e-speak

Install and Configuration Guide

Developer Release 3.01
June 2000

HEWLETT® PACKARD

Contents

Chapter 1 Introduction

Hewlett-Packard's e-speak is a platform for developing, managing, and accessing Internet-based services (e-services). It is based on open standards. With e-speak, e-services can communicate securely at will—even across firewalls—and can combine to perform an unlimited number of functions without the user being involved.

For example, suppose you are planning a business trip. You use various websites to book a flight, rental car, and hotel room. Each time you must re-enter your itinerary. You could use an all-in-one travel website to simplify matters, but that can limit your choices. More importantly, all you find out from each website is what that company is willing to sell you. It's up to you to decide which arrangements best meet your needs.

In an e-speak environment, you tell a travel e-service provider about your needs, such as your itinerary, budget, personal preferences, and any special needs you have. Then, the e-service provider finds the right services and broker among them to get the best possible travel arrangements that meet your needs. With e-speak, your needs drive the process.

Let's take the example a step further. What if the unexpected happens, like the airline cancelling your flight? You might get where you're going but have no rental car waiting for you or no hotel room for the night. But, an e-speak-enabled travel service can adjust the other elements of your trip automatically.

How E-speak Works

E-speak is a programming structure for creating and running global distributed e-services. In e-speak, services seek out needed resources and communicate with all kinds of machines, from large data center servers to portable handheld devices, and even appliances. E-speak-enabled services can communicate through public networks, past network firewalls, and to non-networked devices through low-level serial protocols. An e-speak service can call on other services to perform a task, and it can be called on by clients in specialized environments.

After a service has been written to communicate in e-speak, it can be used as a client or a server for a new service being developed—without changing a line of code in the original service. Unlike most connectivity tools, e-speak mediates communications between client and server. This makes it possible to add new services by inserting an adapter to translate the protocols that each side speaks. Client and server can evolve independently.

E-speak provides four key e-service functions:

- **Discovery—** Finding the right services
- **Brokering—**Negotiating among competing services to find the best solution
- **Composition—**Dynamically creating higher-level services by combining lower-level component services
- **Mediation—**Continuously monitoring and managing changes

Structure and Components

E-speak consists of four main components — a service engine called the *Core*, e-speak Application Programming Interfaces (*APIs*), a collection of *standard services*, and *contributed applications*.

Figure 1 identifies the various pieces of software that form the e-speak environment and shows how they relate to each other.

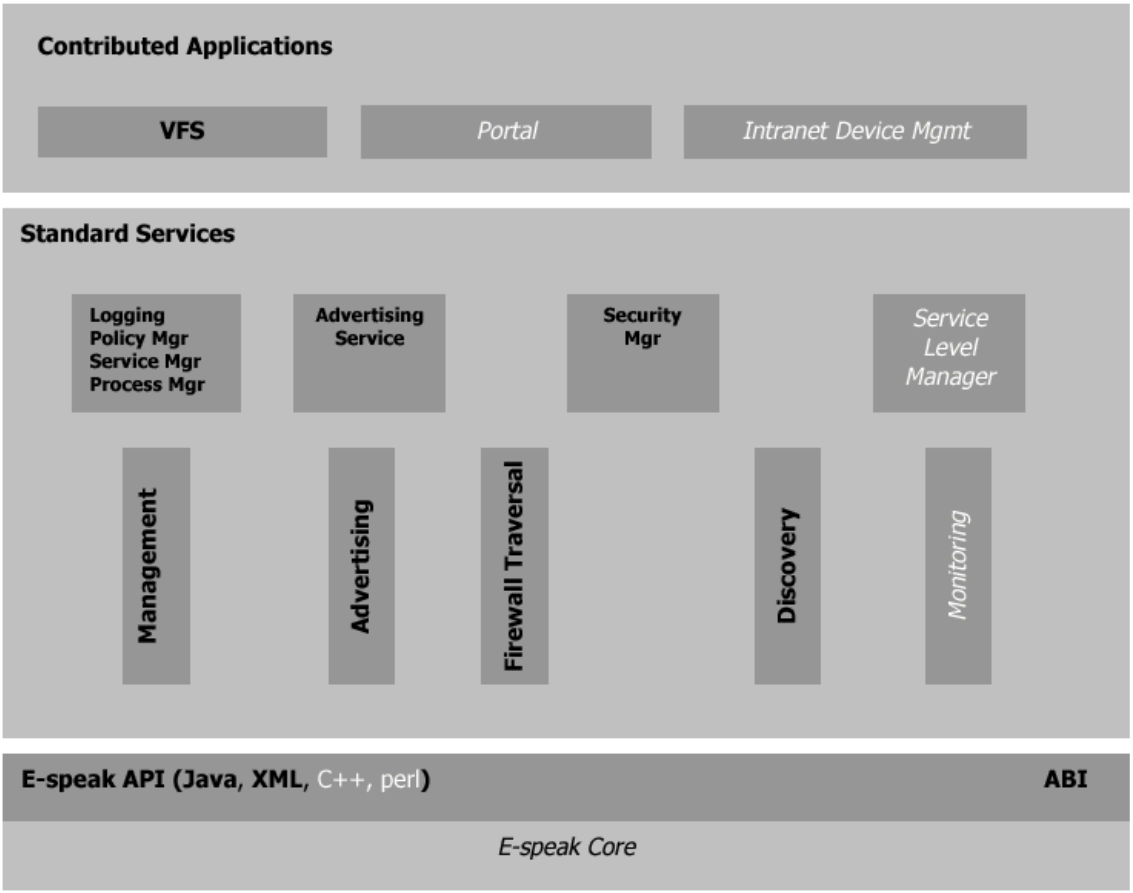


Figure 1 E-speak components: Software included in this release is in bold text; software in white italics will be supported in a future release.

Core

Core is the central piece of software in e-speak. It provides basic capabilities, such as messaging, mediation, naming, and monitoring for the e-speak environment. Cores can be interconnected to form a distributed e-speak environment.

APIs

E-speak APIs act as system call interfaces to the Core and the standard services. The API examples in this guide are written in Java, but the architecture of e-speak is language independent. So, APIs can be written in other languages. For more information, see the e-speak *Programmer's Guide*.

Standard Services

Standard services provide communication, security, discovery, and other functions that most e-services require. Programmers can build upon the standard services or replace them as needed. Standard services in this release and their main purposes are:

- Advertising Service — locates and imports services that are not present in a local core.
- Event Distribution Service — notifies different subscribers (applications interested in receiving events) of significant events.
- System Management Services — a suite of services for managing e-speak applications. The following system management services are included in this release:
  - Logging Service — provides simple API to log messages in persistent store (one per e-speak Core), so that they can be analyzed meaningfully at a later date.
  - Policy Manager — a user environment in which a user may set, get, or remove policies.
  - Service Manager — tracks managed services and any management events generated by a managed service.
  - Process Manager — provides a uniform way for remotely managing processes.

Contributed Applications

Contributed applications supplement the functionality provided by the Core and standard services. One contributed application—VFS—is included in this release. It was created to use all the features of e-speak and in doing so provide a complete test application.

VFS is a virtual file system that allows users to create a workspace of file cabinets containing the folders and files that interest them. This workspace is unique to each user, available from any computer connected to the e-speak infrastructure, and can be shared with other users. Unlike a web browser, e-speak allows you to take all of your file references with you when you change locations. For more information on VFS, see the e-speak *Contributed Services Guide*.

Current E-speak Release Components

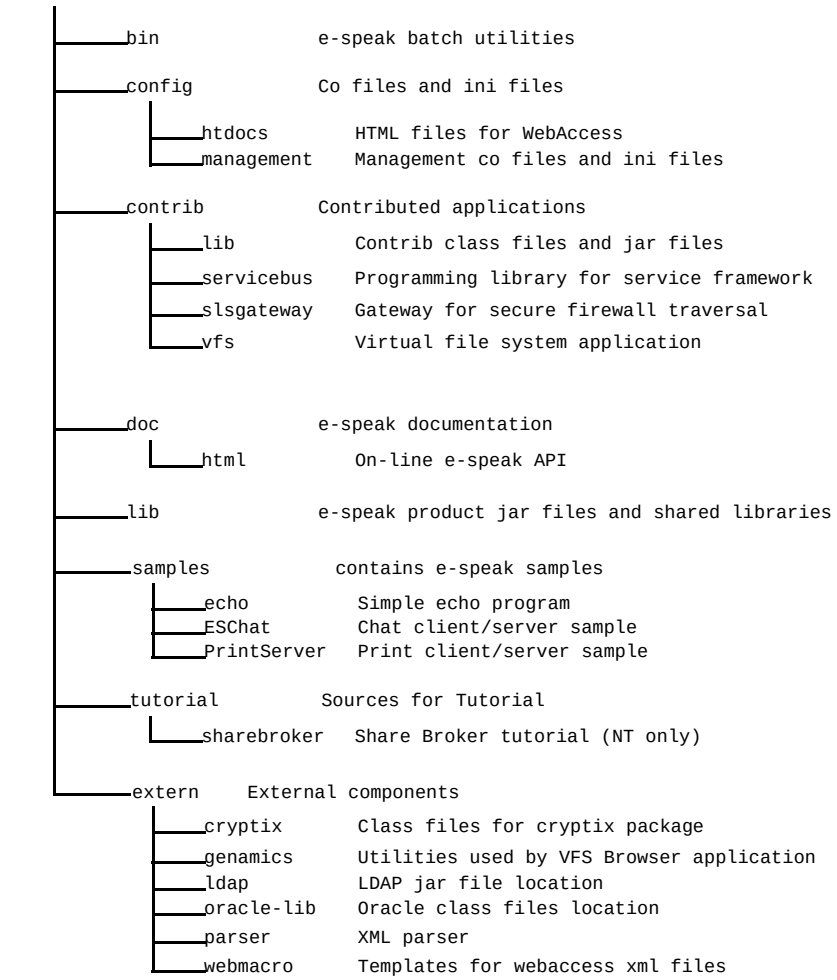
Major components included in the current release of e-speakand their directory locations are:

- Service Engine — the source files, which are written in Java, and the corresponding class files are part of the `escore.jar` file available under the `lib` subdirectory of the installation directory.
- Web Programming Model and Networked Object Model class files — these files are bundled together in the `esclient.jar` file available under the `lib` subdirectory of the installation directory. This includes APIs for accessing the e-speak Service Engine and services over the World Wide Web (WebAccess) or a local area network (J-ESI).
- Advertising Service, Management Services, and Event Distribution Service class files — these files are bundled into the `esclient.jar` file.
- External components — the following files, which are freely available and distributed “as-is,” are located under the `extern` directory of the installation directory: `ldapjdk.jar`, `cryptix.jar`, `xerces.jar`, `webmacro.jar`, and `classes111.zip` (for Oracle-JDBC interface).
- VFS source code and class files — these files are located under the `contrib` subdirectory of the installation directory.

- Sample application source code and utility scripts — these files are located in the `samples` subdirectory of the installation directory.
- Class files and sample programs for E-speak Service Framework programming interface is located in <installDir>/contrib/servicebus directory.
- Source code for the tutorial — these files are located in the `tutorial` subdirectory of the installation directory. The tutorial runs on the NT platform only.
- Security component — these files are integrated with the main system files.
- Online JAVADOC APIs — these files are located in the `html` subdirectory of the `doc` subdirectory.
- Configuration files — the following files are located in the `config` subdirectory of the installation directory: `repository.ini` (for specifying persistence parameters), `espeak.cfg` (contains trace options and security-related options), and `default.ini` (includes all services needed to be started).

Release Directory Structure

The installed e-speak directory looks like this:



About This Guide

The Installation Guide gives you the information you need to get started with e-speak. It includes installation and configuration instructions as well as information about compiling and running a simple e-speak program to verify the installation.

This guide is designed to get you started. It is not a comprehensive reference to the APIs or the e-speak architecture. Application programmers who need detailed information about e-speak APIs should read the e-speak *Programmer's Guide*. For complete information about the structure of e-speak, see the e-speak *Architecture Specification*.

This guide is organized as follows:

- Chapter 1 contains an overview of e-speak and an introduction to this guide.
- Chapter 2 contains procedures for installing and configuring e-speak on Windows NT, HP-UX, and Linux platforms.
- Chapter 3 contains configuration information for expanding e-speak's functionality, including web programming (*WebAccess*), system management, the SysLoader utility, persistence, and integrated development environments.
- Chapter 4 contains instructions for configuring and running the standard e-speak services included in this release.
- Chapter 5 contains instructions for using *Echo*, a sample program included with this release.
- Chapter 6 contains instructions for configuring the security component of e-speak. By default, the installation of e-speak is configured for testing and is not secure.

Where to Get More Information

E-speak includes a complete set of reference documentation. After you install the software, you can find the following documents in the doc subdirectory of your installation directory:

- *E-speak Install and Configuration Guide*
- *E-speak Architecture Specification*
- *E-speak Programmer's Guide*
- *E-speak Contributed Services Guide*
- *E-speak System Management User Guide*
- *E-speak WebAccess User Guide*
- *E-speak Tutorial*
- *Online API documents*

Updated documents are available online at http://www.e-speak.net.

Chapter 2 Installing and Configuring

This e-speak release is designed to run on Windows NT, HP-UX, and Red Hat Linux platforms. However, Hewlett-Packard encourages programmers to build e-speak environments for other platforms by offering source files for download from the http://www.e-speak.net website.

This chapter is divided into three subsections. You can find the information you need as follows:

NT — see “Windows NT Installation” on page 12

HP-UX —see “HP-UX Installation” on page 20

Linux — see “Linux Installation” on page 29

Windows NT Installation

System Requirements

Optimum

Operating system	Windows NT 4.0/SP4
Free disk space	35 MB at runtime (75 MB or more while installing)
RAM	256 MB or higher
Java	Java Development Kit 1.1.8 or later

Minimum

Operating system	Windows NT 4.0/SP3
Free disk space	35 MB at runtime (75 MB or more while installing)
RAM	64 MB
Java	Java Development Kit 1.1.8 (or an equivalent Java runtime and development environment)

Before You Begin

Before you install e-speak, you should:

- Make sure you are installing the latest version
- Install the external components that e-speak requires to run the desired functions or extensions.

This section assists you in those tasks.

Getting the Latest Version

The latest version of e-speak source files and their corresponding binary release versions are available freely at http://www.e-speak.net. If you download the binary release from the e-speak website, you can skip to the "Installing E-speak on Windows NT" section.

For Windows NT, the files come as a self-extracting InstallShield executable file. The file, `es_x0301.exe`, can be downloaded from http://www.e-speak.hp.com/developers.

External Module Dependencies

This table lists the various e-speak components and the external modules and products they depend upon to run properly. If you decide to use a certain e-speak component, make sure you have the corresponding external modules installed on your machine *before* you begin the e-speak installation process.

E-speak Component	External Modules Required
Advertising Service	LDAP Server (See "Advertising Service" on page 61 for more information.) If available, the Advertising Service uses an LDAP server, when suitably configured.
E-speak Core in Persistent mode	Oracle 8.0.x as a backend database (accessible using JDBC)
	Oracle Database Server with thin client JDBC driver
	(See the Release Notes for information on the version of the Database Server, JDBC driver, and Operating System supported by the current release.)
Management services	Java™ Servlet Development Kit (JSDK) 2.0, available at http://java.sun.com/products/servlet
WebAccess solution	Java Servlet Development Kit (JSDK) 2.0
	Apache web server and Apache Jserv
PrintServer sample	Swing classes (come bundled with JDK1.2)

Installing E-speak on Windows NT

NOTE: You see this symbol: `<installDir>` throughout these instructions. Wherever you see this symbol, substitute the actual directory name or pathname for that component as it exists on your machine.

To install e-speak

1. Download the `es_x0301.exe` installation file.
2. Run the `es_x0301.exe` installation file. You can run the file by double-clicking it in Windows Explorer or by typing `run es_x0301.exe` into a command prompt.
3. At the Installation window, follow the install wizard's prompts to input the necessary information.

   NOTE: The default installation location is `C:\E-speak.`

To un-install e-speak

1. In the *E-speak programs* menu, click *Uninstall E-speak.*
2. The uninstall GUI starts and the application is uninstalled automatically.
3. Check the e-speak folders and delete any remaining files manually.

Configuring e-speak for Windows NT

After installation, follow the instructions in this section to configure the e-speak environment. This is necessary to run the e-speak Core and services and to build and run the sample applications included with this release.

