

Command Line Interface User Guide for IPWorks SS

USER GUIDE

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	Target Groups	1
1.2	Related Information	1
2	IPWorks CLI Overview	3
2.1	Security	4
3	Basic Command Shell	5
3.1	Launching the CLI	5
3.2	Command Format	6
3.3	CLI Commands	7
3.4	Command File Support	8
3.5	Initializing the CLI at Start Up Time	9
3.6	Using Qualifiers to Control Output	10
3.7	Specifying Comments	11
3.8	Dealing with Errors	12
3.9	Commands Log	13
3.10	Enablelog	13
4	Basic Concepts and Commands of the CLI	15
4.1	Summary of Storage Server Concepts	15
4.2	Using the Show Commands	15
5	Working with Objects	21
5.1	Worksets	21
5.2	Specifying Expressions for Selecting Objects	24
5.3	Scoping	25
6	Editing Objects	27
6.1	Specifying Field Values for Updates	27
6.2	Transactions	29
6.3	Prompt Mode During Updates	32
6.4	Editing	36
7	Decisions	39



8	User Preferences and Profiles	41
8.1	Setting Preferences at Different Levels	42
8.2	User Profile and Access Control	43
9	Operations	47
9.1	Common Syntax Used for Operations	47
10	Advanced Topics	51
10.1	Customizing Fields Used for Commands	51
10.2	Output Formats	52
11	IPWorks Formatter	53
11.1	Templates	53
11.2	Basics of Writing Templates	55
11.3	Basic Tags	56
11.4	Formatting Data	58
11.5	Iteration	61
11.6	Conditions	63
11.7	Including Other Formats	65
11.8	Prematurely Terminating a Format	65
11.9	Predefined Templates	66
	Reference List	67



1 Introduction

This document provides the information on how to use the IPWorks Command Line Interface (CLI) to configure Storage Server (SS). It also provides information on how to launch and use the interface with descriptive examples. The CLI manipulates SS objects, therefore a section is provided to explain the basic concepts of these objects.

1.1 Target Groups

This document is intended for personnel working with provisioning and configuration for IPWorks.

1.2 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 IPWorks CLI Overview

IPWorks CLI is a command-line utility designed to fit telecommunication network architectures. It is used for the manipulation of DNS, ActiveSelect DNS (ASDNS), E164 Number Mapping (ENUM), AAA, and DHCP configurations. This CLI allows the operator to have a centralized configuration of the above services, and monitor the status of the servers.

The CLI supports a number of features as follows:

- **IPWorks is a multi-user system:** Many instances of the CLI can be used at once. If required, each IPWorks user can have a unique user name and password. Access control profiles can be used to restrict the operations a user can perform. No locking mechanism is provided to prevent changes made by one user from affecting others. Error messages will be prompted in such a case.
- **Ideal for interfacing to customer OSS systems:** The flexible formatting support in the CLI provides a method for the CLI to inter-operate with other management systems in the network. The CLI can format the output of commands in many different formats including American standard Code for International Interchange (ASCII), Berkeley Internet Name Domain (BIND) configuration file syntax (DNS information only) and Extensible Markup Language (XML).
- **Supports automation from scripting, batch jobs, or remote connections:** CLI input and output can be customized to meet the needs of the calling application. XML is supported for formatting data.
- **Allows for repetitive background or regularly scheduled IP address management tasks:** These include importing and exporting data, adding, deleting, or modifying objects, or updating servers. The CLI provides simple but powerful commands that users enter at a shell prompt or invoke in a script.
- **Formatted Output:** The CLI allows a user to define formats for printing objects. This includes the ability to format related objects. The format definition is based on XML.
- **Export Utilities:** The CLI has predefined formats (using the output formatter) for producing the configuration files for each of the PL-VM.
- **Import Utilities:** The CLI has special code written for importing data and loading it into IPWorks. The support formats include DNS as well as customized Text and XML formatted data.



2.1 Security

The Element Manager like other components of IPWorks should be secured using layers. Only authorized personal has access to IPWorks element manager machines. Some operations require root privileges. Considerable damage can be done if untrained or unauthorized users are allowed access.

In addition to operating system authentication and authorization, IPWorks requires separate authorization and authentication. Each user with administration privileges should have an account with a profile that is appropriate for the type of operations the administrator is responsible for. Using individual accounts also allows each operation to be traced to a specific user. Refer to the *User Management* section in *IPWorks Configuration Management* for more information.

2.1.1 SSL and Certificates

In IPWorks, access control is supported by creating user profiles (for instance `dnscreateuser`) and then attaching these profiles to various users. For example, profile `dnscreateuser` can be attached to user `enrico`.

In user sample profile `dnscreateuser` there is an edit rule setting that defines `allow dnsserver create`, which means that when a user attempts to create a `dnsserver` object, the system looks up the profiles attached to the user, and if there is a rule stating `allow dnsserver create`, only then it proceeds with the operation. If the user's profile does not include such a rule, the operation fails.

For detailed command descriptions, see Section 8.2 on page 43.



3 Basic Command Shell

The interactive command shell is a portion of the CLI, which provides a mechanism for operations such as reading commands, processing them and producing output. This section provides some basic functions for the command shell.

3.1 Launching the CLI

3.1.1 Running the CLI Interactively

The IPWorks CLI can be started interactively as follows:

From a shell prompt, start the CLI by executing the `ipwcli` command and log into the Storage Server.

```
#ipwcli
IPWorks> Login: <Storage Server System Username>
IPWorks> Password:
Login to server successful.
IPWorks>
```

An example to log into the storage server is as follows:

```
IPWorks> Login: admin
IPWorks> Password:
Login to server successful.
IPWorks>
```

Example 1 Logging into the Storage Server

If the user is not a root user, the following exception occurs as below:

```
ipwcli: command-not-found ipwcli
```

To avoid this exception, refer to the *Configuring Access to CLI* section in *Configure User Account*.

Note: The password is not displayed; even masked (*) characters are not shown. This behavior is to ensure that unauthorized persons in the vicinity cannot even detect the length of the password.

Strong passwords constraints (such as minimum and maximum length and use of a combination of alphanumeric and special characters) are enforced; for more information on this, and password expiration and password locking after unsuccessful attempts, refer to *Configure User Account*.



3.2 Command Format

To use the CLI, commands to be executed must be specified by the user. CLI commands use a syntax that is similar to many Unix-based command languages. There is a basic command verb followed by zero or more parameters, followed by optional qualifiers that can be used to specify options for the execution of a command. Qualifier names start with a - character.

Some qualifiers have values, and some qualifiers do not. A qualifier that has a value can be specified in one of three different ways:

1. The next item in the command is interpreted as the value for a qualifier if it does not start with a - character.
2. The value can be specified using the = character.
3. The value can be specified using the : character.

All of the following examples are equivalent:

```
IPWorks> list -field description
IPWorks> list -field=description
IPWorks> list -field:description
IPWorks> list -field "description"
IPWorks> list -field="description"
```

If a value for a qualifier contains a special character such as "*" and "_", it can be enclosed in quotes. In other words, a value enclosed in quotes will be as a whole part.

For example:

```
IPWorks> create monitor -set name="abc_def"
1 object(s) created.
IPWorks> list
[Monitor abc_def]
  Partition: active
  Name: abc_def
  Filename: asdnsmon.conf
  Type: Monitor
  ExportNeeded: true
IPWorks>
```

Commands can be specified in two ways:

1. Interactively in the CLI command shell.
2. In a text file that contains a sequence of commands to execute.



3.3 CLI Commands

This section provides a brief introduction of the commands in CLI. For more detailed descriptions, refer to the online help in CLI. The user can get the detailed description for the specific command by the following command in CLI:

```
IPWorks> help [command name] -verbose
```

3.3.1 Commands History

The commands history feature allows users to execute the previous commands given during the current session. Users can reissue the commands using **UP** and **DOWN** arrows for a maximum of 25 distinct commands. The latest command will be added to the top of the list (of 25 most recent commands) and the earliest command is dropped.

The **Backspace** key can be used to edit typographical errors in the commands.

Table 1 shows some basic commands for interacting with the CLI and the Storage Server:

Table 1 Basic CLI Commands

Commands	Description
exit	Exits the CLI.
help	Shows online help for the CLI commands.
import	Imports data from an external file into IPWorks. Refer to the <i>Importing and Exporting Data</i> section in <i>Configure DNS and ENUM</i> .
login	Login logs you into the server.
logout	Logout logs you off the server.
run	Executes a sequence of commands in an external file.
set	Sets user preferences for the session or user or both.
show	Displays information about the schema.
changepassword	Enables the user to change the password of the current user only. To change other users' password, use the command <code>modify</code> .
enablelog	Enables logging; must specify a log level. For details of this command, see Section 3.10 on page 13.

Table 2 shows some CLI commands for manipulating the objects in the Storage Server:

Table 2 CLI Commands to Manipulate Objects

Commands	Description
clear	Releases all resources for the current work set and replaces it with an empty work set.
create	Creates a new object.



Commands	Description
delete	Deletes one (or more) objects.
list	Displays one (or more) objects.
modify	Modifies one (or more) objects.
select	Replaces the current work set with the specified objects.
transaction Cancel	Cancels the current transaction and discards all changes made to any object as part of the transaction.
transaction Commit	Commits all changes made to objects in the current transaction
transaction Start	Starts a manual transaction. All subsequent updates to objects will be made as part of this transaction (until it is committed).

Page 8 shows some CLI commands for manipulating DHCP configuration:

Table 3 CLI Commands to Manipulate DHCP

Commands	Description
reconfig	Triggers the DHCP server to notify clients about configuration changes. For details of parameters, refer to <i>IPWorks DHCP Parameter Description</i> .
reconfigstatus	Returns the number of clients that responded to the <code>reconfig</code> command.

Table 4 shows some CLI commands for sending messages to the AAA server:

Table 4 CLI Commands to Send Message to AAA Server

Commands	Description
send	Sends a message to the AAA server. For more information about the command, execute <code>help send</code> to list the details in IPWorks CLI.

3.4 Command File Support

The CLI also supports the ability to execute a sequence of commands in a stored external file.

The command file can be executed in one of two ways:

- Using the `Run filename` command.
- Entering the name of the file at the `IPWorks>` prompt.

Here are two examples that execute the same command file:

```
IPWorks> run <Path>/sample.cli -trace
IPWorks> <Path>/sample.cli
```

Both of these commands execute all the commands in the file `sample.cli`, then return control back to the user when the commands are completed



(provided that the `exit` command was not encountered in the file). The first example echoes each command, along with its output to the terminal. This is because of the presence of the `-trace` qualifier. The second example simply executes the commands in the file.

3.5 Initializing the CLI at Start Up Time

The CLI provides the following methods of initializing its settings at start up:

- Defining properties in a file to control the CLI behavior.
- Specifying CLI commands in an initialization file that executes when the CLI is started.
- Defining user preference to control the CLI behavior.

3.5.1 Properties File Method

When the CLI starts, it loads several properties files to initialize itself. These properties files are used to load settings that are specific to this instance of the CLI. This file can be used to initialize the settings that are used when the CLI logs in to IPWorks. This allows users to define the default name of the server, or the default name that will be used when they log in to IPWorks.

