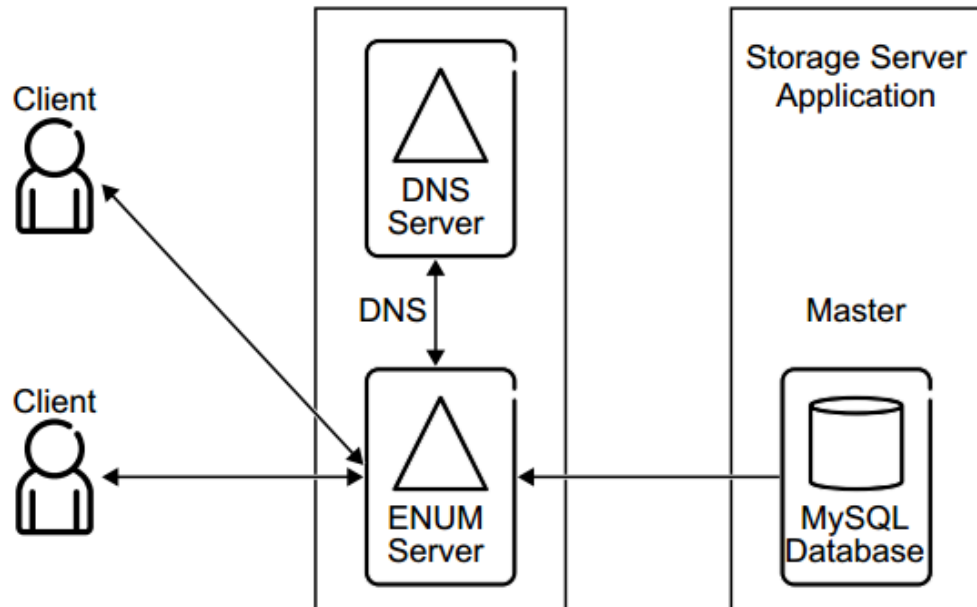


Figure 2 ENUM Relationship with Other Components

7.2 ENUM Managed Objects

Table 7 provides short description of each ENUM managed object in IPWCLI. For more information about the parameters of the objects, refer to the corresponding sections in *IPWorks DNS*, *ASDNS*, *ENUM Parameter Description*.

Table 7 ENUM Managed Objects

Managed Object	Description
<i>EnumServer</i>	An <i>EnumServer</i> object is used to hold ENUM Server configuration data.
<i>EnumZone</i>	An <i>EnumZone</i> object defines the zones that are serviced by the ENUM Server.
<i>EnumAcl</i>	An <i>EnumAcl</i> object stands for the ENUM Access Control List (ACL), which is a set of client source IP addresses allowed or blocked, or both, to access the specific ENUM View(s). An <i>EnumAcl</i> object can be used in different views and servers.
<i>EnumView</i>	An <i>EnumView</i> object provides an access control mechanism for a defined set of ENUM Zones. The <i>EnumView</i> can also be used together with <i>EnumDnRange</i> to implement split namespace.
<i>EnumZVRel</i>	An <i>EnumZVRel</i> object stands for ENUM Zone View Relationship which defines the relationship between an ENUM Zone and an ENUM View.
<i>EnumSOARecord</i>	An <i>EnumSOARecord</i> object is used for DNS Negative Cache.
<i>EnumNSRecord</i>	An <i>EnumNSRecord</i> object is for future use.
<i>EnumDnSched</i>	An <i>EnumDnSched</i> object provides the entry point for routing calls to an e.164 directory number, and also provides NAPTRRecords for endpoints with IP connections to the network.



Managed Object	Description
<i>EnumDnRange</i>	An <i>EnumDnRange</i> object represents a range of directory numbers. This object allows a destination node or a call server to be identified within the range of directory numbers.
<i>DestNode</i>	A <i>DestNode</i> object represents information about Logical Destinations in the IP Telephony network.
<i>AINNode</i>	An <i>AINNode</i> object contains AIN configuration data for the ENUM server. The object is used to communicate with SCP.
<i>AINLNPDData</i>	An <i>AINLNPDData</i> object contains AIN configuration data for each ENUM number series. The object is used in the process of the NP queries and responses.
<i>AINToIFreeData</i>	An <i>AINToIFreeData</i> object contains AIN configuration data for each ENUM number series. The object is used in the process of the TollFree queries and responses.
<i>CountryCode</i>	A <i>CountryCode</i> object contains standard international country codes for transforming an international number to a national number.
<i>INAPData</i>	An <i>INAPData</i> object contains INAP configuration data for each ENUM number series. The object is used in the process of NP queries and responses.
<i>INAPNode</i>	An <i>INAPNode</i> object contains INAP configuration data for the ENUM server. The object is used to communicate with SCP.
<i>MAPData</i>	A <i>MAPData</i> object contains MAP configuration data for each ENUM number series. The object is used in the process of NP queries and responses.
<i>MAPNode</i>	A <i>MAPNode</i> object contains MAP configuration data for the ENUM server. The object is used to communicate with SCP.
<i>EnumOperator</i>	An <i>EnumOperator</i> object is a specific ENUM object. Use the <i>EnumOperator</i> object to hold the operator configuration data.
<i>EnumClientRealm</i>	An <i>EnumClientRealm</i> object distinguishes the original ENUM client as three categories: a local, national, or international ENUM client.
<i>SCCPAddress</i>	An <i>SCCPAddress</i> object holds INAP configuration data to be used in SCCP layer when generating query/response.





8 AAA Management

This section describes the basic concepts to configure IPWorks AAA.

IPWorks AAA uses IPWCLI and ECLI to perform the configuration management.

For more information on:

- AAA Managed Objects in IPWCLI, refer to *IPWorks AAA Parameter Description*.
- AAA Managed Objects in ECLI, refer to the MOC *IPWorksAAARoot* in *Managed Object Model (MOM)*.

8.1 Radius AAA

The IPWorks Radius AAA server provides authentication, authorization and accounting functions for the wired and wireless network access. Authentication verifies the identity of an entity; authorization determines whether a requesting entity is allowed to access a resource; and accounting collects information on resource usage for the purpose of trend analysis, auditing, billing or cost allocation.

3GPP-WLAN Network Architecture

Figure 3 shows the 3GPP-WLAN network architecture. IPWorks Radius AAA supports the Wa, Wm, and D'/Gr' interfaces.

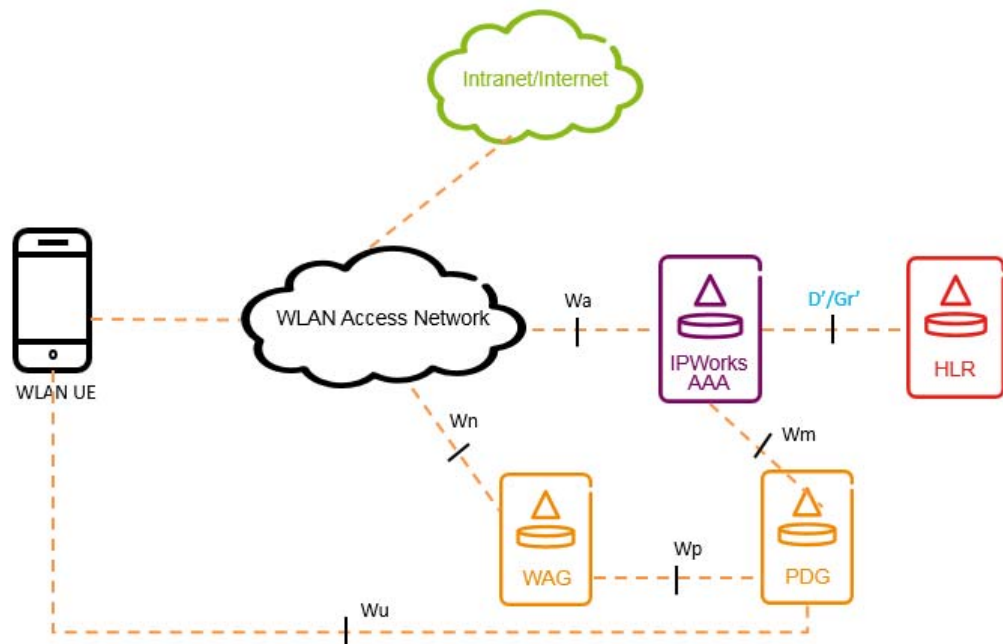


Figure 3 3GPP-WLAN Network Architecture

For more information about the interfaces, refer to the following interface descriptions:

- IPWorks 3GPP AAA Server-WLAN Access Network Wa Interface
- IPWorks 3GPP AAA Server-PDG Wm Interface
- IPWorks 3GPP AAA Server-HLR D'/Gr' Interface

Fixed Access Network Architecture

Radius AAA supports user authentication, authorization, and accounting for fixed access network, as shown in Figure 4.

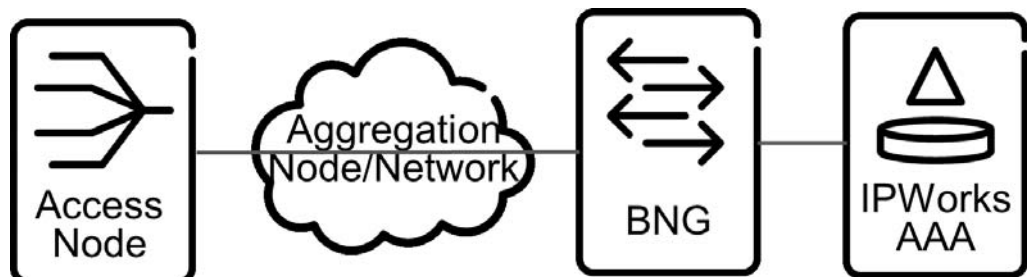


Figure 4 IPWorks AAA in Fixed Access Network

GPRS Network Architecture

Figure 5 shows the network of AAA for GPRS (Gi interface).