E-speak Environment Variables

Find and make a note of the pathnames for these variables, which you use to configure your e-speak installation:

Variable	Pathname Description
JRE	Points to the Java runtime environment. This pathname should point to the executable file named `java`. Do not point this to the `jre` file (Sun JDK installations only).
CLASSPATH	Points to your other classes (e.g. swing, jsdk).
JAVAC	Points to the Java compiler (for compilation only).
ESPEAK_HOME	Points to the e-speak `installation` directory.
PATH	Should contain <installDir>\lib.

If you are using Microsoft Java, find and make a note of the pathnames for these variables in addition to or instead the above variables:

Variable	Pathname Description
JRE	Points to Microsoft's JVM (`jview`).
JVC	Points to the Microsoft VJ++ compiler (usually `jvc.exe`).
VJ++	To use Microsoft VJ++, set this variable to `true` (all lowercase).

Setting the Environment Variables

You can set the environment variables permanently by using the Environment tab of the System Control Panel or by using a command shell.

To Use the System Control Panel

1. Open the System Control Panel and click the Environment tab.
2. Enter the pathnames for these variables where indicated. Remember to use the alternate instructions if using Microsoft JVC:
   - JRE

- JAVAC
- CLASSPATH
- ESPEAK_HOME
- PATH

3 Click Apply or OK.

To Use a Command Shell

1 In a command shell, you can use these commands to set the variables:

```
set JRE=<installDir>\java
set JAVAC=<installDir>\javac
set ESPEAK_HOME=<installDir>
set CLASSPATH=<actual path>\jsdk.jar;<actual path>\swingall.jar
```

If you use JDK 1.2, you do not need to add `swingall.jar` in the CLASSPATH.

2 If you use Microsoft Visual J++, you need to set these additional variables:

```
set JRE=C:\WINNT\jview.exe
set JVC="C:\Microsoft Visual Studio\VJ98\JVC.exe"
set VJ++=true
```

To Set the CLASSPATH Environment Variable Automatically

1 The `envmake.bat` script should be located in the directory `<installDir>\bin\.`

2 Run the `envmake.bat` script. The script sets the `CLASSPATH` environment variable, which contains all necessary e-speak `jar` files.

E-speak Runtime Variables

A lot of e-speak's runtime behavior can be configured using the `espeak.cfg` file. Aspects that can be configured through this file include web proxy configurations, core performance tuning parameters, and security.

This section explains some major parameters of web proxy and core connections. Security configurations are documented in the "Security" section of Chapter 3 and in Chapter 6.

Web-Proxy Configuration

If you need engine-engine connection through a web-proxy for firewall traversal purpose. The following information should be included in the espeak.cfg configuration file:

- `net.espeak.infra.core.connector.webproxyname=proxyhost.yourdomain.com`

  This parameter is a single value being the fully qualified hostname of the http-proxy used to pierce a firewall.

- `net.espeak.infra.core.connector.webproxyport=8088`

  This parameter is the port on which the proxy listens.

- `net.espeak.infra.core.connector.noproxydomain=yourdomain.com`

  This parameter is the domain for the direct connection that should be made. Currently, only one entry is supported. Syntax is the end of the domain that should be matched, such as `hp.com`. The `'*'` wildcard is not supported.

Core Connection Polling Configuration

The connectivity of the engine is improved significantly using the new JNI-based polling mechanism. Precompiled shared libraries are distributed for Windows NT 4.0, Redhat Linux 6.1+, and HP-UX 11.0. A pure Java reference implementation is also available for other platforms.

Configuration parameters for the Polling mechanism can be added to `espeak.cfg`. The parameters are listed below with their default values:

- `net.espeak.infra.core.thread.WorkerThreadFactory.min=10`

  This parameter is the minimum number of threads that must be available for message processing at all times. The default setting is 10.

- `net.espeak.infra.core.thread.WorkerThreadFactory.max=500`

This parameter is the maximum number of threads that will be available for message processing. The default setting is 500.

- `net.espeak.infra.core.thread.WorkerThreadFactory.timeout=2000`

  This parameter is how long a thread must be idle before it is removed from the thread pool and killed, provided there are more than the minimum number of threads in existence at the time. The default setting is 2000 milliseconds.

- `net.espeak.infra.core.comm.poll.ConnectionsQueueManager.useNativeSelect=true`

  This parameter indicates whether the JNI polling library should be used instead of the pure Java version of the polling library. The default setting is true. (Set this to false to NOT use JNI.)

- `net.espeak.infra.core.comm.poll.ConnectionsQueueManager.numDedicated=32`

  This parameter indicates the number of connections which will be serviced by dedicated threads. For these connections, polling is not performed, and a dedicated thread continuously processes messages. The default setting is 32.

- `net.espeak.infra.core.comm.poll.ConnectionsQueueManager.maxConnections=200`

  This parameter indicates the maximum number of connections that the core can handle at any given time. The default setting is 200.

Starting Basic Services

You can quickly test that e-speak is installed correctly and that the basic environment is set up with the espeak utility. This utility starts the e-speak Core and the basic services. If you need help with the espeak utility, type `espeak -h` to view the espeak utility help page.

1. Change to the `<installDir>\bin` directory.
2. Enter `.\espeak` to start the Core and basic services.

You see output similar to this:

```
*****************************************************************
* Running:  Core CoreDistributor ServiceDistributor
AdvertisingService ManagementDistributor
*****************************************************************

ES Core starting with an In-Memory Repository.
coreId = "6bf5c260ca031e38246bdf23e2f9eded"
Starting ES Core Server with Rendezvous of "TCP:12346". Started.
Connection Object: mapped localhost to 15.81.93.137
-- WARNING:  Serializing class
com.hp.es.intercorecom.confactory.co.ConnectionObject with slow,
Java-dependent Java serialization
Created the BaseDistributorVocabulary vocabulary
Advertising service not running
Started Core distributor
switching to no-backend version...
advertising service is ready
Warning: Pls check LDAP configuration if you intended to use LDAP.
Switching to non-LDAP mode now
```

3 Press Ctrl+C in the same command shell if you want to stop these services.

CAUTION: Make sure that all JVMs terminate, because if some JVMs continue running, the TCP ports remain occupied. You can do this by opening the Task Manager window and killing all JVM processes.

HP-UX Installation

NOTE: You see this symbol: `<installDir>` throughout these instructions. Wherever you see this symbol, substitute the actual directory name or pathname for that component as it exists on your machine.

System Requirements

Optimum

Operating system	HP-UX 11.00
OS Kernel Parameters	Use these guidelines for tuning the following kernel parameters: `maxfiles` should be 256 (default 60) `maxuprc` should be 1024 (default 75) `maxusers` should be 1024 (default 32) `NPTY` should be 512 (default 60) `max_thread_proc` should be 2048 (default 64) Some of the values depend on each other. So, if you cannot assign a value, skip it and retry after setting the other parameters.

Operating system patches	For HP-UX 11.00 JDK, these patches are required: PHCO_20765, PHKL_20202, PHCO_19666, PHKL_20016, PHKL_18543, PHKL_21624, PHCO_19047, PHKL_21392, PHKL_20674, and PHCO_20882. For applications that use AWT, these patches are required: PHSS_20275, PHSS_17535, PHSS_20864, PHNE_21433, PHSS_20863, and PHNE_21493. Note: Consult http://www.hp.com/go/java for an up-to-date list of necessary and recommended patches. To determine which patches are already installed on your machine, log in as root and enter this command: `/usr/sbin/swlist -l product.`
Free disk space	30 MB at runtime (60 MB or more while installing)
Java	Java Development Kit 1.2.2.04, which can be downloaded from http://www.unixsolutions.hp.com/products/java

Before You Begin

Before you install e-speak, you should:

- Make sure you are installing the latest version
- Install the components that e-speak requires to run the desired functions or extensions.

This section assists you in those tasks.

Getting the Latest Version

The latest version of e-speak source files and their corresponding binary release versions are available freely at http://www.e-speak.net. If you download the binary release from the e-speak website, you can skip to the "Installing E-speak on HP-UX" section.

External Module Dependencies

This table lists the various e-speak components and the external modules and products they depend upon to run properly. If you decide to use a certain e-speak component, make sure you have the corresponding external modules installed on your machine *before* you begin the e-speak installation process.

E-speak Component	External Modules Required
Advertising Service	LDAP Server (See "Advertising Service" on page 61 for more information.) If available, the Advertising Service uses an LDAP server, when suitably configured.
E-speak Core in Persistent mode	Oracle 8.0.x as a backend database (accessible using JDBC)
	Oracle Database Server with thin client JDBC driver
	Oracle Server
	Oracle Thin JDBC Driver
	(See the Release Notes for information on the version of the Database Server, JDBC driver, and Operating System supported by the current release.)
Management services	Java Servlet Development Kit (JSDK) 2.0, available at http://java.sun.com/products/servlet
WebAccess solution	Java Servlet Development Kit (JSDK) 2.0
	Apache web server
espeak utility	perl 5.003 or later.
PrintServer sample	Swing classes (bundled with JDK1.2)

Installing E-speak on HP-UX

The HP-UX installation is a Software Distribution (SD) depot file named `ex_x0301.Z`. You can download this file from http://www.e-speak.hp.com/developers.

For this installation, you need the swinstall(1M) program. You also need super-user privileges. For details on swinstall and other SD utilities, refer to the swinstall man pages.

To install e-speak

1. Download the `es_x0301.Z` file.
2. Log in as a super user.
3. Move the installation file to a temporary directory, such as `/tmp`, with enough disk space to hold both the compressed and uncompressed versions of the file.
4. Change to that directory using this command: `cd /tmp`
5. Uncompress the file using this command: `uncompress ES_x0301.Z`
6. Install the software using this command:

   ```
   /usr/sbin/swinstall -s $PWD/es_x0301
   ```

7. Follow the instructions to complete installation. The default file installation location is

   ```
   /opt/e-speak.
   ```

To un-install e-speak

1. Log in as a super user.
2. Use the following command to un-install the package without the GUI appearing: `swremove @ <installDir>`

   Or, run `swremove` without any arguments, which starts the uninstallation GUI.
3. The uninstallation process starts after you confirm the operation.

4 Check for any e-speak directories or files which did not get deleted and remove them manually.

Configuring e-speak for HP-UX

After installation, follow the instructions in this section to configure the e-speak environment. This is necessary to run the e-speak Core and services and to build and run the sample applications included with this release.

You need to research the pathnames of several environment variables and then set the variables for e-speak to find and access them properly. This section contains instructions for finding and setting the environment variables, along with a utility program for automatically configuring one of the variables.

E-speak Environment Variables

Find and make a note of the pathnames for these variables, which you use to configure your e-speak installation:

Variable	Pathname Description
JRE	Points to the Java runtime environment. This pathname should point to the executable file named `java`. Do not point this to the `jre` file (Sun JDK installations only).
CLASSPATH	Points to your classes (jsdk.jar for example).
JAVAC	Points to the Java compiler (for compilation only).
ESPEAK_HOME	Points to the e-speak `installation` directory.
SHLIB_PATH	Should contain <installDir>/lib.

Setting the Environment Variables

You set the above mentioned environment variables permanently in your `.profile` (or `.cshrc` if you use the C shell) file. Although there are slight differences in command styles between shells, the basic commands for the Bourne/K shell are:

```
export JRE=/<actual pathname>/java
export JAVAC=/<actual pathname>/javac
export CLASSPATH=/<actual pathname>/jsdk.jar
export ESPEAK_HOME=<installDir>
```

E-speak Runtime Variables

A lot of e-speak's runtime behavior can be configured using the `espeak.cfg` file. Aspects that can be configured through this file include web proxy configurations, core performance tuning parameters, and security.

This section explains some major parameters of web proxy and core connections. Security configurations are documented in the "Security" on page 48 and in Chapter 6, "Using Security in E-speak".

Web-Proxy Configuration

If you need engine-engine connection through a web-proxy for firewall traversal purpose. The following information should be included in the espeak.cfg configuration file:

- `net.espeak.infra.core.connector.webproxyname=proxyhost.yourdomain.com`

  This parameter is a single value being the fully qualified hostname of the http-proxy used to pierce a firewall.

- `net.espeak.infra.core.connector.webproxyport=8088`

  This parameter is the port on which the proxy listens.

- `net.espeak.infra.core.connector.noproxydomain=yourdomain.com`

  This parameter is the domain for the direct connection that should be made. Currently, only one entry is supported. Syntax is the end of the domain that should be matched, such as `hp.com`. The `'*'` wildcard is not supported.

Core Connection Polling Configuration

The connectivity of the engine is improved significantly using the new JNI-based polling mechanism. Precompiled shared libraries are distributed for Windows NT 4.0, Redhat Linux 6.1+, and HP-UX 11.0. A pure Java reference implementation is also available for other platforms.

Configuration parameters for the Polling mechanism can be added to `espeak.cfg`. The parameters are listed below with their default values:

- `net.espeak.infra.core.thread.WorkerThreadFactory.min=10`

  This parameter is the minimum number of threads that must be available for message processing at all times. The default setting is 10.

- `net.espeak.infra.core.thread.WorkerThreadFactory.max=500`

  This parameter is the maximum number of threads that will be available for message processing. The default setting is 500.

- `net.espeak.infra.core.thread.WorkerThreadFactory.timeout=2000`

This parameter is how long a thread must be idle before it is removed from the thread pool and killed, provided there are more than the minimum number of threads in existence at the time. The default setting is 2000 milliseconds.

- `net.espeak.infra.core.comm.poll.ConnectionsQueueManager.useNativeSelect=true`

  This parameter indicates whether the JNI polling library should be used instead of the pure Java version of the polling library. The default setting is true. (Set this to false to NOT use JNI.)

- `net.espeak.infra.core.comm.poll.ConnectionsQueueManager.numDedicated=32`

  This parameter indicates the number of connections which will be serviced by dedicated threads. For these connections, polling is not performed, and a dedicated thread continuously processes messages. The default setting is 32.

- `net.espeak.infra.core.comm.poll.ConnectionsQueueManager.maxConnections=200`

  This parameter indicates the maximum number of connections that the core can handle at any given time. The default setting is 200.

Starting Basic Services

You can quickly test that e-speak is installed correctly and that the basic environment is set up with the espeak utility. This utility starts the e-speak Core and the basic services. If you need help with the espeak utility, type `espeak -h` to view the espeak utility help page.

1. Change to the `<installDir>/bin` directory.
2. Enter `./espeak` to start the Core and basic services.

   You see output similar to this:

```
****************************************************************
* Running:  Core CoreDistributor ServiceDistributor
AdvertisingService ManagementDistributor
```

```
*****************************************************************

ES Core starting with an In-Memory Repository.
coreId = "6bf5c260ca031e38246bdf23e2f9eded"
Starting ES Core Server with Rendezvous of "TCP:12346". Started.
Connection Object: mapped localhost to 15.81.93.137
-- WARNING:  Serializing class
com.hp.es.intercorecom.confactory.co.ConnectionObject with slow,
Java-dependent Java serialization
Created the BaseDistributorVocabulary vocabulary
Advertising service not running
Started Core distributor
switching to no-backend version...
advertising service is ready
Warning: Pls check LDAP configuration if you intended to use LDAP.
Switching to non-LDAP mode now
```

3 Press Ctrl+C in the same command shell if you want to stop these services.

CAUTION: Make sure that all JVMs terminate, because if some JVMs continue running, the TCP ports remain occupied.

Linux Installation

NOTE: You see this symbol: `<installDir>` throughout these instructions. Wherever you see this symbol, substitute the actual directory name or pathname for that component as it exists on your machine.

System Requirements

E-speak has been tested on Red Hat Linux 6.1.

Operating system	Red Hat Linux 6.1
Free disk space	30 MB at runtime (60 MB or more while installing)
Java	Java Development Kit 1.1.8. The Linux JDK can be downloaded from http://www.blackdown.org/java-linux.html.

Before You Begin

Before you install e-speak, you should:

- Make sure you are installing the latest version
- Install components that e-speak requires to run the desired functions or extensions.

This section assists you in those tasks.

Getting the Latest Version

The latest version of e-speak source files and their corresponding binary release versions are available freely at http://www.e-speak.net. If you download the binary release from the e-speak website, you can skip to the "Installing E-speak on Linux" section.

External Module Dependencies

This table lists the various e-speak components and the external modules and products they depend upon to run properly. If you decide to use a certain e-speak component, make sure you have the corresponding external modules installed on your machine *before* you begin the e-speak installation process.

