

e-speak

Tutorial

Release A.03.14.00
August 2001

COPYRIGHT NOTICE

© 2001 HEWLETT-PACKARD COMPANY

To anyone who acknowledges that this document is provided "AS IS" WITH NO EXPRESS OR IMPLIED WARRANTY: permission to copy, modify, and distribute this document for any purpose is hereby granted without fee, provided that the above copyright notice and this notice appear in all copies, and that the name of Hewlett-Packard Company not be used in advertising or publicity pertaining to distribution of this document without specific, written prior permission. Hewlett-Packard Company makes no representations about the suitability of this document for any purpose.

Windows and Windows NT are registered trademarks of Microsoft Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Linux is a trademark of Linus Torvalds. All other trademarks are the properties of their respective owners.

Contents

Chapter 1	Getting Started	1
	Before You Begin	2
	Prerequisite Skills and/or Experience	2
	Tutorial Requirements	2
	Basic Setup	2
	Configuring the Tutorial for a Multi-User System	5
	Implementation	7
	E-speak Tutorial Files	7
	Demonstration	15
	Open Three Command Prompt Windows	15
	Start the Process Manager	16
	Start the e-speak Desktop	16
	Start the e-speak Core	18
	Verify that the Core is Running	19
	Additional Resources	20
Chapter 2	Basic Service	21
	Concepts	21
	Identify the Service Requirements	22
	Identify a Service Contract and Service Vocabulary	22
	Implement the Service	23
	Implement the Service Deployer	23

	Implementation	24
	Define the Service Contract	25
	Implement the Service	26
	Implement the Service Deployer	27
	Establish a Connection to e-speak	27
	Create a Service Description	28
	Create a Service Element and Set the Implementation	29
	Register, Advertise, and Start the Service	29
	Demonstration	30
	Compile the Code (Optional)	30
	Run the Service	30
	Test the Service	31
Chapter 3	Basic Client	33
	Concepts	33
	Implementation	34
	Create a Connection to e-speak	34
	Locate the Service	35
	Invoke the Method	35
	Demonstration	36
	Compile the Code (Optional)	36
	Run the Client	36
	Verify that the Client is Connected to the Core	36
	Verify that the Client can Interact with a Service	36
	Run Multiple Services	37

Chapter 4	Managed Services	39
	Implementation	39
	Build a Desktop-Managed Service	40
	Implement the DefaultManagedServiceIntf Interface	40
	Register the Class as Desktop-Manageable	41
	Set the Initial State	42
	Allow Property Modifications	43
	Demonstration	43
	Compile the Code (Optional)	43
	Start a Managed Service	44
	Verify That the Service is Running	45
	Pause the Managed Service	45
	Reset the Service Invocation Counter	46
	Remove and Restart a Managed Service	46
	Edit Service Configuration at Runtime	47
	Stop the Managed Service (Optional)	48
Chapter 5	Events	49
	Concepts	49
	Event Model	49
	Implementation	50
	Creating an Event Distributor	51
	Create an Events Listener/Subscriber	51
	Create an Event Publisher	53
	Register as an Event Publisher	53

	Publish an Event	54
	Demonstration	54
	Compile the Code (Optional)	54
	Start the Events Distributor	54
	Verify That the Events Distributor is Running	55
	Start the Events Client	55
	Start the Events Service	55
	Generate Additional Event Messages	56
	Stop the Events Service (Optional)	56
Chapter 6	Folders	57
	Concepts	57
	Implementation	57
	Managing Bindings Using Folders	58
	Creating Folders	58
	Creating Transient Folders	58
	Creating Persistent Folders	59
	Naming Found Services	59
	Removing Services From Folders	60
	Demonstration	60
	Compile the Code (Optional)	60
	Start the Folder Client	61
	Save a Service Object in a Folder	61
	Save a Service Object under an Alias	62
	Delete an Alias From Your Folder	63
	Service Aliases	64

	Stop the Folders Client (Optional)	64
Chapter 7	Threads	65
	Concepts	65
	Implementation	66
	Implementing ESRunnable	66
	Using ESThread/ESRunnable	68
	Demonstration	68
	Compile the Code (Optional)	68
	Start the Threads Client	69
	Display Results from All Currently-Running Services	69
	Stop the Threads Client (Optional)	69
Chapter 8	Security	71
	Concepts	71
	The Security Process	71
	About the PSE Manager	72
	Method Tag Used in Certificates	73
	Implementation	74
	Configuration	74
	Certificates	76
	Demonstration	78
	Run the Untrusted Client	78
	Stop the Untrusted Client (Optional)	78

Chapter 9	Vocabularies	79
	Concepts	79
	Implementation	80
	Deploying Vocabularies	80
	Using Vocabularies to Describe Services	81
	Finding Vocabularies	82
	Finding Services Using a Vocabulary	82
	Accessing Descriptions	82
	Demonstration	83
	Compile the Code (Optional)	83
	Start the Vocabulary Deployer	83
	Start the Vocabulary Services	83
	Start the Basic Client	84
	Start the Vocabulary Client	84
	Display Services Using a Custom Vocabulary	85
	Query Services by Attribute	85
	Stop the Vocabulary Client (Optional)	85
Chapter 10	Contracts	87
	Concepts	87
	Implementation	88
	Deploying Contracts	88
	Finding Contracts	89
	Finding Services Using a Contract	89
	Demonstration	89

	Compile the Code (Optional)	89
	Start the Contract Deployer	89
	Start the Contracts Client	90
	Describe the Service Finder	90
	Stop the Contracts Deployer (Optional)	91
Chapter 11	Multiple Cores	93
	Concepts	93
	Demonstration	93
	Start a Second Core	93
	Start a Service on the Second Core	94
	Start the Client	94
	Glossary	97
Appendix A	E-speak Installation Directories	101

Chapter 1 Getting Started

prax-is (*prak'sis*), *n* [*fr. Gr., to do.*]. *An example or exercise, or a set of examples, for practice*

The Praxis tutorial introduces some of the capabilities of e-speak, both from a coding and procedural perspective. As you work through the chapters, keep in mind that the procedures and code samples presented here are not necessarily the only way to accomplish a particular task; instead, they demonstrate one way to approach processes and gather information within a versatile application.

Most chapters contain three sections:

- **Concepts** — a discussion of the concepts covered in the chapter
- **Implementation** — a discussion of how those concepts were applied when implementing the tutorial

NOTE: Chapters that do not require code modifications do not include an implementation section.

- **Demonstration** — step-by-step procedure(s) to demonstrate a concept

The code in the implementation sections does not duplicate the source code exactly. For training purposes, the code has been modified to make the examples more clear. For example, samples often do not include exception handling code.

If you are completing the tutorial to simply learn some of the basic functions of e-speak, you can disregard the code sections; however, if your goal is to learn something about e-speak programming, the code sections will give you a good idea of the types of commands and processes to use in a particular situation.

For more complete information about e-speak programming, refer to the *e-speak Programmers Guide*.

Before You Begin

Prerequisite Skills and/or Experience

To successfully complete this tutorial, you must have the following skills/experience:

- Working knowledge of Java programming
- Familiarity with the e-speak environment and terminology

Tutorial Requirements

Basic Setup

Tutorial setup varies depending on your operating system. The tutorial installation directory is `<espeak_home>/tutorial/praxis`, where `<espeak_home>` is the e-speak install directory. Since you may be building and modifying code, you should **copy** all of the tutorial files to your own directory and run the tutorial locally, as shown in [Table 1](#).

Table 1 Copying the Praxis tutorial to a local drive

Platform	To copy the tutorial files:
Windows NT®	In the Windows® desktop, drag and drop the entire tutorial directory onto a local drive.
Linux™ and HP-UX	At a command prompt: <ol style="list-style-type: none"> 1 Type <code>cd ~</code> 2 Type <code>mkdir tutorial</code> 3 Type <code>cp -R <espeak_home>/tutorial/praxis tutorial</code> 4 Type <code>cd ~/tutorial/praxis/config</code>

The tutorial working directory is `<your base directory>/tutorial/praxis/config`, where `<your base directory>` is the local directory to which you copied the tutorial directory. For example, if you copied the tutorial to a directory called `estutorial`, ensure that you are in the `estutorial/tutorial/praxis/config` directory before working with tutorial examples.

Before you start this tutorial, ensure that the:

- E-speak home environment variable, `ESPEAK_HOME`, points to the e-speak install directory and that following directories are in your path:

- e-speak `bin` directory

Table 2 e-speak bin directories

Platform	Directory
Windows NT	<code>c:\Program Files\e-speak\bin</code>
HP-UX	<code>/opt/e-speak/bin</code>
Linux	<code>/usr/local/e-speak/bin</code>

- Java `bin` directory

- Current working directory

NOTE: Refer to the *e-speak System Administration Guide* for instructions.

- Tutorial classes are included in the Java classpath as shown in [Table 3](#).

Table 3 Tutorial Java classpaths

If you are using...	On...	Then...
The pre-built jar file	HP-UX or Linux™	<pre>export CLASSPATH=\$CLASSPATH:\$ESPEAK_HOME/ lib/tutorial.jar</pre>
	Windows NT®	<p>From the command line:</p> <pre>set CLASSPATH=%CLASSPATH%;%ESPEAK_HOME%\ lib\tutorial.jar</pre> <p>Or in the desktop:</p> <ol style="list-style-type: none"> 1 Right-click My Computer. 2 Select Properties to open the System Properties window. 3 Select the Environment tab. 4 Set the classpath to <code>%ESPEAK_HOME%\lib\tutorial.jar</code> in the System Variables list box. 5 Click Set. 6 Click Apply. 7 Click Ok.

Table 3 Tutorial Java classpaths

If you are using...	On...	Then...
Class files in a local directory	HP-UX or Linux™	<code>export CLASSPATH=\$CLASSPATH:<your base directory>/tutorial/praxis/lib</code>
	Windows NT®	<p>Command line:</p> <pre>set CLASSPATH=%CLASSPATH%;<drive:\your base directory>\tutorial\praxis\lib</pre> <p>Or</p> <p>In the desktop:</p> <ol style="list-style-type: none"> 1 Right-click My Computer. 2 Select Properties to open the System Properties window. 3 Select the Environment tab. 4 Set the classpath to <code><drive:\your base directory>\tutorial\praxis\lib</code>. 5 Click Set. 6 Click Apply. 7 Click Ok.

Refer to your operating system's online help for specific instructions on modifying system variables.

Configuring the Tutorial for a Multi-User System

The tutorial's e-speak components use TCP/IP protocols to communicate. Generally, only one application at a time can use a TCP/IP port number on a system. If you run the tutorial using the standard configuration, only one user at a time can run the tutorial on your system.

To support simultaneous users, modify the tutorial configuration files to use port numbers not in use on your system. Valid port numbers range from 1 to 65535, although you may need special privileges to use numbers below 1024. It is usually safest to choose numbers at the high end of the range. For more information on port number allocation, see www.iana.org.

Table 4 lists the configuration files you must modify for each user.

Table 4 Tutorial files you must modify to allow simultaneous users

File(s)	Modification
<p>The Process Manager, e-speak desktop, and desktop managed services use these files:</p> <p>tutorial/praxis/config/core.cfg</p> <p>and</p> <p>tutorial/praxis/config/service.cfg</p>	<p>In both <code>core.cfg</code> and <code>service.cfg</code>:</p> <ol style="list-style-type: none"> 1 Locate <code>Management.portnumber=51422</code>. 2 Change <code>51422</code> to a different port (for example, <code>51922</code>).
<p>This file is used to start the e-speak core and advertising service:</p> <p>tutorial/praxis/config/core1.ini</p>	<p>In <code>core1.ini</code>:</p> <ol style="list-style-type: none"> 1 Locate <code>esip:51423</code> and <code>portnumber=51423</code>. 2 Change both occurrences of <code>51423</code> to a different port (for example, <code>51923</code>).
<p>The tutorial services and clients use this file to connect to the e-speak core:</p> <p>tutorial/praxis/config/espeak.prop</p>	<p>In <code>espeak.prop</code>:</p> <ol style="list-style-type: none"> 1 Locate <code>portnumber: 51423</code>. 2 Change <code>51423</code> to the same port you specified in <code>core1.ini</code>.
<p>This file is used to start an additional e-speak Core and advertising service:</p> <p>tutorial/praxis/config/core2.ini</p>	<p>In <code>core2.ini</code>:</p> <ol style="list-style-type: none"> 1 Locate <code>esip:51424</code> and <code>portnumber=51424</code>. 2 Change both occurrences of <code>51424</code> to a different port (for example, <code>51924</code>).
<p>The tutorial services and clients use this file to connect to the additional e-speak Core:</p> <p>tutorial/praxis/config/otherEspeak.prop</p>	<p>In <code>otherEspeak.prop</code>:</p> <ol style="list-style-type: none"> 1 Locate <code>portnumber: 51424</code>. 2 Change <code>51424</code> to the same port you specified in <code>core2.ini</code>.

