vocal point logo

Programmer's Reference

Nina API, Release 0.8 Beta

By: Vocal Point Engineering

Draft 2 **Last Modified March 6, 2000 2:43 pm**

DRAFT

Contents

List of Tables

List of Figures

1 Introduction

This document is a programmer's reference for the Nina interface from Vocal Point, Inc.

NOTE: *This document is a preliminary draft.*

1.1 What is Nina?

Nina is a voice-activated browser that allows users to navigate the World Wide Web over the telephone. Using Nina, Internet service and content providers can quickly voice-enable any of their existing HTML content. Nina can synthesize speech for text content, play audio files, or fill out Web form to access back-end databases for purposes such as e-commerce transactions.

Nina includes a custom-developed API interface and set of gateway components. The APIs provide the interface to the browsers using *extended cascading style sheets* (xCSS). The gateway provides the CTI/speech recognition necessary to connect telephone users to the browsing services.

You can use the entire Nina package provided by Vocal Point, Inc., or you can develop your own gateway interface using the Nina API libraries.

1.1.1 Key Features

Key features of Nina include:

- Accessibility from any standard wireline or wireless phone.
- Audio prompts to guide the user through each interaction, similar to a voice menu system.
- Nina supports both speech commands and dial-tone multi-frequency (DTMF) input.
- Users can bypass the navigation prompts, or even enter a sequence of consecutive commands, at any point.
- Text-to-speech and audio file support. Content providers can use either pre-recorded audio files (recommended) or can use synthetic speech output if their content is too dynamic for pre-recorded audio to be feasible.
- Nina supports standard HTML features, including forms, tables, and data input.
- Nina is deployed in clusters of distributed servers in an n-tier architecture for scalability.

1.1.2 Industry Standards

The core of Nina is a semantic processing engine that takes ordinary HTML content and applies the metadata needed to make that content suitable for audio browsing. In order to accomplish this, Vocal Point has created an extension the WWW Consortium's Cascading

Style Sheets standard to facilitate voice-audio navigation. This extended standard is called Extended Cascading Style Sheets, or xCSS.

1.1.3 What Type of Enterprise Uses Nina?

Nina has a wide range of potential uses. For example, organizations that can make use of Nina include:

- Equipment vendors, for example vendors of unified messaging systems [PBX]
- Telecom/wireless service providers
- Internet service and content providers
- Any network service provider
- Third-party site hosts

1.1.4 Audience for this Document

This document is written for systems integrators and developers who will be implementing Nina. You should possess the following background knowledge:

- Strong working knowledge of Microsoft Visual C++ 5.0 or Java 1.2
- Familiarity with development on Microsoft Windows NT environments
- HTML and Version 2 of cascading style sheets
- Appropriate industry background for your organization

1.2 System Requirements

In order to use and develop for Nina, your system must meet these requirements:

- Pentium II class or better CPU
- 128MB RAM
- Microsoft Windows NT 4.0, Service Pack 3 or later
- Sun Microsystems Java JDK 1.2

The API libraries are delivered for the following platforms:

- Java JDK 1.2
- Microsoft development platform:
  - Microsoft Visual C++ 5.0 with Service Pack 3 or later
  - Microsoft Windows NT 4.0 with Service Pack 3 or later

1.3 About This Document

Chapters subsequent to this introduction are as follows:

- Chapter 2: Architecture on page 13 describes Nina components and process flow.
- Chapter 3: Installing Nina on page 23 describes how to install Nina.
- Chapter 4: Using Nina on page 25 describes how to configure Nina for your site.
- Chapter 5: Server Communication on page 33 describes the message streams used by the various servers.
- The Attachments provide supplemental information:
  a. Appendix A: Terms and Acronyms on page 45 contains a glossary.

1.3.1 Typographical Conventions

This document uses the following typefaces to indicate special instructions or values:

- Monospace font indicates code, filenames, pathnames, and other absolute values. For example:

Use a text editor like `vi` or `emacs` to edit this file.

- Monospace bold indicates something that you type.
  `c:\applications\utilities>` **`install.sh mystuff`**
- Italic font indicates a value that you replace with the correct one. Monospace italic occurs where a value within a command string should be replaced by the correct one
  `install.sh` ***`file`***
- The standard conventions apply for command-line syntax such as options.
  `install.sh -user` ***`username`*** `[-m | -write]`
- Special terms appear in *italic* when they are first introduced to the reader. You can find a glossary of all special terms in Appendix A: Terms and Acronyms on page 45.

1.3.2 Text Condition Tags

Conditional text tags mark special portions of text that can be shown or hidden, depending on the document's condition or readership.

- Draft text is purple with a change bar to the left. Hiding Draft also hides the DRAFT notice on the bottom of each page. Draft text may be shown to customers for pre-release versions of the documentation or software.

2 Architecture

This chapter describes Nina core components and process flow. Essentially, the process is to translate onlinear “gestalt” data on a web page into a linear, audio format suitable for presentation over the telephone. Because the audio format is not suitable for playback of large amounts of data, the page must be reduced to its most essential elements, and these elements presented to the user as simple choices, in the optimal priority order. The user’s experience is similar to a hierarchical voice menu system.

2.1 Nina Process Flow

Nina comprises both server and gateway components.

- The Nina Server contains the following:
  - The Nina server application, running as an NT Service. This application provides the extended browsing capability needed for voice interaction. Nina applies the extended cascading style sheets (xCSS) associated with the page to determine which items on the web page should be presented to the user. Nina then converts these web page options into voice menu format.
- The Nina Gateway is a separate server that provides the interface between the telephone user and the browsing component. Vocal Point offers a gateway implementation as a separate option. However, if you want to provide your own, you will need:
  - Computer telephony interface (CTI)
  - Automatic speech recognition interface (ASR)
  - Text-to-speech interface (TTS)
- Additional components are:
  - A resource manager (RMAN) for load balancing and status monitoring
  - A logging server that receives logging messages from all the other servers