The CLI first initializes its properties by loading the file `/opt/ipworks/cli/confs/ipworks_cli_defaults.conf`. This file defines the system-wide default settings for the CLI and it is installed during the installation of the CLI.

Subsequently, the CLI loads any user-specific property settings; it uses the algorithm to locate the user's properties as follows:

1. If the `-properties` qualifier is specified on the CLI command line, that file is loaded.
2. Otherwise, the CLI checks to see if the environment variable `IPWORKS` is defined. If so, its value is used as the name of the properties file to be loaded.
3. If that fails, the current directory is checked for a file called `.ipworks`. If the file is found, it is loaded.
4. If that fails, the users home directory is checked for a file called `.ipworks`. If the file is found, it is loaded.
5. If that fails, the CLI assumes there are no user-specific properties.

3.5.2 Initialization File Method

The CLI also allows for a user-specific command file to be defined that contains CLI commands to be executed when it starts. The CLI uses an algorithm similar



to the properties file method to determine if there is a user-specific command initialization file:

1. If the `-init` qualifier is specified on the CLI command line, that file is used.
2. Otherwise, the CLI checks to see if the environment variable `IPWORKSINI` is defined. If so, it specifies the file to be used.
3. If that fails, the current directory is checked for a file called `.ipworksini`. If the file is found, it is used.
4. If that fails, the users home directory is checked for a file called `.ipworksini`. If the file is found, it is used.
5. If that fails, there is no `init` file.

3.5.3 Defining User Preferences

The CLI also allows the user to define user-specific preferences that customize the CLI's behavior. These preferences are stored in the central IPWorks database (not in an external properties file) and are applied any time they login to IPWorks from any interface and from any system on the network. The preferences do not take effect until the login command is executed. For more information on preference commands, see Section 8 on page 41.

3.6 Using Qualifiers to Control Output

The CLI provides several qualifiers that can be used to control output. These qualifiers can be used on any CLI command.

If the user is executing a sequence of commands from a file using the `Run` command, the user can have the commands echoed in the output by specifying the `-trace` qualifier.

Any of these qualifiers can be specified when the CLI is started and they will apply to all commands executed during that session.

Qualifiers to control the amount of output

There are three qualifiers that can be used to control the amount of output produced by each command. These qualifiers can be assigned the values `ON` or `OFF`, or they can be specified with no value (in which case the default value `ON` is assumed. These qualifiers are:

- `-quiet` : all output is suppressed.
- `-verbose`: additional output is displayed.
- `-stacktrace`: stack traces will be displayed for errors.

Redirection of command output



Command output can be redirected to an external file using the `-output` qualifier, as in this example:

```
IPWorks> list -output=temp.out
IPWorks> exit
exiting.
OS>cat temp.out
[DnsServer ns0]
Partition: active
Name: ns0
Address: 10.0.0.1
PrimaryAddress: 10.0.0.1
DnsName: ns0.domain.com
PrimaryDnsName: ns0.domain.com
Filename: named.conf
```

Example 2 Redirecting Command Output to an External File

This example shows that the output for the `list` command (which listed the `DnsServer` object in the current workset) was redirected to an external file by using the `output` qualifier.

Two other qualifiers that can be specified to further control the CLI output:

`-append`: output is appended to the file.

`-echo`: output will continue to be directed to its current location as well as the specified file.

Here are the examples:

- `IPWorks> list -out=dns.out -append`
- `IPWorks> list -out=dns.out -echo`
- `IPWorks> list -out=dns.out -append -echo`

3.7 Specifying Comments

The CLI allows comments to be placed in command files. The comments must appear on a line with no other commands. Inline comments are not supported at this time.

Any line that begins with either the word `comment` or one of the special symbols, `#` or `//` is considered a comment. Normally, comments are simply discarded. Here are some examples:



```
IPWorks> comment ignore this
IPWorks> // ignore this
IPWorks> # ignore this
IPWorks>
```

Example 3 Specifying Comments

If the user is running a command file using the `-trace` qualifier, the comments are echoed in the output.

3.8 Dealing with Errors

There are many situations where users may encounter an error while working with the CLI. In order to make these situations easier to resolve, the CLI includes a special command to allow users to find out more information about the error. This can be used for most situations (except for simple syntax errors).

The `show error` command displays information about the last error that the CLI encountered (during its current session). This includes a more detailed explanation of what caused the error and usually provides some hints on how to resolve it.

If the user uses the `-verbose` version of the command, it displays a `stacktrace` within the source code where the error was detected. This can be a very important piece of information if the error being dealt with is a bug in the code that needs to be fixed and is to be reported to the IPWorks development team.

Here is an example of the output from this command:

```
IPWorks> create masterzone bogus-server test.com
Prerequisite [DnsServer bogus-server] not found for changes to
[MasterZone test.com]
IPWorks> show error
The last error was:
  Prerequisite [DnsServer bogus-server] not found for changes
  to [MasterZone test.com]
This is the description of the error:
  The values of certain fields on objects imply the existence
  of other prerequisite objects. If those prerequisite
  object do not exist then the values for the related
  field(s) are not valid. To rectify this the user should either
  change the field value in question, or else
  the user may need to create the prerequisite object prior
  modifying the object that is causing the error. The message
  text should contain sufficient information about the
  prerequisite object in question to allow the user to fix the
  problem.
```

Example 4 Showing Error



3.9 Commands Log

The commands log feature logs all the commands executed during a single session from the CLI into a log file. This can be used for administrative purposes. For example, the file can be used to run a set of routine commands as a script.

The name of the log file created is in the format `username_CLICommands_YYYYMMDD_HHMMSS.cmdlog`.

To enable this feature, the user has to set the `CLICommands.CmdLogEnabled` value of the CLI configuration file `/opt/ipworks/cli/confs/ipworks_cli_defaults.conf` to 1.

If a log file does not exist in the location specified as the `CLICommands.Directory`, a log file is created. Now all commands executed in the session are written into the log file. There is a default size and file count set for the log file. If the size exceeds the default size, a new file will be created. If the count exceeds the default count, the oldest file will be deleted and a new file will be created.

3.10 Enablelog

The `enablelog` command provides the user the ability to enable logging by toggling on or off the logging options at any given point of time. This command enables the user to change the logging level during runtime.

```
IPWorks> enablelog <loglevel>
```

Where `loglevel` is one of the following values:

- *None*: Does not log any details. This is the default value. If another logging level is in use, and it is changed to *None*, then the system stops logging into the log file.
- *All*: Logs all the activities performed by the user in the session.
Note: If *All* is specified, server performance can be affected.
- *Errors*: Logs errors that occur in the session.
- *Traces*: Logs stack traces when errors occur.
- *Net*: Logs low-level network communications pertaining to the session.
- *Netdata*: Logs all low-level network packet data pertaining to the session.
- *Netraw*: Logs raw network packet data pertaining to the session.
- *Netthread*: Logs net thread activity pertaining to the session.
- *Schema*: Logs schema loading pertaining to the session.





4 Basic Concepts and Commands of the CLI

In order to use the CLI, users need to understand the basic concepts that are provided by the Storage Server.

4.1 Summary of Storage Server Concepts

- The CLI manipulates Storage Server objects. Each object has a class that defines some basic structural information about the object. The class defines a set of fields that may have values for the object. Some fields may have only one value, some may 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 uniquely identify an object in that class. This unique identifier is known as the key value. In some classes, the key may consist of several fields, while in other it may be a single field.
- Each class also defines a set of relationships that exist between objects of that class and objects in other classes. Relationships can be used to navigate among the stored objects. A relationship can link an object with another object, or it may link it 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 objects that are assigned a name to simplify their retrieval.

4.2 Using the Show Commands

The CLI has several commands to help users learn more about how the objects are structured. This section defines some of these commands and explains the meaning of some of the output from these commands.

All of these commands show additional information if the user runs them using the `-verbose` qualifier. Without this qualifier the commands show a basic description of the items being shown. All the examples presented in this section, include the `-verbose` output so that the additional information can be documented.



4.2.1 Show Class Command

The `show class` command displays information about a class (or set of classes). This includes:

- **Aliases:** Alternate names for the class
- **Information about Class Fields:**
 - **Key Fields:** Identifying fields for objects in this class. These are displayed in the order they should be specified. Included with each field is its preference value. This value is used to determine the fields that are assigned values when only some of the keys are specified. A preference of `n` means that the key field is assumed to be specified only if there are `n` key values specified. The `n` key value appears in the output as `[n]`.
 - **Minimal Fields:** These are the minimal set of fields that must be assigned values when created.
 - **Required Fields:** The fields that are required to have a value when the object is stored. This may include fields that are not in the Minimal list of fields, if they can be derived or have default values based on other fields or system settings.
 - **Prompt Fields:** The fields that the user is prompted for in prompt mode.
 - **Baseline Fields:** The default set of fields used for most operations.
 - **Other Fields:** The fields showing other information about the component.
 - **Relationships:** Information about relationships that are defined for the class.
 - **Description:** A basic description of the class.

Here is an example:



```
IPWorks> show class ARecord -verbose
Alias(es): ARecords
Fields:
Keys:      Partition[4], Container[3], DnsName[1],
           Address[2]
Minimal:   DnsName, Address
Required:  Partition, Container, DnsName, Address
Prompt:    Partition, Container, DnsName, Ttl,
           Address, Zone, Description
Baseline:  Partition, Container, Area, DnsName,
           Ttl, Type, Address,
           Zone, IsZoneSpecific, RData, Description
Other:     Class, DottedName, Format, InternalData, RName
Relationships: PTRRecord, Area, Zone, Partition
Description: The A resource record(s) for a dns name.
```

Example 5 Showing Class

4.2.2 Show Field Command

This command displays information about one or more fields in one or more classes. This includes basic information about the type of values that can be assigned to the field, and a description of the field itself.

Here is an example:

```
IPWorks> show field DnsServer Address -verbose
DnsServer.Address
Type: Multi valued (ordered)
Datatype: Address
Description
An IPv4 address (in dotted decimal notation) or an IPv6
address. You may specify an IPv6 address using the
standard abbreviated forms, but the address will be
stored in its fully expanded form.

Description:
The address(es) associated with the server. When it is
necessary to contact the server, the addresses are tried
in order. Similarly, if during the configuration of this
server there are places where the server's address is
required, the first address will be used.

Example(s)
- 192.168.0.1
- FE80::1.2.3.4
- FE80::0102:0304
- FE80:0000:0000:0000:0000:0102:0304
```

Example 6 Showing Field



4.2.3 Show DNS and DHCP Option Commands

These commands display information about the values that can be assigned to the **Option** fields that is present on most of the DNS and DHCP objects. These are special multi-valued fields where the first part of a value is a keyword (or tag) that defines how the rest of the value is to be interpreted. This is used to define the different configuration options on all the DNS and DHCP objects.