Now that simultaneous users can run the tutorial, assign a unique advertising group to prevent others who are running the tutorial — either on your system or from another location on your local LAN subnet — from being able to “see” your services:

- 1 Open `tutorial/praxis/config/adv.cfg` in a file editor.
- 2 Locate `net.espeak.services.advService.group=PraxisGroup`.
- 3 Change `PraxisGroup` to a name of your choice.
- 4 Save the file.

All local e-speak advertising services with the same group name can potentially access each other’s services. While running the tutorial, if you see services you didn’t start, or if you see duplicate service names, verify that you changed the advertising group name to something unique. You may be seeing another user’s services.

Implementation

E-speak Tutorial Files

The tutorial contains examples of the e-speak source code that pertain to a particular topic. If you would like to view, edit, or print the code from the source files directly, the files are located in subdirectories within the tutorial directory (`tutorial/praxis/`) as shown in [Table 5](#).

NOTE: If you are using HP-UX or Linux™, the `.cmd` extension does not appear as part of the file name. For example, instead of `basicClient.cmd`, the file simply appears as `basicClient`.

Table 5 Praxis tutorial file names and locations

Subdirectory	File Name	Description
	<code>Makefile</code>	Builds the tutorial for HP-UX and Linux™
	<code>compile.bat</code>	Builds the tutorial for Windows NT®

Table 5 Praxis tutorial file names and locations

Subdirectory	File Name	Description
client	CLI.java	The command line interpreter
	ClientAPI.java	The e-speak code for the client
	ClientAPIException.java	The Java exception used by the client
	ClientLoader.java	The client <code>main()</code>
config	adv.cfg	The advertising service configuration properties
	basicClient.cmd	The command file to run the client
	basicClient.ini	The SysLoader ini file
	basicClient.prop	The client properties
	basicService.cmd	The command file to run the service
	basicService.ini	The SysLoader ini file
	basicService1.cmd	The command file to run the service
	basicService1.ini	The SysLoader ini file
	basicService1.prop	The service properties
	basicService2.cmd	The command file to run the service
	basicService2.ini	SysLoader ini file
	basicService2.prop	The service properties
	basicService3.cmd	The command file to run the service
	basicService3.ini	The SysLoader ini file
basicService3.prop	The service properties	

Table 5 Praxis tutorial file names and locations

Subdirectory	File Name	Description
config	client.pse	The security keys used by the client
	contractClient.cmd	Runs the client used in the contracts chapter
	contractClient.ini	The SysLoader ini file
	contractClient.prop	The client properties
	contractDeployer.cmd	Runs the contract deployer used in the contracts chapter
	contractDeployer.ini	The SysLoader ini file
	contractDeployer.prop	The service properties
	core.cfg	The e-speak configuration file used by the Core
	core.pse	The security keys used by the Core
	core1.cmd	Runs the tutorial's primary Core
	core1.ini	The SysLoader ini file
	core2.cmd	Runs the secondary Core used for multi-Core operations
	core2.ini	The SysLoader ini file
	coreaccess.certs	The security certificates used by cores
	coreacl.adr	The default trust relationships for the Core
	desktop.cmd	Runs the e-speak desktop
desktop.ini	The SysLoader ini file	

Table 5 Praxis tutorial file names and locations

Subdirectory	File Name	Description
config	Desktop.xml	The schema used by the e-speak desktop
	espeak.prop	The e-speak connection properties (hostname, portnumber, etc.) used by the tutorial applications
	eventsClient.cmd	Runs the client used in the events chapter
	eventsClient.ini	The SysLoader ini file
	eventsClient.prop	The client properties
	eventsDistrib.cmd	Runs the events distributor used in the events chapter
	eventsDistrib.ini	The SysLoader ini file
	eventsDistrib.prop	The service properties
	eventsService.ini	The SysLoader ini file for the service used in the events chapter
	eventsService.prop	The service properties
	foldersClient.cmd	Runs the client used in the folders chapter
	foldersClient.ini	The SysLoader ini file
	foldersClient.prop	The client properties
	managedService.ini	The SysLoader ini file for the service used in the managed service chapter
	managedService.prop	The service properties

Table 5 Praxis tutorial file names and locations

Subdirectory	File Name	Description
config	otherCoresService1.cmd	Runs the service that connects to another Core in a multi-Core environment
	otherCoresService1.ini	The SysLoader ini file
	otherCoresService1.prop	The service properties
	otherEspeak.prop	The e-speak properties used to connect to other cores
	processmgr.cmd	Runs the e-speak Process Manager
	processmgr.ini	The SysLoader ini file
	service.cfg	The e-speak config file used by services
	service.pse	The security keys used by services
	serviceaccess.certs	The security certificates used by services
	serviceacl.adr	The default trust relationships for services
	templates.xml	The e-speak desktop template
	threadsClient.cmd	Runs the client used in the threads chapter
	threadsClient.ini	The SysLoader ini file
	threadsClient.prop	The client properties
	trusted.cfg	The e-speak configuration file used by the trusted client
	trustedclientaccess.certs	The security certificates used by the trusted client

Table 5 Praxis tutorial file names and locations

Subdirectory	File Name	Description
config	trustedclientacl.adr	The default trust relationships for the trusted client
	untrusted.cfg	The e-speak configuration file used by the untrusted client
	untrustedClient.cmd	Runs the untrusted client
	untrustedClient.ini	The SysLoader ini file
	untrustedClient.prop	The client properties
	untrustedclientaccess.certs	The security certificates used by the untrusted client
	untrustedclientaccess.adr	The default trust relationships for the untrusted client
	vocabClient.cmd	Runs the client used in the vocabulary chapter
	vocabClient.ini	The SysLoader ini file
	vocabClient.prop	The client properties
	vocabDeployer.cmd	Runs the vocabulary deployer used in the vocabulary chapter
	vocabDeployer.ini	The SysLoader ini file
	vocabDeployer.prop	The service properties
	vocabService1.cmd	Runs the service used in the vocabulary chapter
	vocabService1.ini	The SysLoader ini file
vocabService1.prop	The service properties	

Table 5 Praxis tutorial file names and locations

Subdirectory	File Name	Description
config	vocabService2.cmd	Runs the service used in the vocabulary chapter
	vocabService2.ini	The SysLoader ini file
	vocabService2.prop	The service properties
contracts	ContractCLI.java	The command line interpreter extension for the contracts chapter client
	ContractClientAPI.java	The e-speak code extension for the contracts chapter client
	ContractDeployer.java	Creates and deploys an e-speak contract
doc/html	index.html	The index page for the tutorial javadoc
events	EventCLI.java	The command line interpreter extension for the events chapter client
	EventClientAPI.java	The e-speak code extension for the events chapter client
	EventDistributor.java	The e-speak events distributor
	EventPublisher.java	The e-speak code extension that publishes events
folders	FoldersCLI.java	The command line interpreter extension for the folders client
	FoldersClientAPI.java	The e-speak code extension for the folders client

Table 5 Praxis tutorial file names and locations

Subdirectory	File Name	Description
intf	ServiceIntf.esidl	The e-speak service interface definition that inputs to the esidl compiler
	ServiceIntf.java	The e-speak esidl compiler output files
	ServiceIntfMessageRegistry.java	
	ServiceStub.java	
managedservice	ManagedImpl.java	An extension of the service implementation used in the managed service chapter
	ManagedService.java	The e-speak code extension for the service used in the managed service chapter
service	LoaderIntf.java	The interface definition for the classes that can be loaded by ServiceLoader
	Service.java	The e-speak code used by basic services
	ServiceImpl.java	Implements the service
	ServiceLoader.java	The service main()
threads	ThreadCLI.java	The command line interpreter extension for the client used in the threads chapter
	ThreadClientAPI.java	The e-speak code extension for the client used in the threads chapter

Table 5 Praxis tutorial file names and locations

Subdirectory	File Name	Description
vocabulary	VocabCLI.java	The command line interpreter extension for the client used in the vocabulary chapter
	VocabClientAPI.java	The e-speak code extension for the client used in the vocabulary chapter
	VocabDeployer.java	The e-speak code used to create and deploy an e-speak custom vocabulary
	VocabService.java	The e-speak code extension for services used in the vocabulary chapter

Demonstration

Complete the procedures described in this section each time you run the tutorial. All subsequent chapters assume that these processes are already running on your computer.

Open Three Command Prompt Windows

In your operating system, open three command prompt windows. Refer to your operating system manual if you need assistance in opening this type of window. You will use these windows for various processes throughout the tutorial. In general, use each window to display and run one of the following types of tasks: client, desktop, or Process Manager.

TIP: The step-by-step procedures in the tutorial begin by specifying the window in which you should perform a task. For example, if the procedure begins with “In the client window,” perform the task in the command prompt window that is currently running the e-speak client.

Start the Process Manager

NOTE: The e-speak tutorial uses its own Process Manager. Even if you already have a Process Manager currently running on your system, you must complete this procedure.

In a command prompt window:

1 Navigate to the tutorial working directory.

NOTE: Refer to [“Basic Setup” on page 2](#) for the working directory path.

2 Type `processmgr`.

3 Press **Enter**.

Start the e-speak Desktop

In another command prompt window:

1 Navigate to the tutorial working directory.

2 Type `desktop`.

3 Press **Enter**. The desktop opens.

4 Right-click **e-speak**.

5 Select **New Host**. A Host icon appears on the desktop.

6 Right-click **Host**.

7 Select **Create a process from a template** to open the **Please select a template** window.

8 Select **E-speak Core**.

- 9 Click **Create**. E-speak Core appears on the desktop just below Host.

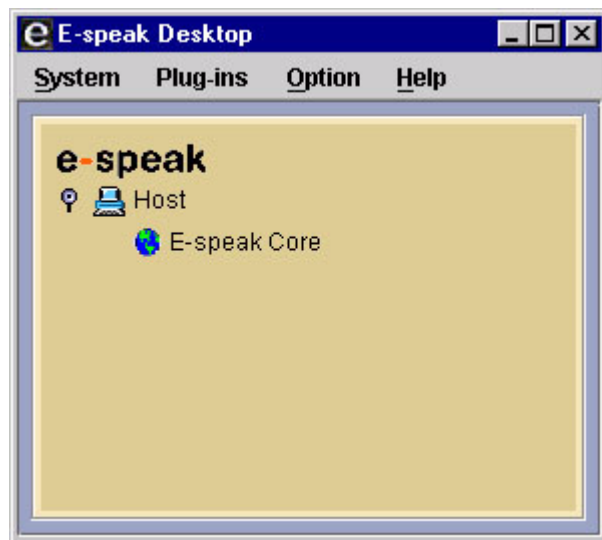


Figure 1 The e-speak Desktop

Start the e-speak Core

In the e-speak desktop:

- 1 Right-click **E-speak Core**.
- 2 Select **Process Management > Start Process**.

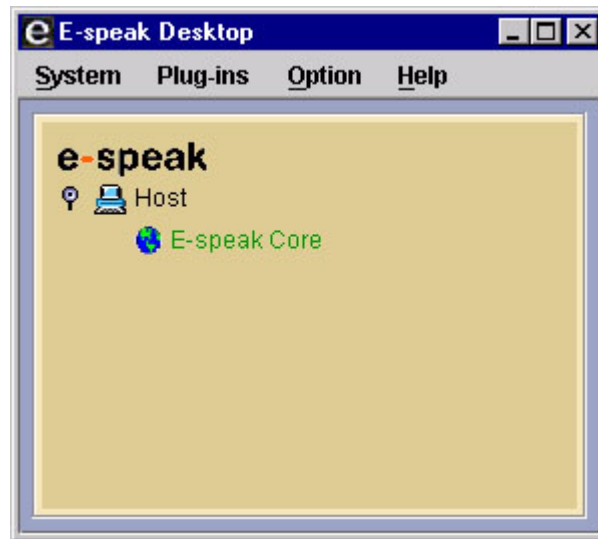


Figure 2 The e-speak Core successfully started

E-speak Core on the desktop turns green if successfully started. If the connection fails, E-speak Core turns red. If this occurs, the procedure, [“Verify that the Core is Running”](#), may provide data to assist you in troubleshooting.