E-speak Component	External Modules Required
Advertising Service	LDAP Server (See "Advertising Service" on page 61 for more information.) If available, the Advertising Service will use an LDAP server, when suitably configured.
E-speak Core in Persistent mode	Oracle 8.0.x as a backend database (accessible using JDBC)
	Oracle Database Server with thin client JDBC driver
	Oracle Server
	Oracle Thin JDBC Driver
	(See the Release Notes for information on the version of the Database Server, JDBC driver, and Operating System supported by the current release.)
Management services	Java Servlet Development Kit (JSDK) 2.0, available at http://java.sun.com/products/servlet
WebAccess solution	Java Servlet Development Kit (JSDK) 2.0
	Apache web server
PrintServer sample	Swing classes (bundled with JDK1.2)

Installing E-speak on Linux

The Linux installation is a compressed rpm package, `es_x0301.rpm`,which can be installed using RPM on Red Hat Linux. You can download this file from http://www.e-speak.hp.com/developers.

You need super-user privileges to perform the installation.

To install e-speak as the root user

1. Download the `es_x0301.rpm` file.
2. Log in as the root user.
3. Locate the `es_x0301.rpm` file.
4. Use rpm to install the software. By default, e-speak is installed under the `/usr/local` directory; however, you can specify a different directory with this command:

```
rpm -i --relocate /usr/local=<installDir> es_x0301.rpm
```

To install e-speak as a normal user

1. Download the `es_x0301.rpm` file.
2. Log in as the root user.
3. Locate the `es_x0301.rpm` file.
4. Use rpm to install the software. By default, e-speak is installed under the `/usr/local` directory; however, you can specify a different directory with this command:

```
rpm --initdb --dbpath /tmp
rpm --relocate /usr/local=<installDir> --dbpath /tmp es_x0301.rpm
```

To un-install e-speak

1. Login as the root user.
2. Query all the installed packages using this command:

```
rpm -qa
```

3. Select the package you want to un-install.
4. Use the following rpm command to uninstall e-speak.

```
rpm -e <e-speak package name>
```

5 Check for any e-speak directories or files which did not get deleted and remove them manually.

Configuring e-speak for Linux

After installation, follow the instructions in this section to configure the e-speak environment. This is necessary to run the e-speak Core and services and to build and run the sample applications included with this release.

E-speak Environment Variables

Find and make a note of the pathnames for these variables, which you use to configure your e-speak installation:

Variable	Pathname Description
JRE	Points to the Java runtime environment. This pathname should point to the executable file named `java`. Do not point this to the `jre` file (Sun JDK installations only).
CLASSPATH	Points to your classes. (e.g. swingall.jar and jsdk.jar)
JAVAC	Points to the Java compiler (for compilation only).
ESPEAK_HOME	Points to the e-speak `installation` directory.
LD_LIBRARY_PATH	Should contain <installDir>/lib.

Setting the Environment Variables

You set the above mentioned environment variables permanently in your `.profile` (or `.cshrc` if you use the C shell) file. Although there are slight differences in command styles between shells, the basic commands for the Bourne/K shell are:

```
export JRE=<actual path>/java
export JAVAC=<actual path>/javac
export CLASSPATH=<actual path>/swingall.jar:<actual path>/jsdk.jar
export ESPEAK_HOME=<installDir>
```

E-speak Runtime Variables

A lot of e-speak's runtime behavior can be configured using the `espeak.cfg` file. Aspects that can be configured through this file include web proxy configurations, core performance tuning parameters, and security.

This section explains some major parameters of web proxy and core connections. Security configurations are documented in "Security" on page 48 and in Chapter 6, "Using Security in E-speak".

Web-Proxy Configuration

If you need engine-engine connection through a web-proxy for firewall traversal purpose. The following information should be included in the espeak.cfg configuration file:

- `net.espeak.infra.core.connector.webproxyname=proxyhost.yourdomain.com`

  This parameter is a single value being the fully qualified hostname of the http-proxy used to pierce a firewall.

- `net.espeak.infra.core.connector.webproxyport=8088`

  This parameter is the port on which the proxy listens.

- `net.espeak.infra.core.connector.noproxydomain=yourdomain.com`

  This parameter is the domain for the direct connection that should be made. Currently, only one entry is supported. Syntax is the end of the domain that should be matched, such as `hp.com`. The `'*'` wildcard is not supported.

Core Connection Polling Configuration

The connectivity of the engine is improved significantly using the new JNI-based polling mechanism. Precompiled shared libraries are distributed for Windows NT 4.0, Redhat Linux 6.1+, and HP-UX 11.0. A pure Java reference implementation is also available for other platforms.

Configuration parameters for the Polling mechanism can be added to `espeak.cfg`. The parameters are listed below with their default values:

- `net.espeak.infra.core.thread.WorkerThreadFactory.min=10`

This parameter is the minimum number of threads that must be available for message processing at all times. The default setting is 10.

- `net.espeak.infra.core.thread.WorkerThreadFactory.max=500`

  This parameter is the maximum number of threads that will be available for message processing. The default setting is 500.

- `net.espeak.infra.core.thread.WorkerThreadFactory.timeout=2000`

  This parameter is how long a thread must be idle before it is removed from the thread pool and killed, provided there are more than the minimum number of threads in existence at the time. The default setting is 2000 milliseconds.

- `net.espeak.infra.core.comm.poll.ConnectionsQueueManager.useNativeSelect=true`

  This parameter indicates whether the JNI polling library should be used instead of the pure Java version of the polling library. The default setting is true. (Set this to false to NOT use JNI.)

- `net.espeak.infra.core.comm.poll.ConnectionsQueueManager.numDedicated=32`

  This parameter indicates the number of connections which will be serviced by dedicated threads. For these connections, polling is not performed, and a dedicated thread continuously processes messages. The default setting is 32.

- `net.espeak.infra.core.comm.poll.ConnectionsQueueManager.maxConnections=200`

  This parameter indicates the maximum number of connections that the core can handle at any given time. The default setting is 200.

Starting Basic Services

You can quickly test that e-speak is installed correctly and that the basic environment is set up with the espeak utility. This utility starts the e-speak Core and the basic services. If you need help with the espeak utility, type `espeak -h` to view the espeak utility help page.

1. Change to the `<installDir>/bin` directory.
2. Enter `./espeak` to start the Core and basic services.

   You see output similar to this:

```
*****************************************************************
* Running:  Core CoreDistributor ServiceDistributor
AdvertisingService ManagementDistributor
*****************************************************************

ES Core starting with an In-Memory Repository.
coreId = "6bf5c260ca031e38246bdf23e2f9eded"
Starting ES Core Server with Rendezvous of "TCP:12346". Started.
Connection Object: mapped localhost to 15.81.93.137
-- WARNING:  Serializing class
com.hp.es.intercorecom.confactory.co.ConnectionObject with slow,
Java-dependent Java serialization
Created the BaseDistributorVocabulary vocabulary
Advertising service not running
Started Core distributor
switching to no-backend version...
advertising service is ready
Warning: Pls check LDAP configuration if you intended to use LDAP.
Switching to non-LDAP mode now
```

3. Press Ctrl+C in the same command shell to stop these services.

CAUTION: Make sure that all JVMs terminate, because if some JVMs continue running, the TCP ports remain occupied.

Chapter 3 Expanding E-speak Functionality

E-speak has many advanced features that must be configured separately from the main installation of the Core and Standard Services. These features include:

- WebAccess — activates e-speak over the Internet
- Security — establishes a secure communication environment
- System Deployment Console — monitors and controls system activity
- SysLoader — controls the services and classes to be loaded
- Integrated Development Environments — allows the establishment of complex e-speak environments

This chapter helps you learn about how these features work and how to install and configure the necessary components to use these features in an e-speak development environment.

WebAccess

E-speak WebAccess allows users to interact with the e-speak Core and services through a standard web-based document exchange model. WebAccess internally uses XML and also provides an interface for XML-enabled applications. It currently provides XML mappings for the whole set of e-speak's Core functionality and a subset of the Core functionality for browser access. Interaction between the browser, XML clients, and WebAccess is based on HTTP protocol.

Restrictions on Implementation

Some restrictions apply for the current implementation. WebAccess services are currently required to be installed as servlets. It is tested with Apache1.3.12 / Jserv1.1 only. Only a single e-speak Core is currently supported.

This section explains the installation and setup of Apache/Jserv.

WebAccess Features

The e-speak WebAccess interface supports all the critical features of e-speak using a document-based interface:

- Service Registration
- Vocabulary Definition
- Service Invocation (synchronous interactions only)
- Client Login Session handling
- Service URL lookup (HTTP post)

Installation

The e-speak WebAccess interface runs as a servlet within a Java enabled web server. The following packages are needed to run WebAccess:

- E-speak *Core* (comes with the release)
- *Apache* Web server (to be installed separately, available from http://www.apache.org/dist/binaries/win32) version 1.3.12
- *Apache/JServ* servlet extension for Apache (available also from http://java.apache.org/jserv/dist) version 1.1
- *JSDK20*, Java Servlet Developer's Kit (available from http://www.sun.com)
- *Xerces* XML parser (comes with the release)
- Webmacro template package (comes with the release)

- JavaCC package (comes with the release)

  NOTE: In the following sections, <installDir> is the e-speak installation directory, <APACHE> is the Apache installation directory, <JSERV> is the Apache JServ installation directory, <JDK> is the installation directory of your JDK, <JSDK> is the Java Servlet Development Kit installation directory, and <JAVACC> is the directory for Java Compiler Compiler software.

Installing Apache Web Server on NT

1. Download `apache_1_3_12_win32.exe` from www.apache.org and follow the installation procedure to install Apache.
2. Edit the `<Apache>/conf/httpd.conf` file to change the `DocumentRoot` and `Directory` values as follows:

```
DocumentRoot "<installDir>/config/htdocs"
   ...
<Directory "<installDir>/config/htdocs">
   ...
   Order allow,deny
   Allow from all
</Directory>
```

You may also need to specify the `ServerName` if you are running on NT or are using a server cluster. For more information, see the Apache server documentation.

NOTE: Examples of configuration files for Apache and JServ are in `<installDir>/config/htdocs`.

Installing Apache JServ on NT

1. Download ApacheJServ-1.1.exe from http://java.apache.org and follow the installation procedure to set up JServ.
2. During setup, JServ prompts for the locations of the following components:
   - your JDK
   - your Apache and its `httpd.conf` file

- your JSDK

3 Select `yes` when JServ installation prompts for changing Apache's `httpd.conf` file.

4 Check the Apache `httpd.conf` file to ensure that the line for Apache JServ configuration is present. If not, enter it manually or correct it, if necessary:

```
Include "<JSERV>/conf/jserv.conf"
```

5 After installation, the following JServ configuration files need to be adapted in order to work with WebAccess:

- `<JSERV>/conf` with `jserv.properties`
- `<JSERV>/servlets` with `zone.properties`
- `<JSERV>/conf` with `jserv.conf if you want to enable logging`

6 Use the procedures below to make the required changes to the two configuration files.

Changes in `jserv.properties`

1 Locate the following sections in the `jserv.properties` file and ensure that the paths are valid:

```
# The Java Virtual Machine interpreter.
wrapper.bin=<jdk>/118/bin/java.exe
# CLASSPATH environment value passed to the JVM
wrapper.classpath=<JSERV>/ApacheJServ.jar
wrapper.classpath=<JSDK>/lib/jsdk.jar
```

2 Add following lines at appropriate places, for path and classpath:

```
# PATH environment value passed to the JVM {ONLY FOR NT}
wrapper.path=<installDir>/lib
# Additional CLASSPATH to be added
wrapper.classpath=<installDir>/lib
wrapper.classpath=<installDir>/lib/esclient.jar
```

```
wrapper.classpath=<installDir>/lib/escore.jar
wrapper.classpath=<installDir>/extern/webmacro/webmacro.jar
wrapper.classpath=<installDir>/extern/webmacro
wrapper.classpath=<installDir>/extern/parser/xerces.jar
wrapper.classpath=<installDir>/extern/cryptix/cryptix.jar
wrapper.classpath=<installDir>/extern/cryptix
wrapper.classpath=<installDir>/extern/ldap/ldapjdk.jar
wrapper.classpath=<installDir>/extern/oracle-lib/
classes111.zip
wrapper.classpath=<JAVACC>/javacc.zip
```

Changes in `zone.properties`

1. Locate the `zone.properties` file under your <JSERV>/servlets directory.
2. Add the following lines at appropriate places in the file:

```
# Startup Servlets
servlets.startup=net.espeak.webaccess.WebAccess

# Servlet Aliases
servlet.WebAccess.code=net.espeak.webaccess.WebAccess
servlet.IsItWorking.code=net.espeak.webaccess.htdocs.servlets.IsIt
Working
servlet.BookBroker.code=net.espeak.webaccess.htdocs.servlets.BookB
roker
servlet.ProxyFatBrain.code=net.espeak.webaccess.htdocs.servlets.Pr
oxyFatBrain
servlet.ProxyBarnes.code=net.espeak.webaccess.htdocs.servlets.Prox
yBarnes
servlet.ProxyAmazon.code=net.espeak.webaccess.htdocs.servlets.Prox
yAmazon
servlet.Forms.code=net.espeak.webaccess.htdocs.servlets.Forms
servlet.Login.code=net.espeak.webaccess.htdocs.servlets.Login
```

When Apache is started, the servlets are accessible at http://<your-machine>/servlets/WebAccess and the BookBroker example is accessible as http://<your-machine>/index.html

Changes in `Webmacro.properties`

1. Locate the `<installDir>/extern/webmacro/Webmacro.properties` file.
2. Change the `TemplatePath` to point to `<installDir>/config/htdocs/templates`. The template files are used for internally generating XML messages and for templating HTML pages.

Changes in `WebAccess.xml`

1. Locate the `<installDir>/extern/webmacro/webaccess.xml` configuration file. This file contains all of the settings used by service developers and deployers to set the password, expiration time, hostname, web proxy name, and database connection information. It can also be used to enable or disable the persistent message queue and debugging.
2. Find the `<TokenManager>` element. Inside this is a `<TokenPassword>` element, which is a key used to encrypt the session tokens given to clients.
3. Change the `<TokenPassword>` as needed. Each site should have a unique `<TokenPassword>` to ensure security.
4. Find the `<ExpireIntervalSecs>` element and change if necessary. This element sets the expiration interval for each token, and the default setting is 14400 seconds.
5. Update the information for JDBCConnection in the configuration file (two places):
   - Name of the host running the Oracle database
   - Login name and password
6. Find the `<webServerInfo>` element and change the `hostname` element to your actual hostname.

7 Find the <webProxyInfo> element and change the hostname element to your actual proxy.

Unix and Linux Platforms

For HP-UX platforms, the shell environment needs to be set up as below:

SHLID_PATH=<installDir>/lib

For Linux platforms, the shell environment nees to be set up as below:

LD_LIBRARY_PATH=<installDir>/lib

Debug Logging

Since Apache servlets do not report messages to the screen, debug messages generated from servlets must be obtained from log files; for example:

```
tail -f   <JSERV>/logs/jserv.log
tail -f   <APACHE>/logs/access.log
```

Setting the debug option for the servlet allows you to trace requests as they go through the WebAccess servlet. This setting is made in the zone.properties file, and the logging level should be lowered to 'info' in the jserv.properties file.

To set the debug option

1 Add the following line for the Servlet Init Parameters to the zone.properties file:

```
servlet.net.espeak.webaccess.WebAccess.initArgs=debug=1
```

2 Make the following changes to the jserv.properties file:

```
log.channel=true
log.channel.info=true
log.channel.debug=true
```

3 Make the following changes to the jserv.conf file:

```
ApJServLogLevel info
```

Testing Web Access Configuration

Now that you've installed all the components and done the configuration work, you can test the WebAccess infrastructure.

Testing WebAccess Configuration

The sequence of commands in this section illustrate how to test on NT. For Linux and HP-UX platforms, use '/' in place of '\'. Make sure no JVMs are running before starting this procedure.

1. Start the Core with the following commands:

```
CD to <installDir>\bin
.\espeak
```

2. Start the Apache server from the `<installDir>\config` directory. For example, here is one set of commands for starting the Apache server:

```
CD to <installDir>\config
<APACHE>\Apache -d "<APACHE>" -s
```

   NOTE: On HP-UX and Linux platforms, the Apache executable is called 'httpd'. You need double quotes only if your <APACHE> has blanks in it. On NT, if you use a shortcut icon, modify the `Start In` box of the shortcut properties to `<installDir>\config`.