There are special commands for displaying additional information about their respective configuration options:

Here is an example of a `show dns option` command:

```
IPWorks> show dns option allow-query -verbose
allow-query
Tag:                allow-query
Class(es):          View, MasterZone, SlaveZone, StubZone, DnsServer
Description:         Specifies which hosts are allowed to ask ordinary q
                    If specified for a zone it overrides the server's
                    allow-query statement. If not specified, the default
                    allow queries from all hosts.
Datatype:            AddrList
  Description:        Specifies a list of clients. The syntax allows you
                    - explicitly specify address(es)
                    - specify subnet(s) to match
                    - specify TSIG key(s) to match
                    - refer to a named ACL
                    - exclude clients using any of the above criteria.

                    The syntax is:
                        <addrlist> = "{" <element> ";" [ <element> ";"
                        <element> = <addrlist> |
                                    <ipaddr> |
                                    <ipnet> |
                                    <aclname> |
                                    "key" <keyid> |
                                    "!" <element>

                        <ipaddr> = an IP address
                        <ipnet>  = an IP network in "/masklength" notat
                        <keyid>  = the (quoted) name of a TSIG key
                        <aclname> = the name of an ACL or one of the sp
                                acl's: (any, none, localhost, localnets)
Example(s):          allow-query { any; }
                    allow-query { 10.0.0.0/8; 192.168/16; 1.2.3.4; }
                    allow-query { key "key1"; !acl2; localhost; }
Server Config:       allow-query <value>;

IPWorks>
```

Example 7 Showing DNS Option



There are other `show` commands that can be used to view information about the IPWorks schema, or about other aspects of the server's behavior. When users are first learning how to use the CLI, it is very beneficial to experiment with these commands to develop a better understanding of the CLI and how it works.





5 Working with Objects

This section describes some of the concepts and issues that are involved in working with objects in the CLI. It does not address the issues involved in editing objects, but rather focuses on how the objects are manipulated. For more information on editing objects, see Section 6 on page 27.

5.1 Worksets

One of the basic concepts used in the CLI is called a workset, which represents a collection of objects that are used as the target for the execution of a CLI command.

This concept is used by most of the object manipulation commands (such as `as`, `list`, `select`, `modify` and `delete`). These commands have a common syntax and common qualifiers for selecting the objects that are in the workset of the command.

The parameters and qualifiers for each of these commands specify object selection criteria that is used to determine the objects on which the command will operate. This section describes basic object selection and the use of qualifiers to filter or reduce the set of objects specified in the operation.

The objects that are commonly used can be found in the following documents. The user can also use the following command (CLI online help) to get the detailed description of the objects:

```
IPWorks> show field <object> -verbose
```

- *IPWorks AAA Parameter Description*
- *IPWorks DNS, ASDNS, ENUM Parameter Description*
- *IPWorks DHCP Parameter Description*

5.1.1 Basic Object Selection Criteria

Selecting a Specific Object

To select a specific object, the user must specify both the class of the object and key values for the object as parameters to the command. An example of a command that selects an object is as follows:

```
list dnsserver ns0
```



This command displays a single object, whose class is `dnsserver` and whose key value is `ns0`. When the command finishes, the current workset consists of just this object (assuming it exists).

Note: To see the status of the server like `dnsserver` and `dhcpv4server`, use the command `show status`. For example:

```
IPWorks> select dnsserver
Selected 1 object(s) .
IPWorks> show status
[DnsServer dns1] (10.170.1.14) On 06/04/07 at
22:21:51 server is 'running'
```

Selecting Objects from an Enumeration

To select an enumeration (known sets of objects that are assigned a name to simplify their retrieval), the user must specify the named set of objects. An example of a command that selects a named set is as follows:

```
IPWorks> list zones -where server=ns0
```

Example 8 Select Objects from an Enumeration

This command displays all the objects in the enumeration `zones` (the set of all zone objects) that have a value for the `server` field of `ns0`. This effectively retrieves all the zones for this server, and then displays them all. The current workset consists of all these objects when the command has completed.

Selecting Related Objects

To select a object that is related or linked to another object, the user must know how the objects are related and know the source object for the relationship. An example of two commands that select related objects are as follows:

```
IPWorks> delete resourcerecords -for:masterzone:ns0:domain.com
```

Example 9 Selecting Related Objects

This command deletes the resource records for the masterzone `domain.com` in the server `ns0` using a relationship to find them. This does not change the current workset (unless this object was part of the workset, in which case it is deleted.)



```
IPWorks> select masterzone ns0 domain.com
IPWorks> list -related resourcerecords
[ARRecord jerry.example.com -> 1.2.3.4]
  Partition: active
  Container: default
  Area: default
  DnsName: jerry.example.com
  Type: A
  Address: 1.2.3.4
  Zone: example.com
  ZoneId: dns1:default:example.com
  IsZoneSpecific: false
  RData: 1.2.3.4
```

Example 10 Displaying Related Resource Records Contained in a Zone

The first command selects a masterzone *domain.com* for the DNS server *ns0*, the second command uses a relationship to display all the resource records that are contained in that zone. The workset will contain all the resource records when the list command is completed.

Using Objects from a Previous Operation

If there is no object selection criteria, the set of objects from the last operation (known as the current workset) will be used for the next command. An example of using objects from a previous operation is as follows:

```
IPWorks> modify -add address=1.2.3.4
```

Example 11 Modifying an Object from a Previous Operation

This command adds the value *1.2.3.4* to the Address field for all the objects that are in the current workset. It does not change the current workset when doing this.

5.1.2 Using Qualifiers to Filter Objects

There are also qualifiers that can be used to specify additional selection criteria to further reduce the set of objects for an operation. These can be combined with any of the selection methods above.

Using Field Values as Selection Criteria

An example of selecting objects that have matching field values is as follows:

```
IPWorks> list arecords -where dnsname=sample.d
omain.com -preserve
```

Example 12 Selecting Objects by using the Where Qualifier



This command finds all the objects that are ARecords and that have a value for the field DnsName that is set to *sample.domain.com*. These objects are then displayed.

The use of the `-preserve` qualifier also prevents the current workset from being changed by this operation (normally, the current workset is all the ARecords that are listed).

Sorting Objects by Specifying Field Values

An example of specifying one or more field values that can be used to sort a selected object is as follows

```
IPWorks> select dnsservers -sort:name -dbfrom:1 -dbto:10
```

Example 13 Selecting Objects by using the Sort Qualifier

This command selects the first ten DNS servers that are sorted alphabetically by their DNS name as the current workset.

5.2 Specifying Expressions for Selecting Objects

One of the ways for selecting objects for a command is by specifying search criteria based on the field values of objects. This is done by specifying a search filter with the `-where` qualifier. A search filter is a logical expression that must be matched by objects in order to be included in the workset.

A search filter is an expression that satisfies the following grammar:

```
<expression> :=      <condition> |
                      '(' <expression> ')' |
                      <expression> <logicalop> <expression> |
                      <notop> <expression>

<condition> :=       field <relop> value |
                      field

<logicalop> :=        '&' | '&&' | 'AND' | '.AND.' |
                      '|' | '||' | 'OR' | '.OR.'

<notop>               :=      '!' | '^' | 'NOT' | '.NOT.'

<relop>               :=      '=' | '==' | 'EQ' | '.EQ.' |
                              '!=' | '<>' | 'NE' | '.NE.' |
                              '>' | 'GT' | '.GT.' |
                              '>=' | 'GE' | '.GE.' |
                              '<' | 'LT' | '.LT.' |
                              '<=' | 'LE' | '.LE.'
```

In a `condition`, the value can be a wild card expression (using the `*` to represent any sequence of 0 or more characters). If the value contains



special characters, such as space or other characters that might confuse the CLI parser, it may be necessary to quote the value or the entire expression. The value cannot be the name of another field (it is not currently possible to compare fields against one another in a search filter).

In a `condition`, there are two forms: one with a `relop` specified, and one without. If not specified, the `relop` defaults to `=` and the value defaults to `*`. This allows users to easily search for all the objects that have any value for a given field (or all the objects that do not have a value), as in the following example:

```
IPWorks> list dnsservers -where "!address"
```

This command displays all the `DnsServer` objects that do not have a value for the `Address` field.

5.3 Scoping

Since the IPWorks database contains data that spans multiple servers within a network, it can contain a number of different types of objects that may have very little relationship to each other. To deal with this, the CLI includes support for a feature called **Scoping**. This allows definition of a virtual boundary around the objects that are of interest. When users perform searches, only the objects within that boundary (or scope) are considered. When new objects are created, they are automatically inserted in that scope (unless they are explicitly created outside of the scope).

5.3.1 Example of Scoping

In order to understand the scoping example, users need to understand the IPWorks concept called a `partition`. A partition is a management container that allows maintaining different sets of data within IPWorks. All objects are placed into a partition. When a new object is created, one specified field is the partition that contains the object. For management purposes, all the objects in one partition are completely independent of the objects in another partition. In other words the partition acts as a data separation mechanism.

If the user has multiple partitions in the database, each containing different sets of unrelated objects, the user will usually want to restrict the work so that it is focused within one partition at a time. This is where scopes can be very useful.

In the provided examples it is assumed that there is one partition called `active` already created. In IPWorks this partition is the default partition for data that represents live network data.

This example shows the use of `show scope` command that is to determine the setting for a scope.



```
IPWorks> show scope partition
Partition
Value(s): active
IPWorks> select dnsservers
Selected 2 object(s).
```

Example 14 Showing Scope

Further assume that an experimental server configuration is likely to be created so that it can be but is not used in the network. To do this, a new partition needs to be created:

```
IPWorks> create partition test
1 object(s) created.
IPWorks> scope partition test
IPWorks> select dnsservers
No matching object(s) found.
```

Example 15 Creating a Partition and Set the Scope

This example shows how to create a new partition called `test` and then set the scope to that partition, using the `scope` command. As seen above, the search for existing DNS servers finds none, because there are no servers within the current scope.

This next example illustrates how the current scope affects objects that are created. Notice that in this example, the DNS server that was created was automatically added to the current scope (the Partition field was set to `test`).

```
IPWorks> create dnsserver test-server
1 object(s) created.
IPWorks> list
[DnsServer test-server]
Partition: test
Name: test-server
Filename: named.conf
ExportNeeded: true
```

Example 16 Showing How the Current Scope Affects the Created Objects

Although these examples focus on the use of the Partition for scoping, there are other scopes used to restrict user's actions. To see the full list, use the `show scope` command. It is also possible to set multiple scopes at the same time (for example, users can scope the Partition to focus on objects in a certain partition and scope the DnsServer to further restrict their actions to operate on objects contained in that DnsServer).



6 Editing Objects

This section describes some of the concepts and issues that are involved in editing (such as creating, modifying, and deleting) objects in the CLI.

6.1 Specifying Field Values for Updates

When updating objects using the create or modify commands, it is necessary to specify values for the fields that will be changed. This is primarily done using three qualifiers which share the same syntax:

- `-add`: add a field value to an object.
- `-remove`: deletes an object's field value.
- `-set`: sets an object's field value

There is a fourth qualifier that can be used (`-edit`) for changing values. For more information on the `edit` command, see Section 6.4 on page 36.