NOTE: *VocalPoint provides separate API reference documentation in HTML format.*

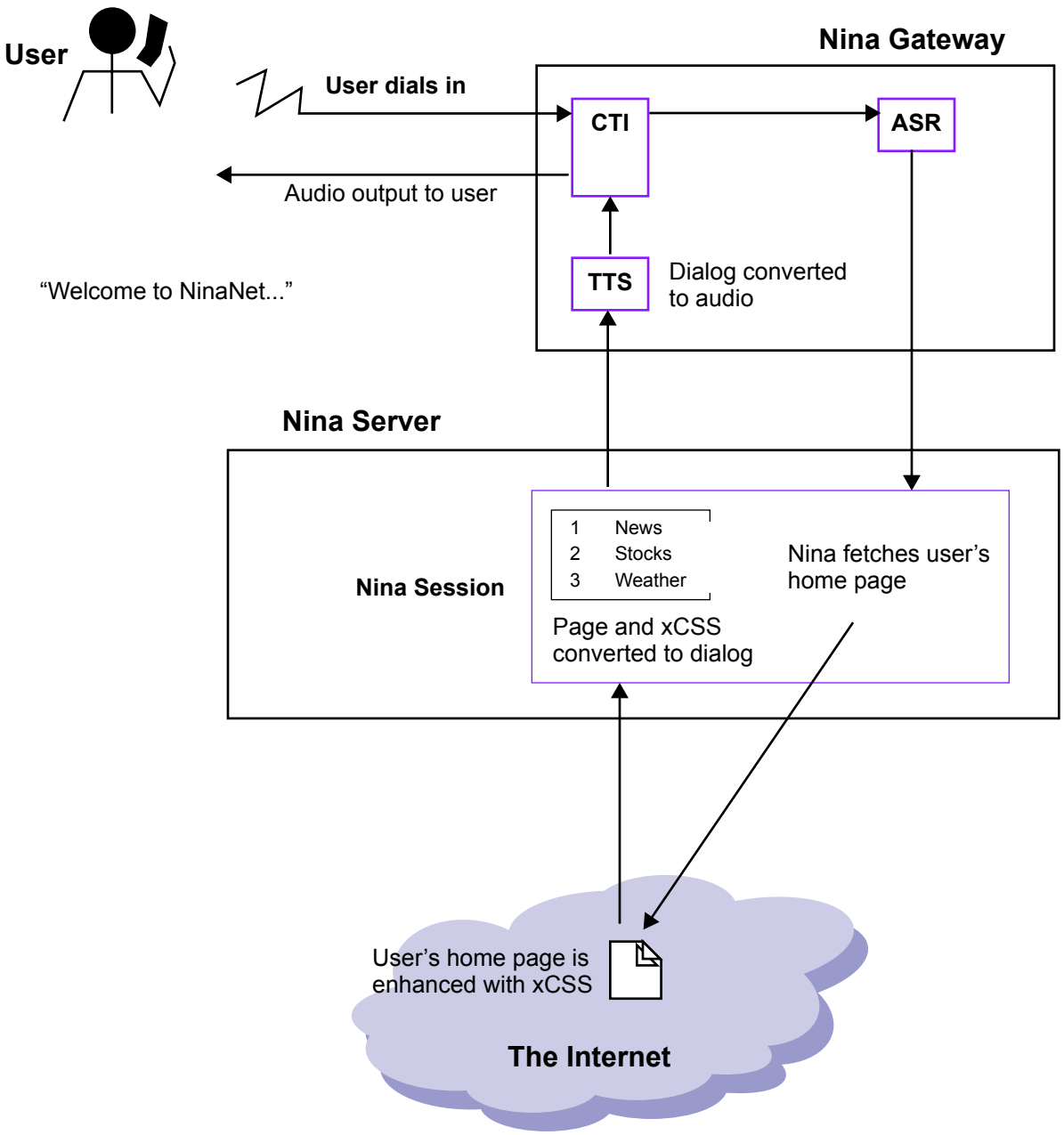
2.1.1 Illustrated Flow

Figure 2.1 on page 15 illustrates the end-to-end process flow between the end user, Nina, and the Internet pages the user wishes to browse. For the purposes of this example, the Internet telephony browsing service will be called “NinaNet”.

➠ *When a user dials in*

1. The user dials into NinaNet using a standard PSTN telephone line or a wireless handset.
2. The call is directed to the CTI component on the gateway. This component manages the call connection.
3. On the Nina gateway:
   a. The Nina CTI component sends the audio stream to the ASR component.
   b. The Nina ASR component translates the audio speech input into a text stream.
4. The Nina gateway initiates a session with the Nina server.
5. The Nina server gets the user’s home page.This page contains special modifications to include the xCSS comments.
6. Nina interprets the xCSS on the user’s homepage to determine which portions of the page content will be presented to the user, and in what order.
7. The Nina dialog generator creates dialog of options that will be presented to the user as prompts. This dialog is a hierarchical list of choices and some instructions on how to make the selection.
8. Nina sends the dialog to the text-to-speech (TTS) component on the Nina gateway.
9. The TTS component outputs the dialog to the CTI in audio format.
10. The CTI on the gateway sends the converted speech message to the user, who listens to the choices and chooses the desired option.
11. Input from user choices determines ongoing Internet navigation.
12. When the user hangs up the phone, the CTI notifies the other Nina components and terminates that particular session.

Figure 2.1 Nina Processing Flow



2.2 Server Architecture

Nina is a standalone server that provides the core functionality of a Web browser, but replacing the graphical navigation normally associated with point-and-click Web browsers with a streamlined series of prompts that the user listens to and then selects from, over the telephone. The gateway provides the interface between the user and the Nina server.

The gateway components and Nina are designed to be deployed as part of an n-tier distributed server architecture. For example, there could be:

- A gateway cluster with one or more servers for the CTI, ASR, and TTS functions. Each instance of these functions can spawn multiple individual sessions (one session of each type per call).
- One or more Nina servers, each running a Nina process. Each Nina process can have multiple sessions. Each session represents one phone call.
- A resource manager (RMAN) to handle load balancing and keep track of all the sessions across all the machines. RMAN can be one or more processes.
- A logging server that handles logging messages from all the other servers.

Each server is configured to specify the maximum number of each type of session it can accept. The resource manager uses this information when deciding which machines to start sessions on for a particular call.

2.2.1 Server Message Streams

All the Nina components (Resource Manager, gateway components, Nina server, and logger) communicate using a well-defined protocol. This protocol uses a message passing scheme where acknowledgments are returned based on conventions defined by Vocal Point, rather than any underlying mechanism such as RPC or CORBA.

The Nina messaging protocol includes the following message streams:

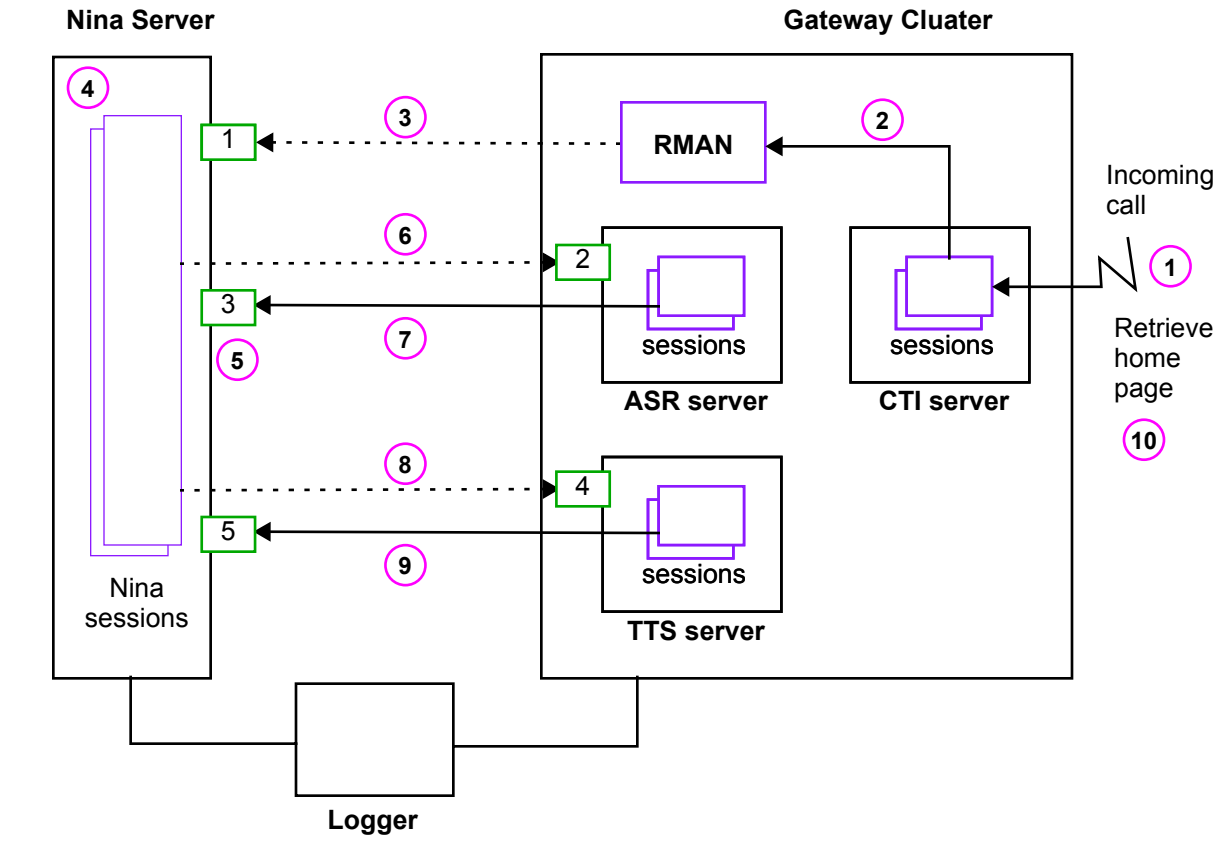
- A control stream to open or close session, check on status or shut down the server. This stream lasts long enough for one message to be sent in both directions.
- An asynchronous bi-directional input stream consisting of an input method and input results. This stream typically remains connected for the duration of a session.
- A synchronous output stream presenting results of parsed HTML/xCSS (i.e., the output from dialog builder) that the gateway text-to-speech converter must translate into audio format for the user on the telephone.
- An asynchronous log message stream.