   Another way to start Apache server is to copy the following files from `<installDir>/config` directory to <APACHE> directory: `espeak.cfg`, `securestore.txt`, `clientcerts.adr` and `servicecerts.adr`. You can then start Apache the normal way.

3. Launch a web browser, such as Netscape or Internet Explorer.

4. Point the browser at this URL: `http://localhost`
   Depending on your configuration, you may need to use your actual hostname instead of localhost if the Login screen does not come up.

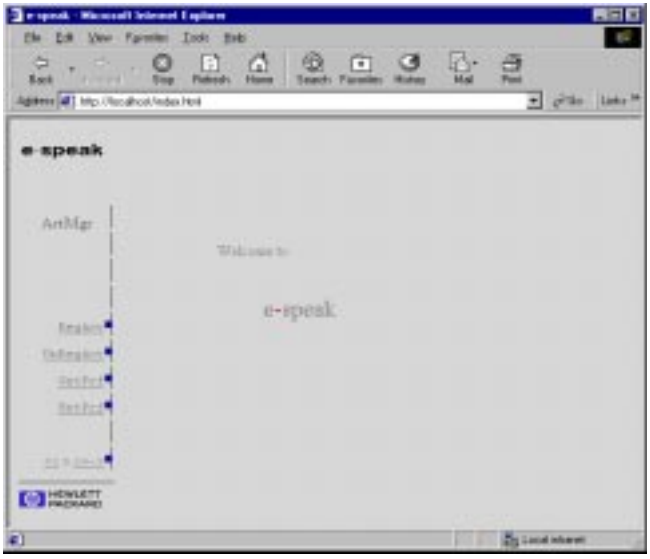
5 The browser window comes up with the Login Form, which looks like this:

6 If you do not see this screen, make sure that:

- Your servlet installation is working properly (even without e-speak)
- Your Apache server is running properly
- Your Apache server is focused on the WebAccess `htdocs` directory.

7 Click the Login link on the Login Form. Another form appears, which requires username and password. Enter `esadmin` for both the username and password,

then click the Login button. If the login is successful, you see a screen that looks like this:



8 From this screen, you can register and locate vocabularies and services, and you can register users. At this point, you have a fully functional e-speak WebAccess system ready for development of web-accessible services.

For more information, including the Book Broker example, see the *E-speak WebAccess Programmers Guide*. For using the BookBroker example, you need to ensure that the Classpath environment variable includes the following for running the BookBroker class that registers the vocabularies and services:

```
CLASSPATH=.;%ESPEAK_HOME%\lib\esclient.jar;%ESPEAK_HOME%\lib\escor
e.jar;%ESPEAK_HOME%\extern\parser\xerces.jar;%ESPEAK_HOME%\extern\
webmacro\webmacro.jar;%ESPEAK_HOME%\extern\ldap\ldapjdk.jar;%ESPEA
K_HOME%\extern\oracle-lib\classes111.zip;<actual path>\jsdk.jar
```

NOTE: On Linux and HP-UX platforms, the classpath looks as below:

```
CLASSPATH=.:$ESPEAK_HOME/lib/ esclient.jar:$ESPEAK_HOME/lib/
escore.jar:$ESPEAK_HOME/extern/parser/xerces.jar:$ESPEAK_HOME/
extern/webmacro/webmacro.jar:$ESPEAK_HOME/extern/ldap/
ldapjdk.jar:$ESPEAK_HOME/extern/oracle-lib/classes111.zip:<actual
path>/jsdk.jar
```

Troubleshooting WebAccess

- Ensure that the configuration files do not have any typos or case differences.
- Before you start the Core, ensure that no JVMs are running. If so, kill all JVMs.
- Look at the logs under <APACHE> and <JSERV>.

  ```
  <APACHE>/logs/access.log
  <APACHE>/logs/error.log
  <JSERV>/logs/jserv.log
  <JSERV>/logs/mod_jserv.log
  ```
- Most problems in using WebAccess are related to faulty configurations in `webaccess.xml`. Make sure you have read the installation and configuration notes on this file.
- Make sure your `webaccess.xml` file is in a directory on your `CLASSPATH`.
- Make sure site-specific plugins implement the right interfaces and have default public constructors.
- Make sure the web server is set up to find the `webaccess` directory.
- Make sure the `webmacro.properties` file has been changed to refer to the location of the `webaccess\htdocs` directory.
- Make sure your proxy server is configured correctly so that the `java.net.URL` class can connect to your service.
- Make sure to configure the database connection information for the persistent message manager in `webaccess.xml`.
- Make sure the web server is started in the `<installDir>\config` directory. This directory contains an `espeak.cfg` file, which must be read by WebAccess to set up security.

- If you are on HP-UX or Linux platforms, make sure that <installDir>/lib is included in your shared library path (SHLIB_PATH for HP-UX and LD_LIBRARY_PATH for Linux).

Security

E-speak security centers around access control to certain services and resources based on a certificate system. When a client machine requests a service, the authorization engine within e-speak's security service looks for a valid certificate on the client-side for authorization. Clients with valid certificates are "trusted."

E-speak also uses cryptographic technology to prevent unauthorized access of messages along the client-server connection. Messages are encrypted and authenticated to prevent interception and/or alterations. This means that a copy of a message cannot be captured and used again at a later date because the attempt is ignored.

E-speak security uses the following components:

- Private Secure Environment
- Certificates

This section describes the security components, how to configure them, and gives a sample security configuration file. For more detailed information, see Chapter 6, "Using Security in E-speak".

Private Secure Environments

E-speak establishes a Private Secure Environment (PSE) containing passphrase protected public-private key pairs. Both clients and servers have public-private key pairs. They authenticate by using a cryptographic protocol to prove they are in possession of the private key corresponding to their public key. No entity must ever share its private key.

The PSE is loaded into the client or server along with any certificates that have been issued to the client or server. The PSE and certificates are contained in separate files.

Certificates

Access rights are controlled by certificates, which are authenticated by the Security Engine. Both client and server may have certificates.

Configuring E-speak Security Services

The default configuration file is `espeak.cfg`. This is looked for in the <installDir>/config directory, current directory, and the user's home directory

The properties supported by the security code are as follows:

1. Master flag controlling security:

   `net.espeak.security.activate, boolean`

   Default value is `off`. This property is set to `true` in espeak.cfg file and thus security is activated.

2. Property controlling whether secure sessions are established with newly encountered resources:

   `net.espeak.security.connectOnContact`

   Default value is `off`. When it is off, sessions are not established unless required (by `SessionRequiredException`) or created explicitly.

3. PSE mode:

   `net.espeak.security.pse.mode`

   Possible values are `gui`, `passphrase`, and `passfile`. Default value is `gui`.

   If the mode is `gui`, a dialog is used to get the PSE passphrase.

   If the mode is `passphrase`, the property `net.espeak.security.pse.passphrase` is used to get the passphrase (default `null`).

If the mode is `passfile`, the property `net.espeak.security.pse.passfile` (default `passfile.txt`) is used to get the name of a file which must contain a `net.espeak.security.pse.passphrase` property defining the passphrase.

4 PSE key file:

`net.espeak.security.pse.storefile`

Default value is `securestore.txt`. This defines the name of the file containing public-private key pairs.

5 PSE role:

`net.espeak.security.pse.role`

Default value is `client`. This defines the default role (symbolic PSE key name).

6 Certificate file suffix:

`net.espeak.security.pse.certfile`

Default value is `certs.adr`. The value of this property is appended to the role name to get the name of the certificate file to load. If the role is `client`, the certificate file is `clientcerts.adr`, for example.

7 ACL file suffix:

`net.espeak.security.pse.aclfile`

Default value is `acl.adr`. The value of this property is appended to the role name to get the name of the ACL file (trust assumptions) to load. If the role is `client`, the ACL file is `clientacl.adr`, for example.

8 Cipher suites:

`net.espeak.security.cipherSuites`

The value is a list of cipher suites in ADR syntax. The default is to use `hmac`, `sha-1`, and `128-bit blowfish`.

Sample Security Configuration File

```
!============================================================
! E-speak security properties file.
!============================================================
net.espeak.security.activate=on
user.name="John Doe"

! Example time value.
foo.timeout = 12h 3m .0001s
! Set a property prefix.
@prefix=net.espeak.security
! Gui mode runs a dialog for the passphrase.
!.pse.mode=gui
! Passphrase mode looks for the passphrase property.
.pse.mode  = passphrase
! Passfile mode looks for a file containing the passphrase
property.
!.pse.mode = passfile
! Define the passphrase.
.pse.passphrase = default passphrase
! Define the default role (PSE key name).
!.pse.role = foo
```

More Information

For more a more detailed description of security in e-speak, see Chapter 6.

System Deployment Console

The e-speak System Deployment Console is a remote management tool implemented in e-speak. It has been designed for the distributed management of both e-speak and native (non-e-speak) processes as well as services. It works with the Management Services described in Chapter 4.

The purpose of the System Deployment Console is to:

- Help service operators to configure and deploy services remotely.
- Provide remote management at task (process) level.
- Help people to debug and test their services remotely.
- Allow easy and intuitive access to web-based management.

The architecture of the Console is based on the Common Information Model (CIM) for system management and features many extensions specific to e-speak. The Console uses a locally stored database, essentially a table of managed elements representing, and interacting with real processes and services.

The Console has a built-in model for e-speak systems. It includes task-level management models for e-speak enabled hosts, cores, clients and services. Such a model forms the basis of the Console's inner-workings. This model can be extended and even replaced to allow custom system management. Normally, the users of the Console are not aware of the existence of the model.

Starting the Deployment Console

The Console itself is an e-speak client. Like all e-speak clients, it must be connected to a local e-speak Core to start.

1. To start the Deployment Console, use this command (after setting CLASSPATH to e-speak required .jar files):

   ```
   <JVM> net.espeak.services.management.configman.ui.Console
     <12346>
   ```

2. The Console is not ready until the local Core connection has been established or timed out.

Working with the Deployment Console

For instructions on how to use the Deployment Console, see the *System Deployment Console User Manual*, available on the e-speak website.

Configuration for Persistence

The current release only supports Oracle Database Server with thin client JDBC driver. The Release Notes provide information on the version of the Database Server, JDBC driver, and Operating System supported by the current release.

Before starting e-speak in a persistence mode

1. Install Oracle Server on a server. Follow the typical installation as documented in the Oracle installation documentation.
2. Install Oracle's Thin JDBC driver on each of the clients that need to run the e-speak Core.
3. Ensure that you are able to make a connection to the Oracle Server from the JDBC driver on the Client.
4. Make sure that Oracle Listener and Server are running on the Database Server.
5. Modify the `repository.ini` file (available in the `<installDir>\config` directory) with the following changes:
   - In the section `[Repository Params]`, comment out `Store Type=INMEMORY` and uncomment `Store Type=JDBC`
   - In the section `[JDBCGlue]`, make the following modification to the `connectionString`: 1.) Make `HOST` equal to the IP of the server where Oracle Database Server is running. 2.) The default `PORT` value should be 1521. Change it as needed if the Oracle server is listening on a different port.
6. Create User(s) in the Database.

Metadata in E-speak

E-speak creates all the metadata it requires. All data and some metadata within e-speak is partitioned based on the Core. Resources registered by a Core can be discovered or unregistered only by that Core. Other Cores do not have any access to these resources.

Removing and Replacing Repository Data and Metadata

A RepositoryReset utility is available to entirely remove all e-speak repository data and the metadata from the persistent repository. You can use this utility to remove an old version of the repository from previous releases of e-speak which are not compatible with the current release. You can then create a new repository by doing a cold start with the '-r' option as described below.

To run the RepositoryReset utility

1 Enter the following command:

```
<jvm> net.espeak.infra.core.repository.RepositoryReset -u ben1
-pswd ben1
```

2 The following text appears:

```
Preparing to reset Repository!
All E-speak data associated with user ben1 will be removed.
Continue?  (Y/N)
```

3 Press `y` to continue. The following text appears, showing the procedure is complete:

```
ES Core starting with a JDBC-based Repository for user ben1.
JDBC-based Repository is doing RESET of Database!
Repository successfully removed.
```

NOTE: You can specify the `-f` option to prevent the interactive confirmation.

Restarting E-speak

You can restart e-speak two ways: cold start with the `-r` option or warm start.

Cold Start `-r` Option

When starting the Core for the first time, use the `-r` option. You can also use this option to clean up all resources for a Core. The `-r` option does the following tasks:

- Deletes all data pertaining to the Core
- Creates the metadata needed for e-speak

- Initializes to 0 the Unique ID for Resources
- Loads onto the database all the initial core managed resources and boot resources

Warm Start

Make sure to perform a Cold Start at least once before doing a Warm Start.

Specifying repository parameters

The `repository.ini` file controls various cache and repository parameters. You can specify your own `repository.ini` file using the `-repository` option to the Core. By default, a `repository.ini` file is searched for in the current directory. There are three caches in the e-speak implementation—the resource description cache, the resource specification cache, and the resource state cache.

A `repository.ini` file is available in `<installDir>\config` directory. Here is a sample `repository.ini` file:

```
[Cache Params]
Description Cache Size=262144
Specification Cache Size=262144
State Cache Size=262144
```

The minimum size for any of these caches is 1024. If you specify a smaller cache size, 1024 is used instead. If you specify a larger size, your setting is used. You can select a particular kind of repository by commenting out the other kinds of repository adapters. In the following example, the e-speak core starts with a JDBC-based persistent repository:

```
[Repository Kind]
;adapterType=INMEMORY
adapterType=JDBC
```

A JDBC-based reference implementation on top of an Oracle database is provided with this release. Database vendor-specific information is captured in the `JDBC-Glue` clause in the `repository.ini` file. For example, the name of the driver is different if an Informix database is deployed.

The JDBC driver uses the connection string to talk to the back end database. It has several components. For example, it identifies that a thin Oracle driver is used. The advantage of a thin driver is that the actual database server can be on a Unix or an NT machine and the Core can interoperate with no problems.

The HOST and PORT strings identify the actual machine name that runs the database. Database vendor-specific error codes are encoded for portability. In the following example, the error code 955 is valid for Oracle to capture the pre-existence of a table. Similarly, noTbl denotes the error code for a missing table and dupRec shows the error code for an attempt to add a duplicate entry to a table.

```
[JDBCGlue]
driverName=oracle.jdbc.driver.OracleDriver
connectionString=jdbc:oracle:thin:@(DESCRIPTION =(ADDRESS =
(PROTOCOL = TCP)(HOST = <hostname>)(PORT = 1521))(CONNECT_DATA =
(SID =ORCL)))
tblExists=955
noTbl=942
dupRec=1
dbFull=1650, 1651, 1652, 1653, 1654, 1655, 1656, 1657, 1658, 1659
```

Configuring Integrated Development Environments

Integrated development environments (IDEs) allow you to run, compile, and debug applications in a user-friendly environment. This section explains how to start the e-speak Core and other components within an IDE.

This section focuses on Microsoft's Visual J++ environment. Syntax and procedures may vary for other IDEs. Refer to their help documents as needed.

To configure Microsoft Visual J++

1. Set the classpath to contain e-speak-related classes.

   a. Click <Project> Properties on the Project menu, where <Project> is the name of your Microsoft Project.

b In the Classpath tab, add the necessary e-speak–specific jar files:

`<installDir>\lib\escore.jar`	E-speak Core
`<installDir>\lib\esclient.jar`	E-speak Standard Services and APIs
`<installDir>\contrib\lib`	Needed if Contributed tools like Core Browser, performance monitor, etc. are used
`<installDir>\extern\ldap\ldapjdk.jar`	Needed if Advertising Service uses external LDAP server
`<installDir>\extern\parser\xerces.jar`	Needed when using e-speak's XML API
`<installDir>\oraclelib\classes111.zip`	Needed if e-speak's Core is started in persistence mode, using Oracle Database as a back-end

2 Start the e-speak Core and services and load the `default.ini` file using the `SysLoader` utility.

NOTE: SysLoader is available in the `escore.jar` file, and `default.ini` is available in the `<installDir>/config` directory.

a Click <Project> Properties on the Project menu, where <Project> is the name of your Microsoft project.

b In the Launch tab, click the Custom radio button and enter the following values for the Program and Arguments settings:

Program JView.exe

Arguments
```
/p /cp:p "<JAVAPACKAGES>"
/d:espeak_home=<installDir>
/d:espeak_port=12346
/d:espeak_group="<your_group_name>"
net.espeak.util.SysLoader
"<installDir>\config\default.ini"
```

3 Start the Core and other default services mentioned in default.ini by choosing Start Without Debugging from the Debug menu.