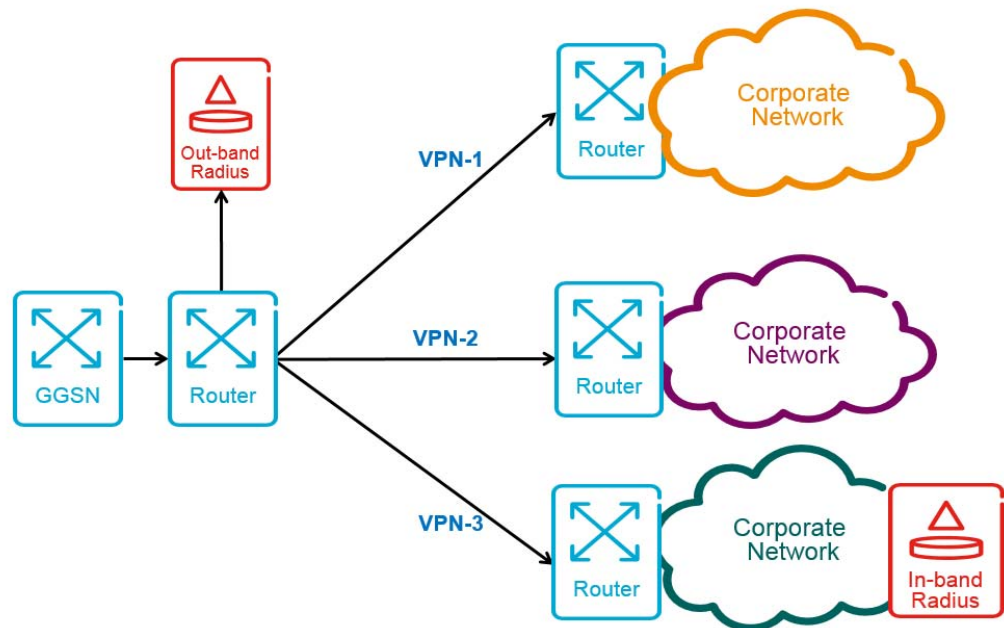


Figure 5 AAA for GPRS (Gi Interface)

For more information about the interface, refer to the following interface description:

- *IPWorks AAA Server-AAA Clients Gi Interface*

Protocol Stack Overview

The relevant protocol stacks shall be configured to achieve IPWorks Radius AAA functions. Figure 6 shows an overview of the protocol stacks used by IPWorks Radius AAA:

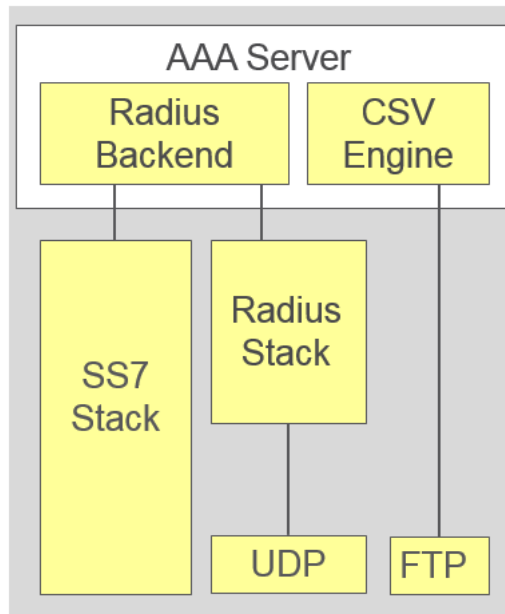


Figure 6 Radius AAA Protocol Stacks Overview

IPWorks Radius AAA configuration is composed of the following parts:

- SS7 Stack configuration.

Certain Radius AAA functions (such as, Wi-Fi AAA if the use case is 2G/3G USIM(SIM) based UE authentication with HLR) require SS7 Stack configuration.

For information on how to configure SS7 Stack, refer to *Configure SS7 for AAA*.

- Radius AAA components, including the following configurations:
 - **Radius Stack:** The module is used to send/receive the radius message
 - **Radius Backend:** The module is used to perform authentication, authorization and accounting functions.
 - **Radius CSV Engine:** The module is used to generate the session based CSV file and provide FTP related function.

The following tools are used to configure Radius AAA:

- ECLI configuration

The Radius AAA function are mainly configured by using ECLI, refer to *Configure Radius AAA*.

- IPWCLI configuration

The AAA server and user provisioning are configured by using IPWCLI.

8.1.1

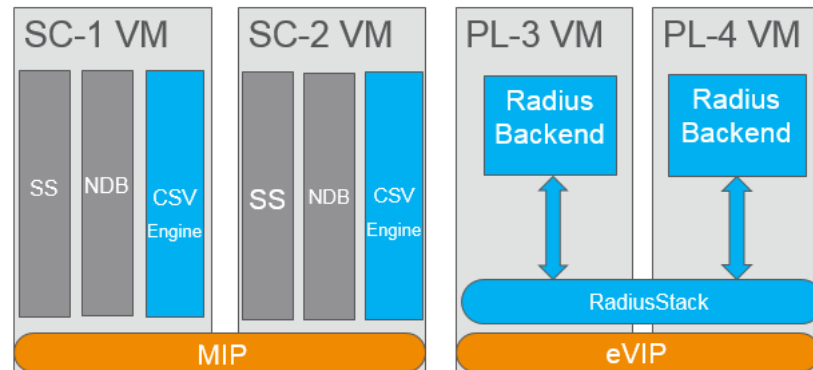


Figure 7 Radius AAA Allocation in IPWorks 2+2 Cluster

Figure 7 shows the typical allocation of Radius AAA in IPWorks 2+2 cluster.

- CSV Engine is integrated with Storage Server.
- CSV Engine is working in active/standby mode.
- Radius Backend and Radius Stack are working on Active/Active mode.
- Radius Stack supports load balancing and failover.

8.2

IPWorks AAA server that is used as the 3GPP AAA server in EPC network is called IPWorks EPC AAA. IPWorks EPC AAA server supports the authentication and authorization of User Equipment (UE) which wants to access EPC through Non-3GPP IP Access Network. It also supports the network-based Mobility Management mechanism over s2a/s2b reference point.

Figure 8 shows the EPC network architecture. IPWorks EPC AAA server supports the STa, S6b, SWm, S13 and SWx interface specified in 3GPP TS 29.273.

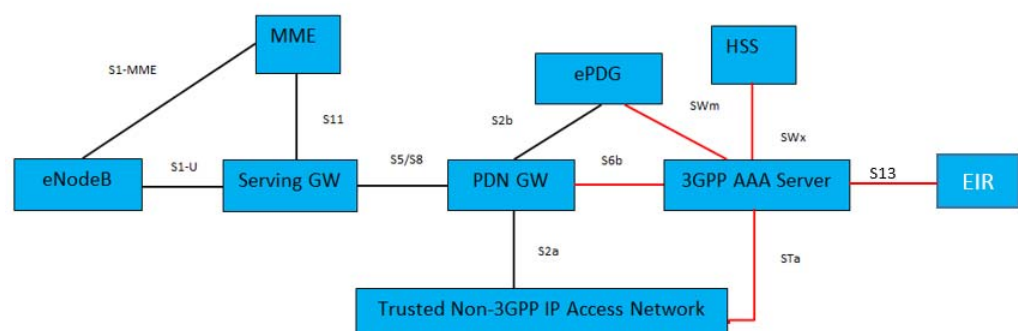


Figure 8 3GPP AAA Server and Neighboring Nodes in EPC Network

For more information about the interfaces, refer to the following interface descriptions:

- *IPWorks 3GPP AAA Server-Non-3GPP Access GW STa Interface*
- *IPWorks 3GPP AAA Server-HSS SWx Interface*
- *IPWorks 3GPP AAA Server-EIR S13 Interface*
- *IPWorks 3GPP AAA Server-PDN GW S6b Interface*
- *IPWorks 3GPP AAA Server-ePDG SWm and SWm+ Interface*

IPWorks supports SES authentication involving D' and SWm' reference points, as shown in Figure 9.

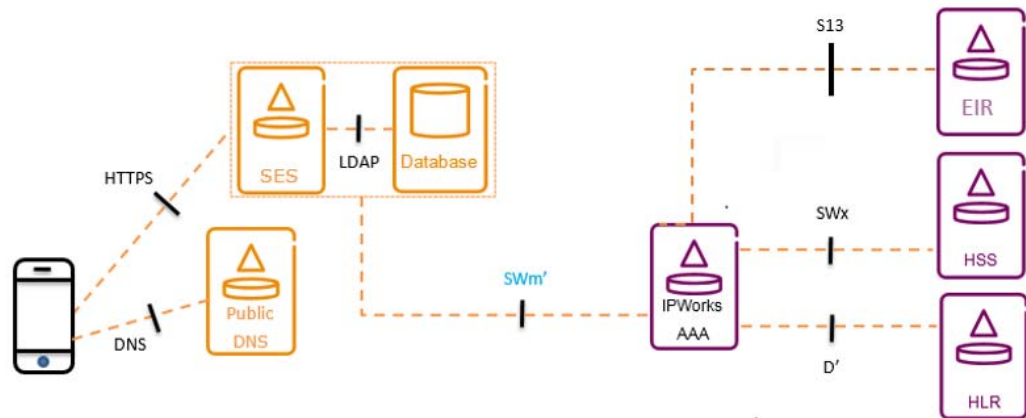


Figure 9 User Authentication for SES

For more information about the interfaces, refer to the following interface descriptions:

- *IPWorks 3GPP AAA Server-HLR D' Interface*
- *IPWorks 3GPP AAA Server-SES SWm' Interface*

The relevant protocol stacks shall be configured to achieve IPWorks EPC AAA functions. Figure 10 shows an overview of the protocol stacks used by IPWorks EPC AAA:

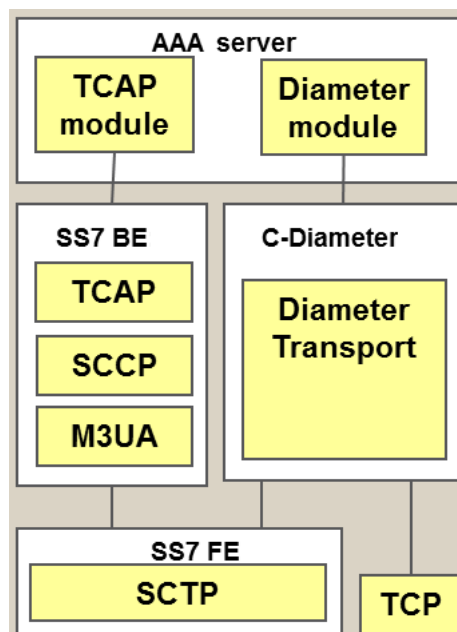


Figure 10 IPWorks EPC AAA Protocol Stacks Overview

IPWorks EPC AAA configuration is composed of the following parts:

- Diameter Stack configuration

Diameter stack configuration is mandatory and needed first for all EPC AAA services, refer to *Diameter Stack Configuration Guide* for details.

For basic concepts about the Diameter Stack, see Section 8.2.1 Diameter Stack on page 46.

- SS7 Stack configuration

Certain AAA functions (such as, Wi-Fi Mobility Management, SES Support, Diameter Over SCTP) require the SS7 stack configuration. For information on how to configure SS7 stack, refer to *Configure SS7 for AAA* according to different IPWorks deployment scenario.

- EPC AAA configuration, including the following:

- ECLI configuration

The AAA function (such as, PKI authentication, Wi-Fi Mobility Management, SES Support, IMEI check support) related configurations are mainly performed by using ECLI, refer to *Configure EPC AAA*.