NOTE: In general, a green desktop label indicates that the Core or service successfully started; however, it may not necessarily be ready. For example, a service label is green, but a client trying to access it receives the message `No service found`. The most likely reason is that the service has not finished the startup process. To verify that a Core/service is ready, check its process output. Refer to the *e-speak System Administration Guide* for details.

Verify that the Core is Running

In the e-speak desktop:

- 1 Right-click **E-speak Core**.
- 2 Select **Process Management > Process Output**.

A process output window opens and displays information about the Core. Use this information to help resolve any issues when starting the e-speak Core:

```
CoreArgs> hostName => <Host>
E-Speak Core : Version NA
Core Starting with URL of esip://<Host>:51423/
Core ID (for core service IDs) is "5?\2323RkL\224\342N"
Repository : In-Memory .
E-Speak Core Started.
Loading Default provider
Taking the default connectiontimeout value...
Warning: Failed loading espeak-wide config file.
Taking the default connectiontimeout value...
Connected to Core.
Registered advertising service
BackendMode is: slp
<Host>:51423:AdvSvc Mcast: Sleeping for 30000 milli-seconds....
to allow advertising services to discover one another.
You may reduce this delay if you're in a faster
network or your program is
prepared to handle the case when it runs faster
than the advertising services to discover one
another (e.g. retry upon failures).
To customize this parameter, set the property
net.espeak.services.advService.mcast.sleepMillis.
(and start advertising service directly with
```

```
net.espeak.services.advService.server.AdvService.)
Lookup Server URL = esip://localhost:51422/core/service/find
Taking the default connectiontimeout value...
Taking the default connectiontimeout value...
Completed Multicasting.
Started advertising service
```

NOTE: In this example, <Host> is the name of your host. Your process output may vary from that shown here.

Refer to the *e-speak System Administration Guide* for help with troubleshooting issues.

Additional Resources

For more information about e-speak, please refer to the:

- *e-speak Programmers Guide*
- *e-speak Architectural Specification*
- *e-speak System Administration Guide*

NOTE: The *e-speak System Administration Guide* contains specific information regarding the e-speak desktop that you will find very useful as you work through the tutorial.

- *e-speak Contributed Software*

Chapter 2 Basic Service

This chapter demonstrates implementing and running a service. To complete this chapter, the basic components of e-speak must be running on your computer, specifically the Process Manager, e-speak Desktop, and Core. Before you begin, ensure that you have completed all of the processes outlined in [Chapter 1, “Getting Started.”](#)

Concepts

E-speak-enabling a process involves six phases:

- Identify the service requirements
- Identify a service contract (interface) and service vocabulary
- Implement a service
- Implement the service deployer
- Run the service
- Test the service

Identify the Service Requirements

When trying to identify (find or implement) a service, you should:

- Understand the service interactions by:
 - Determining how to implement e-service interfaces, particularly synchronous or asynchronous interfaces (messaging or publishing-subscribing)
 - Considering how clients use the service
 - Applying use case analysis to determine service interactions
 - Identifying incoming and outgoing information
 - Defining your e-service interface requirements based on analysis
- Build e-services with the language of your choice (Java™, C++, Python, or C), keeping in mind that:
 - The tutorial is implemented using Java
 - Although e-speak supports developing e-services using both the Network Object Model (J-ESI) and XML-based Document Exchange Model (e-speak Service Framework Specification (SFS)), this tutorial focuses on developing e-services with J-ESI
- Accelerate development by:
 - Leveraging e-speak features such as persistent protection domains for user accounts, folders, access rights, etc.
 - Incorporating third-party e-services as components
 - Converting legacy applications into e-services

Identify a Service Contract and Service Vocabulary

The service must conform to a service contract and advertise itself in a service vocabulary. You can either use an existing service contract and service vocabulary, or create new ones if existing contracts/vocabularies do not fit your needs:

- Specify the interfaces that others will use to access the service and use the e-speak IDL compiler to generate stubs for the interfaces
- Design a vocabulary that allows the service to be effectively advertised to potential users

Implement the Service

The service implementation provides functions that define a service. Once you specify the contract, you can either use an existing service, transform a legacy application into a service, or create a new service based on the contract and vocabulary you selected.

Implement the Service Deployer

The tutorial service deployer uses the Java e-speak Service Interface (J-ESI), which supports a set of basic and extended services to deploy and use Internet-wide e-services. To deploy a service:

- 1 Describe the service using either the base vocabulary or an existing vocabulary.
- 2 Create a service contract or identify an existing service contract.
- 3 Start an e-speak engine or access a currently-running engine.
- 4 Configure e-speak's standard services, connection parameters (such as port numbers), firewall traversal, and repository.
- 5 Create a service element to register your service by describing the e-service attributes using a vocabulary.
- 6 Register the e-service with an e-speak engine.
- 7 Select an advertising method for the service:
 - **On-line** — E-speak's internal advertising service
 - **Off-line** — An alternative advertising method such as an LDAP-based service
 - **Community** — A feature that automatically advertises the service in multiple e-speak engines
 - **E-services Village** — A directory of available services arranged into useful categories
- 8 Use the service element to start/activate the service.

Implementation

The Praxis service tutorial consists of five files.

Table 6 Praxis tutorial service files

File	Description
tutorial/Praxis/intf/ServiceIntf.esidl	This file defines the service interface that specifies how the service can be accessed. The esidl compiler is used to create <code>ServiceIntf.java</code> and other files which, when compiled, are used by the service.
tutorial/Praxis/service/ServiceImpl.java	This file implements the service interface. It contains the logic needed to implement the service and does not contain e-speak code.
tutorial/Praxis/Service/Service.java	<p>This file contains the e-speak code necessary to deploy an e-speak service.</p> <p>To deploy an e-speak service, <code>Service.java</code>:</p> <ol style="list-style-type: none"> 1 Looks for the properties <code>serviceName</code> and <code>serviceString</code>. 2 Uses these properties to set the e-speak service name, which is then used by clients to discover the service. 3 Sets the string that will be returned by the implementation's <code>getString()</code> method. <p>The file implements the <code>LoaderIntf</code> interface and is dynamically loaded by <code>ServiceLoader</code>.</p>
tutorial/Praxis/Service/ServiceLoader.java	The tutorial uses this utility class to provide startup "housekeeping" for the tutorial's services. Through configuration, it dynamically loads the service class being demonstrated in a particular exercise (for example, <code>service.class</code>) and calls its <code>start()</code> method.
tutorial/Praxis/Service/LoaderIntf.java	This file defines the interface that service classes must implement to be dynamically loaded by <code>ServiceLoader</code> . The class defines a single method: <code>start(Properties)</code> .

Define the Service Contract

The service interface uses an e-speak IDL similar to the Java-RMI IDL. The IDL file must have an `.esidl` extension for the IDL compiler to recognize it as an e-speak IDL file. The file `tutorial/Praxis/intf/ServiceIntf.esidl` is the service interface for the Praxis tutorial.

The service contract allows a client to access a service using one of two methods:

```
package tutorial.Praxis.intf;

import net.espeak.infra.cci.exception.EInvocationException;
import net.espeak.jesi.ESService;
import Java.util.Vector;

public interface ServiceIntf extends ESService {
    public Vector getDescription()
        throws EInvocationException;

    public String getString()
        throws EInvocationException;
}
```

Compile the IDL file using the e-speak IDL compiler:

```
java net.espeak.util.esidl.IDLCompiler ServiceIntf.esidl
```

J-ESI then uses the following IDL compiler-generated files:

- `tutorial/Praxis/intf/ServiceIntf.java`

`ServiceIntf.java` is a copy of `ServiceIntf.esidl` with minor changes to make it an e-speak interface.

- `tutorial/Praxis/intf/ServiceStub.java`

`ServiceStub.java` is the stub class the service finder returns to the client when it discovers the look-up service. The stub class contains the code to create messages, marshal parameters, and send information to the service provider for every method defined in the interface.

- `tutorial/Praxis/intf/ServiceIntfMessageRegistry.java`

J-ESI uses `ServiceIntfMessageRegistry.java` to register the object types.

Implement the Service

Once you specify a contract, the tutorial implements the methods declared in the interface. The `ServiceImpl` class implements the `getDescription` and `getString` method:

- `getDescription` — displays information about the service, including the date and time the service was instantiated and the number of times the `getString` method has been invoked since the counter was reset
- `getString` — displays the service string

The following is an abbreviated version of the `tutorial/praxis/service/ServiceImpl.java` code. Notice that the methods throw an `ESInvocationException` if the e-speak engine detects problems.

```
public ServiceImpl implements ServiceIntf {
    private Java.util.Date whenInstantiated = new Date();
    private long stringInvocations = 0;
    private String serviceString;

    public ServiceImpl (String theString) {
        serviceString=theString;
    }

    public Java.util.Vector getDescription()
    throws ESInvocationException {
        Vector description = new Vector(2,2);
        description.add(new String("When instantiated : " +
            whenInstantiated.toString()));
        description.add(new String("Primary invocations: " +
            stringInvocations));
        return description;
    }

    public String getString()
    throws ESInvocationException {
        stringInvocations++;
        return serviceString;
    }
}
```

Although a service developer creates the service implementation, the implementation does not necessarily contain any e-speak-specific code.

Implement the Service Deployer

In general, a service deployer:

- Establishes a connection to the e-speak engine
- Creates a service description using a vocabulary
- Creates a service element
- Sets the service implementation
- Registers, advertises, and starts the service

The server code samples in this section are extracted from `tutorial/praxis/service/Service.java` and contain the e-speak statements that deploy a service.

Establish a Connection to e-speak

In J-ESI, an application interacts with e-speak through an `ESConnection`:

```
ESConnection connection = new ESConnection(props);
```

where `props` is a `java.util.Properties` object containing e-speak properties. [Table 7](#) lists some of the e-speak properties and their default values. A property uses this default value until another is specified.

Table 7 Default values of e-speak properties

Property	Default value
Accountname	guestconnection
Username	guest
Password	null
Protocol	esip
Hostname	localhost
Portnumber	2950

Table 7 Default values of e-speak properties

Property	Default value
Community	null
Eventcontrol	0
Esurl	esip:localhost:2950

The tutorial uses `tutorial/config/espeak.prop` to define e-speak properties used by the tutorial services and clients. The properties contained in this file load at startup into a `Properties` object that is passed to the `ESConnection` constructor. For example, if the e-speak engine is running on a local machine using port number 51423 and an ESIP protocol, the `espeak.prop` file includes:

```
hostname = localhost
portnumber = 51423
protocol = esip
```

Create a Service Description

The service deployer uses an `ESServiceDescription` object to describe the service using a vocabulary. The default e-speak base vocabulary includes these attributes:

- Name
- Type
- Description
- ESGroup
- ESCategory
- Keywords

Initially, the tutorial uses the base vocabulary and only uses the “Name” attribute to distinguish services. Later, the tutorial will use a custom vocabulary to further differentiate services.

```
ESServiceDescription essd = new ESServiceDescription();
essd.addAttribute(ESConstants.SERVICE_NAME, "Basic");
```

Create a Service Element and Set the Implementation

After the service has been described, the service deployer creates an `ESServiceElement` to represent the service:

```
ESServiceElement serviceElement =  
    new ESServiceElement(connection, essd);
```

The service element contains, among other things:

- A description of the service
- An accessor to modify the description
- The actual service implementation
- Information about the service handler, including the:
 - Queue to which messages to the service are sent
 - Number of threads handling service requests

Next, the deployer provides the service element with an object that implements the service contract:

```
serviceElement.setImplementation(new ServiceImpl());
```

In the tutorial, the `ServiceImpl` class implements the service.

Register, Advertise, and Start the Service

Clients can only discover and access registered services. For a service to be visible to other e-speak engines (besides the local engine), you must advertise it with a local advertising service. You must also start the service so that it can begin servicing requests. You can perform all three functions with service element methods:

```
serviceElement.register();  
serviceElement.advertise();  
serviceElement.start();
```

Demonstration

Compile the Code (Optional)

If desired, you can compile the code by making any necessary modifications to the files and running `Makefile` (on HP-UX or Linux™) or `compile.bat` (on Windows NT®) from the `tutorial/praxis` directory.