The Core and other services are started in a separate console window.

4 Press Ctrl+C in the command shell to stop the Core and services.

Chapter 4 Running E-speak Standard Services

E-speak includes three standard services, which are supported tools that will be maintained in future releases. These services are simple to use and serve as an aid to learning more about e-speak. The standard services are bundled into the `<installDir>\lib\esclient.jar` file.

This chapter describes the following standard services, giving instructions on how to run them:

- **Event Distribution Service—**Used to notify different subscribers (applications interested in receiving events) of significant events.
- **Advertising Service—**Used to locate and import services that are not present in a local logical machine.
- **Management Services—**A suite of services used to manage an **e-speak** application. The **E-speak** *System Management User Guide* provides detailed information on the four management services and how to use the Programming Interface to make an **e-speak** component manageable.

NOTE: The examples in this section use the NT notation for path names. On Unix platforms, use forward slashes (/) instead of back slashes (\).

Event Distribution Service

Client applications that need to subscribe to and receive notifications of events generated by event publishers use the e-speak event distribution service. This service uses a publish-subscribe model. There are potentially four entities in this model:

- **Distributor—**The entity that receives notifications from event publishers and manages subscriptions to events. A standard distributor is included in this e-speak release.
- **Publisher—**The originator of events. Publishers come in one of two classes—external services and the Core.
- **Subscriber—**Subscribes to various event types handled by the distributor. Subscribers specify filters to allow distributors to reduce the number of events forwarded to clients.
- **Listener—**In most cases, a subscriber to an event is also the listener. However, in some situations a Subscriber could register interest in an event on behalf of a different Client that implements a Listener interface.

Core-Generated Events

All events for resources are disabled by default. But, services registering resources with the Core can enable event generation on a resource-by-resource basis. The Core assumes the presence of a Core distributor and ships these events to it. Subscribers to these events include the Core proxies and any external clients.

Both external and Core distributors support the same interface. Event types are expressed as dot-delimited strings. Simple prefix matching of the event type string is supported.

Currently, only the contributed VFS application turns on event generation for the resources it uses.

A subscriber to a Core-generated event would simply become a subscriber to the Core and express interest in events of a specific type for an e-speak Resource Locator (ESRL) Core-generated event.

Running the Core Event Distributor

The standard event distributor is started by default in the `<installDir>\config\default.ini` file distributed with e-speak. The relevant lines of the file are:

```
[CoreDistributor]
Class=net.espeak.jesi.event.coredist.ESCoreDistributor
Args=CORE_PORT=%espeak_port%
WaitFor=Core Ready
```

You can replace this distributor with another customized distributor of your own creation if needed. The distributor can also be run stand-alone by invoking the scripts.

To run the Distributor stand-alone

1. Enter `<installDir>\bin\espeak core p=TCP:12346` to start a Core.
2. In another window, enter `<installDir>\bin\espeak cd CORE_PORT=12346` to start the Core event distributor.

Building Event Listeners, Event Subscribers, and Event Generators

The API documentation for the event service is located in `<installDir>\doc\html\index.html`.

To build new applications using this API, add `<installDir>\lib\esclient.jar` to your `CLASSPATH`.

Advertising Service

The advertising service is used to locate and import services that are not present on a local machine. The advertising service makes connections to other advertising services if necessary, imports services that satisfy clients' requirements, and returns the services to the client.

LDAP Server

An advertising service may run in one of two modes:

- Offline with an external lookup directory
- Online without an external lookup directory.

Any lookup directory may be used to implement the offline mode. The advertising service in this release uses a Lightweight Directory Access Protocol (LDAP) server as its external lookup directory. For more information about LDAP, see the LDAP World website at this URL:
`http://www.critical-angle.com/ldapworld/index.html`

Installing a Directory Server

Follow these steps to download and configure a directory server to use with the advertising service:

1. Obtain a directory server program. A 60-day trial version can be downloaded from Netscape at:
   `http://www.iplanet.com/downloads/testdrive/detail_8_7.html`
2. Get the documentation you need, such as an installation guide, to properly install the directory server. If you download the directory server from Netscape as mentioned above, you can find documentation at these locations:
   - http://developer.netscape.com/docs/manuals/index.html
   - http://home.netscape.com/eng/server/directory/4.0
3. Make a note of the following values as they pertain to your installation. You need this information to complete the configuration process:
   - Directory suffix
   - Directory Manager DN

- Directory Manager password

NOTE: The installation process asks for a large amount of configuration data. You can use most of the default values, except for the values noted above.

4. After you install the LDAP server on NT, you need to reboot the system to start the directory server. Check the instructions as appropriate for Unix platforms.

5. Create a schema to store information about the connection object. Creation of object classes and attributes is described in Chapter 3 of the *Netscape Directory Server: Administrator's Guide*.

To create an attribute for Connection Objects

1. On the Directory Server Console, select `Configuration tab -> Database icon -> Schema folder -> Attributes tab`.

2. In the Attribute dialog box, click Create.

3. Enter `cobj` in the Attribute Name text box.

4. Select binary for its syntax.

5. Click OK.

To create an object class

1. On the Directory Server Console, select `Configuration tab -> Database icon -> Schema folder`.

2. Click Create, then enter the name `ESCore` in the Name text box.

3. Add attributes `cn`, `cobj`, and `objectclass` to the object class.

4. Click OK.

How to Download the LDAP JDK

An LDAP JDK provides a set of APIs to access an LDAP directory server. A free implementation of LDAP JDK can be downloaded from Netscape. The latest version is Netscape Directory SDK 3.1 for Java and can be obtained at:
`http://developer.netscape.com/tech/directory/index.html`

Configuring the Advertising Service

Configuring the advertising service involves editing the LDAPDirAgent.java file, inserting information you noted during the LDAP server installation process, and recompiling the package as follows:

1 Open this file:

`<installDir>\src\services\advertise\agents\LDAPDirAgent.java`

2 Assign the values you defined while installing the LDAP server to the following items:

- `OSO_DN = <Directory Manager DN>`
- `OSO_PASSWD = <Directory Manager password>`
- `SEARCH_BASE = <Directory Suffix>`

NOTE: If you do not want to rebuild the system, you can pass these parameters as command line arguments. See "Automatic Connection without a Backend LDAP Directory" on page 70 for more information.

How to Advertise Services

Services can be advertised locally within an enterprise, across the Internet, or in a local domain. This is described in detail in the following sections.

Advertising a service across the Internet

Service providers can advertise their services throughout the world using the HP-hosted global service directory by selecting default values when starting the advertising service.

In this case, when a user invokes the `advertise()` call, the service gets advertised in the global service directory. For example, this enables a service provider located in Los Angeles to advertise a service to the global directory, accessible by any clients in New York. The service provider places the service outside the firewall, but the clients can be inside or outside the firewall. The clients only need to connect through any proxy which supports the `HTTP connect` call.

Advertising a service within an enterprise

Service providers can advertise their services within an enterprise in two ways:

- Using a service directory, such as an LDAP server
- Using an e-speak Core repository

Using a service directory is similar to using the HP-hosted global service directory, except that the advertising service connects to the service directory specified by the service provider. This service directory can be located within the enterprise and spread in different locations. In this case, you start the advertising service by specifying `-beproto <protocol>`, `-behost <hostname>`, and `-beport <port-number>` command line options. If the `-beproto` command line option is not specified, then the global service directory hosted by HP is selected.

By using an e-speak Core repository, service providers who do not want to use service directory like LDAP can still achieve the same results. This is done by starting the advertising service in `With repository` mode. When the user invokes the `advertise()` call, the service is placed in the local advertising service. In this case, clients doing a search in a community can specify fully qualified group names (group name + host name +port number). The infrastructure automatically

connects and makes all the services available in the advertising service identified by host name and port number visible to the client, even when not using any service directory.

Advertising a service in a local domain

Service providers can advertise services in the local domain in two ways:

- Using a service directory, such as an LDAP server
- Using an e-speak Core repository

Using a service directory is similar to using the HP-hosted global service directory, except that the advertising service connects to the service directory specified by the service provider. This service directory can be located within the enterprise and spread in different locations. In this case, you start the advertising service by specifying `-beproto <protocol>`, `-behost <hostname>`, and `-beport <port-number>` command line options. If the `-beproto` command line option is not specified, then the global service directory hosted by HP is selected.

The benefit of using an e-speak Core repository is that users can take advantage of the spontaneous discovery mechanism in the e-speak system. Service provider advertisements are automatically transferred to all the advertising services belonging to the same group. So, clients doing a search in a community would only need to specify the group names. The hostname and portnumber are no longer necessary in the local domain, because the advertising services spontaneously talk with each other in the local domain and exchange information.

Multiple groups in a single service directory

It is possible to have multiple groups in a single service directory, such as LDAP. In this case, different advertising services belonging to different groups can connect to the same service directory and advertise services.

Selecting a group name

Two different service providers may advertise services with exactly the same descriptions (attribute values). To prevent collisions across the advertised services or to protect access to the services advertised, specify a unique group name for the

advertising service. The service provider uses the `-group <groupname>` command line option of advertising service to achieve this. Service providers and clients are responsible for prevent collisions by selecting unique group names for their advertised services.

A client doing a search can specify a community which is a collection of group names. In this case, only services registered in the those groups are returned to the client. Services in other groups, even if matching client's query, are not returned.

A service provider wishing to advertise the service to the whole world can do so by starting the advertising service with group name `speaktome` and using the HP-hosted global service directory. Specifically, the service provider starts the advertising service with command line option `-group "speaktome"` to advertise the service in the HP-hosted global service directory.

Starting the Advertising Service

The e-speak Core must be running before you start the advertising service. The Advertising Service can run in one of two modes — with or without an external lookup repository. Each mode requires a different set of parameters. The next section describes what parameters may be specified. The following sections explain how to start the Advertising Service with these parameters.

Parameters to be Specified

The following parameters can be specified for running the advertising service with or without an external lookup repository:

Specification of Core

The Advertising Service should be connected to a Core. The current release supports two ways to specify a Core with which the advertising service is registered. One is to specify the Core by giving a protocol name, a host name, and a port number. If the communication protocol is not given, TCP/IP is assumed.

`-cp <conn -protocol>`	The communication protocol between the Core and an Advertising Service
`-eshost <core-hostname>`	The name of a host machine on which a Core is running
`-esport <core-port-number>`	The port to which the Core is listening

The other way is to give a URL string for the Core, such as `"tcp:rgelpc016.rgv.hp.com:12346"`.

`-esurl <protocol-hostname-port-number>`	E-speak URL in the following format: "protocol : hostname : port number"

Configuration for Initial Discovery

A group of e-speak Cores that run together and share the same external repository may be given a name. The name is specified with the `-group` option as follows:

```
-group <group-name>
```

Initial discovery of advertising services is implemented using a multicast mechanism. The port the advertising service listens to is specified with the `-mport` option. By default, it uses 1438 (i.e., the SLP multicast port), but any port can be used. However, you may need to specify a port number greater than 1000 if you do not have a root privilege to run the advertising service. The `-mport` option is as follows:

```
-mport <multicast-port-number>
```

Advertising Services get the information about which host an external repository is running on, what protocol the repository uses, and what port it listens to from the initial discovery and join process. Alternatively, advertising services may hard-code the information in a file and read it at start-up time. The file can be specified with the `-beconfig` option as follows:

```
beconfig <backend-configuration-file>
```

A backend configuration file can contain information on host, port, and protocol. The entries can appear in any order, such as the following:

- `Host: <host-name>`
- `Port: <port-number>`
- `Protocol: <protocol-name>`

Specification of Connection Mode

To control the initial discovery and joining mode, you can specify a connection configuration file with the `-connconfig` option.

`-connconfig <connection-configuration-file>`	File which contains connection mode

You can turn on or off the option of multicasting to other advertising services a request to connect to each service's Core (`AutoConnect`). Also, you can turn on or off the option of accepting an auto connect request from other advertising services (`AcceptConnection`). The default value of both options is set to `yes`.

- `AutoConnect: yes/no`
- `AcceptConnection: yes/no`

Specification of LDAP Configuration Information

To allow dynamic discovery and joining, there are advertising services that access the directory as the same user, such as Directory Manager. The login information may be hard-coded in the Advertising Service itself, or it may be given as command-line arguments as follows:

- `-rootdn <Directory Manager DN>`
- `-passwd <Directory Manager password>`

- `-base <Directory suffix>`

NOTE: This information is determined when the LDAP directory is configured. See Chapter 2, "Installing and Configuring." for more information.

Starting Advertising Service With or Without a Backend LDAP Directory

With the configuration options above, you can start up the Advertising Service with or without an external repository using one of the following sets of commands.

Automatic Connection without a Backend LDAP Directory

```
<installDir>\bin\espeak ads eshost=<hostname>
   esport=<portnumber> group=<groupname>
   mport=<multicastport> ypprotocol=slp
```

No Automatic Connection with a Backend LDAP Directory

```
<installDir>\bin\espeak ads eshost=<hostname>
   esport=<portnumber> rootdn=<Directory-Manager-DN>
   passwd=<Directory-Manager-password> base=<Directory-suffix>
   beconfig=<backend-configuration-file> group=<groupname>
   mport=<multicastport> connconfig=<connection-configuration-file>
```

Automatic Connection with a Backend LDAP Repository

```
<installDir>\bin\espeak ads eshost=<hostname>
   esport=<portnumber> rootdn=<Directory-Manager-DN>
   passwd=<Directory-Manager-password> base=<Directory-suffix>
   group=<groupname> mport=<multicastport>
```

Management Services

The current e-speak release contains a suite of four management services, Logging Service, Policy Manager, Service Manager and Process Manager. You must have the Java Servlet Development Kit (JSDK) 2.0 installed and the corresponding `jsdk.jar` file in your CLASSPATH environment variable in order to compile and run any program that uses management services.

Management services include:

- Logging Service
- Process Manager
- Policy Manager Service

Logging Service

The Logging Service provides a simple logging API to e-speak service writers. Messages are logged in a persistent store (one per Core) such that they may be meaningfully analyzed at some later date.

Concepts

Log messages will in practice come from a wide variety of sources and, in the worst case (for the log reader), come from different implementations of functionally identical services, each of which has a different Vocabulary (or dictionary) of log messages. Therefore, there must be some mechanism to differentiate one from another in order for the log to be meaningful. It is thus essential that each logged message must in some way specify the dictionary to which it refers. The implementation of the dictionary is left to the user.

Starting the Logging Service

To start the Logging Service, enter the following command (after setting CLASSPATH to e-speak required .jar files):

```
<JVM>
net.espeak.services.management.logger.manager.ESLogServiceManager
localhost <CORE_PORT> <ESPEAK_HOME>
```

NOTE: In this command, <CORE_PORT> is the port number of the local core that this Logging Service should connect to, <ESPEAK_HOME> is the location of the e-speak installation as set in the ESPEAK_HOME environment variable.

Example Usage

In order to log messages, an `ESLogClient` object is provided. This object has a `logMessage` method that takes `ESLogMessage` objects. Below is an example usage of `ESLogClient`:

```
import com.hp.es.management.services.logger.client.*;
import com.hp.es.management.services.logger.message.*;
import com.hp.es.client.framework.util.*;


ESLogClient lc = new ESLogClient(new FrameworkContext());
ESLogMessage m = new
     ESLogMessage(“id”, “d”, ESLogMessage.INFO, “msg1”);
lc.logMessage(m);
```

The `FrameworkContext` object handles the connection to the Core. The default constructor connects to a Core on the local host. To connect to a remote Core, pass the framework context constructor a `String[]` containing the host name and port number.

Process Manager

The e-speak Process Manager provides a uniform way of remotely managing of processes. Here process refers to a general operating system process, for example, a running Java Virtual Machine is a process, an active e-speak core is a process, and a database server could also be a process. The concept of process should not be confused with the concept of e-speak service. For example, an e-speak advertising service may consist of multiple processes: the backend database engines could be one process, while its e-speak service access front-end could be another process. While one service may constitutes of multiple processes, one process may also host multiple services. The management of processes is often interrelated to the management of services. Process Management is an essential part of the infrastructure that helps service management.