- IPWCLI configuration

The AAA server and user provisioning are performed by using IPWCLI.

8.2.1 Diameter Stack

Diameter Stack is a component that is used by IPWorks AAA server as the protocol front end.

Figure 11 shows Diameter Stack 2+2 deployment in IPWorks Cluster. IPWorks Cluster, called as IPWorks Own Node, uses the same Diameter realm and Diameter host.

Other Diameter Nodes connected to IPWorks Cluster are called as Diameter Peer Nodes.

In IPWorks Cluster:

- Controller daemon of Diameter Stack in SC node manages the connectivity of user processes across the cluster.
- Payload daemon of Diameter Stack in PL node engages in Diameter signaling.
- IPWorks AAA process handles the business logic, for example, authentication.

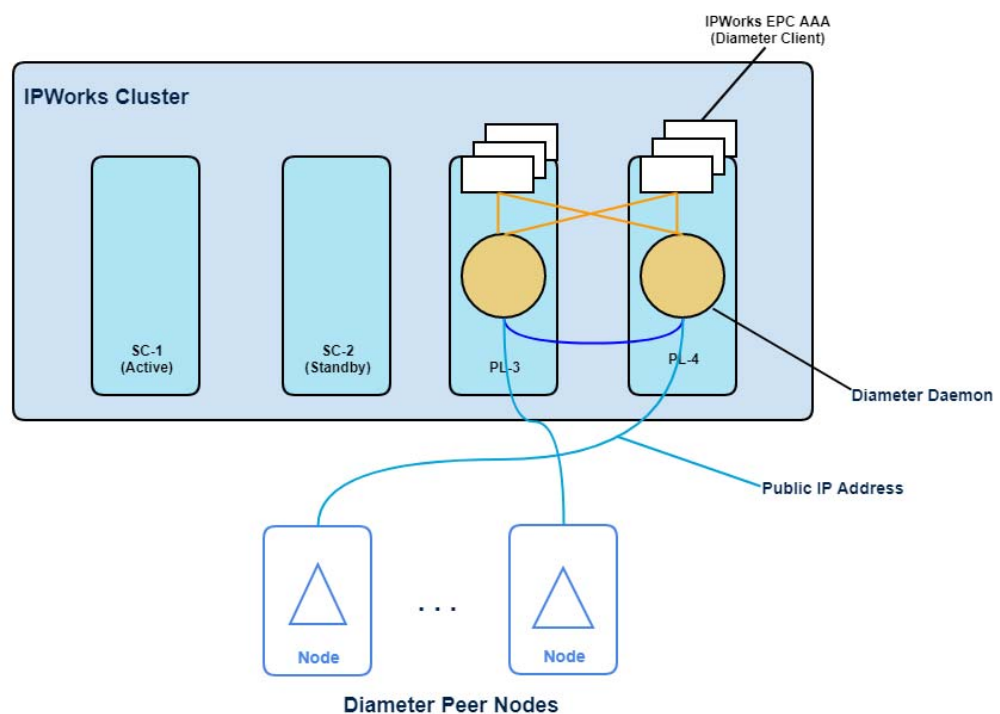


Figure 11 Diameter AAA Allocation in 2+2 IPWorks Cluster

For more details on how to configure diameter stack initially or update diameter Stack configuration in runtime, refer to *Diameter Stack Configuration Guide*.



9 DHCPv4 Management

This section describes the basic concepts to configure IPWorks DHCPv4 server.

IPWorks DHCPv4 uses IPWCLI and ECLI to perform the configuration management. For more information on:

- DHCPv4 Managed Objects in IPWCLI, refer to *IPWorks DHCP Parameter Description*.
- DHCPv4 Managed Objects in ECLI, refer to the *Managed Object Model (MOM)*.

9.1 DHCPv4 Overview

IPWorks DHCPv4 server contains the following functions:

- **DHCPv4 service:** Mainly used for IPv4 address allocation. It is the automatic configuration of IP networking parameters.
- **NACF service:** The Network Access Configuration Function (NACF) is the DHCPv4 Server that interacts with another network element called Connectivity Session Location Function (CLF). The purpose of this interface is to exchange the binding information pertaining to the exchange of the IP address allocated to the Customer Premises Equipment (CPE) and the network location information provided by the NACF.

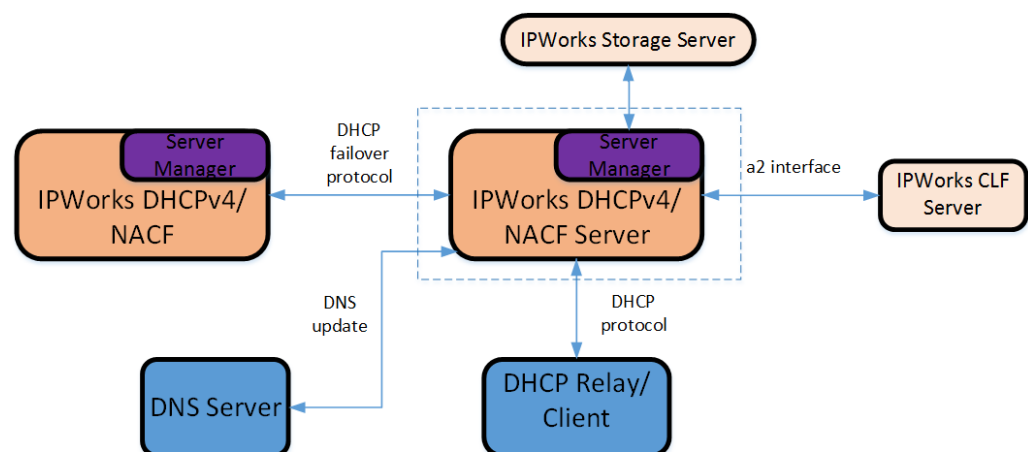


Figure 12 DHCPv4 Overview

For detailed configuration, refer to *Configure DHCP*.



9.2 DHCPv4 Service

The Dynamic Host Configuration Protocol (DHCP) is an extension of the Bootstrap Protocol (BOOTP) that handles dynamic IP address assignments for remote clients. It allows a remote client to plug into a network and broadcast a request, to one or more DHCP servers managing a particular subnet, for network configuration information. One or more DHCP servers reply to the remote client and one server ultimately negotiates a lease of terms including an IP address, a specified period of usage for the address, and other client configuration parameters.

IPWorks DHCP configuration is composed all or part of the following parts according to customer's requirements:

- *Creating DHCPv4 Server*
- *Creating DHCP Policy Objects*
- *Overlapping Lease Pools (Optional)*
- *DDNS (Optional)*
- *Failover (Optional)*
- *Load Balancing (Optional)*
- *Generating Client Host Names (Optional)*
- *Authentication of DHCP Clients (Optional)*
- *User-Defined Options (Optional)*

9.3 NACF Service

The NACF is the DHCPv4 server that interacts with another network element called CLF. The purpose of this interface is to exchange the binding information pertaining to the exchange of the IP address allocated to the Customer Premises Equipment (CPE) and the network location information provided by the NACF. Each IP Addressing Zone contains multiple Topology Zones. CPEs across the Topology Zones are connected to NACF through routers. Each Topology Zone maintains its own router.

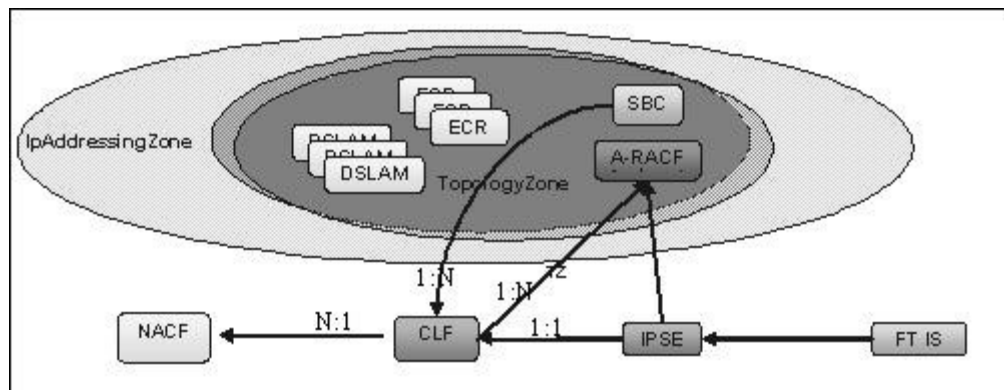


Figure 13 CLF-NACF Interface

The following four routers are supported by NACF by default. This is due to how they form option82 and the way option82 is decoded and reformatted by NACF to form CLID and RemoteId:

- Alcatel
- Cisco
- Juniper
- Red Back

NACF and CLF communicate over a TCP connection where NACF acts as TCP client and CLF as the TCP server.

NACF configuration should be done after the DHCP service configuration and the following extra configurations were completed:

- *Creating Option82 Objects*
- *CLF Interface Specifications*
- *P-CSCF Discovery*
- *User ID Provisioning*





10 MySQL Database Management

IPWorks uses MySQL databases as repositories for storing network configuration data. The Storage Server uses a MySQL database to store static network data. MySQL database on the Storage Server is a centralized resource for managing the network.

This section introduces the basic information of MySQL database.

10.1 MySQL File Locations

All MySQL files are installed to the MySQL installation directory. A symbolic link, `/usr/local/mysql`, is created to point to the MySQL installation directory. This allows fixed pathnames to be used regardless of where MySQL has been installed.

The IPWorks Storage Server database is named as `ipworks`. The database is installed in the following directories on SC-1 and SC-2 as shown in Table 8.

Table 8 Directories

Element	Stored Directory ⁽¹⁾
Data for Data Node	<code>/local/ipworks/mysql-cluster/datanode</code>
Data for SQL Node	<code>/local/ipworks/mysql-cluster/sqlnode</code>
Logs for Data Node	<code>mysql-cluster/datanode</code>
Logs for Management Node	<code>mysql-cluster/mgmnode</code>

(1) User must not change the files in any of the above directories or their subdirectories. Otherwise, this might corrupt the databases.

10.2 MySQL Scripts

The MySQL distribution provides scripts for managing most of the common administrative operations on the MySQL server. However, IPWorks provides its own customized script `ipworks.mysql` in the `/etc/init.d` directory to manage MySQL processes.

To start/stop or show the status of the MySQL NDB cluster nodes, refer to *Configure MySQL NDB Cluster*.

10.3 MySQL Server

When the MySQL server software is installed, it creates one daemon, the MySQL server (`mysqld`), which is started automatically.

For more information on the operation of the MySQL server, refer to the [MySQL Documentation](#). The MySQL manual is available in Info format in the `/usr/local/mysql/docs/` directory on the machine where the server software is installed. User can also access it online in HTML and PDF formats, refer to [MySQL Documentation](#).