Figure 2.2 on page 17 shows the message streams used by Nina.

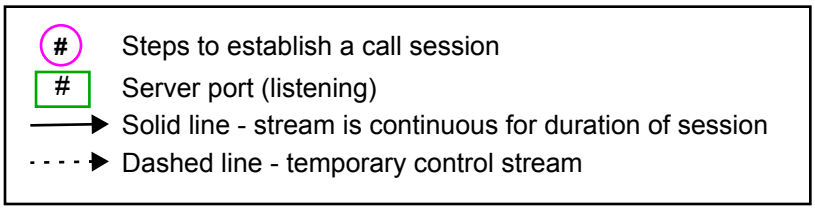
➠ *Message stream sequence*

1. An incoming call connects to the CTI.
2. The CTI generates a unique SessionID and sends a message to RMAN.
3. RMAN opens ASR and TTS sessions on servers with free resources, and sends a control stream to Nina telling Nina to create a session.
4. Nina creates a session and asks RMAN for the [host,port] location of the TTS and ASR resources. RMAN replies specifying specifying ports 2 and 4 .
5. Nina creates two TCP/IP sockets, one for accepting the ASR connection and one for accepting the TTS connection. These are ports 3 and 5 .
6. Nina sends a control stream to the ASR session on port 2 .
7. The ASR connects with the Nina session on port 3 .
8. Nina sends a control stream to the TTS session on port 4 .
9. The TTS connects with Nina on port 5 .
10. Nina loads the user's home page.

Figure 2.2 Nina Messaging Streams



Key



Not shown in figure: This diagram highlights message streams between the gateway and the Nina server. Not shown are:

- Ports for CTI and RMAN.
- RMAN control of each element.
- Individual mesage streams between all the gateway elements.
- Individual message streams between the logger and each server in the cluster.

See Chapter 5: Server Communication on page 33 for details on the messages used by the various message streams.

2.2.2 Sessions

Each time a user connects, a new session is established between the gateway and the Nina server. That session persists only as long as the user is connected. Within each type of server

(CTI, ASR, TTS, and Nina) there is one corresponding session for the call, identified by a unique Session ID.

2.2.3 Ending a User Session

A session is ended when the user hangs up the telephone. Ordinarily, the CTI notes the call being dropped, and informs the ASR. The ASR then informs Nina. To assure proper cleanup, Nina informs the TTS resource as well.

2.3 Resource Manager (RMAN)

The Resource Manager (RMAN) manages resources on a cluster of server machines. Resources are defined as individual sessions for each type of server. RMAN tracks the number of available and used sessions on each server, for example the number of ASR sessions, Nina sessions, etc. The number of available sessions are the *resources* on that server.

RMAN continually broadcasts on the LAN to a well-known UDP port that is listened on by every Service Registry on every machine. The Service Registry responds with a UDP packet describing the state of the machine it is watching. This results in the following:

- RMAN builds up a global view of the current status of the network.
- Each Service Registry builds up a list of available RMANs.

RMAN itself can be configured to have multiple instances, with one RMAN designated as the “primary” instance. This section discusses a configuration where RMAN uses local processes called the Service Registry that run on each individual server.

2.3.1 Service Registry

Each machine has a service registry where services running on that machine register themselves at startup. Each service registry then provides the world (network) with information about what services are active on a given machine.

The Service Registry pings every service that has registered with it (every N seconds) to assess its status.

The Service Registry can run on separate machines, or it can be part of RMAN.

2.3.2 RMAN and Service Registry Functionality

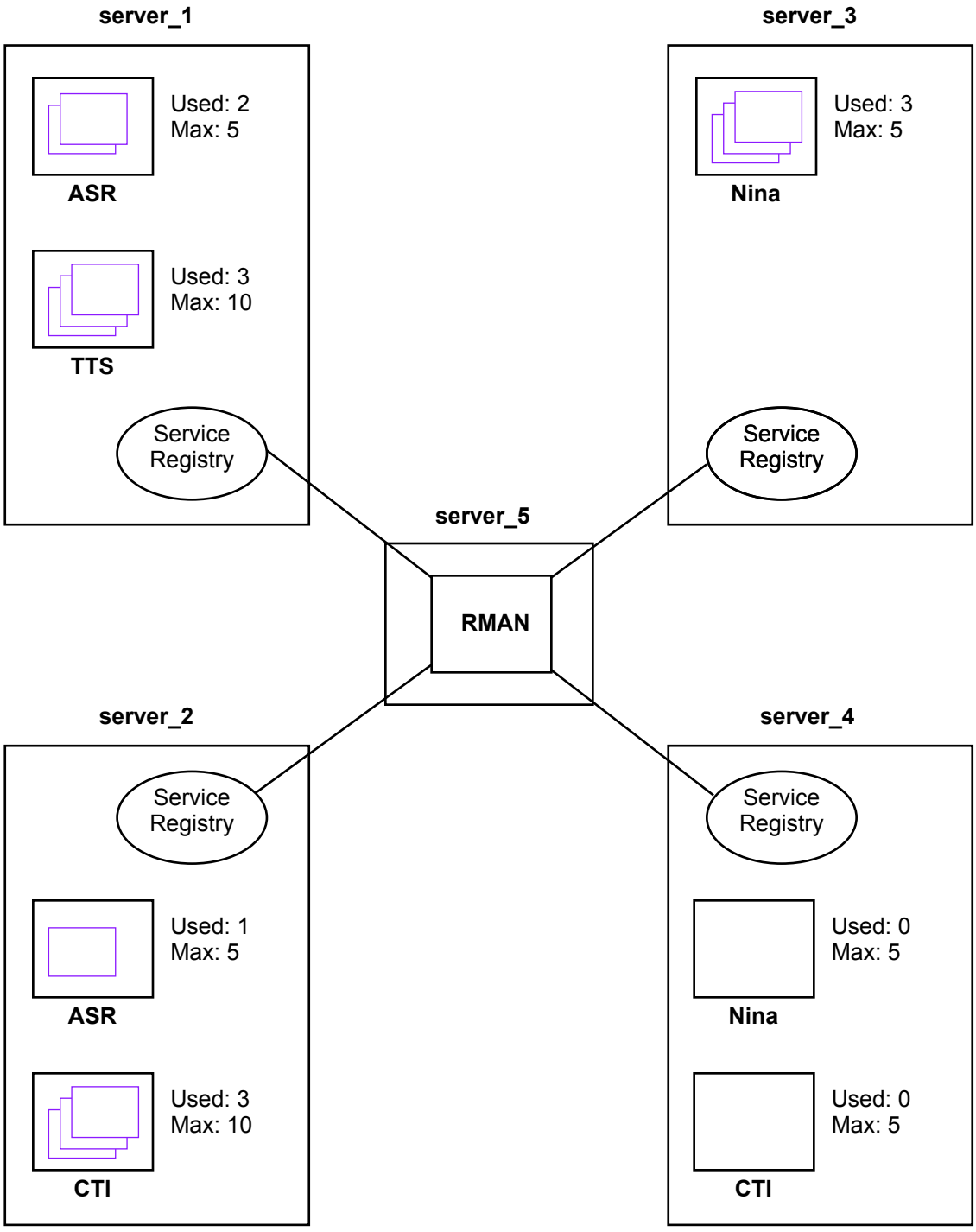
How RMAN works:

- RMAN broadcasts on the same port to all machines on the network.
- The Service Registry on each local machine responds to that port with:
  - The type of services running on that machine (for example, “ASR” or “CTI”)
  - For each service type, the number of active sessions and the maximum capacity for that service type.
- When a call comes in, the CTI detects it and notifies RMAN, which then allocates the appropriate ASR, TTS, and Nina sessions on machines with free resources (available sessions) of the appropriate type.

2.3.3 Sample RMAN and Service Registry Configuration

Figure 2.3 shows a sample configuration with Nina, ASR, TTS, and CTI services distributed over sevaral servers. In this figure, there are three active calls in progress. Note that there are a total of three sessions of each type corresponding to these calls.

Figure 2.3 RMAN and Service Registry



2.3.4 Specifying the Maximum Number of Sessions

You can specify the maximum number of allowable sessions of each service type that is installed on a machine by editing the configuration file for that service. See "Configuration Files" on page 20 for more information.

2.3.5 RMAN and Load Balancing

Currently, RMAN allocates new session to the server with the greates tpercentage of free sessions of that service type.

2.4 Configuration Files

A number of configuration files are used by the various types of servers (CTI, ASR, TTS, and Nina). This section discusses only a few highlights of these files. For details, refer to the comments within the files themselves.

The `config.ini` file is used by servers to connect to the Service Registry. The options `ServiceRegistryHost` and `ServiceRegistryPort` specify the hostname and port for the service registry that the server should connect to. If the host is not defined, it defaults to use the local host.

For example:

```
ServiceRegistryHost: LocalHost
ServiceRegistryPort: 13333
```

2.4.1 Nina Server Configuration Files

Nina server uses the following configuration files:

- `iap.ini` is the Nina configuration file.
- `css.ini` specifies xCSS defaults such as the default greeting. Use this file to customize your message prompts across all pages. (To customize a particular page, edit the xCSS code associated with that page.) See "Using xCSS" on page 26 for details.

Table 2.1 Nina configuration file - key items

Configuration Item	Description
`HomePage`	URL for default home page to load, prior to user login.
`http.ProxyHost` `http.ProxyPort`	Proxy host information, if needed
`MaxSessions`	The maximum number of concurrent sessions to accept
`RegistrationTimeOut`	Maximum length of time to wait for registering with the Service Registry.
`PageLoadTimeout`	The maximum length of time in seconds to wait for a page to load.
`ninatmp`	Location of the Nina temporary directory
`AutoKeyWords`	Specifies whether to attempt to infer keywords from the page items
`ServiceRegistryHost` `ServiceRegistryPort`	Hostname or IP address and port of Service Registry host

Table 2.1 Nina configuration file - key items

Configuration Item	Description

[various]	The user navigation commands and their DTMF equivalents are configurable through the configuration file, as well as a variety of error and help messages.
	For example:
	`# Command for going to top of page:`
	`PageTopCommand: go to top`
	`PageTopDTMF: 02`

3 Installing Nina

4 Using Nina

This chapter describes the various configuration and customization tasks involved in implementing Nina, such as changing the message prompts for various navigation points.

4.1 The User Experience

In visual browsing, the user can scan and locate information within an HTML page in a non-linear fashion. The user can skip ahead, have multiple windows or frames open simultaneously, and can process or evaluate multiple information points simultanesously. In voice-activated browsing, information must be presented in a serial fashion because the medium of sound and voice is linear by its very nature.

To facilitate navigation, Nina generates a dialog-based interaction. Prompts similar to those in telephone voice menu systems provide navigation cues to guide the user at each step.

4.1.1 Nina's Browser Navigation Tree

With Nina, users navigate around an HTML document using a hierarchical menu technique called *step navigation*. In step navigation, the browser leads the user through each HTML document by generating prompts. The user selects one of the presented options and advances to the next step (or HTML page). Nina also includes standard browser commands such as Back and Home.

Step navigation is implemented by representing the HTML document using a *navigation tree*. The user navigates a document by following branches to visit different nodes in the tree.

Different types of *navigation nodes*are used to specify a different type of user-machine interaction.

- At a *routing node*, a prompt informs the user of the possible paths to outgoing child nodes. The user then selects the branch to follow and is advanced to another node in the tree. This approach allows novice users to find their way around a document. At each point, , the user is informed about the current session state and what options are available at that particular point in the navigation tree.
- At a *group node*,
- A *content node* contains text and generic markup elements. Content nodes cannot be visited by navigation commands

4.1.2 Sequence for Navigating a Node

Each time the user visits a node, a particular sequence of events occurs.

➠ When the user first visits a node

1. **Greet.**

   The user hears a short description or label that names the current node. The default greeting is “Welcome to *label*.”

2. **Prompt.**

   The user hears different options depending if the node is a routing node or a group node.

   - ❖ For routing nodes, the default prompt is “Your options are: *label 1*, *label 2*, ... “ where each *label* corresponds to a child node.
   - ❖ For group nodes, the default prompt is to execute the `read` method.

3. **Respond.**

   After prompting the user, a navigation node waits for a response from the user and then processes the response accordingly.

   - ❖ Routing nodes advance the user to a child node when the user speaks the corresponding key.
   - ❖ At a group node, the user can request one of the following special commands:
     - “Repeat” re-lists the contents of the current node.
     - “Continue” advances the user to the “next” node.
     - “Links” generates a reading of the links at that node.
     - “Link me to” follows a link.

4.1.3 Methods Used for Node Navigation

Greet: The greeting is invoked when the user first enters a navigation node. The greeting attribute (or property) is read to the user. Note that content nodes do not greet the user, because they cannot be visited by navigation commands.

Prompt: Each navigation node can define its own prompt. The prompt is executed at the following points during navigation:

- After the greeting is completed, except when the user interrupts (makes a selection) during the greeting.
- When the user invokes the “Repeat” command

The prompt method does the following:

- For routing nodes, the prompt method informs the user of the available options.
- For content nodes, the prompt method plays (reads) the data.

4.2 Using xCSS

Both HTML and CSS are industry standards. It is assumed that readers of this document are already familiar with both of these terms and the use of the standards for displaying Web documents.

Extended cascading style sheets (xCSS) is an extension of the existing CSS 2.0 standards developed by Vocal Point for use with Nina.

You can embed the xCSS in the HTML source, or you can include it as a separate file that is applied by Nina to interpret the page.

4.2.1 CSS Selectors

The xCSS standard does not completely envelop all CSS items. There are CSS properties not supported for xCSS, and there are also new xCSS properties that are not part of CSS.

Existing CSS Selectors Used in xCSS: CSS selectors are used as a unique way to identify a point on the HTML page. Nina can use CSS selectors to find the portion of the page it should "read" to the user. The xCSS standard supports many (but not all) of the CSS selectors. Table 4.1 lists the CSS selectors currently used in xCSS.