Process Management is provided through the e-speak Process Manager Service. It is a bundled e-speak service in this release.

In order to run the Process Manager Service, the user is expected to know:

- How to start e-speak Cores
- How to start the e-speak Advertising Service

Starting the Process Manager Service

To start the Process Manager service, do the following:

1. Make sure there is a local core running, and you know its port number.
2. If you want the Process Manager Service to be accessible remotely, then make sure you have started a local Advertising Service.
3. Start the process manager with the following command line (after setting CLASSPATH to e-speak required .jar files):

   ```
   <JVM> net.espeak.services.management.CPM.service.ProcessManager
   <CORE_PORT>
   ```

   NOTE: In this command, <CORE_PORT> is the port number of the local e-speak core. You can only start one instance of Process Manager on each machine. There is no need for running more than one instance of Process Manager on one machine.

Accessing the Process Manager Service from Your Application

1. The Process Manager Service is an e-speak service. You can discover this service with the following e-speak query:

   ```
   "Type==’ProcessManagerService’ and Name==’<HOST_IP>’"
   ```

   NOTE: In this query, <HOST_IP> is the IP address of the host where the Process Manager is running. The service interface used in discovery shall be:

   ```
   net.espeak.services.management.CPM.service.ProcessManagerIntf
   ```

2. The returned service stub implements the following interface:

```
public interface ProcessManagerServiceIntf extends ESService {

    public String[] allProcesses() throws ESInvocationException;
   public String[] allLiveProcesses() throws ESInvocationException;
    public String ping(String msg) throws ESInvocationException;
    public String run(String path) throws ESInvocationException,
      ProcessManagerException;
    public void stop(String pid) throws ESInvocationException,
      ProcessManagerException;
   public String getStatus(String pid) throws ESInvocationException,
      ProcessManagerException;
    public String getLastStatus(String pid) throws
      ESInvocationException, ProcessManagerException;
    public void remove(String pid) throws ESInvocationException;
    public void setConfig(String s) throws ESInvocationException;
    public String getConfig() throws ESInvocationException;
   public byte[] getOutput(String pid) throws ESInvocationException,
      ProcessManagerException;
    public byte[] getError(String pid) throws ESInvocationException,
      ProcessManagerException;
}
```

The client can use the interface to enumerate, start, stop, get standard output, and/or get status of all processes managed by the Process Manager. Note that a process becomes manageable by the Process Manager if it is started through the Process Manager.

Accessing the Process Manager Service using the System Deployment Console

The System Deployment Console uses the Process Manager to control processes on local and remote machines. Thus it is a client of the Process Manager Service. For more information about how to run the System Deployment Console, please see the e-speak *System Deployment Console User Guide*, which is available on the e-speak website.

Policy Manager

A Policy Manager may be thought of as a user environment, similar to a user environment under Windows NT. A user may set policies, get policies, or remove policies using the Policy Manager.

A Policy Manager Factory is implemented as an e-speak service. A Client would typically discover the Factory Service and then use it to create, remove, link, or unlink a Policy Manager.

Starting the Policy Manager

To start the Policy Manager, use the following command (after setting CLASSPATH to e-speak required .jar files):

```
<JVM>
net.espeak.services.management.policymanager.PolicyManagerFactor
yImpl localhost <CORE_PORT> <ESPEAK_HOME>
```

NOTE: In this command, <CORE_PORT> is the port number of the local core that this Policy Manager Service should connect to, and <ESPEAK_HOME> is the location of the e-speak installation as set in the ESPEAK_HOME environment variable.

Chapter 5 Working With Applications

How Applications Work in E-speak

As mentioned in Chapter 1, an e-speak service can call on other services to perform a task, and it can be called on by clients in specialized environments. Understanding the service/client relationship is important for writing applications in the e-speak environment. This chapter help you gain that understanding.

This chapter describes how to get started with a simple program, *Echo*, which is included in this release.

About The Echo Program

The *Echo* application has two components —`EchoServer` and `EchoClient`. The `EchoServer` program provides one service — it echoes any string sent to it by a Client. `EchoClient` is a sample Client that sends a string to the EchoService and checks if the string is properly echoed back to it. The `EchoServer` and `EchoClient` source code is available under the directory `<installDir>\samples\echo\src.`

Generally speaking, an e-speak service-provider, such as `EchoServer`, does the following:

- Connects to the Core
- Registers a service that can be discovered by Clients
- Waits for a request and responds upon receiving it

Meanwhile, an e-speak Client program, such as `EchoClient`, usually does the following:

- Connects to the Core

- Finds the service it wants to use
- Invokes a method on the service and processes the response

NOTE: The server must be ready before you start a client.

Echo Syntax

NOTE: This section uses the NT notation for pathnames and commands. On Unix platforms, use forward slashes (/) instead of back slashes (\).

EchoClient uses the property files `client.prop`, and EchoServer uses the property files `server.prop`. These files are available under the `singlecore` and `multicore` subdirectories of `<installDir>\samples\echo\config`.

Both files have the following syntax:

- `hostname=`<hostname where e-speak Core is running>
- `portnumber=`<port of the e-speak Core>
- `community=`<name of the group used to identify the group-server for Advertising Services>

You can eliminate the `hostname` field if EchoClient and Core are running on the same machine. You can also eliminate `portnumber` if the Core is started on the default port (`12346`). You can set `community=null` when EchoClient and EchoServer are connected to the same Core.

Building and Running the Echo Program

For building and running this sample program, set up the environment variables `JAVAC`, `JRE`, and `CLASSPATH` (`JVC` and `VJ++` if you are a Microsoft Visual J++ user) as described in Chapter 2, “Installing and Configuring”.

The following sections provide instructions for building and running the sample programs in Windows NT, HP-UX, and Linux.

Running Echo on Windows NT

Under Windows NT, you can run Echo from a command shell or within a Microsoft Visual Java (VJ++) IDE. To assist with compilation, the sample programs come with a batch file, `compile.bat`. When `compile.bat` is run, it calls `<installDir>\bin\envmake.bat`, which sets the `CLASSPATH` environment variables.

To run sample programs in an NT command shell

1. In a Command Prompt window, enter `cd <installDir>\samples\echo`
2. Set environment variable `ESPEAK_HOME` to point to the `<installDir>` and add `<installDir>\samples\echo\lib` to your `CLASSPATH` environment variable.
3. Run `compile` to build the sample programs. This creates the class files under the `<installDir>\samples\echo\lib\samples\echo` directory.
4. Run the following command to start the e-speak Core and EchoServer.

   ```
   <installDir>\bin\espeak -i
   <installDir>\samples\echo\config\singlecore\EchoServer.ini
   ```

   NOTE: An `EchoServer.ini` file is available under `<installDir>\samples\echo\config\singlecore` for this purpose.
5. You see output similar to this:

   ```
   ES Core starting with an In-Memory Repository.
   coreId = "17ef1871f8f7e2b3666541755c6d0c9d"
   Starting ES Core Server with Rendezvous of "TCP:12346". Started.
   Connected to e-speak
   Advertising service not found, defaulting to local repository
   Started Echo service
   ```
6. Open a different Command Prompt window and change directory to the `<installDir>\samples\echo` directory.
7. Run the EchoClient program using the `EchoClient.ini` file through the `espeak` utility.

```
<installDir>\bin\espeak -i
   <installDir>\samples\echo\config\singlecore\EchoClient.ini
```

8 You see output similar to this:

```
Group null will be contacted thru local core
findConnFac: search for conn fac returned null
ResourceName == 'TestServer'
ResourceName == 'TestServer'
EchoClient sent :*Hello World!*
EchoClient got reply : *Hello World!*

  SUCCESS. String returned by echo server matches the string sent.
```

9 You can terminate the Core and echo server by pressing Ctrl+C.

To run sample programs in Microsoft Visual Java (VJ++) IDE

1 Configure your VJ++ IDE as explained in the"Configuring Integrated Development Environments"section in Chapter 3. A Visual J++ project file, `echo.vjp`, under the directory `<installDir>\samples\echo`, is provided for your convenience.

2 Load the project 'echo' in VJ++ IDE, using the supplied project file with this command:
`<installDir>\samples\echo\echo.vjp`.

3 On the Build menu, click Build. This compiles the sample program. If no errors appear, the class files are created under the
`<installDir>\lib` directory.

4 On the Project menu, click Echo Properties… and go to the Launch tab.

5 Verify that the Custom radio button is selected and the Program and Argument fields have the following values. If not, correct the fields. When finished, click OK to close the dialog box.

Program `JView.exe`

Arguments

```
/p /cp:p "<JAVAPACKAGES>"
/d:espeak_home=<installDir>
/d:espeak_port=12346
/d:espeak_group="myEchoGroup"
net.espeak.util.SysLoader
"<installDir>\config\default.ini"
```

6. On the Debug menu, click Start Without Debugging. This starts the Core and other e-speak components running.

7. When all sample applications are complete, a new console window appears, showing output similar to the following:

```
ES Core starting with an In-Memory Repository.
coreId = "6bf5c260ca031e38246bdf23e2f9eded"
Starting ES Core Server with Rendezvous of "TCP:12346". Started.
Connection Object: mapped localhost to 15.81.93.137
-- WARNING:  Serializing class
net.espeak.infra.intercorecom.confactory.co.ConnectionObject
with slow, Java-dependent Java serialization
Created the BaseDistributorVocabulary vocabulary
Advertising service not running
Started Core distributor
switching to no-backend version...
advertising service is ready
```

8. On the Project menu, click Echo Properties and go to the Launch tab in the same IDE.

9. Click the Custom radio button and enter the following information in the two text boxes:

Program	`JView.exe`
Arguments	`/p /cp:p "<JAVAPACKAGES>" echo.EchoServer <install-Dir>\samples\echo\config\singlecore\server.prop EchoServer`

10 On the Debug menu, click Start Without Debugging. This runs the EchoServer program, producing the following output in the new console window:

```
Connected to e-speak
Vocabulary Name in Client Side = default
Started Echo service
```

11 When you are finished with the sample, terminate the service by pressing Ctrl+C.

12 In the same IDE, click Echo Properties on the Project menu and go to the Launch tab. Click the Custom radio button and enter the following information in the two text boxes:

Program `WJView.exe`

Arguments `/p /cp:p "<JAVAPACKAGES>" echo.EchoClient <install-Dir>\samples\echo\config\singlecore\client.prop "Hello World" EchoServer`

13 On the Debug menu, click Start. The Echo Properties dialog box appears.

14 In the Echo Properties dialog box, select echo.EchoClient and click OK.

15 The EchoClient program produces the following output in the Output window: (If the Output window does not appear, press Ctrl+Alt+O.)

```
group server null not in core
connect: group server url not given for group null

Advertising Services now in core:
[0]: name is rgelpc129.rgv.hp.com:12346 and subtype is rgelpc129.rgv.hp.com:12346
*** end of list ***
ResourceName == 'EchoServer'
ResourceName == 'EchoServer'
EchoClient sent :Hello World
EchoClient got reply : Hello World

  SUCCESS. String returned by echo server matches the string sent.
```

To compile in VJ++ IDE and run in an NT Command Shell

To compile the Echo sample in MS VJ++ but run it in an NT Command Shell, follow Step 1 and Step 3 from the ""To run sample programs in Microsoft Visual Java (VJ++) IDE" on page 80" for compilation, then follow Step 6 through Step 9 from the "To run sample programs in an NT command shell" on page 79 to run them.

Running *Echo* on HP-UX or Linux

For HP-UX or Linux, a Makefile is provided for compiling Echo sources. This Makefile can be found under `<installDir>/samples/echo`. You can either use a native `make` utility available on your Unix platform, or GNU's `make` utility (known as `gmake`). The GNU `make` utility can be downloaded freely from `ftp://ftp.gnu.org/gnu/make`.

The following targets are defined in the `makefile` provided:

- `make clean` removes all class files from the `echo/lib` directory.
- `make all` compiles all Java source files in the `echo/src` directory. The class files are created in the `echo/lib` directory.

To compile source files, start e-speak, and run sample programs

1. Copy the *Echo* directory recursively to a temporary location, `/home/myself`, for example.
   ```
   cp -R <installDir>/samples/echo /home/myself
   ```
2. Change to directory `cd /home/myself/echo`. Set the environment variables as discussed in "Setting the Environment Variables" in Chapter 2.
3. Enter `make all` to compile the sources. Class files are created under the `/home/myself/echo/lib/samples/echo` directory. Add this directory to your `CLASSPATH`.
4. Start the e-speak Core and EchoServer using the `EchoServer.ini` file:
   ```
   <installDir>/bin/espeak -i
     /home/myself/echo/config/singlecore/EchoServer.ini
   ```

You see output similar to this:

```
ES Core starting with an In-Memory Repository.
coreId = "17ef1871f8f7e2b3666541755c6d0c9d"
Starting ES Core Server with Rendezvous of "TCP:12346". Started.
Connected to e-speak
Advertising service not found, defaulting to local repository
Started Echo service
```

5. In a new shell window, go to the directory where the sample program Echo is compiled: `cd /home/myself/echo`.

6. Add the `lib` directory to your current settings for the `CLASSPATH` environment variable. Type `export CLASSPATH=${CLASSPATH}:/home/myself/echo/lib` when using ksh, sh, or bash. (You can enter `setenv CLASSPATH ${CLASSPATH}:/home/myself/echo/lib` if you use `csh`.)

7. Run EchoServer and EchoClient, using the `espeak` utility. To run, enter:

```
<installDir>/bin/espeak -i
/home/myself/echo/config/singlecore/EchoClient.ini
```

You see output similar to this:

```
Group null will be contacted thru local core
findConnFac: search for conn fac returned null
ResourceName == 'TestServer'
ResourceName == 'TestServer'
EchoClient sent :*Hello World!*
EchoClient got reply : *Hello World!*

  SUCCESS. String returned by echo server matches the string sent.
```

8. Terminate the Core and echo server by typing Ctrl+C in this window.

Distributed Applications

A distributed e-speak environment consists of any number of interconnected e-speak Cores. Connections between Cores are established in pairs. Each pair of connected Cores agrees to share some resources through an *export* process. Once a resource is exported to a Core, all tasks connected to that Core can discover and use the resource as if it were a local resource.

For a Core to participate in a distributed e-speak environment an export mechanism such as an advertising service, must be available.

The following sections describe how to set up a distributed e-speak environment and how to run the *Echo* sample program that illustrates this functionality.

NOTE: This section uses the NT notation for pathnames and commands. On Unix platforms, use forward slashes (/) instead of back slashes (\).

Connection Object Files

Connections between Cores are established in pairs. Each connection requires a configuration file, called a *Connection Object file,* which specifies the port on which it listens. A Connection Object file contains information about three items:

- The name of the protocol
- The hostname on which the Advertising Service is running
- The port on which it is listening

The Connection Object files are used for two purposes. Each file describes how the Service Engine (Core) should configure the ports to listen on. Connection Object files also describe the machine to which the services are to be exported. The Connection Object files that describe the listening ports should be present on the corresponding machines.

Logical Machines

Typically, each Service Engine, or *Logical Machine*, is created on a separate computer (physical machine). However, it is possible to configure multiple logical machines on the same physical machine. This can be two Cores running on a single computer or on two different computers on a local area network.

When setting up this kind of configuration, each Logical Machine must have a unique Core port number. You can use the espeak utility to quickly set up a Service Engine (Core) ready to participate in a distributed e-speak environment.

To set up a logical machine, you need to provide the socket port number used by the Core and the name of the Connection Object file that contains information used by Service Engine to create Connection Objects.

After creating two logical machines on the same physical machine (as outlined below), you can start running the sample distributed applications, as described in the next section.

To set up Logical Machines

1. Enter `<installDir>\bin\espeak core p=TCP:12345` to run a Core for Logical Machine 1.
2. In a different window, enter `<installDir>\bin\espeak ads esport=12345 copath=<installDir>\config myco=co1` to start a AdvertisingService for Logical Machine 1.
3. In a window on another machine (or in another window on the same machine), enter `<installDir>\bin\espeak core p=TCP:12346` to run a Core for Logical Machine 2.
4. In the second window, enter `<installDir>\bin\espeak ads esport=12346 copath=<installDir>\config myco=co2` to start a AdvertisingService for Logical Machine 2.

Setting Up a Distributed *Echo* Program on Two Logical Machines

NOTE: When running this sample across two physical machines, make sure that the `community` name that the `client.prop` refers to (and `-group` argument of the advertising services under `EchoServer.ini` and `EchoClient.ini`) are not used by some other application within the same Local Area Network.