Run the Service

In the e-speak desktop:

- 1** Right-click **Host**.
- 2** Select **Create a process from a template** to open the **Please select a template** window.
- 3** Select **A Service**.
NOTE: This is the default instance of the service.
- 4** Click **Create**.
- 5** Right-click **A Service**.
- 6** Select **Process Management > Start Process**.

The **A Service** label on the desktop turns green, indicating that it has started.

Test the Service

To test the service, you must implement and run a client. [Chapter 3, “Basic Client”](#) discusses how to implement a client. For now, verify that the service is running by viewing the process output from the desktop:

- 1 Right-click A Service.**
- 2 Select **Process Management > Process Output**.** A window opens to display the process output for A Service:

```
tutorial.praxis.service.Service
Connected to e-speak.
ServiceImpl object instantiated
Describing service with Name == Basic
Created service element.
Set service implementation.
<Host>:51423: Found Adv
Service:esip://<Host>:51423/proc/resource/ExternalResource/81
Registered service.
Advertised service.
Started service.
```

NOTE: In this example, `<Host>` is the name of your host. Your process output may vary from that shown here.

Chapter 3 Basic Client

This chapter demonstrates implementing and running an e-speak Client to access the service implemented in [Chapter 2](#). To complete this chapter, the basic components of e-speak must be running on your computer, specifically the Process Manager, e-speak Desktop, Core, and default service. Before you begin, ensure that you have completed all of the processes outlined in [Chapter 1, “Getting Started”](#) and [Chapter 2, “Basic Service.”](#)

Concepts

To access a service, a client:

- Creates a connection to the e-speak engine
- Identifies a contract and vocabulary to use when searching for services
- Locates the service using `ESServiceFinder`
- Invokes the methods specified by the service interface

A client obtains a service stub the when it performs a query using the e-speak finder service (`ESServiceFinder`). This stub is a local representation of a remote service that the client uses to invoke service methods.

Implementation

The Praxis client tutorial consists of four files.

Table 8 Praxis tutorial client files

File	Description
tutorial/praxis/client/ ClientAPI.java	Contains the e-speak code used by the client
tutorial/praxis/client/CLI.java	Implements the command line interpreter used for the client and invokes methods in <code>ClientAPI.java</code> to perform e-speak functions
tutorial/praxis/client/ ClientAPIException.java	The exception class thrown by <code>ClientAPI.java</code>
tutorial/praxis/client/ ClientLoader.java	The utility class used to provide startup “housekeeping” for tutorial clients Through configuration, it dynamically loads the client command line interpreter class being used in a particular exercise (for example, <code>CLI</code> class) and calls its <code>start()</code> method

Create a Connection to e-speak

Like services, clients connect to e-speak using an `ESConnection` object:

```
ESConnection connection = new ESConnection(props);
```

where `props` is a `java.util.Properties` object loaded with e-speak properties. Refer to [“Establish a Connection to e-speak” on page 27](#) for a list of properties.

In the tutorial, the `ClientLoader` class loads properties from files passed as parameters into a `Properties` object which, in turn, passes to the `ESConnection` constructor.

Locate the Service

Clients use an `ESServiceFinder` object to query services. These queries can be general, or limited to services that implement a particular interface. For example, in this query the finder only locates the interface developed and implemented in [Chapter 2](#):

```
String interfaceName = ServiceIntf.class.getName();
ESServiceFinder finder =
    new ESServiceFinder(connection, interfaceName);
```

The `ESServiceFinder` uses an `ESQuery` object to specify the constraints for a search. The service vocabulary specifies the allowable types of search constraints. Since the service developed in [Chapter 2](#) uses the e-speak base vocabulary, and the only base vocabulary attribute you can change is the “Name” attribute, search for services by name:

```
String serviceName = "Basic";
ESQuery query = new ESQuery("(Name=='" + serviceName + "')");
```

The finder’s `find()` or `findAll()` methods perform the search:

- `find()` — returns the first service object (service stub) found that implements the `ESService` interface and matches the query
- `findAll()` — returns all service objects that implement the `ESService` interface and match the query

In the tutorial, the finder limits the results to services that implement the `ServiceIntf` interface:

```
ServiceIntf service = (ServiceIntf) finder.find(query);
```

Invoke the Method

Once the client locates a service and obtains the service stub, it can invoke interface methods as if the service was a local object:

```
Vector description = service.getDescription();
String result = service.getString();
```

Demonstration

Compile the Code (Optional)

If desired, you can compile the code by making any necessary modifications to the files and running `Makefile` (on HP-UX or Linux™) or `compile.bat` (on Windows NT) from the `tutorial/praxis` directory.

Run the Client

In the client window:

- 1 Navigate to the tutorial working directory.

NOTE: Refer to [page 2](#) for the working directory path.

- 2 Type `basicClient`.

- 3 Press **Enter**. The command prompt changes to the tutorial client prompt `Praxis>`, indicating that you are now running the e-speak tutorial client.

Verify that the Client is Connected to the Core

In the client window:

- 1 Type `describe core`.

- 2 Press **Enter**. The system displays Core attributes such as the port number, group name, host name, etc.

NOTE: This procedure is a good method to use to verify that the client is connected to the port specified in the Core properties. For more information on editing Core attributes, refer to the *e-speak Programmers Guide*.

Verify that the Client can Interact with a Service

In the client window:

- 1 Type `showall`.

- 2 Press **Enter**. The system now shows that a service currently running:

```
Basic
```

3 Type `describe Basic`.

4 Press **Enter**. The system displays the service's instantiation date and the number of times its `getString()` method has been called. Notice that the number of invocations is `0`.

```
Welcome to service Basic.  
  When instantiated: <date>  
  Primary invocations: 0
```

5 Type `get Basic`.

6 Press **Enter**. The system displays the string associated with the service. In this case, the service string is set to the default value:

```
You have successfully connected to an e-speak service!
```

7 Type `describe Basic`.

8 Press **Enter**. Notice that the number of invocations is `1`.

9 Type `get Basic`.

10 Press **Enter**.

11 Type `describe Basic`.

12 Press **Enter**. The primary invocation number increments by one each time you invoke the `get` command.

Run Multiple Services

In the e-speak desktop:

1 Right-click **Host**.

2 Select **Create a process from a template** to open the **Please select a template** window.

3 Select **Basic Service 1**.

4 Click **Create**. The application adds **Basic Service 1** to the desktop.

5 Right-click **Basic Service 1**.

6 Select **Process Management > Start Process**. The **Basic Service 1** label on the desktop turns green, indicating that it is running.

- 7** Repeat steps 1 through 6 to create **Basic Service 2** and **Basic Service 3**.
- 8** Go to the client window.
- 9** Type `showall`.
- 10** Press **Enter**. The system displays the name of the currently-running services:

```
Basic
```

```
pi-1
```

```
pi-2
```

```
pi-3
```

- 11** Use the `get` and `describe` commands to access the services, for example:

```
get pi-1
```

```
describe pi-3
```

Chapter 4 Managed Services

The e-speak Desktop allows you to control the behavior of a service beyond simply starting and stopping it. To complete this chapter, the basic components of e-speak must be running on your computer, specifically the e-speak Desktop and Core. Before you begin, ensure that you have completed all of the processes outlined in the first three chapters of the tutorial.

Implementation

The Praxis managed service tutorial consists of two files.

Table 9 Praxis tutorial managed service files

File	Description
tutorial/praxis/managedService/ManagedService.java	An extension of tutorial/praxis/service/Service.java that contains the additional code necessary to interact with the e-speak Desktop
tutorial/praxis/managedService/ManagedImpl.java	An extension of tutorial/praxis/service/ServiceImpl.java that allows you to manipulate service strings at runtime

Build a Desktop-Managed Service

In general, to build a desktop-managed service:

- 1** Create a Java class that implements the `net.espeak.management.service.DefaultManagedServiceIntf` interface.
- 2** Register the class as desktop-manageable.
- 3** Set the initial service state. (optional)
- 4** Designate system properties as remotely-modifiable by the **Manage Runtime Configuration** window. (optional)

Implement the `DefaultManagedServiceIntf` Interface

The tutorial implements the `DefaultManagedServiceIntf` interface in the `tutorial.praxis.managedservice.ManagedService` class (`tutorial/praxis/managedservice/ManagedService.java`). The desktop invokes interface-defined methods when you click the buttons in the desktop **Service Manager Control** window, as shown:

```
public void init() {
    // Code to execute when the "init" button is pressed.
    // Tutorial action: Create and deploy the service.
}

public void pause() {
    // Code to execute when the "pause" button is pressed.
    // Tutorial action: Stop the service from accepting
    // client requests.
}

public void resume() {
    // Code to execute when the "resume" button is pressed.
    // Tutorial action: Resume accepting client requests.
}
```

```
public void reset() {
    // Code to execute when the "reset" button is pressed.
    // Tutorial action: reset service invocation counter.
}

public void remove() {
    // Code to execute when the "remove" button is pressed.
    // Tutorial action: remove service from e-speak.
}
```

The desktop calls an additional method to retrieve the service's managed name:

```
public String getServiceName() {
    return "ManagedService";
}
```

This is the name that appears on the desktop display. The application uses this name when setting the service's state; it is not necessarily related to the e-speak service name used when deploying the service.

Register the Class as Desktop-Manageable

Registering a class as desktop-manageable is as simple as passing an instance of the class to a static method of the `net.espeak.management.service.ServiceManagement` class. In the tutorial, the object that performs the registration also implements the `DefaultManagedServiceIntf` interface, so registration is accomplished as shown:

```
net.espeak.management.service.ServiceManagement.export(this);
```

Set the Initial State

As the desktop manages a service, it tracks the service's state. When started, a desktop-managed service's state is initially "Service Closed." The state does not change to "Service Running" until you click **init** in the **Service Manager Control** window. You can, however, instruct the desktop to start a service in a "Service Running" state by setting its state:

```
net.espeak.management.service.ServiceManagement.setServiceState
("ManagedService", "Service Running");
```

where "ManagedService" is the same name as that returned by the `getServiceName()` method and "Service Running" is the new state.

NOTE: Setting the state does not invoke interface methods. The state only determines which buttons are active in the **Service Manager Control** window.

The available service states are defined as a `public static final String` in the `net.espeak.management.service.DefaultServiceManagementImpl` class. [Table 10](#) lists additional states and their values.

Table 10 Service state default values

Variable Name	Value
CLOSED	"Service Closed"
ERROR	"Service Error"
STOPPED	"Service Stopped"
RUNNING	"Service Running"
DEGRADED	"Service Degraded"

You could use this instead of "Service Running," for example:

```
net.espeak.management.service.DefaultServiceManagementImpl.RUNNING
```

Allow Property Modifications

The desktop's Manage Runtime Configuration feature allows you to view, and possibly modify, a service's system properties at run-time. To be able to modify a property using the desktop, you must register the service property as a modifiable property:

```
net.espeak.management.config.RemoteConfiguration
    .allowRuntimeChange("ServiceString");
```

where "ServiceString" is the name of the property.

The tutorial's `tutorial.praxis.managedservice.ManagedImpl` service implementation (`tutorial/praxis/managedservice/ManagedImpl.java`) uses the system properties object to track the string value it is configured to return. When the service's `getString()` method is invoked, it calls the `getServiceString()` method, which retrieves the string value from the properties object and returns it to the caller:

```
public String getServiceString() {
    return System.getProperties().getProperty("ServiceString");
}
```

This allows you to change, at runtime, the value of the string that the tutorial service returns.

Demonstration

Compile the Code (Optional)

If desired, you can compile the code by making any necessary modifications to the files and running `Makefile` (on HP-UX or Linux™) or `compile.bat` (on Windows NT) from the `tutorial/praxis` directory.

Start a Managed Service

In the e-speak Desktop:

- 1 Right-click **Host**.
- 2 Select **Create a process from a template** to open the **Please select a template to create on host** window.
- 3 Select **Desktop Managed**.
- 4 Click **Create**. Desktop Managed appears on the desktop.
- 5 Right-click **Desktop Managed**.
- 6 Select **Process Management > Start Process**. The Desktop Managed service option on the desktop turns green, indicating that it is running. After a few moments, an icon and “E-speak Service” label appears below the service, indicating that it is a desktop-managed service.