10.4 MySQL NDB Cluster

MySQL NDB Cluster is a technology that enables clustering of in-memory databases in a shared-nothing system. It integrates the standard MySQL server with an in-memory clustered storage engine called NDB. The MySQL NDB Cluster allows high availability and also provides high performance, while allowing for near linear scalability.

A MySQL Cluster consists of a set of computers, each running a number of processes including MySQL servers, data nodes for NDB Cluster, management servers, and possibly specialized data access programs. All these programs work together to form a MySQL Cluster.

When data is stored in MySQL NDB Cluster, the tables are stored in the data nodes. Such tables are directly accessible from all other MySQL servers in the cluster. The data stored in the data nodes for the cluster can be mirrored, and the cluster can handle failure of individual data nodes with no other impact than that a few ended transactions because of losing the transaction state. Since transactional are expected to handle transaction failure, this is not a problem.

IPWorks stores the persistent data in MySQL NDB Cluster. The database tables in the cluster are created with engine type `ndb`. All data is provisioned by Storage Server on the SQL Node. SQL Node saves data to Data Node automatically.

The MySQL NDB Cluster is organized in units called "nodes". In MySQL NDB Cluster terminology, "nodes" correspond to processes rather than machines. There might be more than one node on the same machine. A cluster has three types of node, as follows:

MySQL NDB Cluster may have a minimum of three nodes (one Management Node, one Data Node, and one SQL Node) and a maximum of 63 nodes.

- Management Node

The role of this type of node is to manage the other nodes within the cluster, such as providing configuration data, starting and stopping nodes, running backup. The location of the Management Node has no impact upon the ENUM Server.

- Data Node

There can be multiple Data Nodes in a cluster. Data Nodes contain in-memory copies of the database. Data Nodes can contain replicas of all or part of the database. A large database can be segmented (or



partitioned) over several Data Nodes if it does not fit into the RAM available on a single server. Replication and partitioning within a cluster are hidden from the clients of the database.

The Data Node stores the ENUM tables using the in-memory NDB storage engine.

The data stored in the Data Nodes for MySQL NDB Cluster can be mirrored. A cluster can handle failure of an individual Data Node with no impact other than the loss of a few transactions as a result of losing the transaction state. This is not a problem because transactional applications are expected to handle transaction failure.

- SQL Node

This is the node that accesses the cluster data. It is a traditional MySQL server that uses the NDB storage engine.

MySQL NDB cluster has two SQL Nodes with multiple instances. All cluster data is accessed by the NDB API or SQL Node. The ENUM Server accesses the ENUM-related data using the MySQL NDB API interface.

In IPWorks, there is also a type of node can be called NDB Node. This is similar to SQL Node, but with the SQL interface replaced by a lower-level interface named `MySQL NDB API`. Database clients can access the NDB Node directly by using the interface. The ENUM Server uses the `MySQL NDB API` to locate NDB Node to gain access to the MySQL NDB Cluster.

Provisioning of the database is by the IPWorks application on the Storage Server. It uses SQL Nodes (one per cluster) to populate the in-memory database.

For more information about on MySQL Cluster, refer to [MySQL 5.5 Reference Manual](#).

Figure 14 illustrates the architecture of the MySQL database that is composed of MySQL NDB Cluster.

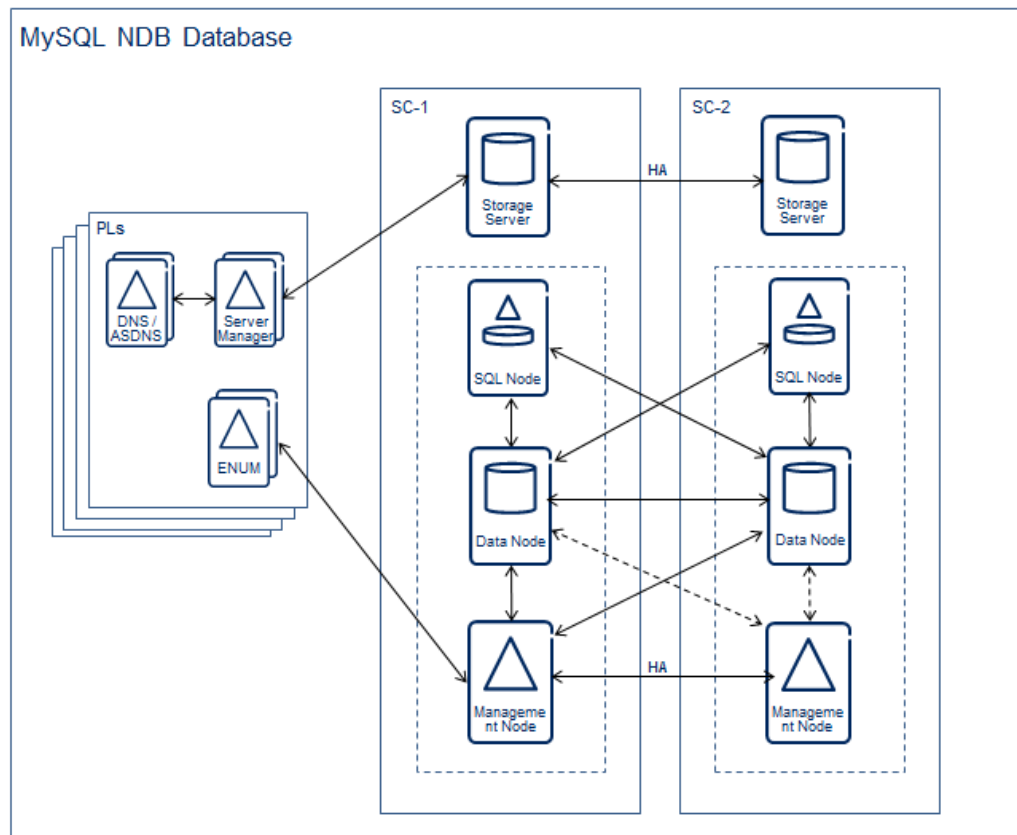


Figure 14 MySQL NDB Database

For more information about how to manage the MySQL NDB Cluster nodes, refer to *Configure MySQL NDB Cluster*.

10.5 MySQL CLI

The MySQL distribution provides a user interface for performing tasks to administer and monitors the MySQL server. The MySQL tool is an SQL shell and supports interactive and non-interactive use. When used interactively, it presents query results in an ASCII-table format. When used non-interactively (for example, as a filter), it presents the results in tab-separated format.

The MySQL CLI program is located in the `/usr/local/mysql/bin` directory.

To start the MySQL CLI, refer to *Configure MySQL NDB Cluster*.



11 Service Life Cycle Management

Several types of Availability Management Framework (AMF) tools are available to help to manage AMF services or AMF components. The command **ipw-ctr** is recommended to control the life cycle of service application from high level for user view.

This tool wraps `immadm` and `immlist` to communicate with AMF.

The usage is as follows:

```
SC-X:~ # ipw-ctr
ipw-ctr[17963]: Usage: ipw-ctr <option> <component> [<hostname>]
Usage:=====
Usage: ipw-ctr $1 $2 $3
Usage: $0 {ipw-ctr}
Usage: $1 {start | stop | restart | status}
Usage: $2 {ss | dns | dnssm | asdns | asdnssm | enum | fesync | aaasm | aaa_diameter | =>
aaa_radius_stack | aaa_radius_backend | sqlnodemgr | [status] all...}
Usage: $3 {hostname}
Usage: eg: ipw-ctr start ss SC-1
Usage: eg: ipw-ctr stop dns PL-3
Usage: eg: ipw-ctr status enum PL-3
Usage: eg: ipw-ctr status all
Usage:=====
```

Table 9

Parameter Index	Parameter Option	Description
1	start	Start one IPWorks component
	stop	Stop one IPWorks component
	restart	Stop and start one IPWorks component
	status	Check whether one IPWorks component is started or stopped.
	status all	Check the status of all IPWorks components that configured in the attribute <i>serviceType</i> .
2	component	ss, sqlnodemgr, dns, dnssm, asdns, asdnssm, enum, fesync, aaasm, aaa_diameter, ⁽¹⁾ aaa_radius_stack, aaa_radius_backend, csvengine.
3	hostname	hostname can be: SC-1/SC-2/PL-3/PL-4. <ul style="list-style-type: none"> ss, sqlnodemgr, csvengine can be in SC-1/SC-2. aaasm, aaa_diameter, aaa_radius_stack, aaa_radius_backend, dns, dnssm, asdns, asdnssm, enum, and fesync can be in PL-3/PL-4.

(1) To make sure that the ENUM FE Sync provides service, enable ENUM FE in ECLI first. Refer to Section Configuring ENUM Front End in *Configure DNS and ENUM* for details.





12 Transaction Logging

The IPWorks DNS and ActiveSelect DNS Monitor provide transaction logging. The purpose of transaction logging is to provide a historical (long-term) log of all major activity for these IPWorks components.

Transaction logging is not detailed operational logging and is not likely useful for diagnosing problems. Rather, components standard log file is used for diagnosing problems. It exists to provide an administrator a means to determine what has happened to network in the past. For example, if an administrator needs to determine who was using an IP address at a given time or who deleted a DNS resource record, they can do this by viewing the transaction log.

For more information about the Transaction Logging, refer to *DnsTransLog* and *AsdnsMonTransLog*.

12.1 Transaction Logging Periodic Maintenance Operations

As only a limited number of transaction log files are retained, it is necessary to back up the files for a long-term historical archiving. The files must be backed up before they are automatically deleted.

As the files are automatically deleted, it does not need to delete them manually. However, they can be manually deleted if disk space should become an issue or there is a desire to delete the files once backed up.

Note: If these files are deleted, the highest numbered files must be deleted before deleting lower numbered files (for example, first delete the file `ipworks_service_trans.log.<max-1>`).

12.2 Transaction Logging Message Format

The format of each record written to the transaction log is as follows:

```
date-time | service | event | event-data
```

where:

date-time	is the date and time of the event. The time is reported as local time and to the second.
service	is the type of service (dns, asdns).
event	is one of the general or service-specific events. For more information on events, see Section 12.3 on page 58.



event-data depends on the event. The format of event-data is also event-dependent — fixed formats are used, but field=value is used with optional or variable data.

12.3 Transaction Logging Events

This section describes the general or service-specific events written to the transaction log.