Figure 2 shows the setup structure for running a distributed Echo program.

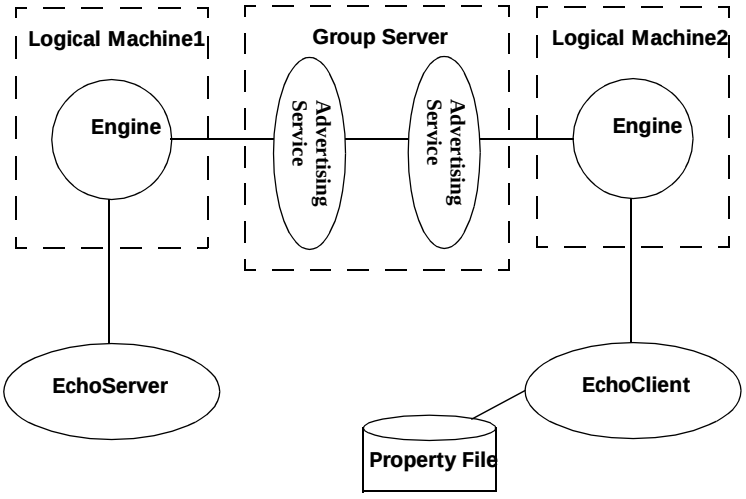


Figure 2 Client and Service in distributed environment

To set up the program

1 Start two Service Engines (Cores) on either the same machine or on two different machines:

- A server application, `EchoServer`, started on `logical machine1`
- A Client application, `EchoClient`, run on `logical machine2`

In this example, two Connection Object files, `co1.MYCO` and `co2.MYCO`, are used to configure the Service Engines (Cores) for `machine1` and `machine2`. These files are available in the directory `<installDir>\samples\echo\config`. The contents of the two Connection Object files are shown below:

- `co1.MYCO`
  `TCP: machine1.domain.com 23456`
- `co2.MYCO`
  `TCP: machine2.domain.com 34567`

You can replace `machine1.domain.com` and `machine2.domain.com` as appropriate. They can point to the same physical machine (local host).

Regardless, `co1.MYCO` should be present on `machine1` and `co2.MYCO` should be present on `machine2`. If you run the distributed application on the same machine, `machine1` and `machine2` can both be identical as long as the port numbers are different.

To run the program

1. In separate windows on each machine, change to the `<installDir>\bin` directory and add `<installDir>\samples\echo\lib` to the `CLASSPATH` environment variable.
2. Edit the Connection Object files `co1.MYCO` and `co2.MYCO` in the `<installDir>\samples\echo\config` directory to reflect the correct machine names.
3. On machine1, start the Core, Advertising Service, and EchoServer using the `EchoServer.ini` file available in the `<installDir>\samples\echo\config\multicore` directory:

   ```
   <installDir>\bin\espeak -i
   <installDir>\samples\echo\config\multicore\EchoServer.ini
     hostname=<hostname>
   ```

   You see output that looks like this:

```
************************************************************
* Running:  Core AdvertisingService echo.EchoServer
```

```
*************************************************************

ES Core starting with an In-Memory Repository.
coreId = "e648caf27cdd37788fabba57337bbc7b"
Starting ES Core Server with Rendezvous of "TCP:12346". Started.
Connection Object: mapped localhost to 15.81.93.137
-- WARNING:  Serializing class
net.espeak.infra.intercorecom.confactory.co.ConnectionObject with slow, Java-
dependent Java serialization
Warning: Pls check LDAP configuration if you intended to use LDAP. Switching to
non-LDAP mode now
switching to no-backend version...
advertising service is ready
i am processing 3
Querying for advertising services with query: ( ResourceType == 'Advertising
Service' and ResourceSubType == 'echogrp' )
my entries... 0
rgelpc129.rgv.hp.com:12346: agent found 0 services for query (ResourceType ==
'ESVocabulary') and (ResourceName == 'BaseDistributorVocabulary')
i am processing 1
Connected to e-speak
Vocabulary Name in Client Side = default
Started Echo service
```

4 On machine2, in a new shell window, enter the following command to start another Core, Advertising Service, and EchoClient:

```
<installDir>\bin\run -i
   <installDir>\samples\echo\config\multicore\EchoClient.ini
   hostname=<hostname>
```

You see output that looks like this:

```
*************************************************************
* Running:  Core AdvertisingService echo.EchoClient
*************************************************************

ES Core starting with an In-Memory Repository.
coreId = "fb05569e0bbc814f1efb341214d880c0"
Starting ES Core Server with Rendezvous of "TCP:43210". Started.
Connection Object: mapped localhost to 15.81.93.137
-- WARNING:  Serializing class
net.espeak.infra.intercorecom.confactory.co.ConnectionObject with slow, Java-
dependent Java serialization
switching to no-backend version...
advertising service is ready
Warning: Pls check LDAP configuration if you intended to use LDAP. Switching to
non-LDAP mode now
```

```
-- WARNING:  Serializing class
net.espeak.infra.intercorecom.ESIP.ESIPControlMessage with slow, Java-dependent
Java serialization
Group echogrp will be contacted thru local core
findConnFac: conn fac returned is
net.espeak.infra.client.coreproxy.ESConnectionFactory@155
Trying to connect to CF specified by null
connect: group server echogrp already in core

Advertising Services now in core:
[0]: name is rgelpc129.rgv.hp.com:12346 and subtype is echogrp
[1]: name is rgelpc129.rgv.hp.com:43210 and subtype is echogrp
*** end of list ***
(ResourceType==’Advertising Service’) and (ResourceSubType ==’echogrp’)
(ResourceType==’Advertising Service’) and (ResourceSubType ==’echogrp’)
ResourceName == ’RemoteServer’
ResourceName == ’RemoteServer’
i am processing 3
Querying for advertising services with query: ( ResourceType == ’Advertising
Service’ and ResourceSubType == ’echogrp’ )
my entries... 0
rgelpc129.rgv.hp.com:43210: agent found 0 services for query (ResourceType ==
’ESVocabulary’) and (ResourceName == ’BaseDistributorVocabulary’)
EchoClient sent :*Hello!*
Connection Dropped By the Client
i am processing 3
Querying for advertising services with query: ( ResourceType == ’Advertising
Service’ and ResourceSubType == ’echogrp’ )
my entries... 0
rgelpc129.rgv.hp.com:43210: agent found 1 services for query ResourceName ==
’RemoteServer’
EchoClient got reply : *Hello!*

  SUCCESS. String returned by echo server matches the string sent.
```

Chapter 6 Using Security in E-speak

This chapter describes the current basic setup for using e-speak securely. Security is subtle and it is dangerous to treat it as just another thing to "select the check box" — it can bite you hard!

There are four sections to this chapter.

- "The Basic Security Model" deals with the basic aspects of the security model, and in particular PSEs and certificates.
- "Bootstrap Process for Testing" discusses a bootstrap process used for testing purposes.
- "Configuration Files" discusses security configuration files.
- "Security Examples"provides a number of basic examples of tag attributes and protection masks.

The Basic Security Model

Everyone (and everything) has a set of public/private keys. Entities are distributed and interact with one another by means of secure sessions using the SLS protocol – this includes firewall traversal technology. All entities can both *use* services offered by others and also *provide* services to others. This means that all parties in secure sessions have to be authenticated to each other. In particular, SLS secure sessions authenticate both parties involved by using challenge-response negotiations based on public-key cryptography.

Access control to services is done by exchanging digitally signed certificates as a part of the SLS protocol providing secure sessions. These certificates act like "tickets" that grant entities with authorization to access and make use of services. Certif-

icates are signed by issuing entities (or Principals) and are issued to subject principals who may use them. The subject of the certificate is the entity which is being authorized. For example, if the issuer is the Core, and the subject is the client, and the certificate contains the tag attribute (`net.espeak.method (*) (*)`), the Core is issuing a certificate authorizing the client to access any method within any service in the Core.

These certificates can also be chained together (using *delegation*) to give composite authorizations. Refer to the J-ESI documentation and the *E-speak Architecture Specification* for more information about the security model.

PSEs and Certificates

A Private Secure Environment (PSE) represents a keystore containing public/private key pairs. Each principal e-speak entity needs to have its own set of keys and needs to store them securely within a PSE. The PSE itself can be stored as a binary file in your local file system. This data is encrypted and a passphrase is required to lock/unlock the data it contains.

The PSE is responsible for generating its own key pairs – in particular, it has been designed so that private keys should never be exposed.

The other main function of a PSE involves validating and signing certificates. Validating a certificate involves checking the signature of the certificate using the issuers public-key (embedded within the certificate). Signing a certificate involves using a private key held within a PSE to create a digital signature, based upon a message digest of the certificate data.

PSE Manager

The PSE Manager is a GUI tool that supports these basic tasks:

- Creating a new PSE and saving it as a binary file. This involves choosing a passphrase that is used as an encryption/decryption key. It is important to keep this passphrase information secure – anyone capturing your PSE will be able to *perfectly* masquerade as you and access everything that you can access. Also,

losing or forgetting your passphrase means that you will be unable to unlock or access your own PSE. For automatic operation, the PSE passphrase can be kept in a pass file stored on a floppy disk etc. There is a configuration option for this.

- Creating new key-pairs. The PSE Manager can create new key-pairs, each of which are given a symbol label. These labels can then be used when constructing certificates.
- Creating and editing certificates. Attribute certificates typically contain information about the issuer, the subject, what is being authorized and the validity period. For convenience, the PSEs symbolic labels (or *roles*) for keys can be used to refer to known keys when constructing certificates – thus avoiding tedious and error-prone data entry of key information.
- Validating and signing certificates as described above.

PSE data can be saved to binary files (using a passphrase for the encryption key) and certificates can be saved to text files etc.

For further information on the PSE Manager, see the PSE Manager user documentation.

Bootstrap Process for Testing

The bootstrap process *for testing purposes* is shown below. When writing and deploying secure applications, refer to the J-ESI documentation and the E-speak *Architecture Specification*. This configuration is only for testing purposes to get the application programmer started. In a live deployment, all entities (users) or services that need to be distinguished for access control purposes need to have separate PSEs. Sharing a PSE is comparable to sharing a login password to a Unix or Windows NT system and is generally accepted to be a bad security practice.

In the following, the subject of the certificate is the entity which is being authorized. For example, if the issuer is the core and the subject is the client and the certificate contains the tag attribute (net.espeak.method (*) (*)), the core is issuing a certificate authorizing the client to access any method within any service in the core.

1 Use the PSE Manager GUI tool to do the following:

a Generate a keystore object (i.e. a Private Secure Environment) and is typically called `securestore.txt`. This is presently shared by all participants – the core, the client and the service. Therefore, this configuration is *not* distributed.

b Each participant will have their own key-pairs. The current simple approach is to generate three different key-pairs, one for each participant, with the following labels: `client`, `core`, and `service`, all within the same PSE.

c Generate a basic attribute certificate, one for each pair of distinct participants (i.e. client as issuer, core as subject and so on for all distinct combinations) which gives each participant *arbitrary* permission to perform operations. Each certificate will contain the all-powerful e-speak tag attribute:

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

The PSE Manager can be used to conveniently generate these attribute certificates – it has access to all the keys that were generated. The PSE labels associated with the key-pairs can be used to refer to the keys within the certificates for convenience.

d After it is generated, the certificate must be issued – this means it is *signed* by the issuer (client, Core, or service). Again, the PSE Manager can perform this function of signing these certificates by any one of the participants.

2 To operate the core with security turned on, a security configuration file must be correctly loaded containing the appropriate attributes. The configuration file is more fully explained in the following sections. A high-level snapshot is as follows:

a The configuration file is like a Java properties file and is typically named `espeak.cfg`. It is searched for in the current directory, the user's home directory or on the Java CLASSPATH.

b A very simple espeak.cfg file is shown in Figure 3.

```
!============================================================================
! E-speak properties file.
!============================================================================

!----------------------------------------------------------------------------
! Security properties.
!----------------------------------------------------------------------------
! Master flag controlling whether security is on or off.
net.espeak.security.activate=on

! Set a property prefix.
@prefix=net.espeak.security

! Default name of the keystore file
.pse.storefile=securestore.txt

! Gui mode runs a dialog for the passphrase.
!.pse.mode=gui
! Passphrase mode looks for the passphrase property.
.pse.mode  = passphrase
! Passfile mode looks for a file containing the passphrase property.
!.pse.mode = passfile

! Define the passphrase.
.pse.passphrase = default passphrase

! Define the default role (i.e. the default PSE key label).
!.pse.role = client
```

Figure 3 A very simple `espeak.cfg` file

The following section discusses configuration files in more detail.

Configuration Files

The default configuration file is espeak.cfg. The file is looked for in the following places: the config directory under e-speak home as defined by the property `espeak_home`, the directory specified by property `net.espeak.util.config.file`, the current directory (from the system property 'user.dir') if the property is not set, or the directory specified by the `user.home` system property as a system resource from the classpath.

Java system properties can be set on the java command line using this syntax:

`Dproperty=value.`

The name of the file to look for can be specified using the `net.espeak.util.config.file` property.

The file defined by the `property net.espeak.util.config.master` is always loaded on top of all other files, if specified. The default for this property is null.

All files found are loaded, in reverse order, with files found earlier being merged on top of properties from files found later. The format of the files is java properties file format, with the following additions.

`@prefix=<prefix>`

This sets a property prefix to apply to properties starting with a dot. For example:

```
@prefix=net.espeak.security
.pse.mode  = passphrase
```

results in `net.espeak.security.pse.passphrase` being set to `passphrase`. Once set, a prefix remains in force until changed or set null.

`@mode=<mode>`

If the `<mode>` is “override” (default), the values found in this file will be used and all previous values will be ignored. Once the espeak.cfg parser encounters a file with mode set to “override,” no more files will be parsed. If the mode is “merge,” espeak.cfg files will be combined. But if two files specify values for the same property, only the value in the last file to be parsed will be used.

The name of the configuration file to look for can be set using the system property `net.espeak.util.config.file,` which has the default value `espeak.cfg`. If the system property `net.espeak.util.config.master` is set, the file of that name will be loaded on top of all other files found.

You can get the configuration by calling `ConfigIntf Config.getInstance()`, which returns a reference to a static instance of the default configuration. Other files can be loaded directly if wanted, see `util.Config` for the API. Single property files can be loaded using `ConfigProps`.

Property File Syntax

A java properties file contains property names and definitions.

- The name is separated from the definition by '='.
- Spaces before the property name and around the = are ignored.
- The value of the property extends to the end of line, and includes trailing spaces.
- Long property values can be broken across lines using \ to escape new lines.
- The characters ! and # introduce end-of line comments.
- The character : may be used as an alternative to =.

Property Conversion

The class util.Convert provides methods to convert property strings to and from common types. The types int, boolean, and long are supported. The duration converters accept times in the format 12h3m1.001s and convert them to longs in milliseconds. Any zero component of a time can be omitted, and spaces may be included. A zero time may be given as 0s.

The boolean converter accepts on, true, yes for true and off, false, no for false, regardless of case.

Argument Specifications

The mapping or argument switches to properties can be defined using util.ArgSpec. This provides methods to process command-line arguments and map them onto properties in a configuration.

Security Properties

The following are the properties supported by the security code.

- Master flag controlling security: `net.espeak.security.activate`, boolean, default off. If this property converts to true, security is activated.

- Property `net.espeak.security.connectOnContact`, default off, controls whether secure sessions are established with newly encountered resources. When it is off, sessions are not established unless required (by SessionRequiredException) or created explicitly.
- PSE mode: `net.espeak.security.pse.mode`. Values: gui, passphrase, passfile. Default gui. If the mode is gui, a dialog is used to get the PSE passphrase. If the mode is passphrase the property `net.espeak.security.pse.passphrase` is used to get the passphrase (default null). If the mode is passfile the property `net.espeak.security.pse.passfile` (default passfile.txt) is used to get the name of a file which must contain a `net.espeak.security.pse.passphrase` property defining the passphrase.
- PSE key file: `net.espeak.security.pse.storefile`, default securestore.txt. Defines the name of the file containing public-private key pairs.
- PSE role: `net.espeak.security.pse.role`, default client. Define the default role (symbolic PSE key name).
- PSE file protection mode: `net.espeak.security.pse.OSfileprotection`, default true. This property specifies whether local OS file protection should be applied and is supplied purely as an aid for testing purposes. For full security protection, this option should be true.
- Certificate file suffix: `net.espeak.security.pse.certfile`, default certs.adr. The value of this property is appended to the role name to get the name of the certificate file to load. If the role is 'client' the certificate file is 'clientcerts.adr' for example.
- ACL file suffix: `net.espeak.security.pse.aclfile`, default acl.adr. The value of this property is appended to the role name to get the name of the ACL file (trust assumptions) to load. If the role is 'client' the ACL file is 'clientacl.adr' for example.
- Cipher suites: `net.espeak.security.cipherSuites`. The value is a list of cipher suites in ADR syntax. The default is to use hmac, sha-1, and 128-bit blowfish.