Managed Services Displayed on the Desktop

When you first start a desktop-managed service, the service appears on the desktop as shown in [Figure 3](#). This is the name returned by the `getServiceName()` method.



Figure 3 E-speak service label as displayed when you start a managed service

However, when you right-click the e-speak service and select a menu option, the label changes to ManagedService as shown in [Figure 4](#):

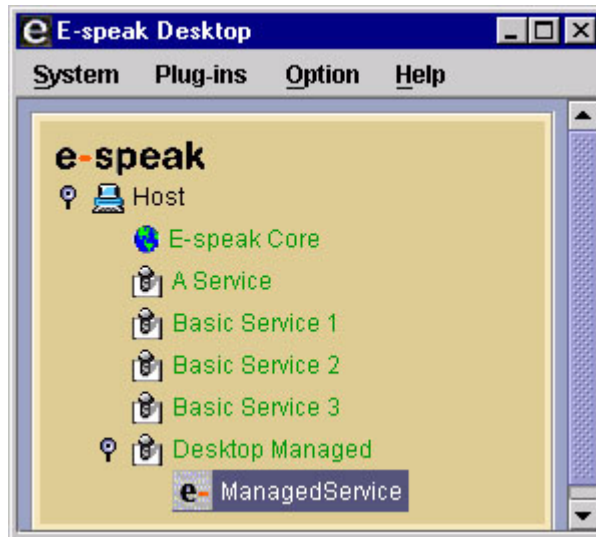


Figure 4 E-speak service label as displayed when you right-click a managed service

Verify That the Service is Running

In the client window:

- 1 Type `showall`.
- 2 Press **Enter**. The managed service, `oxy-1`, now appears in the services list.
- 3 Type `get oxy-1`.
- 4 Press **Enter**. The system displays the string associated with `oxy-1`:

```
exact estimate
```

Pause the Managed Service

In the e-speak Desktop:

- 1 Right-click **E-Speak Service**.
- 2 Select **Service Control** to open the **Service Manager Control** window.

- 3** Click **Pause**.
- 4** Type `showall` in the client window.
- 5** Press **Enter**. Because you paused `oxy-1`, it does not appear in the services list.
- 6** Type `describe oxy-1`.
- 7** Press **Enter**. The system generates an error since you cannot display the attributes of a paused service.
- 8** Click **Resume** in the Service Manager Control window.
- 9** Type `showall` in the client window.
- 10** Press **Enter**. The service, `oxy-1`, once again appears in the list.
- 11** Type `describe oxy-1`.
- 12** Press **Enter**. The system displays the service attributes. Notice that pausing and resuming a service does not change its instantiation date.

Reset the Service Invocation Counter

In the Service Manager Control window:

- 1** Click **Pause**.
- 2** Click **Reset**.
- 3** Click **Init**.
- 4** Type `describe oxy-1` in the client window.
- 5** Press **Enter**. Notice that the service invocation counter has been reset to zero.

Remove and Restart a Managed Service

In the Service Manager Control window:

- 1** Click **Pause**.
- 2** Click **Reset**.
- 3** Click **Remove**.

- 4 Type `showall` in the client window.
- 5 Press **Enter**. Oxy-1 no longer appears in the services list.
- 6 Click **Init** in the Service Manager Control window.
- 7 Type `showall` in the client window.
- 8 Press **Enter**. Oxy-1 again appears in the services list.
- 9 Type `describe oxy-1`.
- 10 Press **Enter**. Since you removed and restarted the service, notice that the instantiation date has been updated to the current day and time.

Edit Service Configuration at Runtime

In the e-speak Desktop:

- 1 Right-click **Desktop Managed**.
- 2 Select **Configuration > Manage Runtime Configuration** to open the Configuration for Managed Service window. This window displays the system properties associated with the service.
- 3 Locate the ServiceString property in the list.
- 4 Type a new oxymoron into the text field:
`modern history`
- 5 Click **Apply**.
- 6 Click **Yes**.
- 7 Click **Close**.
- 8 Type `get oxy-1` in the client window.
- 9 Press **Enter**. The service now displays the new value:
`modern history`

Stop the Managed Service (Optional)

In the e-speak Desktop:

- 1** Right-click **Desktop Managed**.
- 2** Select **Process Management>Stop Process**.

Chapter 5 Events

This chapter demonstrates implementing and running a service that uses e-speak events. To complete this chapter, the basic components of e-speak must be running on your computer, specifically the e-speak Desktop and Core. Before you begin, ensure that you have completed all of the processes outlined in the first three chapters of the tutorial.

Concepts

E-speak provides an events-service to provide a publish-subscribe mechanism for communication built upon e-speak messaging.

NOTE: For detailed information on events, please refer to the *e-speak Programmers Guide*.

Event Model

The e-speak event model consists of four entities.

Table 11 E-speak event model

Entity	Function
Publisher	Generates event messages
Listener	Receives events
Distributor	Receives events from publishers and forwards them to listeners
Subscriber	Registers interest in an event with a distributor and designates a listener to receive the event

The subscriber and listener are typically the same entity (object). Similarly, a publisher may act as a distributor of its own events.

Implementation

The Praxis event tutorial consists of four files.

Table 12 Praxis tutorial event files

File	Description
tutorial/praxis/events/ EventDistributor.java	The e-speak events distributor
tutorial/praxis/events/ EventPublisher.java	An extension of tutorial/praxis/ managedService/ ManagedService.java that publishes events when you press Service Manager Control buttons
tutorial/praxis/events/ EventCLI.java	An extension of tutorial/praxis/ client/CLI.java that instantiates and initializes EventClientAPI.java
tutorial/praxis/events/ EventClientAPI.java	An extension of tutorial/praxis/ client/ClientAPI.java that supports subscribing to, unsubscribing from, and receiving notification of e-speak events

Creating an Event Distributor

The `tutorial.praxis.events.EventDistributor` class (`tutorial/praxis/events/EventDistributor.java`) implements the tutorial's event distributor. The `ServiceLoader` class initially loads the distributor. When the event distributor's `start()` method is invoked, it:

- Connects to e-speak:

```
ESConnection connection = new ESConnection(properties);
```

where `properties` is a `java.util.Properties` object containing e-speak properties.

NOTE: For more details, refer to [, "Chapter 2, "Basic Service."](#)

- Creates an `ESDistributor` object to perform the distributor function:

```
ESDistributor distributor = new ESDistributor(connection);
```

- Specifies the names of the events to be distributed:

```
distributor.addEvent("Praxis");
```

NOTE: A distributor can distribute more than one type of event by calling the `addEvent()` method multiple times using additional event names.

- Starts distributing events:

```
distributor.start();
```

Create an Events Listener/Subscriber

A listener implements the `ESListenerIntf` interface and defines the `notify()` and `notifySync()` methods. The distributor calls these methods when a subscriber registers events. The tutorial implements the event listener in the `tutorial.praxis.events.EventClientAPI` class (`tutorial/praxis/events/EventClientAPI.java`). It is instantiated by the `EventCLI` class and initialized when the `EventClientAPI.subscribe()` method:

- Creates an `ESSubscriber` instance:

```
ESSubscriber subscriber = new ESSubscriber(connection);
```

where `connection` is a reference to an e-speak `ESConnection` object.

- Registers the object that acts as the events listener:

```
subscriber.setImplementation(this);
```

NOTE: In the tutorial, the same object performs both the subscriber and listener functions. This is the object that implements the `ESListenerIntf` interface and receives notification via its `notify()` and `notifySync()` methods.

- Specifies the names of the events that are of interest:

```
subscriber.addEvent("Praxis");
```

NOTE: A subscriber can register for more than one type of event by calling the `addEvent()` method multiple times using additional event names. The event names specified here correspond to the event names distributed by the events distributor.

- Accepts event notifications:

```
subscriber.subscribe();
```

You can stop a subscriber from receiving events by unsubscribing it:

```
subscriber.unsubscribe();
```

When the subscriber receives events, the distributor invokes the listener's `notify()` or `notifySync()` methods. The tutorial implements these methods as shown, which displays a message when an event arrives:

```
public void notify(Event event)
throws ESInvocationException {
    handleEvent(event);
}
```

```
public String notifySync(Event event)
throws ESInvocationException {
    handleEvent(event);
    return "Notified";
}
```

```
private void handleEvent(Event event) {
    String type = event.getEventType();
    String payload = event.getPayload().toString();

    System.out.print("\n\n" + type + " event received: " + payload
        + "\n\n" + prompt);
}
```

Create an Event Publisher

The `tutorial.praxis.events.EventPublisher` class (`tutorial/praxis/events/EventPublisher.java`) implements the tutorial's event publisher. It extends the tutorial `ManagedService` class and publishes events when you click the buttons in the **Service Manager Control** window.

Register as an Event Publisher

To register as an event publisher, a service:

- Creates an `ESPublisher` object:

```
ESPublisher publisher = new ESPublisher(connection);
```

where `connection` is a reference to an e-speak `ESConnection` object

- Lists the names of the events it will publish:

```
publisher.addEvent("Praxis");
```

NOTE: A publisher can register to publish more than one type of event by calling the `addEvent()` method multiple times using additional event names. The event names specified here correspond to the event names distributed by the events distributor.

- Starts the publisher:

```
publisher.publish();
```

Publish an Event

To publish an event, a service:

- Creates an event object for a particular event name:

```
Event event = new Event("Praxis");
```

NOTE: The name should be one of the names specified when you initialized the publisher.

- Sets the event payload:

```
event.setPayload("EventPublisher initialized");
```

NOTE: The payload can be any object that can be serialized.

- Publishes the event:

```
publisher.sendNotify(event);
```

The event is received by the listener using its `notify()` method.

Demonstration

Compile the Code (Optional)

If desired, you can compile the code by making any necessary modifications to the files and running `Makefile` (on HP-UX or Linux™) or `compile.bat` (on Windows NT) from the `tutorial/praxis` directory.

Start the Events Distributor

In the e-speak Desktop:

- 1 Right-click **Host**.
- 2 Select **Create a process from a template** to open the **Please select a template to create on host** window.
- 3 Select **Events Distributor**.
- 4 Click **Create**. The application adds the Events Distributor to the desktop.

5 Right-click **Events Distributor**.

6 Select **Process Management > Start Process**.

The Events Distribute option on the desktop turns green, indicating that it is running.

Verify That the Events Distributor is Running

In the e-speak Desktop:

1 Right-click **Events Distributor**.

2 Select **Process Management > Process Output**.

The Process Management window opens and displays information about the events distributor. Look for the message:

```
Distributor started for event 'Praxis'
```

Start the Events Client

In the client window:

TIP: If you are already running a client, type `exit` and then press **Enter** to return to the command prompt.

1 Type `eventsClient`.

2 Press **Enter**. Once the events client successfully subscribes to the event, it displays the message:

```
Subscribed to event 'Praxis'
```

Start the Events Service

In the e-speak Desktop:

1 Right-click **Host**.

2 Select **Create a process from a template** to open the **Please select a template to create on host** window.

3 Select **Events Service**.

- 4 Click **Create**. The Events Service appears on the desktop.
- 5 Right-click **Events Service**.
- 6 Select **Process Management > Start Process**. The Events Service label on the desktop turns green, indicating that it is running. The client window displays the message:

```
Praxis event received: EventPublisher initialized
```

Generate Additional Event Messages

In the e-speak Desktop:

- 1 Right-click **E-Speak Service**.
- 2 Select **Service Control** to open the **Service Manager Control** window.
- 3 Click **Pause**. The client window displays the message:

```
Praxis event received: EventPublisher paused
```

- 4 Click **Resume**. The client window displays the message:

```
Praxis event received: EventPublisher resumed
```

Stop the Events Service (Optional)

In the client window:

- 1 Type `exit`.
- 2 Press **Enter** to return to the command prompt.

In the e-speak Desktop:

- 1 Right-click **Events Service**.
- 2 Select **Process Management > Stop Process**.
- 3 Right-click **Events Distributor**.
- 4 Select **Process Management > Stop Process**.

Chapter 6 Folders

This chapter demonstrates implementing and using e-speak folders. To complete this chapter, the basic components of e-speak must be running on your computer, specifically the e-speak Desktop and Core. Before you begin, ensure that you have completed all of the processes outlined in the first three chapters of the tutorial.

Concepts

The folder service allows you to manage services similar to the way you manage local files in a standard operating system. Persistent folders appear when you reconnect and behave like persistent bookmarks organized hierarchically.

Implementation

The Praxis tutorial folders consists of two files.