12.3.1 Standard Events

The following events are reported by all servers unless not appropriate to the service:

- **Startup:** This indicates when the server started. Event-data provides the version of the server.
- **Stop:** This indicates when the server stopped. Event-data reports the reason for the stop (such as operator requested or invalid configuration).
- **Reload:** This indicates when a reload was requested. Event-data usually indicates what was reloaded and who or what requested the reload.
- **Newfile:** This indicates when a new transaction log file was created and should be the last record in the old file (event-data reports the reason the file was closed) and first record in the new file. A transaction log starts with either a `Startup` or `Newfile` message.
- **Stats:** This periodically (every 15 minutes) reports operating statistics on the service. This information is used to enforce capacity-based licenses. The event-data reports the statistics used to calculate the transactions per second and transaction type mix for the service. The `pt=value` statistic is special and is the number of seconds that have elapsed for the reported statistics.

12.3.2 ActiveSelect DNS Monitor Events

- **Node down:** This indicates that a resource is down. The event-data identifies the resource that is down.
- **Node up:** This indicates that a resource is up. The event-data identifies the address of resource that is up.
- **Stats:** The ActiveSelect DNS Monitor logs the following statistics:
 - `p1` = Number of times resources were probed.
 - `p2` = Number of ASDNS notifications transmitted to DNS servers.

The ActiveSelect DNS Monitor only reports status changes.



An excerpt from an ActiveSelect DNS monitor's transaction log file is as follows:

```
2008/12/11 21:58:40|ASDNSMON|startup|IPWorks ASDNS Monitor
2008/12/11 22:00:00|ASDNSMON|stats|pt=80|p1=64|p2=4
2008/12/11 22:01:56|ASDNSMON|NODE UP|10.5.1.1
2008/12/11 22:01:56|ASDNSMON|NODE UP|10.5.1.2
2008/12/11 22:01:56|ASDNSMON|NODE UP|10.5.1.3
2008/12/11 22:01:56|ASDNSMON|NODE UP|10.5.10.1
2008/12/11 22:02:11|ASDNSMON|stop|Stopping Transaction log
```

12.3.3

DNS Events

- **AddRR:** This indicates that a resource record was added through a dynamic update. The event-data identifies the zone, added Resource Record (at least the domain name and RR type) and the source of the request (IP address). Format for event-data is source-ip, domain name, RR type, and Rdata.
- **DeletedRR:** This indicates that a resource record was deleted through a dynamic update from a zone. The event-data identifies the zone, deleted Resource Record, and the source of the request (IP address). Format for event-data is source-ip, domain name, RR type, and Rdata
- **Zxfr:** This indicates that a zone transfer was done. The event-data identifies the direction (received or send), partner of the transfer (IP address) and status. Format for event-data is direction, zone, peer (master) address, and status.
- **Stats:** The DNS server logs the following statistics:
 - p1 denotes Number of Interactive queries processed.
 - p2 denotes Number of Recursive queries processed.
 - p3 denotes Number of DDNS updates processed.
 - p4 denotes Number of dropped requests (number of requests received which had little or no processing, because of security violations).
 - p5 denotes Incremental zone transfers initiated (number the server started to other servers).
 - p6 denotes Incremental zone transfers honored (number the server received from other servers).
 - p7 denotes Full zone transfers initiated (see p5 and p6).
 - p8 denotes Full zone transfers honored.
 - p9 denotes Notices sent.
 - p10 denotes Notices received.



- p11 denotes ActiveSelect DNS notification received (status/load updates).

Note: Counters are mutually exclusive.

An example from a DNS servers transaction log file is as follows:

```
2008/12/15 12:30:33|DNS|startup|IPWorks DNS Server BIND 9.9.7
2008/12/15 12:30:49|DNS|reload|user requested reload
...
2008/12/22 14:33:52|DNS|AddRR|iptelco.com host992.
iptelco.com. 84600 IN A 11.5.4.68
2008/12/22 14:33:52|DNS|AddRR|iptelco.com host993.
iptelco.com. 84600 IN A 11.5.4.69
2008/12/22 14:33:52|DNS|AddRR|iptelco.com host994.
iptelco.com. 84600 IN A 11.5.4.70
2008/12/22 14:33:52|DNS|AddRR|5.11.in-addr.arpa 68.4.5.11.
inaddr.arpa. 84600 IN PTR host992.iptelco.com.
2008/12/22 14:33:52|DNS|AddRR|5.11.in-addr.arpa 69.4.5.11.
inaddr.arpa. 84600 IN PTR host993.iptelco.com.
2008/12/22 14:33:52|DNS|AddRR|5.11.in-addr.arpa 70.4.5.11.
inaddr.arpa. 84600 IN PTR host994.iptelco.com
...
2008/12/22
14:45:00|DNS|stats|pt=900|p1=0|p2=0|p3=102|p4=0|p5=0|p6=0|
p7=0|p8=0|p9=0|p10=0|p11=0
2008/12/15 12:46:06|DNS|stop|Stopping Transaction log|user
requested shutdown
```



13 Server Manager Configuration Properties

The Server Manager (SM) is the IPWorks component that resides on the same PL and links it to the rest of the IPWorks management system. It has the following functions:

- Update the PL with the latest configuration data for DNS server.
- Retrieve dynamic resource records from the PL for DNS server.
- Retrieve dynamic AAA sessions from PL for AAA server.
- Update the PL with the latest configuration data for DHCPv4 server.
- Retrieve dynamic leases status from the PL for DHCPv4 server.

13.1 Log Files for Server Manager

In DNS, AAA, and DHCPv4 services, SM is working as an interface between SC and PL, the Server Manager's debug logging is helpful in debugging problems with the PL and SC as well as with the Server Manager itself. The Server Manager can be configured to use debug logging through the ECLI interface. By default, the name of the Server Manager's log file is `<server_name>sm.log`, where `<server_name>` is the type of service, such as DNS, AAA, or DHCP.

The directory of the SM log files is designated by the parameter `directory` of the correlative SM Log MOM. For example, the `DnsSMLog` class for DNS SM logs in IPWorks MOM. These MOMs can be accessed through ECLI interface. The default directory of the SM log files is `/cluster/storage/no-backup/ipworks/logs`.

Note: Currently SM log directory cannot be changed.

13.2 Configuration Files

There are two configuration files that control the behavior of the Server Manager. The file `/opt/ipworks/sm/confs/ipworks_sm_defaults.conf`, which is located on the Payload, contains the default values for properties used for all the Server Managers. This file is changed rarely. The file `/etc/ipworks/ipworks_*sm.conf` contains the Server Manager properties that are used most often. The `*` in the filename is the type of server, such as DNS. Properties in this file override those in the `ipworks_sm_defaults.conf` file. Configuration changes made via ECLI are written to this file.

Some of the Server Manager configuration properties are discussed in the following sections. For a complete list of the properties, see the description of each individual property in the configuration files cited above.

13.3 Server Manager Instances

There is one instance of the Server Manager running for each type of server being managed on a given machine. Each instance is a separate process running in a Java Virtual Machine. When the Server Manager process is created, the command line contains the parameter `App=ipw*sm`, where `*` represents the type of server, such as DNS. This allows the processes for the Server Managers in a `ps` command [output] to be differentiated.

For any given type of server, there should only be one instance of the Server Manager running. When the Server Manager starts, it creates a file `/var/run/*sm.port`. This file serves the following purpose:

- Serves as a sentry file that prevents another Server Manager for the same type of server from being started. If the Server Manager is stopped in an abnormal manner (for example `kill -9`), this file will not be deleted when the Java Virtual Machine exits.

The port that the Server Manager listens on is chosen by OS as an available port. To make the Server Manager listen on a particular port, set the `Sm.ListenPort` property in the `/etc/ipworks/ipworks_*sm.conf` file. The loopback address can also be changed from the default value with the `ServerAddress` property.

13.4 Startup of Server Manager

When the Server Manager starts up, it connects to the Storage Server, logs in, registers as a remote agent for the server, and reports the status of the server to the Storage Server. It might also create a Storage Server object for the server.

For more information on the problems that can occur in this chain of events, refer to:

- *IPWorks Troubleshooting Guideline*
- *Command Line Interface User Guide for IPWorks SS*

After the Server Manager has logged in, it searches for a server object in the Storage Server that has the same IP address as the machine on which it is running. It uses the IP address from the local side of the TCP connection that it has made to the Storage Server. If it has the connection using an IPv4 or IPv6 address for the Storage Server, the address for the server is an IPv4 or IPv6 address correspondingly. For a multihomed host, the Server Manager can be configured to use a particular IP address with the `Sm.LocalAddress` property in the `ipworks_*sm.conf` file.

If the Server Manager does not find a server object in the Storage Server with the address of the machine, it does not create a server object by default. This behavior is defined by the parameter `Sm.CreateServer` in file `/etc/ipworks/ipworks_*sm.conf`. The default value of `Sm.CreateServer` is `false` that indicates that Server Manager cannot



automatically create a server object. Therefore, set the parameter `Sm.CreateServer` as `true` to allow Server Manager to create a server object.

When creating the server object, the Server Manager uses the hostname configured for the machine on which it is running for the name field of the object. This is the name that is returned by the `hostname` command. The Server Manager sets the address field of the server object to be a list of all the IP addresses on the machine, unless it is configured to use a particular address with the `Sm.LocalAddress` property. The Server Manager sets the `dnsname` field of the server object to be a list of all the DNS names of the machine. The DNS `domainname` parameter must be properly configured in the `/etc/resolv.conf` file for this to work.

Note: In the Linux system, DNS server is configured through the `/etc/resolv.conf` file. However, in IPWorks the `resolv.conf` file is generated from the configuration entries in the `/cluster/etc/cluster.conf` file. For details, refer to LDE Management Guide.

13.5 Communication with DNS Server

The Server Manager communicates with the DNS server using the standard DNS and BIND interfaces. DNS queries and dynamic updates are sent to the DNS server, which must be listening for DNS traffic on the loopback address. The loopback address and port can be changed from the default values using the `Sm.DnsLoopbackAddress` and `Sm.DnsLoopbackPort` properties in the `/etc/ipworks/ipworks_dnssm.conf` file. The Server Manager manages the dynamic zones in the DNS server for the IPWorks user. When IPWorks CLI queries for resource records in a dynamic zone or adds/deletes a resource record in a dynamic zone, the Server Manager forms a DNS packet and sends it to the DNS server on the loopback interface. TSIGKeys are used to select the proper view in the DNS server. These TSIGKeys are automatically generated by the Storage Server and used in creating the proper ACLs in the `named.conf` file. The Server Manager's TSIGKey must appear in an `allow-query`, `allow-update`, and `allow-transfer` statement for every dynamic zone as well as the `match-clients` statement for the view.

The Server Manager uses the BIND RNDc facility to send control commands, such as `stop`, `reload`, `stats`, to the DNS server. This utility is shipped with IPWorks and is installed with a link from `/usr/bin`. The RNDc client can be run from the command line independent of the Server Manager. Typing `rndc` from a terminal window gives a list of the commands that can be sent to the DNS server. RNDc requires the use of a TSIGKey. This TSIGKey is created when the DNS server is installed. It resides at `/etc/rndc.key` and should not be deleted or modified. ERH does not have a Server Manager.