Table 4.1 Existing CSS selectors supported for xCSS

CSS Selector	Description
`*`	Universal selector
E	Element selector. Example: H3
E F	Descendant selector. Example: H3 P
E > F	Child selector. Example: H3 > P
E + F	Preceded-by selector. Example: TABLE + UL
E:*first_child*	First-child pseudo-class
E[*attribute*}	General attribute selector Use the *attribute* selectors if you are a third party content enabler (i.e., you are not authorized to change the HTML source).
E[*attribute*="*name*"]	Named attribute selector. Example: `TABLE [width=70]`
E.*attribute*	Shorthand for E[attribute]
E#*id*	Use an ID in the HTML source to find the item. This is the preferred method, if you have access to the HTML source.

Notes on Using CSS Selectors: There are two ways to identify an area of the page that Nina should use.

- You can specify some sort of attribute, such as a table column of a particular width, that is unique to that portion of the page. This method refers to the HTML source, but you don't need to insert anything into the HTML source. However, if someone changes the HTML source and changes the attribute you are using, your xCSS will not work properly until you update it.
- You can use an ID that is stored in the HTML source. This is the preferred method.

4.2.2 New xCSS Properties

Table 4.2 lists the new properties that have been added to xCSS. These properties are not part of CSS.

Table 4.2 New properties for xCSS

New xCSS Property	Description
`prompt: {`*string*`}`	Provide a prompt for the user, where string is the word or phrase that the user should hear as the prompt.

Table 4.2 New properties for xCSS

New xCSS Property	Description
`greeting: {string}`	Provide a greeting the user will hear when visiting the node, where string is the word or phrase to use as the greeting.
`name: {string}`	Provide a label to use for this node. >
`clip: true|false`	If set to true, this node (and its subtree) will be clipped and SAVED.
`prune: true|false`	If set to true, this node (and its subtree) will be clipped and REMOVED.
`filter: true|false`	If set to true, this node will be removed and its children promoted into its place.
`pause-after: {time}`	Specify a time interval in milliseconds to pause after the content at this node is read (played)
`keyword: {string}`	Provide an access keyword for this node.
`convert-to: div|span`	Convert this node. The only acceptable targets currently supported are DIV and SPAN

4.2.3 Customizing the Message Prompts

The Nina system prompts can be modified through the configuration file.

All other prompts are generated from HTML content . Generally, every node has three important attributes:

- `label`
- `greeting`
- `prompt`

If a node is listed as a possible option for selection, its `label` is read to the user. Hence, the `label` should be a short, one- or two-word description of the purpose or content of the node. When a user visits a node by selecting it as an option, the `greeting` is read to the user, followed by the `prompt`.

➠ *To provide a label*

The preferred method of providing a label is through the xCSS `label` attribute. However, if no label is provided, Nina attempts to infer one using the following algorithm:

- If the node is a link, then the name of the link is used as a label.
- If the **`TITLE`** attribute is set, then the value of that attribute is used as a label.

➠ *To provide a greeting*

The preferred method of providing a greeting is through the xCSS greeting attribute. However, if no greeting is provided, Nina constructs a default greeting:

"`Welcome to` *`label`*`.`"

where *`label`* is the xCSS label attribute, or the default label.

You can change the default greeting by editing the `css.ini` file.

➠ ***To provide a prompt***

You can provide a prompt using the xCSS `prompt` attribute. However, if no prompt is provided, Nina constructs a default prompt:

"Your options are."

You can change the default prompt by editing the `css.ini` file.

4.2.4 Default CSS Style Sheet

Figure 4.3 shows the default style sheet from Vocal Point. The CSS defaults are provided as a file called `css.ini`. Nina loads the xCSS defaults from this file.

Figure 4.1 Example xCSS Style Sheet

```
/* Default style sheet */

@media speech {
  * {
    barge-in: true;
    label: "";
    keyword: "";
    node-next: "";
    greeting: "Welcome to $label";
    prompt: "Your options are:";
    pause-before: 0ms;
    pause-after: 0ms;
    filter-single-branch: true;
    prune: false;
    clip: false;
    filter: false;
  }

  FORM { prompt: " "; }

  INPUT { prompt: "Please input $label:" }
}
```

4.3 Sample Page Using xCSS

This section presents a sample HTML page that uses xCSS, and shows how that page is presented through a visual browser vs. a Nina-interpreted audio interaction.

4.3.1 Sample Code

Figure 4.2 shows a sample style HTML page with xCSS. (Line numbers have been added for discussion purposes, and are not part of the HTML.)

Figure 4.2 Sample HTML page with xCSS

```
1.   <html>
2.   <head>
3.     <title>My SpokenWeb!</title>
4.
5.     <style>
6.      @media speech {
7.         hr + table { label: "My Spoken Web, brought to you by Widgets.com.";
8.                      prompt: "Please select one of the following:" }
```

```
9.         td[width="30%"] ul { clip: true; label: "Movie Listings" }
10.        tr tr:first-child { prune: true; }
11.        table { clip: true; convert-to: div; label: "Weather Forecast";
12.                prompt: "Please select a city." }
13.        td + td > b + ul { clip: true; label: "Headlines" }
14.        td * { pause-after: small }
15.     }
16.    </style>
17.  </head>
18.
19.  <body>
20.  <center>
21.     <font size=+1>
22.       <a href=widgets.com>Buy cheap widgets at Widgets.com!</a>
23.     </font>
24.  </center>
25.  <hr>
26.  <center><font size=+2>Welcome to MySpokenWeb</font></center>
27.  <hr>
28.  <table>
29.   <tr>
30.    <td width="30%" valign=top>
31.     <b>Movie Listings:</b>
32.     <ul>
33.      <li><a href=a.html>Star Wars</a>
34.      <li><a href=b.html>Reservoir Dogs</a>
35.      <li><a href=c.html>Solaris</a>
36.      <li><a href=d.html>Superfly</a>
37.     </ul>
38.
39.    <td width="40%" valign=top>
40.
41.     <b>Weather:</b><p>
42.     <table border=1>
43.      <tr><td>City<td>Temperature<td>Outlook
44.      <tr><td>San Francisco <td> 56 <td>Sunny
45.      <tr><td>Chicago <td>17 <td>Snow
46.      <tr><td>Seattle <td>43 <td>Rain
47.     </table>
48.
49.    <td width="33%" valign=top>
50.
51.     <b>Headlines:</b>
52.     <ul>
53.      <li title=Microsoft>Judge rules against Microsoft.
54.      <li title=Superbowl>Vikings win the SuperBowl.
55.      <li title=Coffee>Coffee prices on the rise.
56.     </ul>
57.
58.  </table>
59.  <hr>
60.  <center><small>
61.      Copyright (c) <a href=foobar.com>Foobar.com</a>.
62.  </small></center>
```

```
63.
64.  </body>
65.  </html>
```

4.3.2 Sample Visual Browser Version

A typical visual browser would render this page as shown in Figure 4.3.

Figure 4.3 Sample HTML/xCSS page in a typical browser

Note that there are elements on this page that would probably not be relevant to the voice browsing experience. Such elements might include tables used for formatting, images, horizontal rules, etc.

The xCSS style sheet performs the dual functions of eliminating extraneous aspects of the page, as well as decorating the page with the appropriate prompts.

4.3.3 Sample User Audio Experience

A sample transcript of a user interaction with this page could be as follows.