When these qualifiers are used, the value of the qualifier takes the form of an assignment expression, or possibly a list of assignment expressions. The assignment expression defines both the field to be updated, and the values that are involved in the update. The syntax of the value is defined as follows:

```
<assignment-list> := <assignment> [ ';' <assignment> ]
<assignment> := <field> '=' <value> [ ',' <value> ]
```

The next five examples show how to use this syntax:

```
IPWorks> modify -set description="this is a test"
```

Example 17 Using the -set qualifier

Note: When specifying the value for a field, use the quotation marks (", ', or `) to contain space or escape characters. Escape sequences are defined in Table 5.

Table 5 IPWorks Escape Sequences

Escape Sequence	Character
\n	new line
\t	tab
\b	backspace
\f	form feed



Escape Sequence	Character
\r	return
\x ⁽¹⁾	x

(1) x designates all other characters that are not specified in the table. For example, \" represents \"; \\ represents \; and \a represents a.

This command would set the `Description` field to the value this is a test for all the objects in the current workset. If there was a previous value for this field, it would be replaced by this value because the `-set` qualifier has been used, which always replaces existing values.

```
IPWorks> modify dnsserver sample -add address=
10.0.0.201,10.0.0.202
```

Example 18 Using the -add qualifier

This command modifies the `DnsServer` object named `sample` by adding two new values to the `Address` field. This is a non-destructive update, any existing addresses that were previously defined for this server will still be present as well.

```
IPWorks> modify -set address=10.0.0.1;descrip
tion="Modified"
```

Example 19 Using the -set qualifier to modify two fields

This command modifies the objects in the current workset by setting two fields to the values specified.

Note: It is important that there are no space characters before or after the semicolon.

```
IPWorks> modify -set address=10.0.0.1 -set
description="Modified"
```

Example 20 Using the -set qualifier to specify multiple assignments

This command does exactly the same thing as the previous command, but it shows that the `-set` qualifier can be repeated in the same command to specify multiple assignments. This also applies to the `-add` and `-remove` qualifiers as well.

```
IPWorks> modify -remove address=10.0.0.2
```

Example 21 Using the -remove qualifier

This command deletes the value `10.0.0.2` as a value for the `Address` field for all the objects in the current workset. If a given object does not have this as a value, no changes will be made to that object.



6.1.1 Loading Field Values from External Files

There are a few fields in IPWorks where the value can be a very large, multi-line text value. Under certain situations it may be easier to set the value for these fields by reading the contents of an external text file and loading those contents into the field. The CLI provides a mechanism for doing this. Both the `modify` and `create` commands support the `-load` qualifier. The syntax for this qualifier is similar to the syntax for the `-set`, `add` and `-remove` qualifiers:

```
<load-list>:= <load-spec> [ ';' <load-spec> ]
<load-spec>:= <field> '=' <filename>
```

This syntax is demonstrated with the example as follows:

```
IPWorks> modify -load contents="test.sh"
```

Example 22 Using the -load qualifier

This command sets the Contents field so that its value is the same as the text in the external file `test.sh`. If there was a previous value for this field it is replaced.

Note: This qualifier can only be used to load the contents of a single field. If the user wants to load the entire definition of an object, or multiple objects from a single file, review the Import commands. For more information on import, refer to the *Importing and Exporting Data* section in *Configure DNS and ENUM*.

6.2 Transactions

When objects are updated by CLI commands, the updates are collected together and performed as a single transaction. The frequency with which the CLI commits transactions can be set using global settings or with command qualifiers.

There are two types of transactions: manual and automatic. Manual transactions are normally managed by the user with the `Transaction Start`, `Transaction Commit` and `Transaction Cancel` commands. The default management of transactions is as follows:

- If a transaction has already been started by the user, when a command that modifies, deletes or creates objects is performed, the changes are added to the current manual transaction. The changes are not sent to the Storage server and made permanent until the transaction is committed (by the user).
- If there is no manual transaction, the changes are performed in transactions that are automatically started and committed for each object in the command's workset as part of the command execution.

These default behaviors can be overridden by the following operations:

- Commit Once



- Commit Per Object
- Commit Manually

Note: In the examples presented in subsections, the verbose output is turned on to monitor the transaction level details of what is happening.

6.2.1 Commit Once

The `-commit:once` transaction (either manual or automatic) will be committed once after processing all objects in the command workset.

```
IPWorks> modify -set description="This is a
test" -commit:once -verbose
Working on 2 object(s).
Starting automatic transaction.
Updated [MasterZone dynamic.com]
Updated [MasterZone static.com]
Committing transaction.
Modified [MasterZone dynamic.com].
Modified [MasterZone static.com].
2 object(s) were updated.
IPWorks>
```

Example 23 Using -commit:once transaction

In this example, the Description field is modified for the two objects that are present in the current workset. One transaction is being used for both objects in the workset.

6.2.2 Commit Per Object

The `-commit:perobject` transaction is committed (and another one automatically started) for each object in the command workset. If there was a manual transaction already started, it will be committed before changes are made to the first object.



```
IPWorks> modify -set description="This is a test"
" -commit:perobject -verbose
Working on 2 object(s).
Starting automatic transaction.
Updated [MasterZone dynamic.com]
Committing transaction.
Modified [MasterZone dynamic.com].
Starting automatic transaction.
Updated [MasterZone static.com]
Committing transaction.
Modified [MasterZone static.com].
2 object(s) were updated.
IPWorks>
```

Example 24 Using -commit:perobject transaction

In this example, the `Description` field is modified for the two objects that are present in the current workset. Transactions are being created for each object in the workset.

6.2.3

Commit Manually

The `-commit:manual` transaction will not be committed automatically. If there is no manual transaction active, then one will be started.

```
IPWorks> Transaction start
Transaction started.
IPWorks>
modify -set description="This is a test" -commit:manual -verbose
Working on 2 object(s).
Updated [MasterZone dynamic.com]
Updated [MasterZone static.com]
2 object(s) were updated.
IPWorks> Transaction commit
Transaction committed.
IPWorks>
```

Example 25 Using -commit:manual transaction

In this example, the `Description` field is modified for the two objects that are present in the current workset. The transaction is being managed manually by the user.

Currently the CLI doesn't support creating and modifying, or creating and deleting, the same objects in the `-commit:manual` transaction.



```
IPWorks> Transaction start
Transaction started.
IPWorks> create dnsserver dns1 -set address=10.0.0.1 -commit:manual
1 object(s) created.
IPWorks> modify dnsserver dns1 -set address=10.0.0.2 -commit:manual
Working on 1 object(s).
1 object(s) were updated.
IPWorks> Transaction commit
Transaction committed.
```

Example 26 Non-supported function for -commit:manual transaction

6.3 Prompt Mode During Updates

When modifying or creating new objects interactively, it is sometimes easier to be prompted for field values instead of specifying them all on the command line. The CLI includes a prompting mode to facilitate this, as in this example:

```
IPWorks> create arecord -prompt
Container? [next]>
DnsName: [next] >sample3.domain.com
Ttl? [next] >
Address: [next] > 10.0.0.3
Description: [next] >
1 object(s) created.
IPWorks>
```

Example 27 Prompt Mode

Note: When specifying the value for a field in the prompt mode, quotation marks (" , ' , or `) are not necessary for values that contain space characters. However, values containing escape characters must be quoted. Escape sequences are defined in Table 5.

This example shows a series of prompts asking for field values in the `ARRecord` object that is being created.

The prompting mode allows users to fully edit values, not just enter new ones. For details on the editing commands available during prompt mode, see Section 6.4.1 on page 37. Users can also specify other updates to be made using the `-set`, `-add` and `-remove` qualifiers on a command when using prompt mode. These fields are not included in the prompting.

6.3.1 Enabling Different Prompting Mode Types

The prompting mode is turned off by default, but can be enabled for a command by specifying certain qualifiers. The `-prompt` qualifier turns prompting mode on, and is used to select the type of prompting that is required. The user can specify a set of fields to work using the `-fields` qualifier.



Examples of the type of prompting that can be used are as follows:

Once

The user is prompted one time for each field, and the updates entered are applied to all the objects in the workset for that command. This is the default prompt mode if the user does not specify one.

```
IPWorks> modify -fields=description -prompt:once
Working on 2 object(s).
Description: [next] >this is a sample
2 object(s) were updated.
IPWorks>
```

Example 28 Using the Once Prompting Mode Type

This example shows the prompt mode where the prompting has been restricted to a single field (the Description), and is only going to prompt for that field one time. The changes that are entered will be applied to all objects in the current workset.

PerObject

Users are prompted for each field on each object in the workset, one at a time. If the prompt mode is used with the `create` command, it is always set to prompt per-object.

```
IPWorks> modify -fields=description -prompt:perobject
Working on 2 object(s). Description: this is a sample
Description: [next] > the first sample
Description: this is the sample
Description: [next] > the second sample

2 object(s) were updated.
IPWorks>
```

Example 29 Using the PerObject Prompting Mode Type

This example shows the prompt mode where the prompting has been restricted to a single field (the Description). However, the prompt mode is going to ask about this field for each object in the current workset. In this example, the Description in the first object is set to the `first sample` and in the second object it is set to the `second sample`.

6.3.2

Asking for Help while in the Prompting Mode

During prompting, question mark `?` can be entered for help or a single character `Q` to quit:



```
IPWorks> modify -fields=ttl -prompt  
Working on 1 object(s).  
Ttl: [next] > ?  
Enter a value (it may be quoted), or an edit command, or  
HELP EDITING for detailed help on edit commands. Field  
specific help follows:
```

The resource record 'time to live', or TTL, as specified in RFC1035: a 32 bit signed integer that specifies the time interval that the resource record may be cached before the source of the information should again be consulted. Zero values are interpreted to mean that the resource record can only be used for the transaction in progress, and should not be cached. If no value is specified, the default TTL for the zone is used.

The type of value expected is:

A numeric value that represents the number of seconds.

You can enter this value as a numeric value, or the user can use a shorthand notation for specifying times.

The basic format is #w#d#h#m#s, where w (or W) is preceeded by the number of weeks, d (or D) by number of days, etc.

A number without units is accepted as the time interval in seconds.

Example value(s):

- 2w
- 1d12h

```
Ttl: [next] > q  
Operation terminated by user.  
No object(s) were updated.  
IPWorks>
```

Example 30 Using question mark in the Prompting Mode

This example shows a prompt mode session where the user asked for help for the TTL field. After reading the help message, the user chose to end the modify operation.

```
IPWorks> modify -fields=option -prompt  
Working on 1 object(s).  
Option? [next] > ?  
Enter a value (it may be quoted), or an edit command,  
or HELP EDITING for detailed help on edit commands.  
Field specific help follows:
```

Configuration setting(s) for this zone specific to this server. These settings appear in the zone's declaration.



The type of value expected is:

A Dns configuration option, where each option setting is in the form "option value [value...]". The option and subsequent value(s) are separated by a single space. The "option" is specified by the option tag. The format of the option value(s) are determined by the option definition.