13.6 Communication with ActiveSelect DNS Monitor

The Server Manager does not directly communicate with the ASDNS Monitor. The ASDNS Monitor does not store dynamic network data like resource records



(DNS server). The Server Manager starts, stops, updates the configuration and reports status of the ASDNS Monitor using the UNIX command line and watches the `pid` of the Monitor's process.

13.7 Communication with AAA Server

The Server Manager communicates with the AAA server over a TCP connection, using a proprietary protocol. The protocol supports querying the server for sessions (dynamic AAA session data). The AAA server listens on the default loopback interface. When the Server Manager starts, it attempts to connect to the AAA server. If the Server Manager cannot connect, it assumes that the status of the server is down. Next time the Server Manager needs to communicate with the AAA server, it will again attempt to connect.

13.8 Communication with DHCPv4 Server

The Server Manager communicates with the DHCPv4 servers over a TCP connection, using a proprietary protocol. The protocol covers sending control commands to the DHCPv4 server and querying the server for leases (dynamic DHCP data). The DHCPv4 servers listen on the loopback interface, and the port is 17071.

When the Server Manager starts, it attempts to connect to the DHCPv4 server. If connection fails, it assumes that the status of the DHCPv4 server is down. Next time the Server Manager needs to communicate with the DHCPv4 server, it will again attempt to connect.

13.9 Server Status

When the Server Manager attempts to contact the server (DNS, ASDNS, AAA, or DHCP) and fails, it assumes that the status of the server is down. In addition, a server might have any number of other status strings that it returns when it is queried for its state. The status can be *down* or *running*.

The servers do not automatically inform the Server Manager of a change in status. When the Server Manager starts, it checks the status of the server. For example, if the Server Manager has already established a connection to the DHCPv4 server, when the connection is broken, the Server Manager assumes that the status of the server is down.

For some applications, it is desired to have a relatively current status of a server stored in the Storage Server. The Server Manager can be configured to poll the server for its status. The rate at which the Server Manager polls can be set with the `Sm.MonitorInterval` property. Set this property to the number of seconds between queries for the servers status.

- For the DNS server, the Server Manager uses the `BIND RND` command line utility to get the status.



- For the ASDNS Monitor, the Server Manager checks the `pid` of the monitor's process.
- For the AAA server, the Server Manager checks the `pid` of the process.
- For the DHCPv4 server, the Server Manager sends a request on a TCP connection to get the status.

In all cases mentioned previously, there is no network traffic generated by the Server Manager's query for status. If the status has not changed, the Server Manager does not send the results to the Storage Server. Only if the status of the server has changed, the Server Manager generates network traffic by polling the server for status.

Configuring the Server Manager to poll for status every 30 seconds gives a relatively up-to-date status for the server in the Storage Server. The impact of this configuration on the server's performance is not significant.

13.10 Query Limits

The IPWorks product is designed for high performance of servers. When there are no IPWorks users interacting with a server, the server performance is same as an unmanaged server. That is, the server does not send any information such as newly generated dynamic network data or server status to the Server Manager. However, the IPWorks user can interact with a server to affect the servers' performance potentially. This would most likely happen when a query is made to a DNS or DHCPv4 server for a large number of dynamic resource records.

The queries made by the Server Manager for dynamic data can be limited in a number of ways. The IPWorks administrator can limit the time required for performing the query. This is configured with the `Sm.QueryTimeLimit` property, which is used for the DNS and DHCPv4 servers. This property limits the time in seconds taken to construct the response to the query. If this time is exceeded, no results are returned to the user. Instead the user receives a message giving the time limit that has been exceeded.

The IPWorks administrator can limit the number of objects (resource records or leases) in a query response. For the DNS server, this is configured with the `Sm.DnsAxfrCountLimit` property. Large DNS queries cause the Server Manager to perform a zone transfer from the DNS server. When the number of resource records in the transfer exceeds this number, the AXFR is terminated and a limit-exceeded message is returned. No resource records are returned. For a DHCPv4 Server, the limit on the number of leases for a response is configured with the attribute `maxLeaseQuery` of `MO DHCPServerManager`. This attribute is sent to the DHCPv4 server when the Server Manager connects to it. The DHCPv4 server enforces the limit and returns no leases if the limit is exceeded. A message about exceeded lease limit will also be received.

There is a separate property `Sm.DnsAllowAxfr` that the IPWorks administrator can use to disallow zone transfers by the Server Manager. This



prevents all zone transfers without regard to the size of the returned data or the time for the zone transfer.

IPWorks CLI provides queries for data, especially dynamic data, with filters that minimize the amount of data that is retrieved. Queries for resource records must specify the dnsname of the record wherever possible.

For example the CLI command:

```
IPWorks> list arecord -source iptelco.com -where  
address=10.0.0.1
```

Note: -source must be used only for arecords of zones that are dynamic.

This requires that the Server Manager performs a zone transfer for the iptelco.com zone.

The IPWorks CLI command:

```
IPWorks> list arecord -source iptelco.com -where  
dnsname=fred.iptelco.com
```

This requires only the Server Manager perform a simple owner query. For a large amount of data in the iptelco.com zone, the latter query has a much smaller effect on the performance of the DNS server.



14 Configuration Management

This section lists the operating instructions (OPI) CPIs that the user must follow:

User Account Configuration Management

- *Configure User Account*

DNS, ASDNS, and ENUM Configuration Management

- *Configure DNS and ENUM*

AAA Configuration Management

- *Configure Radius AAA*
- *Configure EPC AAA*

DHCP Configuration Management

- *Configure DHCP*

MySQL NDB Cluster Configuration Management

- *Configure MySQL NDB Cluster*





15 Appendix

15.1 DNS Options

Table 10 BIND 9 DNS Options

Option	Description
additional-from-cache	<p>The two options control the behavior of an authoritative server when answering queries that have additional data, or when following CNAME and DNAME chains.</p> <p>When both of the options are set to yes (the default) and a query is being answered from authoritative data (a zone configured into the server), the additional data section of the reply will be filled by using data from other authoritative zones and from the cache. In some situations this is undesirable, such as when there is concern over the correctness of the cache, or in servers where slave zones might be added and modified by third parties.</p> <p>Benefits of Avoiding Searching:</p> <p>Avoiding the search for this additional data speeds up server operations at the possible expense of additional queries to resolve what would otherwise be provided in the additional section.</p> <p>For example, if a query asks for an MXRecord for host <i>foo.example.com</i> and the record found is <i>MX 10 mail.example.net</i>, normally the address records (A, A6 and AAAA) for <i>mail.example.net</i> will be provided as well, if known. Setting these options to NO disables this behavior.</p> <p>Target Groups:</p> <p>These options are intended for use in authoritative-only servers, or in authoritative-only views. Attempts to set them to NO without also specifying <i>recursion no</i>, causes the server to ignore the options and log a warning message.</p>
allow-notify	<p>The option Specifies which hosts are allowed to notify slaves of a zone change in addition to the zone masters:</p> <ul style="list-style-type: none"> • If specified in the zone statement, in which case it overrides the options <i>allow-notify</i> statement. It is only meaningful for a slave zone. • If not specified in the zone statement, the default is to process notify messages only from a zone's master.
allow-query	<p>The option specifies which hosts are allowed to ask ordinary questions.</p> <ul style="list-style-type: none"> • If specified for a zone, it overrides the servers <i>allow-query</i> statement. • If not specified for a zone, the default is to allow queries from all hosts.
allow-query-cache	<p>The option specifies which hosts are allowed to query the cache.</p> <ul style="list-style-type: none"> • If specified for a zone it overrides the server's <i>allow-query-cache</i> statement. • If not specified, the default is to allow queries from localhost to localhost.
allow-recursion	<p>The option specifies which hosts are allowed to make recursive queries through this server.</p> <p>If not specified, the default is to allow recursive queries from localhost to localhost.</p> <p>Disallowing recursive queries for a host does not prevent the host from retrieving data that is already in the servers cache.</p>
allow-transfer	<p>The option specifies which hosts are allowed to receive zone transfers from the server.</p> <ul style="list-style-type: none"> • If specified for a zone, it overrides the servers <i>allow-transfer</i> statement. • If not specified for a zone, the default is to allow transfers to all hosts.
allow-update	<p>The option specifies which hosts are allowed to submit dynamic DNS updates to the server.</p> <p>The default is to deny updates from all hosts.</p>



Option	Description
allow-update-forwarding	<p>The option specifies which hosts are allowed to submit Dynamic DNS updates to slave zones to be forwarded to the master. The default is <code>{none;}</code>, which means that no update forwarding is performed.</p> <p>To enable update forwarding, specify <code>allow-update-forwarding {any;}</code>. Specifying values other than <code>{none;}</code> or <code>{any;}</code> is counterproductive, since the responsibility for update access control should rest with the master server, not the slaves. Enabling the update forwarding feature on a slave server might expose master servers relying on insecure IP address based access control to attacks.</p>
also-notify	<p>The option defines a list of IP addresses of name servers that are also sent NOTIFY messages whenever a fresh copy of the zone is loaded, in addition to the servers listed in the zone's NSRecords. This ensures that copies of the zones will quickly converge on stealth servers.</p> <p>A port might be specified with each <code>also-notify</code> address to send the notify messages to a port other than the default of 53. This option is only relevant when the notify option is NOT set to <code>no</code>. The default is the empty list (no notification list).</p>
auth-nxdomain	<p>If enabled, then the AA bit is always set on NXDOMAIN responses, even if the server is not authoritative. The default is NO. This is a change from BIND 8. If old DNS software is being used, it must be set to <code>yes</code>.</p>
blackhole	<p>The option specifies a list of addresses that the server does not accept queries from or use to resolve a query. Queries from these addresses are not responded to. The default is <code>none</code>.</p>
bogus	<p>If it is discovered that a remote server is giving out bad data, marking it prevents further queries to it. The default value is <code>no</code>.</p>
cleaning-interval	<p>The interval (in minutes) by which the server deletes expired resource records from the cache. The default is 60 minutes. If set to 0, no periodic cleaning occurs.</p>
coresize	<p>The maximum size of a core dump. The default depends upon the limits of the operating system.</p>
database	<p>The option specifies the type of database to be used for storing the zone data. The string is interpreted as a list of white space-delimited words. The first word identifies the database type and any subsequent words are passed as arguments to the database to be interpreted in a way specific to the database type.</p>
datasize	<p>The maximum amount of data memory the server uses. The default depends upon the limits of the operating system.</p> <p>This is a hard limit on server memory use. If the server attempts to allocate memory in excess of this limit, the allocation fails, which may in turn leave the server unable to perform DNS service. Therefore, this option is rarely useful as a way of limiting the amount of memory used by the server, but it can be used to raise an operating system data size limit that is too small by default. If the amount of memory needs to be limited by the server, the <code>max-cache-size</code> and <code>recursive-clients</code> options can be used instead.</p>
dialup	<p>If enabled, the server treats all zones as if they are doing zone transfers across a dial on-demand dialup link, which can be brought up by traffic originating from this server. This has different effects according to zone type and concentrates the zone maintenance so that it all happens in a short interval, once every <code>heartbeat-interval</code> and hopefully during the one call. It also suppresses some of the normal zone maintenance traffic. The default value is disabled.</p> <p>If the zone is a master zone, the server sends out NOTIFY request to all slaves. This triggers the zone up-to-date checking in the slave (providing it supports NOTIFY), allowing the slave to verify the zone while the call is up. If the zone is a slave or stub zone, the server suppresses the regular zone up-to-date queries and only performs them when the <code>heartbeat-interval</code> expires.</p>
directory	<p>The working directory of the server. Any non-absolute pathnames in the configuration file are specified as relative to this directory.</p>
dump-file	<p>This is the pathname of the file to which the server dumps the database when it receives a dump database message. The default value is <code>named_dump.db</code>.</p>
edns	<p>The edns clause determines whether the local server attempts to use EDNS when communicating with the remote server. The default value is <code>yes</code>.</p>
files	<p>The maximum number of files the server will have open concurrently. The default value is unlimited.</p>