Table 13 Praxis tutorial folder files

File	Description
<code>tutorial/praxis/folders/FoldersCLI.java</code>	An extension of <code>tutorial/praxis/client/CLI.java</code> that contains commands to save service stubs, delete stubs from e-speak folders, and instantiate the <code>FoldersClientAPI</code> object
<code>tutorial/praxis/folders/FoldersClientAPI.java</code>	An extension of <code>tutorial/praxis/service/ServiceImpl.java</code> that allows you to manipulate service strings at runtime

Managing Bindings Using Folders

One of the primary reasons e-speak can support loosely-coupled, distributed services is that it does not rely on global naming. In many traditional distributed systems, the distribution is possible only because clients know the names of service providers and use those names to access particular services. Because there are no global names in e-speak, clients can assign names to the services they find or create. This name is independent of a name that another client may use for the same service.

For example, one client may refer to a printer service as “Marketing Printer,” while another may refer to the same printer service as “Engineering Printer.” This type of independent naming allows services to be migrated or upgraded without having to change the client-side applications that use them.

In J-ESI, clients manage local names using folders. Folders are similar to directories in traditional file systems. Clients can create bindings between the names and services they find, then store the bindings in a folder.

Essentially, folders enable clients to build a local hierarchical name tree. Every user account in an e-speak Core has a folder that is its root (typically denoted by /). Clients can access the root folder in their session by invoking the `getRootFolder` method in `ESConnection`. For example:

```
ESFolder root = connection.getRootFolder();
```

Creating Folders

Client-created folders are either persistent or transient. A transient folder exists only for the duration of the connection in which you create it. A persistent folder exists indefinitely and is available each time you connect. If you back up the Core in a database, persistent folders are still available after Core reboots since the Core, not the client, maintains folder states.

Creating Transient Folders

Use the folder constructor to create subfolders of the root folder; use the `createSubFolder` method to create subfolders of non-root folders. To create a transient folder called `/guest` in the root folder, for example, use the following constructor of `ESFolder`:

```
String userName = "guest";  
ESFolder serviceDir = new ESFolder(connection, userName);
```

Creating Persistent Folders

To create a persistent folder called `/guest` in the root folder:

```
String userName = "guest";  
ESFolder home = new ESFolder(connection, userName, true);
```

Because a root folder is persistent by default, you must simply invoke the `createSubFolder` method on the root folder to create a persistent folder. Because the folder `/guest` is a persistent folder, any subfolder of `/guest` is also persistent.

Naming Found Services

Folders typically manage name bindings on Services you discover or create. For example, you may add bindings for discovered print Services in a persistent folder such as `/home/services/printers`. Because the bindings are stored in a persistent folder, you can access previously discovered printers anytime you reconnect by simply looking in your printers folder.

The following code fragment illustrates how the tutorial client locates the subfolder `/guest` in the root folder. If the subfolder does not exist, the client creates a subfolder before finding the service and creating a name binding within it:

```
String name = "pi-1";  
String userName = "guest";  
ESFolder serviceDir = null;  
try {  
    ESFolder root = connection.getRootFolder();  
    serviceDir = root.getSubFolder(userName);  
} catch (InvalidNameException ine) {  
    serviceDir = new ESFolder(connection, userName, true);  
}  
ServiceIntf service = getServiceByName(name);  
serviceDir.add(name, service);
```

By changing the name used in the add method, the client can bind the service using any name:

```
String name = "pi-1";
String newName = "MyPiService";
String userName = "guest";
ESFolder serviceDir = null;
try {
    ESFolder root = connection.getRootFolder();
    serviceDir = root.getSubFolder(userName);
} catch (InvalidNameException ine) {
    serviceDir = new ESFolder(connection, userName, true);
}
ServiceIntf service = getServiceByName(name);
serviceDir.add(newName, service);
```

Removing Services From Folders

The client removes a name binding for a service from a folder using:

```
serviceDir.remove("MyPiService");
```

Demonstration

Compile the Code (Optional)

If desired, you can compile the code by making any necessary modifications to the files and running Makefile (on HP-UX or Linux%) or compile.bat (on Windows NT®) from the tutorial/praxis directory.

Start the Folder Client

In the client window:

TIP: If you are already running a client, type `exit` and then press `Enter` to return to the command prompt.

- 1** Type `foldersClient`.
- 2** Press **Enter**.

Save a Service Object in a Folder

In the client window:

- 1** Type `showall`.
- 2** Press **Enter**. The system lists the currently-running services:

```
Basic  
pi-1  
pi-2  
pi-3
```
- 3** Type `save pi-1`.
- 4** Press **Enter**. The system saves a copy of the service into the local folder.
- 5** Type `showall`.

- 6 Press **Enter**. The service pi-1 now appears in both the standard services list and in your folder:

```
Services found in home folder:
```

```
pi-1
```

```
Available service names:
```

```
Basic
```

```
pi-1
```

```
pi-2
```

```
pi-3
```

NOTE: If you started services other than **A Service**, **Basic Service 1**, **Basic Service 2**, and **Basic Service 3**, your service list may vary from that shown here.

Save a Service Object under an Alias

The syntax to use to save a service to a folder under an alias is:

```
save servicename newname
```

Remember, creating an alias simply creates a new reference to an existing service.

In the client window:

- 1 Type `save pi-2 MyFavoriteService`.
- 2 Press **Enter**. The system creates an alias called `MyFavoriteService` for the service `pi-2` in your folder.
- 3 Type `showall`.

4 Press **Enter.** The system displays:

```
Services found in home folder:
```

```
pi-1
```

```
MyFavoriteService
```

```
Available service names:
```

```
Basic
```

```
pi-1
```

```
pi-2
```

```
pi-3
```

Pi-2 and MyFavoriteService are simply two names for the same service. If you get both service names, they return the same string: the value of PI, or 3.14159265.

Delete an Alias From Your Folder

In the client window:

1 Type `delete MyFavoriteService`.

2 Press **Enter**. The system removes the alias from your folder.

3 Type `showall`.

4 Press **Enter**. The system displays:

```
Services found in home folder:
```

```
pi-1
```

```
Available service names:
```

```
Basic
```

```
pi-1
```

```
pi-2
```

```
pi-3
```

MyFavoriteService no longer appears in the folder list. Notice that this action has no affect on the original service, pi-2.

Service Aliases

You can use the same alias name to access different services. The alias accesses the service to which it is currently bound. In this example, you use the alias `myPi` to retrieve two different values of `pi` from two different services.

In the client window:

- 1** Type `save pi-1 myPi`.
- 2** Press **Enter**. The system creates an alias called `myPi`.
- 3** Type `get myPi`.
- 4** Press **Enter**. The client window displays the very accurate value of `pi`.
- 5** Type `delete myPi`.
- 6** Press **Enter** to delete `myPi` from the folder.
- 7** Type `save pi-3 myPi`.
- 8** Press **Enter**.
- 9** Type `get myPi`.
- 10** Press **Enter**. The client window displays the less accurate value of `pi`.

Stop the Folders Client (Optional)

In the client window:

- 1** Type `exit`.
- 2** Press **Enter** to return to the command prompt.

Chapter 7 Threads

This chapter demonstrates implementing and using e-speak threads. To complete this chapter, the basic components of e-speak must be running on your computer, specifically the e-speak Desktop and Core. Before you begin, ensure that you have completed all of the processes outlined in the first three chapters of the tutorial.

Concepts

Thread programming using J-ESI includes:

- Threads the service handler uses to handle service requests
- Threads in the client program that may share a connection to the e-speak Core

This tutorial discusses threads in the client environment.

In a multi thread environment, it is more efficient for a client application to share a Core connection among threads. Unfortunately, in this case, threads also share any state information stored in the ESConnection. An application can, however, use ESThread to clone its ESConnection, allowing each thread to maintain separate state information while sharing the underlying connection to the e-speak Core.

Similar to Java threads, you can use J-ESI threads to create classes that:

- Extend ESThread directly (similar to extending `Thread`)
- or
- Implement the ESRunnable interface (similar to the implementing the Runnable interface) and pass it to the constructor of ESThread

Implementing the `ESRunnable` interface is the preferred method of using `ESThread`. The `tutorial.praxis.threads.ThreadClientAPI` and `tutorial.praxis.threads.StringGetter` classes demonstrate this method. Both are located in the file `tutorial/praxis/threads/ThreadClientAPI.java`.

Implementation

The Praxis threads tutorial consists of two files.

Table 14 Praxis tutorial threads files

File	Description
<code>tutorial/praxis/threads/ThreadCLI.java</code>	An extension of <code>tutorial/praxis/client/CLI.java</code> that adds the <code>getall</code> command and instantiates <code>ThreadClientAPI</code>
<code>tutorial/praxis/threads/ThreadClientAPI.java</code>	Demonstrates: <ul style="list-style-type: none"> • Creating a class that implements the <code>ESRunnable</code> interface • Using <code>ESThread</code>

Implementing `ESRunnable`

The `ESRunnable` interface extends the Java `Runnable` interface. A class implementing `ESRunnable` must provide three methods:

- `public void run()`

This method is called when the owning `ESThread`'s `start()` method is called.

- `public ESConnection getConnection()`

The constructor of the owning `ESThread` calls this method. It returns the original `ESConnection` object created by the parent thread.

- `public void setConnection(ESConnection clonedConnection)`

The `ESThread` constructor calls this method after cloning the `ESConnection` object returned by `getConnection()`. The new thread uses this `ESConnection` for all e-speak activities.

Table 15 `ESRunnable` example

Code	Function
<pre>class StringGetter implements ESRunnable { private ESConnection connection; public StringGetter(ESConnection originalConnection) { connection = originalConnection; } }</pre>	<p>This method implements the <code>ESRunnable</code> interface. The constructor receives and saves the parent object's e-speak <code>ESConnection</code> object.</p>
<pre>public ESConnection getConnection() { return connection; }</pre>	<p>The <code>ESThread</code>'s constructor calls this method, which returns the original <code>ESConnection</code> object.</p>
<pre>public void setConnection(ESConnection clonedConnection) { connection = clonedConnection; }</pre>	<p>The <code>ESThread</code>'s constructor calls this method to pass a cloned copy of the <code>ESConnection</code> object. The thread uses this cloned copy when interacting with the e-speak.</p>
<pre>public void run() { // // Do whatever this thread's supposed to do // }</pre>	<p>This method is called when the <code>ESThread</code>'s <code>start()</code> method is invoked.</p>

Using ESThread/ESRunnable

To use the ESThread/ESRunnable technique, an application must:

1 Connect to e-speak:

```
ESConnection connection = new ESConnection(props);
```

2 Create an instance of a class that implements the ESRunnable interface and give it access to the ESConnection object.

NOTE: You can accomplish this using the class constructor.

3 Create an instance of ESThread and pass the ESRunnable object to its constructor:

```
ESRunnable esrunnable = new StringGetter(connection);  
ESThread theThread = new ESThread(esrunnable);
```

4 Start the thread:

```
theThread.start();
```

The tutorial calls the `ThreadClientAPI.getAllByName()` method when you enter the `getall` command in the tutorial client. It retrieves a list of the currently-running tutorial services, then launches `StringGetter` threads to find and retrieve the service's assigned strings simultaneously.

Demonstration

Compile the Code (Optional)

If desired, you can compile the code by making any necessary modifications to the files and running `Makefile` (on HP-UX or Linux™) or `compile.bat` (on Windows NT®) from the `tutorial/praxis` directory.

Start the Threads Client

In the client window:

TIP: If you are already running a client, type `exit` and then press **Enter** to return to the command prompt.

- 1 Type `threadsClient`.
- 2 Press **Enter**.

Display Results from All Currently-Running Services

In the client window:

- 1 Type `showall`.
- 2 Press **Enter** to display all currently-running services.
- 3 Type `getall`.
- 4 Press **Enter**. The system retrieves and displays strings from all of the services in the list in random order.

Stop the Threads Client (Optional)

In the client window:

- 1 Type `exit`.
- 2 Press **Enter** to return to the command prompt.

Chapter 8 Security

This chapter demonstrates implementing and using e-speak security. To complete this chapter, the basic components of e-speak must be running on your computer, specifically the e-speak Desktop and Core. Before you begin, ensure that you have completed all of the processes outlined in the first three chapters of the tutorial.