Options that can be set for this object:

```
allow-notify
allow-query
allow-transfer
allow-update
allow-update-forwarding
also-notify
database
  dialup
forward
forwarders
max-journal-size
max-refresh-time
max-retry-time
max-transfer-idle-out
max-transfer-time-out
min-refresh-time
min-retry-time
notify
notify-delay
notify-source
notify-source-v6
sig-validity-interval
update-policy
zone-statistics
```

Option? [next] > **forwarders**

Option forwarders:?

The IP addresses to be used for forwarding. The default 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 anyway. Forwarding occurs only on those queries for which the server is not authoritative and does not have the answer in its cache.

Specifies a list of addresses and an optional port for each address. The syntax is:

```
= "{ " ";" [ ";" ] ... "}"
```



```
= [ "port" ]  
Option? [next] >
```

Example 31 Using question mark for asking help for the Option field

6.4 Editing

The CLI provides editing capabilities to help do more complex operations than simply adding and removing values for fields. There are editing commands, which can be used during the interactive prompting mode and which can be used with the `-edit` qualifier on certain commands.

In these commands, the `#` represents an optional position of the value to edit in a multi-valued field. A sequence of commands can be appended together by separating them with the semicolon character `;`. Commands can be specified in upper or lower case. The syntax of the editing commands are as follows:

- `a#/<value>/`

Adds the specified `<value>` to the set of values. If specified, the position `#` is the position before which the value is added. If no position is specified, the value is added to the end.

- `d#/<pattern>/`

Deletes one or more values. If specified the position `#` indicates the position of the value to be removed. If not specified, all the values that match `<pattern>` uses the `*` character as a wildcard are deleted. The `<pattern>` can be omitted, in which case the value in the specified position is deleted. If both the position and `<pattern>` are omitted (as in "d"), then all values are deleted.

- `m<pos>/<topos>/`

Moves value in `<pos>` to `<topos>`. This command can only be used if the field is an ordered list of values and both positions must be specified.

- `r#/<pattern>/<rep>/`

Replaces values that entirely match the `<pattern>` with the string `<rep>`. If `<rep>` is omitted, the matching values are removed. If the `#` position is specified, the replacement is only applied to that value, otherwise it is applied to all the values. The `<pattern>` uses the `*` character as a wildcard.

- `s#/<substring>/<rep>/`

Substitutes `<rep>` for all occurrences of `<substring>` in the existing values. If `<rep>` is omitted, the sub-string is removed from the existing values. If the `#` position is specified, the substitution is only applied to that value, otherwise it is applied to all the values. Wildcards cannot be used in the sub-string.



Here are some valid examples of `edit` commands:

```
r/foo*bar/foobar/  
d4  
d;a/newval/;a/newval2/;a/newval3/;s/new//
```

To specify edit commands using the `-edit` qualifier, the field to be edited followed by a colon `:` character, followed by the `edit` commands to apply to the field must be specified. If multiple fields need to be edited, the `-edit` qualifier must be specified multiple times.

Here is a sample command:

```
IPWorks> modify -edit DnsName:s/com/edu/
```

This example modifies all of the objects that are in the current workset. For each object, if the `DnsName` contains the string `com` then it is replaced by the string `edu`. If it does not contain the string `com`, then nothing is done. If a transaction was previously started, these changes are added to that transaction. If not, each object is updated as its own transaction.

6.4.1 Edit Command in Prompt Mode

To specify `edit` commands in prompt mode, simply enter command when prompted for a value. IPWorks recognizes that the value is an `edit` command rather than a value, and applies edits to the existing values for the field. In the event that the user wants to enter a value that is the same as a valid `edit` command, the user can quote the value and IPWorks interprets it as a value instead of an `edit` command. After the `edit` commands are applied to the existing values for the field, the user is prompted for that field again.



```
IPWorks> modify -prompt -field=address
Working on 1 object(s).
Address: 10.0.0.1
Address? [next] > 10.0.0.2
Address: 10.0.0.1
Address: 10.0.0.2
Address? [next] > 10.0.0.3
Address: 10.0.0.1
Address: 10.0.0.2
Address: 10.0.0.3
Address? [next] > s/10/12/
Address: 12.0.0.1
Address: 12.0.0.2
Address: 12.0.0.3
Address? [next] > d3
Address: 12.0.0.1
Address: 12.0.0.2
Address? [next] >
1 object(s) were updated.
```

Example 32 Editing Command in Prompt Mode



7 Decisions

There are many places in the IPWorks product where some automated behavior is used to assist in the configuration of the management objects. One such example is in the automatic management of NS and SOA records for zones. When a `MasterZone` is created, the product attempts to automatically create the NS and SOA records for the zone based on the configuration of the `DnsServer` that contains the zone. This behavior is automated because these records are required for the proper functioning of the DNS protocol and therefore there is really no reason not to do this.

However, there are also some situations where an automated behavior may be of assistance to a user, but the question whether it should be applied may not be obvious. In situations like this, a decision needs to be made in order to determine if the behavior should be applied. IPWorks uses the following algorithm for making decisions:

1. Check the configuration settings in the central server to see if there is a setting that defines what should be done. This gives the IPWorks administrator the opportunity to configure certain behaviors across the product if that is what's desired.
2. Check the user's preference settings to see if there's a setting that defines what should be done. This gives the user the opportunity to preselect certain automated behaviors that they always see. For more information on user preferences, see Section 8 on page 41.
3. Ask the user. If all else fails, IPWorks simply asks the user what to do.

Most automated behaviors occur when a transaction is committed, so most decisions are not asked until the transaction is committed.

Here is an example of a decision that resulted in the user being asked a question:

```
IPWorks> create arecord test.domain.com 10.0.0.1
Would the user like to create a PTR Record
for the assignment of the address "10.0.0.1"
to the dns name "test.domain.com" (it will be added to
zone " 0.0.10.in-addr.arpa")? [yes] > ?
This determines if PTR records should be created
automatically when an A or AAAA record is created.
PTR records are used for reverse lookup
of name-address bindings.
```

When an A or AAAA record is created,
this setting is only checked
if (1) there is a zone that exists that could contain
the potential PTR record and (2) the Subnet (or Prefix)



for the associated address indicates that the user creating the record should be asked.

The valid answer(s) for this question are:

(Yes, No)

These can be abbreviated,

or the user can use the "*" wildcard.

To abort this transaction (without answering), enter "q".

To reuse the user's answer for subsequent occurrences of the question, append "!" to use it for the rest of this command, or "!!" for the rest of this session, or "!!!" to save it for future sessions.

To preanswer the question, set preference "Auto.CreatePTRRecord".

```
Would the user like to create a PTR Record
for the assignment of the
address "10.0.0.1" to the dns name "test.domain.com"
(it will be added to zone "0.0.10.in-addr.arpa")?
[yes] >
1 object(s) created.
```

This example shows:

- A question being asked of a user when an A record is being created. It is often the case (but not always) that a *PTR* record is created at the same time that A records are created, because the *PTR* record provides the reverse lookup capabilities for the name or address binding defined in the A record.
- The user can enter ? at any time when the user is asked a question in the CLI, and it will give a further explanation of what is being asked. The user can also enter the "Q" character to abort the transaction if the user is not sure. Once the question is answered, the transaction will continue.

Note: The help text for this question includes the name of the setting that can be used to predefine answers for the decision that is being made. For more information on user preferences or predefined settings, see Section 8 on page 41.



8 User Preferences and Profiles

As mentioned in previous sections, the CLI has a number of settings that can be used to customize its behavior. Behavior of these settings can be defined and saved on a per-user basis. These settings are collectively referred to as user preferences.

Note: All preference settings are case-sensitive and there is no checking when the user sets a preference. If the user mistypes the name, or use incorrect case, then the preference setting is not used.

User behavior can be controlled through profiles which determine what they are allowed to do within the system. For more information, refer to *User Management in IPWorks Configuration Management*.

To see all the preferences that can be set, the user can use the `show preferences` command. In its default mode, it only displays the values of any preferences the user has defined. However, if the user uses the `-verbose` qualifier, it displays all the preferences that can be set, with descriptions and with their default settings.

```
IPWorks> show preferences -verbose
... <text removed> ...
#----- Auto.CreatePTRRecord -----
# This determines if PTR records
should be created automatically
# when an A or AAAA record is created.
PTR records are used for
# reverse lookup of name- address bindings.
#
# When an A or AAAA record is created,
this setting is only checked
# if (1) there is a zone that exists
that could contain the #
# potential PTR record and
(2) the Subnet (or Prefix) for the
# associated address indicates that
the user creating the record
# should be asked.
#
#Auto.CreatePTRRecord=yes
... <text removed> ...
```

Example 33 Using the `-verbose` qualifier

This example represents an extract of the output. There is normally somewhat more text displayed than the example shown. For the purposes of this document, a portion of the output has been extracted to show the description of only one of the set table preferences. In addition to the description of the



preference setting, the default value is displayed (and commented out). If a value was set for this preference it would be shown without the comment character before the setting.

8.1 Setting Preferences at Different Levels

The CLI allows users to set preferences at three different levels, where each level represents the length of time that the preference setting will be used:

1. Set a preference permanently. This setting is saved in the database and is restored each time the user logs in. These settings are also used if the user uses other IPWorks management interfaces.
2. Set a preference for the current session. These settings last until the user logs out.
3. Set a preference for a single command.

Setting a Preference Permanently or for a Current Session

To set a preference permanently, or for the current session, the user can use the `set` command. If the user wants to save the preference permanently the user can use the `-save` qualifier and it is saved. The user can delete the preference value set and leave it blank to undo the preference.

```
IPWorks> set CLI.Verbose=on
```

Example 34 Turning on preference verbose

This example turns on verbose output for all subsequent command in this session.

```
IPWorks> set Auto.DeleteUnmanagedObject=yes
```

Example 35 Turning on preference Auto.DeleteUnmanagedObject

This example sets preference `Auto.DeleteUnmanagedObject` to `yes` for all subsequent commands in this session. When an object is deleted, the related unmanaged object will be deleted automatically. For example, delete `masterzone` will automatically delete the related `SOARecord` and `NSRecord`.

```
IPWorks> set CLI.Verbose=off
```

Example 36 Turning off preference verbose

This example turns off the preference set for the subsequent command.

Setting a preference for a single command

To set a preference for a single command, the user can use the `-preferences` qualifier. If the user is using the command to preanswer a decision, its important to remember that most decisions occur when the transaction is committed.



Therefore, if this is a manual transaction, set the preferences when the user who commits the transaction, not when the user is making other changes to the object.

```
IPWorks> create ARecord test2.domain.com 10.0.0.2
-pref="Auto.CreatePTRRecord=yes"
Starting automatic transaction.
Committing transaction.
Created [ARecord test2.domain.com -> 10.0.0.2].
Created [PTRRecord 2.0.0.10.in-addr.arpa ->
test2.domain.com].
1 object(s) created.
IPWorks>
```

Example 37 Turning on preference Auto.CreatePTRRecord

This example shows how to set a preference for a single command. Notice the verbose output for this command as a result of the previous change to the `CLI.Verbose` setting. The verbose output shows that the PTR record is created without asking the user.

```
IPWorks> delete masterzone dns1 9.9.4.3.e164.arpa
-pref="Auto.DeleteUnmanagedObject=yes"
```

Example 38 Turning on preference Auto.DeleteUnmanagedObject

This example shows that when `masterzone` is deleted, it will automatically delete the related object such as `SOARecord` and `NSRecord`.