Option	Description
forward	The forwarding facility can be used to create a large, site-wide cache on a few servers, thus reducing traffic over links to external nameservers. A value of <code>Only</code> means complete reliance on forwarders for query responses. A value of <code>First</code> means rely on forwarders first, but if no response query elsewhere.
forwarders	The IP addresses to be used for forwarding. The default value is the empty list (no forwarding). The forwarding facility can be used to create a large site-wide cache on a few servers, reducing traffic over links to external nameservers. It can also be used to allow queries by servers that do not have direct access to the Internet, but wish to look up exterior names. Forwarding occurs only on those queries for which the server is not authoritative and does not have the answer in its cache.
heartbeat-interval	Whenever this interval expires, the server performs zone maintenance tasks for all zones marked <code>diagnose=yes</code> . The default value is 60 minutes. Reasonable values are up to one day (1440 minutes). If set to 0, no zone maintenance for these zones occurs.
interface-interval	The interval (in minutes) with which the server scans the network interface list. The default value is 60 minutes. If set to 0, interface scanning occurs only when the configuration file is loaded. After the scan, listeners will be started on any new interfaces (provided they are allowed by the <code>listen-on</code> configuration). Listeners on interfaces that have gone away are cleaned up.
keys	The keys clause is used to identify a <code>key-id</code> defined by the key statement, to be used for transaction security when talking to the remote server. When a request is sent to the remote server, a request signature is generated using the key specified here and appended to the message. A request originating from the remote server is not required to be signed by this key. Although the grammar of the keys clause allows for multiple keys, only a single key per server is supported.
lame-ttl	Sets the number of seconds to cache a lame server indication. 0 seconds disables caching. The default value is 600 (10 minutes). Maximum value is 1800 (30 minutes).
listen-on	This option can be used to specify the interfaces (and port) from which the server answers queries. This option takes an optional port and an <code>address-match-list</code> . The server listens on all interfaces allowed by the <code>address-match-list</code> . If a port is not specified, port 53 is used. Multiple <code>listen-on</code> statements are allowed.
listen-on-v6	The <code>listen-on-v6</code> option is used to specify the ports on which the server listens for incoming queries sent using IPv6. The server does not bind a separate socket to each IPv6 interface address as it does for IPv4. Instead, it always listens on the IPv6 wildcard address. Therefore, the only values allowed for the <code>address-match-list</code> argument to the <code>listen-on-v6</code> statement are <code>{ any; }</code> and <code>{ none; }</code> . Multiple <code>listen-on-v6</code> options can be used to listen on multiple ports. If no <code>listen-on-v6</code> statement is specified, the server does not listen on any IPv6 address.
masterfile-format	Specifies the file format of zone files. <ul style="list-style-type: none">• For slave zone files, the default value is <code>raw</code>.• For other zone files, the default value is <code>text</code>, which is standard textual representation.
match-clients	Each view statement defines a view of the DNS name space that is seen by a subset of clients. A client matches a view if its source IP address matches the <code>address-match-list</code> of the view's <code>match-clients</code> clause and its destination IP address matches the <code>address-match-list</code> of the view's <code>match-destinations</code> clause. If not specified, both <code>match-clients</code> and <code>match-destinations</code> default to matching all addresses. A view can also be specified as <code>match-recursive-only</code> , which means that only recursive requests from matching clients match that view. The order of the view statements is significant. A client request is resolved in the context of the first view that it matches.
match-destinations	Each view statement defines a view of the DNS name space that is seen by a subset of clients. A client matches a view, if its source IP address matches the <code>address-match-list</code> of the view's <code>match-clients</code> clause and its destination IP address matches the <code>address-match-list</code> of the view's <code>match-destinations</code> clause. If not specified, both <code>match-clients</code> and <code>match-destinations</code> default to matching all addresses. A view can also be specified as <code>match-recursive-only</code> , which means that only recursive requests from matching clients match that view. The order of the view statements is significant: a client request is resolved in the context of the first view that it matches.



Option	Description
match-mapped-addresses	If yes, then an IPv4-mapped IPv6 address matches any address match list entries that match the corresponding IPv4 address. Enabling this option is sometimes useful on IPv6-enabled Linux systems, to work around a kernel quirk that causes IPv4 TCP connections such as zone transfers to be accepted on an IPv6 socket using mapped addresses, causing address match lists designed for IPv4 to fail to match. The use of this option for any other purpose is discouraged.
match-recursive-only	Each view statement defines a view of the DNS name space that is seen by a subset of clients. A client matches a view if its source IP address matches the <code>address_match_list</code> of the view's <code>match-clients</code> clause and its destination IP address matches the <code>address_match_list</code> of the view's <code>match-destinations</code> clause. If not specified, both <code>match-clients</code> and <code>match-destinations</code> default to matching all addresses. A view can also be specified as <code>match-recursive-only</code> , which means that only recursive requests from matching clients match that view. The order of the view statements is significant: a client request is resolved in the context of the first view that it matches.
max-cache-size	The maximum amount of memory to use for the servers cache (in bytes). When the amount of data in the cache reaches this limit, the server causes records to expire prematurely so that the limit is not exceeded. In a server with multiple views, the limit applies separately to the cache of each view. The default value is <i>unlimited</i> , meaning that records are purged from the cache only when their TTLs expire.
max-cache-ttl	This sets the maximum time for which the server caches ordinary (positive) answers. The default value is one week (7 days). This should not be set to zero. Doing so can cause some records such as NS to expire before they are used and can cause DNS clients to get a SERVFAIL return code.
max-recursion-depth	This sets the maximum number of recursion levels that are permitted at any one time while servicing a recursive query. Resolving a name may require looking up a name server address, which in turn requires resolving another name. For example, if the number of indirections exceeds this value, the recursive query is terminated and returns SERVFAIL. The default is 7.
max-recursion-queries	This sets the maximum number of iterative queries that may be sent while servicing a recursive query. If more queries are sent, the recursive query is terminated and returns SERVFAIL. Queries to look up top level domains such as "com" and "net" and the DNS root zone is exempt from this limitation. The default is 50.
max-journal-size	Sets a maximum size in bytes for each DNS dynamic update journal file. When the journal file approaches the specified size, the transaction journal is automatically deleted. The default value is <i>unlimited</i> .
max-ncache-ttl	To reduce network traffic and increase performance the server stores negative answers. <code>max-ncache-ttl</code> is used to set a maximum retention time for these answers in the server in seconds. The default <code>max-ncache-ttl</code> is 10,800 seconds (3 hours). <code>max-ncache-ttl</code> cannot exceed 7 days and is silently truncated to 7 days if set to a greater value.
max-refresh-time	The <code>min-refresh-time</code> , <code>max-refresh-time</code> , <code>min-retry-time</code> , <code>max-retry-time</code> options control the servers behavior on refreshing a zone (querying for SOA changes) or retrying failed transfers. Usually the SOA values for the zone are used, but these values are set by the master, giving slave server administrators little control over their contents. These options allow the administrator to set a minimum and maximum refresh and retry time either per-zone, per-view or globally. These options are valid for slave and stub zones and clamp the SOA refresh and retry times to the specified values.
max-retry-time	The <code>min-refresh-time</code> , <code>max-refresh-time</code> , <code>min-retry-time</code> , <code>max-retry-time</code> options control the servers behavior on refreshing a zone (querying for SOA changes) or retrying failed transfers. Usually the SOA values for the zone are used, but these values are set by the master, giving slave server administrators little control over their contents. These options allow the administrator to set a minimum and maximum refresh and retry time either per-zone, per-view or globally. These options are valid for slave and stub zones and clamp the SOA refresh and retry times to the specified values.
max-transfer-idle-in	Inbound zone transfers making no progress in this many minutes are terminated. The default value is 60 minutes (1 hour).
max-transfer-idle-out	Outbound zone transfers making no progress in this many minutes are terminated. The default value is 60 minutes (1 hour).