Concepts

E-speak security provides the following features:

- Distributed authorization and access control using public-key certificates
- Authorization delegation
- SLS (Session Layer Security), a security protocol that provides:
 - Privacy
 - Message integrity
 - Authentication
- Ability to create secure end-to-end connections through gateways
- Tools to create key pairs and manage certificates

The Security Process

E-speak entities interact with one another through secure sessions using the SLS protocol. Since both entities in a secure session must be authenticated to each other, SLS-secure sessions accomplish this by using challenge-response negotiations based on public-key cryptography. Every entity has a set of public/private keys.

Digitally-signed certificates, part of the SLS protocol, control access to services. These certificates authorize entities to access and make use of services. Principals (issuing entities) sign certificates and grant them to the entities who require them. Certificates can also be linked together through delegation to grant composite access to a service.

When security is enabled, each time you start an e-speak client (a service client or service provider), the application reads two files: trust assumptions and certificates. These files simply contain lists of certificates. In general, a service provider requires only one certificate to access the e-speak engine. The client generally requires two certificates: one to access the service engine and another to access the actual service.

To interact with a security-enabled service, a client must obtain the appropriate certificate, signed by the service provider's principal or by another principal who is linked (via delegation) to a principal listed in the service provider's trust assumption. Service providers and clients require certificates to allow them to access APIs. For example, to register a service and perform a query, the e-speak engine's principal issues certificates to both the service provider and client, granting them access.

About the PSE Manager

The Private Security Environment (PSE) Manager is a graphical interface that allows you to:

- Create and save new PSE binary files
 - Create and store public and private key pairs
- NOTE: The PSE is an encrypted, password-protected environment to keep the data secure.
- Create and edit certificates
 - Validate certificates by checking the signature of the certificate using the issuer's public key
 - Sign certificates using a private key

Please refer to the e-speak System Administration Guide for details on how to use PSE Manager.

Method Tag Used in Certificates

You can create masks and associate them with service elements or certificates. Using masks, you can allow unauthorized operations; that is, any client can perform any operation that is specified in the mask for a particular service. If an operation is not included in a mask, only those clients that have certificates authorizing access can invoke the operation.

E-speak contains two types of masks:

- Metadata masks for metadata operations
- Resource masks for resource-specific operations

Masks are specified as tags. The basic method tag format is:

```
(net.espeak.method <interface name> <method name>)
```

The interface name is the fully-qualified name of an interface class; the method name is the name of the method in the interface, plus the argument types.

Masks are completely general tags. Fields can use the tag matching features such as sets, prefixes, and ranges. The interface and method names do not have to be literal strings. For example, the following tag masks methods beginning with “search”:

```
(net.espeak.method LookupServiceIntf (* prefix search))
```

To include all methods in the interface, set the tag as:

```
(net.espeak.method LookupServiceIntf (*))
```

To include all methods in all interfaces, set the tag as:

```
(net.espeak.method (*) (*))
```

To allow all access, set the tag as:

```
(*)
```

Implementation

Configuration

The Security Properties section of the e-speak configuration file allows you to set up e-speak's security environment. The attributes in the file are accompanied by descriptions to explain their meanings.

To define the default PSE file, add the `.pse.storefile` attribute:

```
! Default name of the keystore file.
net.espeak.security.pse.storefile = core.pse
```

NOTE: The tutorial uses four e-speak configuration files, all located in `tutorial/praxis/config` directory.

Table 16 Security configuration files

Configuration file...	Used by...	Associated with...
core.cfg	e-speak cores	core.pse
service.cfg	e-speak services	service.pse
trusted.cfg	most clients	client.pse
untrusted.cfg	untrusted client demo	client.pse

You can set the mode for passphrase inquiry as follows:

```
! Passphrase mode looks for the passphrase property.
.pse.mode = passphrase
! Define the passphrase.
net.espeak.security.pse.passphrase = default passphrase
```

For more information on configuration files, please refer to the *e-speak System Administration Guide*.

The security-related aspects of the tutorial are outlined in [Table 17](#) and [Table 18](#).

Table 17 Security configuration files, PSE files, passphrases, and roles

Components	Cores	Services	Trusted Clients	Untrusted Clients
Configuration file	core.cfg	service.cfg	trusted.cfg	untrusted.cfg
PSE file	core.pse	service.pse	client.pse	client.pse
Passphrase PSE	core	service	client	client
Role	core	service	trustedclient	untrustedclient

Table 18 Security certificate files

Certificates	
Core	coreaccess.certs
Services	serviceaccess.certs
Trusted Clients	trustedclientaccess.certs
Untrusted Clients	untrustedclientaccess.certs

Please refer to Chapter 6, “Configuring Security,” in the *e-speak System Administration Guide* for specific instructions on how to:

- Create keys and certificates
- Use the PSE Manager
- Define security parameters in the configuration file

Certificates

Certificates grant access to resources and services. Each component of the tutorial uses a separate certificate file containing certificates issued by an entity (issuer) to give the component (subject) access to particular resources (attribute tag). The tutorial certificate files are located in `tutorial/praxis/config`.

Table 19 coreaccess.certs

Issuer	Subject	Attribute Tag
core	core	(net.espeak.method (*)(*)(*))
core	core	(net.espeak.action (*)(*)(*))

Table 20 serviceaccess.certs

Issuer	Subject	Attribute Tag
service	service	(net.espeak.method (*)(*)(*))
core	service	(net.espeak.method (*)(*)(*))
trustedclient	service	(net.espeak.method (*)(*)(*))
service	service	(net.espeak.action (*)(*)(*))
core	service	(net.espeak.action (*)(*)(*))
trustedclient	service	(net.espeak.action (*)(*)(*))

Table 21 trustedclientaccess.certs

Issuer	Subject	Attribute Tag
trustedclient	trustedclient	(net.espeak.method (*)(*)(*))
core	trustedclient	(net.espeak.method (*)(*)(*))
service	trustedclient	(net.espeak.method (*)(*)(*))
trustedclient	trustedclient	(net.espeak.action (*)(*)(*))

Table 21 trustedclientaccess.certs

Issuer	Subject	Attribute Tag
core	trustedclient	(net.espeak.action (*)*)(*)
service	trustedclient	(net.espeak.action (*)*)(*)

Table 22 untrustedClientAccess.certs

Issuer	Subject	Attribute Tag
untrustedclient	untrustedclient	(net.espeak.method (*)*)(*)
core	untrustedclient	(net.espeak.method (*)*)(*)
service	untrustedclient	(net.method tutorial.praxis.intf.ServiceIntf getDescription() (*))
untrustedclient	untrustedclient	(net.espeak.action (*)*)(*)
core	untrustedclient	(net.espeak.action (*)*)(*)
service	untrustedclient	(net.espeak.action (*)*)(*)

Demonstration

Run the Untrusted Client

Use this procedure to verify that the `untrustedClient` has only limited access to e-speak functions.

In the client window:

TIP: If you are already running a client, type `exit` and then press **Enter** to return to the command prompt.

- 1 Type `untrustedClient`.
- 2 Press **Enter**.
- 3 Type `describe Basic`.
- 4 Press **Enter**. E-speak successfully displays the service attributes.
- 5 Type `get Basic`.
- 6 Press **Enter**. The command fails because this client is not authorized to `get` the service.

Stop the Untrusted Client (Optional)

In the client window:

- 1 Type `exit`.
- 2 Press **Enter** to return to the command prompt.

Chapter 9 Vocabularies

This chapter demonstrates implementing, deploying, and using an e-speak service vocabulary. To complete this chapter, the basic components of e-speak must be running on your computer, specifically the e-speak Desktop and Core. Before you begin, ensure that you have completed all of the processes outlined in the first three chapters of the tutorial.

Concepts

A vocabulary is a set of attributes and properties that defines a service. The vocabulary structure is based on the market for which it was defined. For example, the vocabulary that defines a printer service might include attributes such as manufacturer, model name, DPI, speed, color, and cost. A vocabulary that defines apparel merchandise might include size, color, material, and cost. Clients and services alike can find and use vocabularies registered in the community.

This chapter demonstrates:

- How to deploy a custom vocabulary (PraxisVocab)
- How a service finds a vocabulary and uses the vocabulary to describe itself
- How a client uses a vocabulary to locate services

Implementation

The Praxis vocabulary tutorial consists of four files.

Table 23 Praxis tutorial vocabulary files

File	Description
tutorial/praxis/vocabulary/VocabCLI.java	An extension of tutorial/praxis/client/CLI.java that adds the show command and instantiates VocabClientAPI.java
tutorial/praxis/vocabulary/VocabClientAPI.java	An extension of tutorial/praxis/client/ClientAPI.java that supports service queries based on a custom vocabulary
tutorial/praxis/vocabulary/VocabDeployer.java	The e-speak code used to create and deploy an e-speak custom vocabulary
tutorial/praxis/vocabulary/VocabService.java	An extension of tutorial/praxis/service/Service.java that deploys a service using a custom vocabulary

Deploying Vocabularies

Service descriptions are the sets of attribute value pairs in a vocabulary that describe a service. The vocabulary determines the names and types of attributes used in the service description.

Example

A printer service is described in a printer vocabulary. The printer vocabulary contains these attributes:

- DPI (an integer value)
- Manufacturer (a string value)
- Modelname (a string value)

A client can locate this printer among other printer services because of its identity, which is a particular combination of manufacturer name, model name, and DPI value.

All service descriptions contain two types of attributes:

- **Searchable attributes** — the vocabulary attributes Clients use to query services
- **Service-specific data (Data)** — service information that a client can access, but not search for

NOTE: This data can be as simple as string entries or as complex as arbitrary byte code. For example, one datum in the printer description might be the administrator's phone number.

Vocabularies are services that are registered with the e-speak infrastructure. The `ESVocabularyElement` class allows a vocabulary to be registered. The following code sample from the vocabulary deployer registers a vocabulary called `PraxisVocab` that has three searchable attributes: `Name`, `serviceType`, and `stringLength`.

```
String vocabName = "PraxisVocab";
ESVocabularyDescription vocabDesc = new
    ESVocabularyDescription();
vocabDesc.addAttribute("Name", vocabName);
vocabDesc.addStringProperty("serviceType");
vocabDesc.addIntegerProperty("stringLength");
ESVocabularyElement vocabElement = new
    ESVocabularyElement(connection, vocabDesc);
ESVocabulary vocabulary = vocabElement.register();
```

Using Vocabularies to Describe Services

When you create a service description, decide in what vocabulary you want to use to describe the service. Once you select a vocabulary, the service description takes on the vocabulary's attributes:

```
ESServiceDescription essd = new
    ESServiceDescription(vocabulary);
String serviceType = "Palindrome";
int stringLength = serviceType.length();
essd.addAttribute("serviceType", serviceType);
essd.addAttribute("stringLength", stringLength);
```

Finding Vocabularies

Services and clients use the `ESVocabularyFinder` class to locate vocabularies. To find the vocabulary named `PraxisVocab`, the tutorial uses this code:

```
String queryString = "Name=='PraxisVocab'";

ESVocabularyFinder vocabFinder = new
    ESVocabularyFinder(connection);

ESVocabulary vocab = vocabFinder.find(new ESQuery(queryString));
```

Finding Services Using a Vocabulary

Once you have identified the vocabulary (or vocabularies) you want to use to search for a service, you can use any of the attributes in the vocabulary to perform the query. For example, if the you want to find all services for which the string length is greater than five:

```
ESServiceFinder finder = new ESServiceFinder(connection,
    interfaceName);

ServiceIntf service = finder.find(new ESQuery(vocab,
    "stringLength > 5"));
```

For details on creating and using vocabularies, please refer to the [e-speak Programmers Guide](#).

Accessing Descriptions

Use the `ESAccessor` class to view or change the attributes of a service.

Typically, you search for a service that matches some desired attribute(s). Once you find a service, however, you might be interested in viewing the values of other attributes as well. (For example, you search for a printer based on the `Name` attribute, but want to view its `DPI` attribute value once you find it). To view a more complete service description, get the accessor for the service and query the accessor for the information of interest:

```
ESAccessor accessor = ((ESAccessorHandle)service).getAccessor();

String name = accessor.getServiceName();

String type = (String)accessor.getAttribute("serviceType", vocab).getValue();

Integer length = (Integer)accessor.getAttribute("stringLength", vocab).getValue();
```

Demonstration

Compile the Code (Optional)

If desired, you can compile the code by making any necessary modifications to the files and running `Makefile` (on HP-UX or Linux%) or `compile.bat` (on Windows NT®) from the `tutorial/praxis` directory.