Speak	Interaction Content	Notes
Nina:	Welcome to My Spoken Web, brought to you by Widgets.com. Please select one of the following: 1 Movie Listings 2 Weather Forecast 3 Headlines	Greeting uses label specified in Line 7. Line 8 Line 9 Line 11 Line 13
User	Weather forecast	

Speak	Interaction Content	Notes
Nina	Welcome to Weather Forecast. Please select a city: 1 San Francisco 2 Chicago 3 Seattle	Default greeting from `css.ini` file. Line 12 The city prompts are built using a CSS selector. <explain - theone we wrote down points to the movies.> The city names come from Line 44, Line 45, and Line 46.
User	San Francisco	
Nina	Chicago ... 17 ... snow What would you like to do next?	
User	Go to top	Standard navigation command. User can use these commands at any point.
Nina	Welcome to My Spoken...	The entire greeting starts to play again.
User	Movie Listings	This is a barge-in.
NIna	Welcome to Movie Listings. Your options are: 1 Star Wars 2 Reservoir Dogs 3 Solaris 4 Superfly	
User	[User presses 2# on touch-tone keypad]	The # sends all previous input to Nina. In this case, the “2” selects Reservoir Dogs.
Nina	[Follows this link and loads a new page]	

4.4 Site Implementation Strategies

Site implementation examples include:

- Third-party authoring and hosting of site-specific content via a proxy server allows voice-enabling of any content with no engineering or production involvement from the content site.
- Authoring and hosting of site-specific content by a partner of the original content provider. The partner’s service overrides the content provider’s normal proxy and allows the partner to create a highly customized voice-user experience.

5 Server Communication

This chapter describes the message streams used by the various types of servers. The gateway messaging details will only be of interest if you are building your own gateway rather than using the gateway from Vocal Point. Servers that are part of the gateway functionality are the CTI, ASR, and TTS portions.

NOTE: *The Nina API libraries are described in a separate document provided by Vocal Point.*

5.1 Nina Server

This is the core server component, the “brains” of the process.

5.1.1 Nina Server-Level Processing

Nina Server Initialization: Upon initializing, the Nina server performs the following operations:

1. Load a configuration file
2. Register itself with a service registry
3. Connect to a logging server
4. Listen on the control stream for connections.

5.1.2 Nina Session-Level Processing

Nina Session Initialization: When the Nina server receives an `OPEN_IAP_SESSION_METHOD` message, it creates a session as follows:

1. Registers itself with a wll-known service registry.
2. Asks RMAN for the locations of TTS and ASR resources (available sessions).
3. Sets up two server sockets: one for accepting a connection from a TTS session and another for accepting a connection from an ASR session.
4. Waits until the ASR and tTS have successfully connected back on the server sockets created in Step 3.
5. Loads the home page and initiates a session.

5.1.3 Nina Message Streams

Table 5.1 describes the message streams used by Nina.

Table 5.1 Nina message streams

Stream Type	Description
Log stream	The Nina server uses this stream to communicate with the log server to send errors, warnings, status, and other logging messages.
Control stream	The control stream is used to initiate sessions. The Nina server listens on the control stream socket for connections and processes control messages.
Nina-ASR stream	This stream connects a Nina session with an ASR session. It is used by Nina to set new grammars at the ASR, and by the ASR to return recognitions.
Nina-TTS stream	This stream connects an Nina session with a TTS session. This stream is used to make text-to-speech conversion requests, and well as play-URL requests. It is a one-way stream from Nina ->TTS.

5.1.4 Nina Control Stream Messages

The Nina server listens on the control stream socket for connections. When a connection occurs , a control stream is established. The Nina server responds to the control stream messages it receives.

Table 5.2 describes the Nina control stream messages. The message names for C++ use all caps with underbars, and for Java use initial caps with no underbars. For example `STATUS_METHOD` is the C++ name and `StatusMethod` is the Java name.

Table 5.2 Nina control stream messages

Message Name (C++ and Java)	Description
`STATUS_METHOD` `StatusMethod`	When the Nina server receives this message, it sends a `StatusReturnValue` which identifies: • Service type as “IAP” • The number of active Nina sessions running on the server • The maximum number of Nina sessions allowed.
`SHUT_DOWN_METHOD` `ShutDownMethod`	When the Nina server receives this message, it closes down all sessions and returns `IntegerReturnValue(NINA_SUCCESS)`.
`OPEN_IAP_SESSION_METHOD` `OpenIapSessionMethod`	When the Nina server receives this message, it attempts to create (or allocate) a new Nina session.

Table 5.2 Nina control stream messages

Message Name (C++ and Java)	Description
`CLOSE_SESSION_METHOD` `CloseSessionMethod`	When the Nina server receives this message, it closes the session with the specified session ID. The Nina server returns an `IntegerReturnValue`: • `NINA_SUCCESS` if session successfully shuts down • `NINA_NOSUCH_SESSION` if sessiondoes not exist

5.2 Logger

5.3 CTI Server

5.4 ASR Server

The automatic speech recognition (ASR) server is responsible for creating sessions to perform speech recognition, DTMF (touch-tone input) string processing, and audio recording. Each ASR server is capable of creating dynamic grammars on-the-fly, or loading pre-compiled static grammars. At present, the ASR server does not support language modeling or natural language processing.

See "ASR Message Streams" on page 37 for a description of the individual streams and messages mentioned in this section.

5.4.1 ASR Functions

Key functions of the ASR are:

- Recognize speech and return text strings matching the Java Speech Grammar Format (JSGF) entries. It returns a 0-100 confidence score with the result.
- Create complete DTMF strings in the following form:
  - `1234#` for pound (#) terminated multi-key strings
  - `*2` for star key (*) prefixed, single-key strings
- Record audio

5.4.2 ASR Server-Level Processing

The ASR server is responsible for processing message received on the ASR control streams and for creating and managing active ASR sessions. Each ASR server maintains its own record of the maximum number of ASR sessions it can operate at any given time.

ASR Server Initialization: The ASR server initialization sequence is:

1. Connect with the logging server.
2. Load ASR configuration.
3. Connect with service registry.
4. Listen on control stream socket for control stream connections.

ASR Server Shutdown: The ASR server shuts down when it receives a `SHUT_DOWN_METHOD` over the ASR control stream. It then does the following:

- Stop accepting `OPEN_ASR_SESSION_METHOD` requests.

- Wait for all active ASR sessions to complete.
- Stop.

5.4.3 ASR Session-Level Processing

Each ASR session performs speech recognition, DTMF string collection, and audio recording for a single call connection. Once started and initialized, the ASR session normally idles (idle state) waiting for commands from the Nina server sent over the ASR-Nina stream.

Opening ASR Sessions: The Nina server initiates ASR sessions by sending an `OPEN_ASR_SESSION_METHOD` over the ASR control stream.

When the ASR server receives this message, the following occurs:

- If the maximum number of sessions has been reached, return an `IntegerReturnValue` of `NINA_NOSESSIONS_AVAILABLE`.
- Otherwise, open an ASR session, connecting to the ASR-Nina stream at the IP address of the ASR control stream.

Because the `OPEN_ASR_SESSION_METHOD` is sent from Nina, the ASR session should attempt to connect the ASR-Nina stream at the IP address of this control stream.

ASR Session Initialization: When an ASR session is started, the following initialization sequence occurs:

1. Connect to the ASR-Nina stream at the IP address of the ASR control stream and specified Nina port.
2. Connect to the ASR-CTI stream at the specified IP address and CTI port.
3. Initialize session variables.
4. Reset audio channel adaptation (if available).
5. Enter the idle state and wait to process Nina requests.

ASR Session Idle State: In the idle state, the ASR session is waiting for message on the ASR-Nina stream. During this time, the session shuld also perform the following tasks:

- Buffer audio/silence blocks received from the ASR-CTI stream.
- Buffer all DTMF events received from the ASR-CTI stream into the DTMF buffer.