Option	Description
max-transfer-time-in	Inbound zone transfers running longer than this many minutes are terminated. The default value is 120 minutes (2 hours).
max-transfer-time-out	Outbound zone transfers running longer than this many minutes are terminated. The default value is 120 minutes (2 hours).
min-refresh-time	<p>The <code>min-refresh-time</code>, <code>max-refresh-time</code>, <code>min-retry-time</code>, <code>max-retry-time</code> options control the servers behavior on refreshing a zone (querying for SOA changes) or retrying failed transfers. Usually the SOA values for the zone are used, but these values are set by the master, giving slave server administrators little control over their contents.</p> <p>These options allow the administrator to set a minimum and maximum refresh and retry time either per-zone, per-view or globally. These options are valid for slave and stub zones and clamp the SOA refresh and retry times to the specified values.</p>
min-retry-time	<p>The <code>min-refresh-time</code>, <code>max-refresh-time</code>, <code>min-retry-time</code>, <code>max-retry-time</code> options control the servers behavior on refreshing a zone (querying for SOA changes) or retrying failed transfers. Usually the SOA values for the zone are used, but these values are set by the master, giving slave server administrators little control over their contents.</p> <p>These options allow the administrator to set a minimum and maximum refresh and retry time either per-zone, per-view, or globally. These options are valid for slave and stub zones, and clamp the SOA refresh and retry times to the specified values.</p>
minimal-responses	If <i>yes</i> , then when generating responses the server only adds records to the authority and additional data sections when they are required (such as delegations, negative responses). This might improve the performance of the server. The default value is <i>no</i> .
notify	If <i>yes</i> (the default), DNS NOTIFY messages are sent when a zone the server is authoritative for changes. The messages are sent to the servers listed in the zone's NSRecords (except the master server identified in the SOA MNAME field) and to any servers listed in the <code>also-notify</code> option. If <i>explicit</i> , notifies are sent only to servers explicitly listed using <code>also-notify</code> . If <i>no</i> , no notifications are sent.
notify-delay	<code>Notify-delay</code> causes a master server to delay sending DNS notify packets for a number of seconds. By default the IPWorks DNS server sends out a single DNS notify packet for each dynamic DNS update. If a large number of updates are to be sent over a sustained period it is more efficient to delay sending notifies to slave servers, to cut down on the number of zone transfers taking places.
notify-source	Determines which local source address and optionally UDP port, is used to send NOTIFY messages. This address must appear in the slave server's masters zone clause or in an <code>allow-notify</code> clause.
notify-source-v6	Determines which local source IPv6 address and optionally UDP port, are used to send NOTIFY messages. This address must appear in the slave server's masters zone clause or in an <code>allow-notify</code> clause.
pid-file	This is the pathname of the file in which the server writes its process ID. If the option is not specified, the default value is operating system dependent.
port	The UDP or TCP port number the server uses for receiving and sending DNS protocol traffic. The default value is 53. This option is intended for server testing. A server using a port other than 53 cannot communicate with the global DNS.
provide-ixfr	<p>The <code>provide-ixfr</code> clause determines whether the local server, acting as master, responds with an incremental zone transfer when the given remote server, a slave, requests it. If set to <i>yes</i>, incremental transfer is provided whenever possible. If set to <i>no</i>, all transfers to the remote server is non-incremental.</p> <p>IXFR requests to servers that do not support IXFR automatically falls back to AXFR. Therefore, there is no need to manually list which servers support IXFR and which ones do not; the global default of <i>yes</i> always work. The purpose of the <code>provide-ixfr</code> and <code>request-ixfr</code> clauses is to make it possible to disable the use of IXFR even when both master and slave claim to support it, for example if one of the servers is buggy and crashes or corrupts data when IXFR is used.</p>
query-source	Specifies the address and port used for queries sent to other nameservers (when the server does not know the answer to a question).
query-source-v6	Specifies the address and port used for queries sent to other nameservers over IPv6 (when the server does not know the answer to a question).



Option	Description
random-device	The source of entropy to be used by the server. Entropy is primarily needed for DNSSEC operations, such as TKEY transactions and dynamic update of signed zones. This option specifies the device (or file) from which to read entropy. If this is a file, operations requiring entropy fails when the file has been exhausted. If not specified, the default value is <code>/dev/random</code> (or equivalent) when present and none otherwise. The <code>random-device</code> option takes effect during the initial configuration load at server startup time and is ignored on subsequent reloads.
recursion	If enabled and a DNS query requests recursion, then the server attempts to do all the work required to answer the query. The default value is enabled. The setting <code>recursion no</code> ; does not prevent clients from getting data from the servers cache; it only prevents new data from being cached as an effect of client queries. Caching might still occur as an effect the servers internal operation, such as NOTIFY address lookups.
recursive-clients	The maximum number of simultaneous recursive lookups the server performs for clients. The default value is 1000. When each recurring client uses a fair bit of memory, on the order of 20 kilobytes, the value of the <code>recursive-clients</code> option has to be decreased on hosts with limited memory.
request-ixfr	The <code>request-ixfr</code> clause determines whether the local server, acting as a slave, will request incremental zone transfers from the given remote server, a master. IXFR requests to servers that do not support IXFR automatically fall back to AXFR. Therefore, there is no need to manually list which servers support IXFR and which ones do not; the global default of yes is always work. The purpose of the <code>provide-ixfr</code> and <code>request-ixfr</code> clauses is to make it possible to disable the use of IXFR even when both master and slave claim to support it, for example if one of the servers is buggy and crashes or corrupts data when IXFR is used.
rrset-order	The option permits configuration of ordering of the records in a multiple record response. Following are the legal values for ordering: <ul style="list-style-type: none"> • fixed: Records are returned in the order they are defined in the zone. • random: Records are returned in some random order. • cyclic: Records are returned in a cyclic round-robin order. Note : If multiple <code>rrset-order</code> statements appear, they are not combined, only the last one applies. If this option is omitted, records are returned in random order.
serial-query-rate	Slave servers periodically query master servers to find out if zone serial numbers have changed. Each such query uses a small amount of the slave servers network bandwidth. To limit the amount of bandwidth used, BIND 9 limits the rate at which queries are sent. The value of the <code>serial-query-rate</code> option, an integer, is the maximum number of queries sent per second. The default value is 20.
sig-validity-interval	Specifies the number of days into the future when DNSSEC signatures automatically generated as a result of dynamic updates will expire. The default value is 30 days. The signature inception time is unconditionally set to one hour before the current time to allow for a limited amount of clock skew.
sortlist	The <code>sortlist</code> statement takes an address match list and interprets it even more specially than the topology statement does. Each top-level statement in the sortlist must itself be an explicit address match list with one or two elements. The first element (which may be an IP address, an IP prefix, an ACL name, or nested address match list) of each top-level list is checked against the source address of the query until a match is found. Once the source address of the query has been matched, if the top-level statement contains only one element, the primitive element that matched the source address is used to select the address in the response to move to the beginning of the response. If the statement is a list of two elements, the second element is treated like the address match list in a topology statement. Each top-level element is assigned a distance and the address in the response with the minimum distance is moved to the beginning of the response.
stacksize	The maximum amount of stack memory the server uses. The default depends upon the limits of the operating system.
statistics-file	This is the pathname of the file the server appends statistics to when it receives a dump statistics message. If the option is not specified, the default value is <code>named.stats</code> .
statistics-interval	The interval (in minutes) at which name server statistics are logged. The default value is 60. If set to 0, no statistics are logged.



Option	Description
tcp-clients	The maximum number of simultaneous client TCP connections that the server accepts. The default value is 100.
tkey-dhkey	The <i>Diffie-Hellman</i> key used by the server to generate shared keys with clients using the <i>Diffie-Hellman</i> mode of TKEY. The server must be able to load the public and private keys from files in the working directory. In most cases, the key name is the server's hostname.
tkey-domain	The domain appended to the names of all shared keys generated with TKEY. When a client requests a TKEY exchange, it may or may not specify the desired name for the key. If present, the name of the shared key is "client specified part" + "tkey-domain". Otherwise, the name of the shared key is "random hex digits" + "tkey-domain". In most cases, the domain name is the server's domain name.
transfer-format	The server supports two zone transfer methods. The <i>one-answer</i> method transfers one resource record per message. The <i>many-answers</i> method puts as many resource records as possible into a message. Only BIND 8.x and some patched versions of BIND 4.9.5 servers support <i>many-answers</i> .
transfer-source	Determines which local address is bound to IPv4 TCP connections used to fetch zones transferred inbound by the server. It also determines the source IPv4 address and optionally the UDP port, used for the refresh queries and forwarded dynamic updates. If not set, it defaults to a system controlled value which will usually be the address of the interface <i>closest</i> to the remote end. This address must appear in the remote end's <i>allow-transfer</i> option for the zone being transferred, if one is specified.
transfer-source-v6	Determines which local address is bound to IPv6 TCP connections used to fetch zones transferred inbound by the server. It also determines the source IPv6 address and optionally the UDP port, used for the refresh queries and forwarded dynamic updates. If not set, it defaults to a system controlled value which will usually be the address of the interface <i>closest</i> to the remote end. This address must appear in the remote end's <i>allow-transfer</i> option for the zone being transferred, if one is specified.
transfers	<i>transfers</i> is used to limit the number of concurrent inbound zone transfers from the specified server. If no <i>transfers</i> clause is specified, the limit is set according to the <i>transfers-per-ns</i> option.
transfers-in	The maximum number of inbound zone transfers that can be running concurrently. The default value is 10. Increasing <i>transfers-in</i> may speed up the convergence of slave zones, but it also may increase the load on the local system.
transfers-out	The maximum number of outbound zone transfers that can be running concurrently. Zone transfer requests in excess of the limit will be refused. The default value is 10.
transfers-per-ns	The maximum number of inbound zone transfers that can be concurrently transferring from a given remote nameserver. The default value is 2. Increasing <i>transfers-per-ns</i> may speed up the convergence of slave zones, but it also may increase the load on the remote nameserver.
update-policy	<p>The <i>update-policy</i> clause allows fine-grained control over what updates are allowed. A set of rules is specified, where each rule either grants or denies permissions for one or more names to be updated by one or more identities. If the dynamic update request message is signed (that is, it includes either a TSIG or SIG(0) record), the identity of the signer can be determined.</p> <p>Rules are specified in the <i>update-policy</i> option and are only meaningful for master zones. When the <i>update-policy</i> statement is present, it is a configuration error for the <i>allow-update</i> statement to be present. The <i>update-policy</i> statement only examines the signer of a message; the source address is not relevant.</p> <p>Each rule grants or denies privileges. Once a message has successfully matched a rule, the operation is immediately granted or denied and no further rules are examined. A rule is matched when the signer matches the <i>id</i> field, the name matches the <i>name</i> field (according to the <i>nametype</i>) and the type is specified in the <i>type</i> field.</p>
version	The version the server reports through the <i>ndc</i> command or through a query of name <i>version.bind</i> in class <i>chaos</i> . The default value is the real version number of the server.
zone-statistics	If <i>yes</i> , the server keeps statistical information for this zone, which can be dumped to the <i>statistics-file</i> defined in the server options.





Reference List

Ericsson Documents

- [1] *Trademark Information*
- [2] *Typographic Conventions*
- [3] *Glossary of Terms and Acronyms*
- [4] *IPWorks Troubleshooting Guideline*
- [5] *Command Line Interface User Guide for IPWorks SS*
- [6] *Ericsson Command-Line Interface User Guide*
- [7] *IPWorks DNS, ASDNS, ENUM Parameter Description*
- [8] *IPWorks AAA Parameter Description*
- [9] *Managed Object Model (MOM)*
- [10] *Configure MySQL NDB Cluster*
- [11] *Configure DNS and ENUM*
- [12] *Configure EPC AAA*
- [13] *Configure User Account*
- [14] *IPWorks Geographic Redundancy*
- [15] *Configure SS7 for ENUM Number Portability*
- [16] *Configure SS7 for AAA*
- [17] *LDE Management Guide, 1/1553-CAA 901 2978/4 Uen*

Online References

- [18] [SUDO Website](#)
- [19] [Net-SNMP Website](#)
- [20] [MySQL Documentation](#)
- [21] [MySQL 5.5 Reference Manual](#)