8.2 User Profile and Access Control

The IPWorks administrator creates the profiles and attaches them to users; the access control feature in IPWorks verifies the permissions (rules) specified for the user and accordingly allows or disallows the operation.

Permissions are defined in terms of create or delete or modify objects: users can perform these operations on other objects which depend on the objects. For example, if a user has `create-dnsserver` permission, then the user can create a DNS view, `masterZone`, `ACL`, `TSIG` key, and so on.

A profile can be created to allow permissions at a subclass object level such as resource record; in that case, users with that profile can create a `MasterZone`, `ARecord`, or a `CNAMERecord`, but not a `DnsServer`.

The following sub-sections discuss the administrator's commands and the user's commands pertaining to user profile creation.

The IPWorks user is subject to the profiles created by the administrator and the access control mechanism.



8.2.1 Creating Profiles in Administrator Login

Only an administrator (user logged in as `administrator`) can create profiles and assign them to users. Multiple profiles can be attached to a single user: the advantage of this is that a set of features can be defined (and new ones can be added as required) and users can be given permissions by assigning one or more of the profiles as and when required.

In the following example, a profile called `dnscreator` is being created and it is being assigned to a user called `user1`.

```
IPWorks> create profile dnscreator -set editrule
="allow dnsserver create"
1 object(s) created.
IPWorks> create profile dnshandler
1 object(s) created.
```

Example 39 Creating Profiles

Note: If an operation (`create`, `delete`, or `modify`) is not specified in the `allow` statement as in the second `create` profile, then all operations are allowed. The following example illustrates this point.

```
IPWorks> create user user1 -set password=
Shanghai1 -set profile=dnscreator,dnshandler
1 object(s) created.
```

`Shanghai1` is an example for the password to be set. For more information about the password constraints, refer to the *Password* section in *IPWorks Configuration Management*.

In the following examples, the profile `dnscreator` is updated.

```
IPWorks> modify user user1 -set profile=dnsc
reator,dnshandler
1 object(s) created.
```

Example 40 Specifying multiple profiles for a given user or add profiles at a later point

```
IPWorks> modify user user1 -remove profile=dn
shandler,dnscreator
1 object(s) created.
```

Example 41 Detaching a profile an updated command

The following example creates a profile called `resrec` (with permissions to manipulate resource records) at the subclass object level, and assign the profile to `user2`.

Note: In the first example of this section, the privilege to create `dnsserver` objects has been deleted from `user1`.



```
IPWorks>create profile resrec -set editrule="allow
ResourceRecord"
1 object(s) created.
IPWorks> create user user2 -set password=Shanghai1 -set profile=res
1 object(s) created.
IPWorks> modify user user2 -set password=Admin123 -set profile=resr
Working on 1 object(s) .
1 object(s) created.
```

Example 42 Creating a profile at the subclass object level

In this example, no specific operation is included in the `allow` statement; as seen in the following section, `user2` can create, delete or modify resource records.

8.2.2 Access Control of User Profiles

Once the profiles are created and are assigned to users as described in Section 8.2.1 on page 43, access control is enforced on operations requested by users.

In the examples described in Section 8.2.1 on page 43, `user1` has permissions to create a `dnsserver` object; when `user1` issues a command to delete an object, the system returns an error.

When `user1` logs into the system and issues a command to create a `dnsserver` object, the operation proceeds as follows:

```
IPWorks> create dnsserver -set name=dns_U1;address=10.0.
0.2
1 object(s) created.
```

The following is an example of `user1` having insufficient permissions:

```
IPWorks> delete dnsserver dns_U1
Permission to create/modify/delete object denied.
[Profile user1]
```

In the case of `user2` with profile `resrec`, access control allows create, delete, and modify operations at the resource record level such as `masterZone` or `arecord`, as shown in the following examples:

```
IPWorks> create masterzone -set name=eric.com;server=dns
_U2; authoritativename=ns_U2.eric.com;sourcename=dns_U2
.eric.com
1 object(s) created.
```

In the above example, `user2` creates a `masterZone` on an existing DNS server `dns_U1` because `user2` does not have permission to create `dnsserver` object.

The `list` command can be used to see the available `dnsserver` objects.



```
IPWorks> list dnsserver
```

As mentioned above, `user2` can also delete (and modify) resource records. An example is below:

```
IPWorks> delete arecord dns_U2.eric.com;address=10.0.0.2
Working on 1 object(s).
1 object(s) were updated.
```



9 Operations

A significant subset of the commands in the CLI fall into a special category of commands that are called operations. These are commands that perform a domain-specific action on one of the IPWorks management objects. Operations can only be applied to specific types of objects (unlike most of the commands that have been described thus far that are generic and can be used with any object).

Table 6 lists the common operations.

Table 6 Commands Called Operations

CLI Command	Object Type	Description
export	MasterZone and Server	Exports the configuration files for the selected objects to external files in the local file system.
partnerdown	DhcpV4Server	Sets the selected servers to "partner down" mode.
runscript	DnsServer, DhcpV4Server	Executes a remote script on the selected server systems.
show status	DnsServer, AAAServer, DhcpV4Server	Displays operational status for the selected servers.
update	DnsServer, DhcpV4Server	Updates the selected servers with the latest configuration information from the central IPWorks database.

9.1 Common Syntax Used for Operations

All operations use a common syntax as the basis for their command definition.

The syntax is as follows:

```
<cli-command> [ <selection- criteria> ] [ <operation-qualifiers> ]
```

The `<selection criteria>` is a set of parameters and qualifiers that defines the set of objects that the operation is going to be applied to. It is the same set of qualifiers (and parameters) that are used for all the other CLI commands that operate on worksets. For more information on worksets, see Section 5.1 on page 21. This allows the user to use all the same searching and selection qualifiers that the user utilizes on other CLI commands (such as `list` and `modify`).



The `<operation-qualifiers>` are operation-specific qualifiers that specify additional information to use when the operation is performed. Some of these qualifiers may be required, some may be optional, and some commands may not have any qualifiers at all. To get more information on a particular operation, the user can consult the `help` command in the CLI, or the user can consult the output from the `show operation` command (which tends to have even more detailed descriptions of the operations and their parameters).

To further clarify how the operations and their parameters are specified in the CLI, here are a few examples:

```
IPWorks> export MasterZone domain.com -dir=export  
Exported the zone [MasterZone domain.com]
```

This command selects the MasterZone called `domain.com` and then exports its configuration file (`zonefile`) to the directory called `export`. The actual name of the exported file is determined from the configuration of the MasterZone. When the command ends the current workset changes so it contains only this MasterZone object (just like any other commands that operate on or use worksets).

```
IPWorks> update DnsServers -where ExportNeeded=true  
Exported the zone [MasterZone iptelco.com]  
Exported configuration for [DnsServer dns1]  
Updated the configuration for 'DNS' server 'dns1'.
```

This is a very useful command that updates any DnsServers whose configuration has been changed in such a way that the live server configuration on the network is different from the configuration in the central IPWorks database. Notice that the actual output from this command is not included in this example.

```
IPWorks> select dnsserver ns0  
Selected 1 object(s).  
IPWorks>show log  
[DnsServer ns0] (12.0.0.1) is not currently available. Operation show  
IPWorks>
```

This command shows that the user can use operations on the objects that are in the current workset by simply not specifying any other selection criteria. Most operations on servers require that the server is active on the network — this shows the error that the user would receive if the user tries to perform an operation on a server for which the Server Manager is not running.

```
IPWorks>update dnsserver dns1 -rebuild=true
```

This command forcedly updates dnsserver and exports the configuration from the central IPWorks database.



Note: This command (with the option `-rebuild=true`) takes much time to update the DNS server. If the operator tries to incrementally update the configuration and provisioning data, use the `update dnsserver <server name>` (without the option `-rebuild=true`) instead to save time.





10 Advanced Topics

This section describes some of the more advanced topics in the CLI. It is not critical to understand any of these topics, but they provide levels of customization for advanced users that may be useful.

10.1 Customizing Fields Used for Commands

Many of the commands in the CLI operate on sets of fields for the objects while they are being executed. Some commands allow users to specify the list of fields to use (with the `-fields` qualifier). The CLI provides a second mechanism to customize the set of fields that will be used for such operations on a per-class basis, known as field lists.

A `fieldlist` is an ordered list of field names that is specific to a particular class. Each field list has a name, some of which are predefined. Users can specify the set of fields to use for a particular command by using the `-fields` qualifier. To do this, simply include the fieldlist name in the list of fields (with a `$` character preceding it) and all the fields in that field list will be inserted in its place.

If there are no fields specified by the `-fields` qualifiers, then the CLI will use a predefined fieldlist. The fieldlist used depends on the context:

- **Baseline:** Under most circumstances this is the default set of fields that are used for an operation.
- **Prompt:** In prompting mode, this is the set of fields that the user will be prompted for. If not defined, defaults to baseline.
- **Minimal:** This is the minimal set of fields that must have values specified by the user in order to create an object
- **All:** All the fields defined for a class
- **Keys:** The fields that are key fields

Fieldlists can also be defined as properties in the startup file, and a fieldlist definition can include a reference to other fieldlists in its definition. Here is an example line that could be included in a startup file to override the default definition of the baseline fieldlist for the `DnsName` class:

```
FieldList.DnsName.baseline=$keys,Address,Description
```



10.2 Output Formats

The CLI uses XML to define the output formats, including the default output formats that are included with the product. A user can define their own formats.

For more information on the XML formatting language, see Section 11.4 on page 58.

The following file defines the default output format for the CLI:

```
<ipworks:body>
<ipworks:class/><ipworks:key prefix=" " delimiter=":"/>
<ipworks:newline/><ipworks:for each="fieldinselection">
<ipworks:field prefix=" "/>:
<ipworks:value prefix=" " delimiter=", " count="*"/>
<ipworks:newline/>
</ipworks:for>
</ipworks:body>
```

This format would produce output similar to the following:

```
IPWorks> list dnsserver ns0 -format=sample.fmt
[DnsServer ns0]
Partition: active
Name: ns0
Address: 10.0.0.1, 10.1.0.1, 10.2.0.1
PrimaryAddress: 10.0.0.1
DnsName: ns0.domain.com
PrimaryDnsName: ns0.domain.com
Filename: named.conf
```

This example looks up a single DnsServer that has the name `ns0` . It then lists this object using the format defined in the external file called `sample.fmt`. The built-in formats that are included by default include `brief`, `default`, `full`, `ldif`, and `xml`.



11 IPWorks Formatter

This section provides an introduction to the IPWorks Formatter. The Formatter was built into the CLI to control and customize the CLI output. Understanding how it works can be very helpful for controlling the way the user can view managed objects.

11.1 Templates

The Formatter starts with a textual representation of what the formatted output will look like. Embedded in that textual representation are special formatting tags that are interpreted by the formatter, so that it can produce the formatted text. The original, unprocessed textual representation is called a template. Some templates consist almost entirely of formatting tags, while others have very few formatting tags.