When a `RecognitionEvent` or `RecordingEvent` message is received on the ASR-Nina stream, the ASR session will begin to process messages on the ASR-CTI stream to produce a recognition or recording event.

ASR Session Shutdown: The ASR session immediately closes when:

- The ASR server receives a `CLOSE_SESSION_METHOD` on the ASR control stream.
- The ASR session receives a `TERMINATE_EVENT` from the ASR control stream. ”

During the session shutdown, any recognition and recording operation is stopped immediately. The ASR session shuts down by:

- Closing all streams
- Freeing all allocated memory

5.4.4 ASR Speech Recognition

ASR Speech Recognition Sequence: The ASR session begins recognition upon receiving the `RECOGNIZE_EVENT` message on the ASR-Nina stream. While performing recognition, the ASR session buffers any command received on the ASR-Nina stream and does not process these messages until the recognition process is completed.

During speech recognition, the following operations are performed:

1. Clear any buffered audio/silence blocks received from the ASR-CTI stream.
2. Select/compile specified grammar context.
3. Start endpoint detection.
4. Process audio/silence blocks received from the ASR-CTI until the endpoint has been detected.
5. Return a RecognitionResultEvent message of type “speech”, along with the recognition text result and the confidence score (from 0-100).

The recognition process is immediately aborted as soon as any DTMF is buffered.

ASR DTMF Handling: The ASR session maintains a DTMF buffer for touch-tone input. DTMF buffering occurs under the following conditions:

- When the user presses the pound key (#), a string consisting of all DTMF keys preceding and including the “#” key is sent.
- When the user presses the star key (*), a string consisting of the “*” key followed by the next DTMF key is sent.

Whenever DTMF is buffered, and the ASR session does the following:

- Aborts any speech recognition event in process
- Generates a `RecognitionResultEvent` message of type “DTMF” and confidence 100.

ASR Speech Recognition Exception and Error Handling: If an exception or error occurs during recogntion, the ASR session returns a `RecognitionResultEvent` of type “speech” with confidence 0, and text “Rejected”. For example:

```
rre = new RecognitionResultEvent(0, "speech", "Rejected", grammarID);
```

Completing the Recognition Process: The recognition process is completed when the `RecognitionResultEvent` is sent on the ASR-Nina stream . The ASR session should then return back to the idle state.

ASR-Nina Stream Messages Buffered During Recognition: During recognition, the ASR session may receive messages on the ASR-Nina stream. These messages are buffered and processed after the recognition process has completed.

5.4.5 ASR Speech Recording

When a START_RECORDING_EVENT is received and the ASR session is in the idle state, the ASR session begins to send audio/silence blocks received on the ASR-CTI stream to Nina on the ASR-Nina stream. The ASR session continues to transmit audio/silence blocks until the specified termination condition is satisfied.

ASR Speech Recording Termination Conditions: Possible speech recognition termination conditions are when:

- The specified DTMF key is pressed.
- The ASR-CTI stream is silent (audio magnitude below an activity threshold) for a duration exceeding a specified duration threshold.

5.4.6 ASR Configuration File

5.4.7 ASR Message Streams

The ASR server uses several message streams to communicate with the other gateway components and with Nina.

Table 5.3 describes the message streams used by ASR.

Table 5.3 ASR message streams

Stream Type	Description
Log stream	The ASR server uses this stream to communicate with the log server to send errors, warnings, status, and other logging messages.
ASR control stream	The control stream is used to open or close sessions, check on status, or shut down the ASR server. The ASR server listens on the control stream socket for connections. RMAN and Nina establish connections on this socket. The stream typically lasts long enough for one message to be sent in both directions.
ASR-Nina stream	This stream is used to connect an ASR session with a Nina session. Nina sends commands to ASR and results are returned from ASR to Nina. This stream typically remains connected for the duration of a session.
ASR-CTI stream	This stream connects an ASR session with a CTI session. This stream typically remains connected for the duration of a session.

NOTE: *See the API reference documentation provided by Vocal Point for details on the methods and return values for each message stream.*

5.4.8 ASR Log Stream Messages

5.4.9 ASR Control Stream Messages

The ASR server listens on the control stream socket for connections. When a connection occurs , a control stream is established. The ASR server responds to the control stream messages it receives.

Table 5.4 describes the ASR control stream messages. The message names for C++ use all caps with underbars, and for Java use initial caps with no underbars. For example `STATUS_METHOD` is the C++ name and `StatusMethod` is the Java name.

Table 5.4 ASR control stream messages

Message Name (C++ and Java)	Description
`STATUS_METHOD` `StatusMethod`	When the ASR server receives this message, it sends a `StatusReturnValue` which identifies: • Service type as “ASR” • The number of active ASR sessions running on the server • The maximum number of ASR sessions allowed.

Table 5.4 ASR control stream messages

Message Name (C++ and Java)	Description
SHUT_DOWN_METHOD ShutDownMethod	When the ASR server receives this message, it closes down all sessions and returns IntegerReturnValue(NINA_SUCCESS).
OPEN_ASR_SESSION_METHOD OpenAsrSessionMethod	When the ASR server receives this message, it attempts to create (or allocate) a new ASR session. The ASR server returns an IntegerReturnValue: • NINA_SUCCESS if session is successfully created • An appropriate error value otherwise.
CLOSE_SESSION_METHOD CloseSessionMethod	When the ASR server receives this message, it closes the session with the specified session ID. The ASR server returns an IntegerReturnValue: • NINA_SUCCESS if session successfully shuts down • NINA_NOSUCH_SESSION if sessiondoes not exist

5.4.10 ASR-Nina Stream Messages

The ASR server creates an ASR-Nina stream before creating an ASR session. This stream is used to pass command messages from Nina to the ASR, and to return results from the ASR to Nina.

Table 5.5 describes the ASR-Nina stream messages. The message names for C++ use all caps with underbars, and for Java use initial caps with no underbars. For example RECOGNIZE_EVENT is the C++ name and RecognizeEvent is the Java name.

Table 5.5 ASR-Nina messages

Message Name (C++ and Java)	Description
RECOGNIZE_EVENT RecognizeEvent	Upon receiving this command, the ASR session begins a recognition event using the specified JavaSpeech Grammar Format (JSGF).
START_RECORDING_EVENT StartRecordingEvent	When ASR receives this command, it starts recording until the specified termination condition is recognized.
MUTE_EVENT MuteEvent	Mute or unmute the system. When ASR receives this message during recognition or recording, it suspends the processing of audio until an unmute command.
TERMINATE_EVENT TerminateEvent	When ASR receives this message it terminates the session.

5.4.11 ASR-CTI Stream Messages

Before the ASR session is created, the ASR-CTI connection is established .

Table 5.6 describes the ASR-CTI stream messages. The message names for C++ use all caps with underbars, and for Java use initial caps with no underbars. For example `TERMINATE_EVENT` is the C++ name and `TerminateEvent` is the Java name.

Table 5.6 ASR-CTI messages

Message Name (C++ and Java)	Description
`TERMINATE_EVENT` `TerminateEvent`	The ASR session responds by shutting down.
`AUDIO_BLOCK` `AudioBlock`	Only used while recognizing or recording. Otherwise, message is discarded.
`SILENCE_BLOCK` `SilenceBlock`	Only used while recognizing or recording. Otherwise, message is discarded.

5.5 TTS Server

The text-to-speech (TTS) server is responsible for creating sessions to perform text-to-speech conversion, and passing the results (in the form of audio blocks) to the CTI. Each TTS server is capable of dynamically converting text to speech on-the-fly, or loading pre-converted (cached) phrases. The TTS server also informs the CTI if some event (such as a barge-in) should prevent the CTI from playing the queued speech it has already received.