Start the Vocabulary Deployer

In the e-speak Desktop:

- 1 Right-click **Host**.
- 2 Select **Create a process from a template** to open the **Please select a template to create on Host** window.
- 3 Select **Vocabulary Deployer**.
- 4 Click **Create** to add Vocabulary Deployer to the Desktop.
- 5 Right-click **Vocabulary Deployer**.
- 6 Select **Process Management > Start Process**. The Vocabulary Deployer label on the Desktop turns green, indicating that it is running.

Start the Vocabulary Services

In the e-speak Desktop:

- 1 Right-click **Host**.
- 2 Select **Create a process from a template** to open the **Please select a template to create on Host** window.
- 3 Select **Vocabulary Service 1**.
- 4 Click **Create** to add Vocabulary Service 1 to the Desktop.
- 5 Right-click **Vocabulary Service 1**.
- 6 Select **Process Management > Start Process**. The Vocabulary Service 1 label on the Desktop turns green, indicating that it is running.

- 7 Repeat steps 1 through 8 to load **Vocabulary Service 2**.

Start the Basic Client

Use this procedure to verify that a basic service, in addition to vocabulary services, is running.

In the client window:

- 1 Type `basicClient`.
- 2 Press **Enter**.
- 3 Type `showall`.
- 4 Press **Enter**. Notice that only services using the e-speak base vocabulary appear in the list:

```
Basic
```

```
pi-1
```

```
pi-2
```

```
pi-3
```

- 5 Type `Exit`.
- 6 Press **Enter**.

Start the Vocabulary Client

In the client window:

- 1 Type `vocabClient`.
- 2 Press **Enter**.

Display Services Using a Custom Vocabulary

In the client window:

- 1 Type `showall`.
- 2 Press **Enter**. The system displays those services that use the custom vocabulary `PraxisVocab`:

```
pal-2
```

```
oxy-2
```

Query Services by Attribute

This procedure demonstrates how to locate a service based on a particular service attribute. In this case, you will locate a service based on its service type.

In the client window:

- 1 Type `show serviceType == 'Palindrome'`.
- 2 Press **Enter**. The system queries all services using the custom vocabulary and locates one service whose service type equals `Palindrome`:

```
pal-2
```

- 3 Type `show stringLength < 20`.

- 4 Press **Enter**. The system finds one service whose string length attribute is less than twenty:

```
oxy-2
```

E-speak supports any boolean search operator (and, or, not...), allowing you to search for services based on multiple attributes. Refer to the *e-speak Programmers Guide* for more information.

Stop the Vocabulary Client (Optional)

In the client window:

- 1 Type `exit`.
- 2 Press **Enter** to return to the command prompt.

Chapter 10 Contracts

This chapter demonstrates implementing and finding an e-speak service contract. To complete this chapter, the basic components of e-speak must be running on your computer, specifically the e-speak Desktop and Core. Before you begin, ensure that you have completed all of the processes outlined in the first three chapters of the tutorial.

Concepts

Each e-speak service implements a set of interfaces defined in a service contract. These interfaces use the e-speak IDL, which is similar to the Java-RMI IDL. You can create and search for services contracts using the base set of interfaces included with each contract and advertise contracts in vocabularies.

Clients and services alike can find and use contracts registered in the community. This chapter demonstrates:

- How to deploy a custom contract (PraxisContract)
- How a service finds a contract and uses the contract to describe itself
- How a client finds and uses a contract to access services that conform to that contract

Implementation

The Praxis contracts tutorial consists of three files.

Table 24 Praxis tutorial contract files

File	Description
tutorial/praxis/contracts/ContractCLI.java	An extension of tutorial/praxis/client/CLI.java that instantiates ContractClientAPI
tutorial/praxis/contracts/ContractClientAPI.java	An extension of tutorial/praxis/ClientClientAPI.java that finds services using contracts
tutorial/praxis/contracts/ContractDeployer.java	An e-speak service that creates and deploys a contract

Deploying Contracts

The `ESContractElement` class allows you to register a contract with the e-speak infrastructure. The following code sample from the contract deployer registers the `PraxisContract` contract:

```
String contractName = "PraxisContract";

ESContractDescription contractDesc = new
    ESContractDescription();

contractDesc.addAttribute("Name", contractName);

contractDesc.setInterfaceName(ServiceIntf.class.getName());

contractDesc.setInterfaceDefinition(
    convertStringToByteArray("Praxis Contract"));

ESContractElement contractElement =
    new ESContractElement(connection, contractDesc);

ESContract contract = contractElement.register();
```

Finding Contracts

Services and clients use the `ESContractFinder` class to find contracts. To locate the contract named `PraxisContract`, the tutorial uses the following code:

```
String query = "Name=='PraxisContract'";

ESContractFinder contractFinder = new
    ESContractFinder(connection);

ESContract contract = contractFinder.find(new ESQuery(query));
```

Finding Services Using a Contract

You can search for services based on a specific contract:

```
ESServiceFinder finder = new
    ESServiceFinder(connection,contract);

ServiceIntf service = finder.find(new ESQuery("Name=='pi-2'"));
```

Demonstration

Compile the Code (Optional)

If desired, you can compile the code by making any necessary modifications to the files and running `Makefile` (on HP-UX or Linux™) or `compile.bat` (on Windows NT®) from the `tutorial/praxis` directory.

Start the Contract Deployer

In the e-speak Desktop:

- 1 Right-click **Host**.
- 2 Select **Create a process from a template** to open the **Please select a template to create on host** window.
- 3 Select **Contract Deployer**.
- 4 Click **Create**. The application adds the Contract Deployer to the desktop.
- 5 Right-click **Contract Deployer**.

6 Select Process Management > Start Process.

The Contract Deployer label on the desktop turns green, indicating that it is running.

Start the Contracts Client

In the client window:

- 1 Type `contractClient`.
- 2 Press **Enter**.

Describe the Service Finder

In the client window:

- 1 Type `showall`.
- 2 Press **Enter**. The system lists the currently-running services:

```
Basic
pi-1
pi-2
pi-3
```

- 3 Type `describe finder`.
- 4 Press **Enter**. The system displays the finder description:

```
Using the base vocabulary and "PraxisContract" to find services
PraxisContract attributes:
Interface name: tutorial.praxis.intf.ServiceIntf
Interface definition: Praxis Contract
```

Notice that the contract used to find services has changed from "tutorial.praxis.interface.ServiceIntf" to "PraxisContract."

Stop the Contracts Deployer (Optional)

In the e-speak Desktop:

- 1 Right-click **Contract Deployer**.
- 2 Select **Process Management>Stop Process**.

Chapter 11 Multiple Cores

This chapter demonstrates using the client and e-speak Core to access a service running on a new Core. To complete this chapter, specific components of e-speak must be running on your computer: the e-speak Desktop, Core, and at least one service. Before you begin, ensure that you have completed all of the processes outlined in the first three chapters of the tutorial.

Concepts

Using the e-speak advertising service, services running on one Core are available to clients connected to other Cores. Clients can access a service without being aware of the network location from which the services is actually running.

Demonstration

Start a Second Core

In the e-speak Desktop:

- 1 Right-click **Host**.
- 2 Select **Create a process from a template** to open the **Please select a template** window.
- 3 Highlight **E-speak Core 2**.
- 4 Click **Create** to add Espeak Core 2 to the desktop.
- 5 Right-click **E-speak Core 2**.

- 6 Select **Process Management > Start Process**. When the label turns green, the process successfully starts.

TIP: You can view the process output (**Process Management > Process Output**) to verify that start-up is complete.

Start a Service on the Second Core

In the e-speak Desktop:

- 1 Right-click **Host**.
- 2 Select **Create a process from a template** to open the **Please select a template** window.
- 3 Highlight **Service of Core 2**.
- 4 Click **Create** to add Service of Core 2 to the desktop.
- 5 Right-click **Service of Core 2**.
- 6 Select **Process Management > Start Process**. The label turns green when the process successfully starts.

TIP: You can view the process output (**Process Management > Process Output**) to verify that start-up is complete.

NOTE: This service is configured to connect to **E-speak Core 2**.

Start the Client

In the client window:

- 1 Type `basicClient`.
- 2 Press **Enter**. The client connects to E-speak Core.
- 3 Type `showall`.

- 4** Press **Enter**. The application displays any services currently running on the e-speak Core and the service you just started on E-speak Core 2:

```
Basic
```

```
pi-1
```

```
pi-2
```

```
pi-4
```

- 5** Type `get pi-4`.
- 6** Press **Enter**. The application displays the string from the `pi-4` service running on E-speak Core 2. The service's location is transparent to the client.

Glossary

Advertising Service. A service used to look up resources not registered in the local repository. It returns zero or more connection objects.

Base Vocabulary. A vocabulary provided at system start-up.

Certificate. A data structure assigning a tag or name to a subject. Certificates are signed using cryptographic techniques so they cannot be modified without permission.

Certificate Issuer (CI). A service issuing certificates to subjects.

Client. Any active entity (a process, thread, service provider, etc.) that uses the e-speak infrastructure to process a request for a resource.

Contract. See [Resource Contract](#).

Core. The active entity of a logical machine that mediates access to resources registered in the local repository.

Core Event Distributor. A Core-managed resource who collects information on e-speak events and makes the information available to management tools within the infrastructures.

Core-managed Resource. A resource with an internal state managed by the Core.

Distributor Service. A service that forwards published events to subscribers.

Event. A message that results in the recipient invoking a registered callback.

Issuer. An entity issuing a certificate. The issuer is designated in a certificate by its Public Key.

Lookup Service. The component that performs look-up requests to find resources that match attribute-value pairs in the resource description of resources registered in the repository.

Message. The method by which clients and cores communicate.

Metadata. Data that is not part of the resource's implementation, but is used to describe and protect the resource.

Principal. The entity holding the Private Key corresponding to a given Public Key.

Private Key. Secure data. An entity demonstrates knowledge of secure data by cryptographic techniques to authenticate itself.

Private Security Environment (PSE). A cryptographically secure store for Private Keys.

Protection Domain. The environment associated with a particular outbox from which resources can be accessed.

Publish. A request sent to the distributor service to publish events.

Public Key. Non-secure data associated with a given Private Key by cryptographic techniques.

Public Key Infrastructure (PKI). A set of services and protocols that supports the use of public and private key pairs by applications for security.

Repository. A passive entity in the Core that stores resource metadata and the internal state of Core-managed resources.

Resource Contract. An agreement between the client and the resource handler for use of a particular resource. The agreement includes a provision for the client to use an API known to the resource handler when making the request for the resource.

Resource. The fundamental abstraction in e-speak that consists of state and metadata.

Resource Description. The data specified for the Attribute field of the metadata as represented by the client to the Core. See also [Resource Specification](#).

Resource Factory. An entity that can build the internal state of a resource requested by a client.

Resource Handler. A client responsible for responding to requests for access to one or more resources.

Resource Specific Data. A metadata field of a resource that contains information about the resource. This can be public or private to the resource handler.

Resource Specification. All metadata fields, except the Attributes field, as represented by the client to the Core.

Session Layer Security Protocol (SLS). The low-level message protocol used by all e-speak Cores and Clients for remote communication.

Simple Public Key Infrastructure (SPKI). A specific variant of PKI developed within the Internet Engineering Task Force and used by e-speak.

State. Data a resource needs to implement its abstraction.

Subject. The entity to which the access right or name has been issued, designated by its Public Key in a certificate.

Tag. The field in a certificate expressing an access right.

Vocabulary. A resource that contains the set of attributes and value types that describe a resource.

Vocabulary Builder. A Core-managed resource registered by the Lookup Service used to create new value types, attributes, and vocabularies.

Appendix E-speak Installation Directories

The following table lists the e-speak installation directories and a brief description of the files found in each.

Table 25 E-speak installation files

File	Description
bin	Contains helper utilities, such as "espeak"
config	Contains <code>ini</code> files and various configuration files for different e-speak components
contrib	Contains contributed applications and some pre-beta e-speak components
doc	Contains various documents including release notes (<code>RELNOTES.txt</code>) and javadoc APIs for development
extern	Contains various freely-redistributable components external to e-speak
lib	Contains the e-speak <code>jar</code> files
logs	Contains the e-speak log files
samples	Contains samples to aid in e-speak programming
tutorial	Contains tutorial files to help you use and create services for e-speak