11.1.1 Types of Templates

IPWorks uses two types of templates:

1. **Named templates** are managed centrally (in the Storage Server), typically by an administrator. They are stored in a directory, as files (so that they can be easily edited). The product includes a number of predefined templates that are included in the `jarfile` for the Storage Server. Each of these templates has a unique name (the filename) that can be used to refer to the template.
2. **Unnamed templates** are associated with specific objects. They do not have names, and the textual definition of the template is stored in a field in the object. Unnamed templates are typically used to define the text based configuration of the objects they are associated with. For more information about configuring objects with templates, see Section 11.1.3 on page 54.

11.1.2 CLI Default Output Template Formatting Example

Format templates are expressed using XML. The example shows the default output template for the CLI. It is composed almost entirely of formatting tags.

```
<ipworks:body>
<ipworks:class/>:<ipworks:key prefix=" " delimiter=":"/>
<ipworks :newline/>
<ipworks:for each="fieldinselection">
<ipworks:field prefix=" " />:
<ipworks:value prefix=" " delimiter=", " count="*"/>
<ipworks:newline/></ipworks:for>
</ipworks:body>
```



This format would produce output similar to the following:

```
IPWorks> list dnsserver test.com
Dnsname: test.com
DnsName: test.com
HostName: test
Domain: com
Address: 1.2.3.4, 1.2.3.5
```

11.1.3 Configuring Objects with Templates

One of the most important uses of the format templates is in defining how objects are configured in the PL VM. The VMs are configured by loading text-based files. These files are created by IPWorks and the format templates are used to create these files.

The algorithm used is as follows:

Each object has a special field containing the definition of the configuration template for that object. This is an unnamed template, and defines the text that is written to the PL VM for that object. If the field does not have a value, IPWorks uses a named template to define the text for that object. The named template that is used is the template that has the same name as the class of the object in question.

Here is an example of what may be used as a template for a DNS server's named `named.conf`.

```
// This file was written by IPWorks
on <ipworks:date/>
// Generated for server
<ipworks:value field="ServerIdentifier"/>

<!--
Insert statements here that are not managed by IPWorks.
This currently would include acl,
controls, logging, and server statements
-->

logging {
category default { default_debug; default_stderr;
default_syslog; };
category queries { default_debug; default_stderr;
default_syslog; };
category db { default_debug; default_stderr;
default_syslog; };
};

<!-- The rest of the config file -->
```



```
<!-- is generated by IPWorks -->
<ipworks:tsig-keys/>
<ipworks:options/>
<ipworks:views/>
```

This example illustrates two important aspects of the templates:

1. The templates provide a method for configuring statements that are not otherwise managed by IPWorks.
2. A template can include formatting information for other objects. In this example, the three tags at the bottom of the template include the formatted configuration of the TSIG keys, options, and views for the server in question. For more information on formatting data, see Section 11.4 on page 58.

11.2 Basics of Writing Templates

All format templates are xml documents and must conform to the rules of valid xml documents. This section is not a complete review of those rules, but will cover some of the basics in writing a valid xml document, specifically addressing the issues that are important in writing templates.

In general, the formatter works by starting with the template and then replacing all the formatting tags in the template with the appropriate text given the current context. The rest of the text in the template is preserved, with the exception of whitespace which can be preserved, but is not always preserved. For more information on the `<ipworks:body>` tag, see Section 11.3.1 on page 56.

In XML, all tags must be properly terminated. Each start tag must have a corresponding end tag or the tag must use the syntax for an *empty tag*. The term element is used to refer to the start tag, the end tag and the contents between the tags. Elements can contain other elements, but they must be wholly contained — it is not valid to start an element inside another element and then end it outside of that element. In XML, the start tag can also include attributes to specify additional information about the element (besides the tag name).

Here are some examples of valid XML:

```
<tag>This is the content</tag>
<empty-tag/>
<outer><inner><innermost/></inner></outer>
<tag attr="val1" attr2="val2"/>
<!-- This is an xml comment -->
```

Another important thing to understand about xml, is that there are some characters in xml that have special meaning to the xml parser. If the user wishes to include these in a template, there are standard constants (known as entities) that the user can use in their place. These entities are summarized in Table 7:



Table 7 Entities (Standard Constants)

Character	Entity	Xml example	Generated output text
<	<	3 < 5	3<5
>	>	<test>	<test>
&	&	now&then	now&then

There are many more features to xml and defining xml documents. This section provides a basic overview to develop formatting templates.

When the formatter begins to format text using a template, it uses an xml parser to process the template. However, it only processes the xml tags that begin with the prefix `ipworks:`. All other elements are ignored. This allows the user to include other xml tags in the templates, or even html tags, and they will not interfere with the parser. The only other processing that is handled by the formatter is that any xml comments in the template are deleted.

11.3 Basic Tags

This section introduces some of the basic tags, particularly those that are used for managing whitespace in the formatted output.

11.3.1 <ipworks:body>

This tag is the top-level element that contains all the text and other formatting tags that define a formatting template. All templates must contain only one of these elements. If a template does not include a `<ipworks:body>` tag, the entire template text is automatically wrapped in one of these elements.

Besides providing a container for the template, this element serves one other purpose — it defines how whitespace (spaces, tabs and newlines) must be managed within the formatted output. There are basically two strategies:

1. Design the template so that all the whitespace (spaces and newlines) are preserved, and then only the `ipworks` formatting tags will be processed by the formatter.
2. Design the template so that all whitespace in the template is condensed, and then control how whitespace is inserted into the formatted output text by using special tags for inserting whitespace where it is needed.

There is no right way or wrong way to approach this. In some templates, it is easier to use the first approach and in some templates it is easier to use the second. The formatter allows use of either approach.

If there was no `<ipworks:body>` tag the default behavior is to preserve whitespace. If the `<ipworks:body>` tag is specified, the default behavior



is to not preserve whitespace unless the tag explicitly specifies the `textmode` attribute indicating that it should be preserved. A description of the `<ipworks:body>` attributes is presented in Table 8.

Table 8 Attributes for <ipworks:body>

Attribute	Description
textmode	Defines how the text inside the body should be processed. There is currently only one legal value - literal - which means that all whitespace is preserved exactly as it appears. If not specified, all occurrences of whitespace in the text will be condensed to a single space, and text inside tags is trimmed so the leading and trailing whitespace is deleted.
ref	Allows the user to define a reference name that can be used to refer to the root object for the format. For information on References, see Section 11.5.3 on page 63.

11.3.2 <ipworks:newline>

This element is used for inserting newlines into the formatted text. It can be used even if whitespace is being preserved. A description of the `<ipworks:newline>` attributes is presented in Table 9.

Table 9 Attributes for <ipworks:newline>

Attribute	Description
count	Specifies the number of newlines to be inserted.

11.3.3 Prefixes and Suffixes

Many of the formatting tags allow the user to define prefixes or suffixes to be inserted in the formatted text. These are literal text values that are inserted either before (prefixes) or after (suffixes) the rest of the formatted content of that element. These attributes can also be used to control whitespace if the literal value to be inserted contains whitespace in it. See Section 11.1.2 on page 53.

A description of the common attributes for managing whitespace is presented in Table 10.

Table 10 Common Attributes for Managing Whitespace

Attribute	Description
prefix	A literal text value to be inserted into the formatted output before the contents of the formatting element that specifies the prefix.
suffix	A literal text value to be inserted into the formatted output after the contents of the formatting element that specifies the suffix.



11.4 Formatting Data

This section covers some of the basic formatting elements that are used to insert data into the formatted output. One thing that is important to understand is that the Formatter maintains some context information during the processing of the formatting template. This context always includes information such as, which object is the default object to use for applying formatting tags. Under some situations, there may even be a default field, and even a default value if a field has multiple values. To understand how these defaults are determined, the user may wish to review the section that describes iteration, though this is not necessary to fully understand how the elements described in this section work. For more information on Iteration, see Section 11.5 on page 61.

11.4.1 `<ipworks: class>`

This element inserts the name of the storage class of an object. If an object reference is not specified with the object attribute, it prints the class for the default object. A description of the `<ipworks: class>` attributes is presented in Table 11.

Table 11 Attributes for `<ipworks:class>`

Attribute	Description
object	The named reference of the object to use. For more information on References, see Section 11.5.3 on page 63.
prefix	Text to insert before this element's formatted output. For more information on Prefixes and Suffixes, see Section 11.3.3 on page 57.
suffix	Text to insert after this element's formatted output. For more information on Prefixes and Suffixes, see Section 11.3.3 on page 57.

11.4.2 `<ipworks:key>`

This element inserts the key values of an object. If an object reference is not specified with the object attribute, it prints the key values for the default object. A description of the `<ipworks:key>` attributes is presented in Table 12.

Table 12 Attributes for `<ipworks:key>`

Attribute	Description
delimiter	The text to use between key values if there are multiple key values for the object. If not specified it defaults to a single space " ".
scoping	Specifies whether the key values determined by the current scope are hidden. Since all objects that are being formatted have the same scope, this reduces the output text and simplifies the formatted text. The value must be "yes" or "no" and the default is "yes".



Table 12 Attributes for `<ipworks:key>`

Attribute	Description
object	The named reference of the object to use. For more information on References, see Section 11.5.3 on page 63.
prefix	Text to insert before this element's formatted output. For more information on Prefixes and Suffixes, see Section 11.3.3 on page 57.
suffix	Text to insert after this element's formatted output. For more information on Prefixes and Suffixes, see Section 11.3.3 on page 57.

11.4.3 `<ipworks:field>`

This element inserts the name of a field. If a field name is not specified with the field attribute, it prints the name for the default field. A description of the `<ipworks:field>` attributes is presented in Table 13.

Table 13 Attributes for `<ipworks:field>`

Attribute	Description
field	The name of the field to use.
prefix	Text to insert before this element's formatted output. For more information on References, see Section 11.5.3 on page 63.
suffix	Text to insert after this element's formatted output. For more information on References, see Section 11.5.3 on page 63.

11.4.4 `<ipworks:value>`

This element inserts the values of a field for an object. It can be used to insert a single value or multiple values, or a subset of the values for a field. It can also be used to insert a literal value in the formatted output. The algorithm for processing this tag is as follows:

- If a literal value is specified with the value attribute, that is inserted in the formatted output.
- If a field is explicitly specified, the values for that field for the appropriate object are formatted (based on the other attributes). If an object reference is not specified with the object attribute, it uses the default object
- If a field name is not specified with the field attribute, it uses the default field for the default object. If there is a default value, only that value is formatted, otherwise all the values for the default field are formatted (based on the other attributes).

A description of the `<ipworks:value>` attributes is presented in Table 14.



Table 14 Attributes for <ipworks:value>

Attribute	Description
delimiter	The text to use between values if there are multiple values to format. If not specified it defaults to a single space " ".
count	If specified, the number of values that must be formatted. If there is only one value to be formatted this is ignored. In addition to a numeric value, the user can also use the values "all" or "*" to denote all values. The default is *.
format	If specified, this is a sprintf-style formatting expression that is applied to each value before it is printed. This allows control of various features such as width and left or right justification.
object	The named reference of the object to use. This is only used if a field is explicitly specified. For more information on References, see Section 11.5.3 on page 63.
field	The name of the field to use.
value	If specified, this is a literal text value to insert into the formatted text.
prefix	Text to insert before this element's formatted output. For more information on Prefixes and Suffixes, see Section 11.3.3 on page 57.