5.5.1 TTS Functions

Key functions of the TTS are:

- Convert text to speech sound blocks
- Convert Web sound content to CTI-compatible sound blocks

5.5.2 TTS Server-Level Processing

Starting the TTS Server:

TTS Server Initialization: Upon initializing, the TTS server performs the following operations:

1. Connect with the logging server.
2. Load TTS configuration.
3. Connect with the service registry.
4. Listen on socket for control stream connections.

5.5.3 TTS Message Streams

Table 5.7 describes the message streams used by the TTS.

Table 5.7 TTS message streams

Stream Type	Description
Log stream	The TTS server uses this stream to communicate with the log server to send errors, warnings, status, and other logging messages.

Table 5.7 TTS message streams

Stream Type	Description
Control stream	The control stream is used to initiate sessions. The TTS server listens on the control stream socket for connections and processes control messages.
TTS-CTI stream	This stream connects a TTS session with a CTI session.

5.5.4 TTS Control Stream Messages

The TTS server listens on the control stream socket for connections. When a connection occurs, a control stream is established. The TTS server responds to the control stream messages it receives.

Table 5.8 describes the TTS control stream messages. The message names for C++ use all caps with underbars, and for Java use initial caps with no underbars. For example `STATUS_METHOD` is the C++ name and `StatusMethod` is the Java name.

Table 5.8 TTS control stream messages

Message Name (C++ and Java)	Description
`STATUS_METHOD` `StatusMethod`	When the TTS server receives this message, it sends a `StatusReturnValue` which identifies: • Service type as “TTS” • The number of active TTS sessions running on the server • The maximum number of TTS sessions allowed.
`SHUT_DOWN_METHOD` `ShutDownMethod`	When the TTS server receives this message, it closes down all sessions and returns `IntegerReturnValue(NINA_SUCCESS).`
`OPEN_TTS_SESSION_METHOD` `OpenTtsSessionMethod`	When the TTS server receives this message, it attempts to create (or allocate) a new TTS session.
	When the TTS server receives this message, it closes the session with the specified session ID. The TTS server returns an `IntegerReturnValue`: • `NINA_SUCCESS` if session successfully shuts down • `NINA_NOSUCH_SESSION` if sessiondoes not exist

5.5.5 TTS-Nina Stream Messages

The TTS server creates a TTS-Nina stream before creating a TTS session.

Table 5.9 describes the TTS control stream messages. The message names for C++ use all caps with underbars, and for Java use initial caps with no underbars. For example `PLAY_METHOD` is the C++ name and `PlayMethod` is the Java name.

Table 5.9 TTS-Nina stream messages

Message Name (C++ and Java)	Description
`PLAY_METHOD` `PlayMethod`	Convert the given text to speech, and pass the result along (in the appropriate sound block format) to the CTI so that the user can hear the result.
`PLAY_URL_METHOD` `PlayUrlMethod`	Convert the audio format (such as a RealAudio file) found at the given URL to a sound block format the CTI can understand, and pass the result along to the CTI so that the user can hear the result.
`CLEAR_ALL_METHOD` `ClearAllMethod`	Stop converting any text in progress for the given sesion. Pass the CLEAR_ALL_METHOD message along to the CTI. Most often, this is becaus of a user barge-in.
`TERMINATE_EVENT` `TerminateEvent`	Upon receiving this message, the TTS server closes the given sesison.

5.5.6 TTS-CTI Stream Messages

Table 5.10 describes the TTS control stream messages. The message names for C++ use all caps with underbars, and for Java use initial caps with no underbars. For example `PLAY_METHOD` is the C++ name and `PlayMethod` is the Java name.

Table 5.10 TTS-CTI stream messages

Message Name (C++ and Java)	Description
`AUDIOBLOCKEVENT` `AudioBlockEvent`	Send a block of sound data to the CTI to be played to the user.
`SILENCE_BLOCK_EVENT` `SilenceBlockEvent`	Send a block of silence to the CTI to be played to the user. This takes less bandwidth than an audio block and essentially acts as a temporal filler between audio blocks during a phone call.
`TERMINATE_EVENT` `TerminateEvent`	Sends a request to the CTI to terminate a phone call if the CTI has not already done so.
`CLEAR_EVENT` `ClearEvent`	Requests that the CTI abort transmission of queued audio blocks to the user, usually in response to a barge-in.

5.6 Message Protocol

5.6.1 General Message Protocol

Each message contains the following:

- **Standard header** (30 characters) consisting of:
  - **Identifier** (20 characters)
  - **Message length** (10 characters)
- **Message body** (length specified by message length in header), starting with a message name and subsequent arguments separates by newline characters.
  - `MessageName\n`
  - `arg1\n` ...
  - `argn\n`

The receiver of the message knows, based on the `MessageName`, how to parse the arguments. As a rule, most data is represented as strings, to avoid problems with byte order and to enhance readbility.

5.6.2 Method Message Protocol

Method structure is based on the general message protocol:

- Standard header as above
- Message body
  - `MethodName` (example: "`openMediaStreamIn`")
  - `numArgs`
  - `type1`
  - `arg1` ...
  - `typen`
  - **`argn`**

5.6.3 Return Value Message Protocol

Return value message structure is based on the general message protocol:

- Standard header as above
- Message body
  - `ReturnValueName`
  - `numArgs`
  - `type1`
  - `arg1` ...
  - `typen`
  - **`argn`**

5.6.4 Supported Data Types

Currently, the following data types are supported:

- 32-bit signed integers (Integer)
- 64-bit signed integers (Long, after the Java convention)
- Strings
- ByteArrays

5.6.5 Restrictions

When constructing message, the following restrictions apply:

- Strings may NOT contain newlines, because newlines are used to determine data boundaries inside of the packet.
- ByteArrays are uninterpreted sequences of byte data, and are useful for transporting audio data.
- Because of the newline restriction, ByteArrays MUST be the last item in a message.

5.7 Building a Compliant Server

The first thing a compliant server must do is register itself with the Service Registry, which lives on a well-known port. The Service Registry is the only server that must run on a well-known port. From then on, the Service Registry pings every service that has registered with it (every N seconds) to assess its status.

Appendix A: Terms and Acronyms

ACS
ASR-CTI stream.

AIS
ASR-IAP stream. Better known as the ASR-Nina message stream.

ASR
Automatic speech recognition

BOB
Alternate term for the Resource Manager.

browser
An application such as Netscape Navigator that allows a user to navigate the World Wide Web.

content node

CTI
Computer telephony interface

CSS
Cascading style sheets

dialog
Within Nina, a dialog is a hierarchical numbered list of choices and navigation prompts presented by the audio browser.

DTMF
Dial-tone multi-frequency, or telephone touch-tone input.

grammars

group node

HTML
Hypertext markup language

IAP
Alternate term for the Nina server. Stands for Internet Application Platform.

Java JDK
Java Developer Kit

JSGF
Java Speech Grammar Format

navigation node

nagivation tree

node

PSTN

resource

With regard to a Nina server or gateway component server, a resource represents an available session on a particular server.

routing node

session

1. A telephone call and the resources collectively associated with this telephone call. This type of session is uniquely identified by a session ID.
2. An instance of one of the Nina or gateway server processes that was initiated to handle a particular call session.

session ID

A globally unique ID associated with each call session. Generated by CTI.

sink port

Port in a CTI session to which outbound audio data is written.

source port

The port in a CTI session where inbound audio data is read.

step navigation

TTS

Text-to-speech recognition

URL

Universal resource locator

xCSS

Vocal Point's custom extensions to the standard CSS definition.

XML

Extended markup language