11.4.5 Applying Template to a Single Object Formatting Example

This template called `example2`, is written in a way to show how some of the formatting directives work. The resulting output represents what would be generated by applying that template to a single object:

For example:

```
<ipworks:body>
<!-- The first line -->
<ipworks:class suffix=":"/><ipworks:key prefix=" " />
<ipworks:newline/>
<!-- The second line -->
...DnsName:<ipworks:value field="DnsName" prefix=" " />
<ipworks:newline/>
<!-- The third line -->
<ipworks:field field="hostname" prefix="..." suffix=": " />
<ipworks:value field="HostName"/><ipworks:newline/>
<!-- The fourth line -->
<!-- The fourth line -->
<ipworks:value value="...Domain: " />
<ipworks:value field="Domain"/>
<ipworks:newline/>
<!-- The fifth line -->
...Address:
<ipworks:value field="Address" delimiter=","
format="%10s" />
<ipworks:newline/>
```



```
</ipworks:body>
```

This format would produce the following output:

```
IPWorks> list dnsserver test.com -format=example2
Dnsname: test.com
...DnsName: test.com
...HostName: test
...Domain: com
...Address: 1.2.3.4, 1.2.3.5
```

11.5 Iteration

One of the most powerful aspects of the Formatter is that it allows the user to apply a set of formatting elements to sets of objects, fields or values by iterating over all the items in a set.

11.5.1 **<ipworks:for>**

This element provides the basis for iteration in the Formatter. The attributes in the start tag define either a set of objects, a set of fields, or a set of values. The contents of the `<ipworks:for>` element (which can include other formatting elements) are then applied to each item in the set, one at a time.

The `<ipworks:for>` elements can be nested within each other, so that loops could contain other loops for example. During the processing of an `<ipworks:for>` element, the Formatter maintains a scoped stack, much like a compiled programming language does, so that the state and context of the inner loops does not interfere with the context of the outer loops.

When a new loop is started, the current context gets changed, depending on the type of loop. If the loop is iterating over a set of objects, the default object inside the loop will be the one that is the current object from the iteration. The object that was the default object prior to the start of the loop can only be referenced inside the loop using references. For more information on References, see Section 11.5.3 on page 63. Similarly, if the loop is iterating over fields, or over values, the default field or value or both will be changed as iterated.

Note: A loop that iterates over a set of fields does not change the default object.

To iterate over a set of objects, the user must specify a relationship (with the related attribute) that will be used to retrieve the objects that will be used. Thus the user can only iterate over objects that are related to another object. To iterate over fields or values the user must use the each attribute to specify the type of iteration. One of these two attributes must be specified on each occurrence of a `<ipworks:for>` element.

A description of the `<ipworks:for>` attributes is presented in Table 15.



Table 15 Attributes for <ipworks:for>

Attribute	Description
related	The name of the relationship that will be used. If specified, this implies that the loop will be iterating over a set of objects.
object	The named reference of the object to use as the source object to find related objects. If not specified, the default object is used. This may also be used when iterating over fields to specify the object whose fields should be used for the iteration. For more information on References, see Section 11.5.3 on page 63.
each	The type of iteration to perform. Must be one of the following values: <ul style="list-style-type: none">• value: Iterates through all the values for a field.• fieldinobject: Iterates through all the fields that have values in an object.• fieldinclass: Iterates through all the fields that are defined for a class.• fieldinselection: Iterates through all the fields that were selected by the user.• fieldinlist: Iterates through the fields in a list.
field	The name of the field to use when iterating over values (each="value"). If not specified it uses the default field.
class	The name of the class to use when iterating over the fields in a class (each="fieldinclass"). If not specified, the class of the object specified in the object attribute is used, and if that is not specified, the default object's class is used.
list	The list of fields to use when iterating over the fields in a list (each="fieldlist"). This should be a comma-separated list of fields, and can include named fieldlists.
ref	Allows the user to define a reference name that can be used to refer to the current object for this loop from within inner loops if needed. For more information on References, see Section 11.5.3 on page 63.

11.5.2 Formatting Example Including Nested Loops

Here is a format template that includes nested loops, using two different types of loops and shows the use of the data formatting tags working with the default items during the iteration:

```
<ipworks:body>
<ipworks:for each="fieldinselection">
<ipworks:for each="value">
<ipworks:field/>:<ipworks:value prefix=" " />
<ipworks:newline/>
</ipworks:for>
</ipworks:for>
<ipworks:newline/>
</ipworks:body>
```



This format would produce output similar to the following:

```
IPWorks> list dnsserver test.com
DnsName: test.com
HostName: test
Domain: com
Address: 1.2.3.4
Address: 1.2.3.5
IPWorks>
```

11.5.3 References

There may be situations within a formatting template where the user is inside a loop and the user would like to insert a piece of data from an object that is no longer the default object, but is still within the scope of an outer loop, or perhaps the root object that the entire formatting template is being applied to. The formatter provides the ability to do this using references.

When an object is first referenced in a formatting tag (either the `<ipworks:body>` or `<ipworks:for>` tag), the user can associate a reference name with that object, using the `ref` attribute. All the data formatting tags that can format data from an object have an object attribute that the user can use to specify the reference name for the object the user would like to use instead of the default object. This allows users to refer to any object that is in scope at any level, provided the reference name for the object is known.

11.6 Conditions

The Formatter also provides some tags that can be used for conditionally including parts of the formatted text.

11.6.1 `<ipworks:if>` and `<ipworks:ifnot>`

These two elements are identical in syntax. Both elements test a condition to determine whether the contents are processed by the formatter for inclusion in the formatted output. If the condition is true for the `<ipworks:if>` element, its contents are included. Otherwise they are completely omitted. The behavior for the `<ipworks: ifnot>` element is exactly the opposite.

The type of test to be performed is specified with either the `matchclass`, `matchkey` or `matches` attribute. They cannot be combined. If none of these attributes are specified, the test is considered to match if the test field has a non-null value. Table 16 presents a description of the `<ipworks:if>` and `<ipworks:ifnot>` attributes.

Table 16 Attributes for `<ipworks:if>` and `<ipworks:ifnot>`

Attribute	Description
object	The named reference of the object to use for the test. If not specified, the default object is used. For more information on References, see Section 11.5.3 on page 63.
matchclass	A wild card expression that must match the class name of the test object in order to pass the test.
matchkey	A wild card expression that must match the key value. If there are multiple key fields, it must match the last key value.
matches	A wild card expression that must match the value of the test field.
field	The name of the field to use for the test. If not specified it uses the default field.

11.6.2 Skipping Fields without Values Formatting Example

To refer to the formatting example that included nested loops, see Section 11.5.2 on page 62. This formatting example can be improved slightly by using a `condition` in the format. The original definition has a minor problem in that it prints out fields even if they do not have values. This can be improved by changing the template to the following, so that fields without values are skipped:

```
<ipworks:body>
<ipworks:for each="fieldinselection">
  <ipworks:if>
    <ipworks:for each="value">
      <ipworks:field/>:<ipworks:value prefix=" "/>
      <ipworks:newline/>
    </ipworks:for>
  </ipworks:if>
</ipworks:for>
<ipworks:newline/>
</ipworks:body>
```

11.6.3 `<ipworks:else>`

This element can only be used in conjunction with the `<ipworks:if>` and `<ipworks:ifnot>` elements. Although it is somewhat awkward to express this using the xml syntax, the `<ipworks:else>` element should be included in the content of the test element it is being used with. The contents of the `<ipworks:else>` element only is used if the test that encloses it has failed. Here is a fragment of a template that illustrates how these tags are used:

```
<ipworks:if field="dnsname" matches="*.domain.com">
  The name is in domain.com!
<ipworks:else>
```

```
        The name is not in domain.com!  
    </ipworks:else>  
</ipworks:if>
```

11.7 Including Other Formats

The Formatter provides a mechanism for inserting formatted text from another template into the text that is being formatted for the current template. As defined earlier, the formatter ignores all xml tags that do not begin with the `ipworks:` prefix. However, when it is processing a tag that does begin with this prefix, if it is not one of the predefined tags (all of which are defined in this document), the Formatter assumes the tag is a reference to a named template, where the name of the template is the name that follows the `ipworks:` prefix

The Formatter then processes the named template and applies the template to the default object, adding all of the text to the current formatted output. This makes it very easy to structure the formats and include them within one another.

<ipworks:conf>

This tag is used to insert the configuration text for the default object into the current formatted output. This is a second method that can insert the contents of other formats into the current formatted output. For a detailed description of how an object's configuration text is determined, see Section 11.1.3 on page 54.

11.8 Prematurely Terminating a Format

There may be situations during the formatting of a template where the user would simply like to terminate the formatting. This can be accomplished with the elements as follows:

<ipworks:exit>

This element can be used to completely terminate the formatting. If the current format was invoked by another format then formatting of all templates is terminated immediately. If the user specifies an error attribute, not only will the formatting be terminated, but an error message (including the text the user specified) will be displayed to the user. A description of the `<ipworks>` attributes is presented in Table 17.

Table 17 Attributes for `<ipworks:exit>`

Attribute	Description
error	An error message to be displayed to the user. If not specified, the formatting is terminated, but no error is displayed.

<ipworks:return>



This element is very similar to the `<ipworks:exit>` element, except that it only terminate the current format. If the current template was invoked by another template, the formatting will continue in that template after the tag that invoked the current format.

This can be particularly useful in the beginning of templates that only apply to certain types of objects, as in the following example:

```
<ipworks:ifnot matchclass="DnsServer"><ipworks:return/>
</ipworks:if>
```

11.9 Predefined Templates

IPWorks includes several predefined templates that can be used to format any object. There are also many other predefined templates that are used for configuring the managed objects. The generalized formats that can be used with any object are listed in Table 18.

Table 18 Generalized Formats for Objects

Name	Description
cli	The default format for CLI output. Prints the object's class, key values, with fields indented on subsequent lines — multi-valued fields have all values on one line separated by commas.
brief	Displays only the object's class and key values.
conf	Displays the object's configuration text.
ldif	Similar to the LDIF interchange format. Also similar to the default CLI output format, but multi-valued fields are displayed with one value per line, with the field name repeated on each line.
xml	Displays the object using an XML format. This format can be used with the CLI's <code>import xml</code> command to reload the object.



Reference List

Ericsson Documents

- [1] *Trademark Information*
- [2] *Typographic Conventions*
- [3] *Glossary of Terms and Acronyms*
- [4] *IPWorks Configuration Management*
- [5] *Configure DNS and ENUM*
- [6] *Configure User Account*
- [7] *IPWorks AAA Parameter Description*
- [8] *IPWorks DNS, ASDNS, ENUM Parameter Description*
- [9] *IPWorks DHCP Parameter Description*
- [10] *Configure DHCP*