Sample espeak.cfg file

```
!============================================================
! E-speak security properties file.
!============================================================
net.espeak.security.activate=on
user.name="John Doe"

! Example time value.
foo.timeout = 12h 3m .0001s
! Set a property prefix.
@prefix=net.espeak.security
! Gui mode runs a dialog for the passphrase.
!.pse.mode=gui
! Passphrase mode looks for the passphrase property.
.pse.mode  = passphrase
! Passfile mode looks for a file containing the passphrase property.
!.pse.mode = passfile
! Define the passphrase.
.pse.passphrase = default passphrase
! Define the default role (PSE key name).
!.pse.role = foo
```

Security Examples

At the moment, the only thing a JESI service provider needs to worry about from a security point of view is the metadata and resource masks they set up for their resources. When security is enabled, the default behavior is to require authorization for all operations. The masks control this. If a mask is set, operations matching the mask are permitted whether the requestor is authorized or not.

There are two masks:

- the metadata mask for metadata operations
- the resource mask for resource specific operations.

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

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

In the metadata mask, the interface name is the core interface being specified, and the method name is the operation in that interface. For metadata, the interface is likely to be ResourceManipulationInterface, and the method name one of its methods.

In the resource mask for a J-ESI service, the interface name is the fully-qualified name of the interface class. The method name is the name of the method in the interface, plus the concatenated argument types. This allows overloaded methods to be distinguished.

The metadata mask is used by the in-core metaresource when performing metadata operations. The resource mask is passed to the service handler by the core for the service handler to use when performing operations on the service itself.

The masks are completely general tags, so the mask tag itself, or any of its fields, may use the tag matching features such as sets, prefixes and ranges. The interface and method names, for example, do not have to be string literals, they can be sets or prefixes. The general tag format is defined in the e-speak *Architecture Specification* chapter on Access Control.

This tag masks method foo in interface net.espeak.examples.ExampleIntf:

```
(net.espeak.method net.espeak.examples.ExampleIntf foo)
```

This tag masks all methods beginning with foo:

```
(net.espeak.method net.espeak.examples.ExampleIntf  (* prefix foo))
```

This tag masks methods foo and bar:

```
(net.espeak.method net.espeak.examples.ExampleIntf  (* set foo bar))
```

Methods with prefix foo or bar:

```
(net.espeak.method net.espeak.examples.ExampleIntf
  (* set (* prefix foo) (* prefix  bar)))
```

All methods in the interface:

```
(net.espeak.method net.espeak.examples.ExampleIntf )
```

This is equivalent to:

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

since missing trailing elements match anything.

Methods foo in InterfaceA and bar in InterfaceB:

```
(* set (net.espeak.method InterfaceA foo)
       (net.espeak.method InterfaceB bar))
```

All methods:

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

or simply:

```
(*)
```

The full form of the method tag is actually:

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

In the normal case, the service handler is only interested in its own operations, so it does not care what the service field is.

When a message invoking an operation is received, the service handler extracts the interface and method from it, and gets the service identifier from the information passed to the handler by the core. The service handler then constructs a method tag using this data and queries the service authorizer to see if the tag is authorized. The authorizer first checks to see if the tag matches the resource mask, and if it does, the operation is permitted. If the tag does not match the resource mask, the authorizer uses the current security session to see if the tag is authorized.

Normal tag matching rules are used throughout, which is why the service part of a mask tag was omitted above. A tag is authorized at a server in the current security session if a client has presented a valid certificate or certificates that contain the tag. A certificate is valid if it has not expired and its signature is valid. Certificate validity and tag matching are explained in detail in the e-speak *Architecture Specification* chapter on Access Control.

General tags can be constructed using the following method in ESSecurityEnv:

```
ADR createTag(String s) throws IOException
```

The IOException subclass net.espeak.security.adr.ADRParseException is thrown on a parse error. The parameter s is a string containing the input syntax for the tag.

Method tags can be created using

```
ADR createMethodTag(String interfaceName, String methodName,
                    ADR service)
```

For the purposes of resource masks, tags normally contain (*) as the service parameter. In advanced applications, the service ca set the service parameter to its service id, but this is not necessary.

After a mask tag is constructed, it is used in ESAbstractElement methods:

```
void setResourceMask(ADR tag) throws ESException
void setMetadataMask(ADR tag) throws ESException
```

Before a service is registered, these simply affect the local state. After registration, these set the local state and update the service metadata.

Masking can be turned on or off using ESAuthorizer:

```
void setMasking(Boolean x)
```

When masking is off, the resource mask is ignored by the service authorizer even if set. Setting masking off in the authorizer has no effect on the resource metadata, or the in-core metaresource handling metadata operations. Masking can be turned off completely, in the core and handler, by setting a mask to null.

ESConnection has methods for controlling the default resource and metadata masks used when services are registered:

```
void setDefaultResourceMask(ADR mask)
ADR getDefaultResourceMask()
void setDefaultMetadataMask(ADR mask)
ADR getDefaultMetadataMask()
void setMasks(ADR metadataMask, ADR resourceMask)
```

After a default mask is set, all resources registered use it until it is changed. Unless the default masks are set explicitly, ESConnection uses null for them, causing authorization to be checked for all operations.

Appendix A SysLoader Utility

This appendix provides help on the e-speak `SysLoader` utility and information needed to configure e-speak for using persistence.

SysLoader Utility

The e-speak `main` function is located in the class `SysLoader,` which belongs to the package `net.espeak.util`. The required start-up parameter is the name of the `.ini` file, which you can use to control e-speak services and classes to be loaded. The `SysLoader` utility can also print version information on the e-speak product.

Type `<JRE> net.espeak.Util.SysLoader –h` for available options, where `<JRE>` is your preferred Java run-time environment. Note that the `CLASSPATH` environment variable must contain e-speak's `escore.jar` file, available under `<installDir>\lib`. For example,

```
java net.espeak.util.SysLoader -h
Usage: <JRE> SysLoader [-h] [-v] [ini file]
-h : Prints this screen.
-v : Prints product version number.
ini file: Loads and runs commands from ini file specified.
     Looks for core.ini in current directory if none is
     supplied.
```

See the following sections for details on `.ini` file syntax.

Controlling the Classes Loaded at Start-Up

The default file is `core.ini` in the `current` directory. This default can be overridden on the command line with this command:

```
<JRE> net.espeak.cci.util.SysLoader [<your_ini_file>]
```

The `.ini` file must contain a `[Tasks]` section with a `Start=task_list` assignment. This assignment should list tasks in the order they should be started. A name should appear only once. The rest of the file consists of sections named by the task name found in the start assignment.

Recognized assignments in the `.ini` file include:

- `Args=<list of arguments to class>`

  Defaults to an empty list of arguments. This is optional.

- `Background=<true | false>`

  If `true` and the task is in another JVM, the task can continue executing after the JVM that started the task terminates. The default is `false`. This assignment is valid only for tasks started in another JVM. This is optional.

- `Class=<class to execute>`

  The path of the class to be executed in this task. The main method for the class is invoked. This is required.

- `LogErr=<null | file <errorFile> | stderr | stdout>`

  Specifies where to send error output. This assignment affects only tasks started in another JVM. The default is `stderr`. This is optional.

- `LogOut=<null | file <outputFile> | stderr | stdout>`

  Specifies where to send normal output. This assignment affects only tasks started in another JVM. The default is `stdout`. This is optional.

- `OnExit=<closeall | closeself>`

  Specifies what should happen when this tasks finishes. If `closeall` is specified, all tasks are terminated and the JVM exits. The setting `closeself` means that only the current task terminates when it finishes. The default is `closeself`. This is optional.

- `Pause=<seconds to pause>`

  Specifies the number of seconds to wait after starting this task. The default is `don't wait` (0 seconds). This is optional.

- `Run=<injvm | newjvm | on <host>>`

  Specifies how the task should be started. The setting `injvm` means to start the task in the current JVM. The setting `newjvm` means to start the task in a separate JVM on the local machine[1]. The setting `on <host>` is used to start the task in a separate JVM on the remote host using the `rsh` command. The default is `injvm`. This is optional.

- `WaitFor=<<task> loaded | started | ready | exited>`

  Waits for `<task>` to achieve at least the selected status before starting this task. Status values are ordered: loaded, started, ready, and finally exited. Therefore, if a `WaitFor= OtherTask` started is given and `OtherTask` is currently ready, then this task can proceed. This is optional.

  The status of `ready` must be indicated by inserting the following code into the source for the task at the appropriate point:

  ```
  Task.setStatus(Task.STATUS_READY);
  ```

The `.ini` file can also contain a `[Properties]` section that is used to add to or replace system properties. Any assignment in the `[Properties]` section is reflected in system properties of the initial JVM.

1 On NT platform, set Background=true if you set Run=newjvm.

Example .ini File

These are the contents of a typical .ini file. This example starts the Core in the current JVM. After the Core is ready, it starts a TS server in a new JVM, discarding both normal and error output. When the server is ready, it starts a Client using the class tests.apitests.TC, also to be run in a new JVM. When the Client task has finished, all tasks as well as the main JVM terminates.

```
[Tasks]
Start=<Core>,<Server>,<Client>
[Core]
Class=net.espeak.infra.core.startup.TestCore
Args=12345
[Server]
Class=tests.apitests.TS
Run=newjvm
LogOut=null
LogErr=null
WaitFor=Core Ready
[Client]
Class=tests.apitests.TC
WaitFor=Server ready
OnExit=closeall
```

Appendix B 'espeak' Utility

The *espeak*[1] utility, available under `<installDir>/bin`, is provided to help you start the e-speak Core and other components. It is available as a perl5 script, `espeak.pl`, on all supported platforms. An executable, `espeak.exe` on NT, is provided for convenience. Linux and HP-UX users must have perl 5.003 or later to run *espeak* utility.

NOTE: The default version which comes with standard HP-UX installation is Perl4 and not Perl5.

While the *espeak* program can be used to start basic e-speak components, it can also be used to load user-defined `.ini` files. See Appendix A, "SysLoader Utility" to learn how to write e-speak `.ini` files. The following section describes the syntax and usage of the *espeak* program.

Help Page for *espeak*

You can run the help page by typing `espeak -h`. The following output appears:

```
Usage: Run E-speak component(s).
Syntax:
    espeak [-v]
    espeak [help | -h | -help | /?]
    espeak [-c] [-j(opt) <JVM arguments>] [-r <repository specs>]
            [esport=<port number>] [<name>=<value> ...]
            [-i inifile | <E-speak component <args>> ...]
```

1 Experimental utility espeak3 is also available under <installDir>/bin directory, which allows you to start advanced e-speak components.

```
Details:
  help, -h, -help, /?        Print this screen.
  -d, -debug                 Sets debu mode ON
  -j, -jopt <JVM Arguments>  Extra arguments to your JVM.
                             Quotes neede if contains spaces.
  -r <repository specs>Complete path to the file containing backend database
                           specs. Typically, repository.ini
  -i <ini file>User supplied Ini file.
  -v                          Print e-speak version number and exit.
  <esport=<port number>TCP Port number where e-speak Core is started.
  <name>=<value>Assign 'value' to JVM Property 'name'. Typically used it user
                           supplied ini file which contains JVM properties
                           (specified by %<name>%)
  <E-speak components>One or more E-speak components and their arguments
                           These are ignored if -i is specified. Following
                           are currently valid E-speak components and their
                           arguments. By default Core, AdvertisingService,
                           CoreDistributor, ServiceDistributor &
                           ManagementDistributor are started.

Arguments that can be specified with any of the components:
run=newjvm|injvm|on <host>Optional. This will start the component in a new
                           JVM, or same JVM or on another host.Default
                           injvm

Valid E-speak Component(s) and their respective arguments :
--------------------------------------------------------------------------------
  c | Core
--------------------------------------------------------------------------------
 p=<Protocol info>Is either TCP:<tcpport> or IVM:ivmpop1
 user=<user>User name to access backend DB
 passowrd=<Password>Password to access backend DB
 load=<plugin class>Load a plugin class.
 rep=<Repository specs>Full path to repository specification file
 restartRestart the core purging the repository.

--------------------------------------------------------------------------------
    ads | AdvertisingService
--------------------------------------------------------------------------------
 eshost=<host>Host where e-speak core is started
```

```
 esport=<port>port number where core is started
 esurl=<es-url>Alternative to eshost:esport
 beconfig=<backend-config-file>Optional.
 beproto=<backend-protocol>Optional, one of ldap or slp
 behost=<backend-host-name>Mandatory if beproto
 beport=<backend-port-number>Mandatory if beproto
 copath=<path to co>Path to Connection object file.
 myco=<co file>Name of Connection object file.
 rootdn=<backend-root-distinguished-name>Mandatory if beproto or beconfig
 passwd=<backend-password>Mandatory if beproto or beconfig
 base=<backend-base>Mandatory if beproto or beconfig
 mport=<multicast-port>Mandatory if no backend info.    (default 1438)
 group=<group-name>Mandatory if no backend info      (default: current host
                                          name)
 connconfig=<connection-config-file-name> Optional.
 cp=<communication-protocol>Optional.
 sp=<session-protocol>Optional.
 maxconntime=<max-ldap-connection-time>Optional.
 proxyhost=<web-proxy-hostname>Optional.
 proxyport=<web-proxy-portnumber>Optional.

-------------------------------------------------------------------------
   cd | CoreDistributor, sd | ServiceDistributor, md | ManagementDistributor
-------------------------------------------------------------------------
 CORE_PORT=<port>Port number where core was started.
 CORE_HOST=<host>Host name where the core was started.
 CORE_URL=<URL>CORE_HOST:CORE_PORT instead of specifying CORE_PORT &
                            CORE_HOST
Environment Variables:
 Following Environment variables should be defined.
   ESPEAK_HOMELocation where E-speak is installed.
   JREFull path to your JVM executable file.
   VJ++Set to true if using Visual J++ (JRE=<path>\jview)

Examples:
o To run the default E-speak components
  espeak

o To run only Core at TCP port 12390.
  espeak C p=TCP:12390
```

```
o Run Core and Advertising Service at specified group.
  espeak C Ads group=myPrivateGroup copath=/opt/espeak/config myco=co1

Note:
  In order to terminate the program, you must hit <CTRL>C

Caution:
  All JVM's that get started do not always get terminated, particularly on
  Windows NT. Please kill those JVM processes, else the port numbers remain
  occupied and you hit exceptions when starting the services again.
```

Appendix C Introduction to PSE Manager

This chapter introduces and shows the use of a PSE Manager tool. You need few prerequisites to read this – however, you do need to be aware that e-speak security is an example of a Public Key Infrastructure (PKI). The public key approach involves each participant having at least one pair of keys, one private that you keep to yourself and must never divulge and the other key is public which can be widely known. Typically, a private key is used by you to digitally sign data, so that others then use the corresponding public key to check this signature.

You should not need a deep background in cryptography or any knowledge of its theory in order to make sense of this chapter. However, a thumbnail sketch of the security model goes as follows:

- Everyone (and everything) has a set of public/private keys. Entities are distributed and interact with one another by means of secure sessions using the SLS protocol – this includes firewall traversal technology. All entities can both *use* services offered by others and also *provide* services to others. This means that all parties in secure sessions have to be authenticated to each other. In particular, SLS secure sessions authenticate both parties involved by using challenge-response negotiations based on public-key cryptography.
- Access control to services is done by exchanging digitally signed certificates as a part of the SLS protocol providing secure sessions. These certificates act like “tickets” that grant entities with authorization to access and make use of services. Certificates are signed by issuing entities (or Principals) and are issued to subject principals who can use them. These certificates can also be chained together (via *delegation*) to give composite authorizations.

See the J-ESI documentation and the E-speak Architecture Specification for further details concerning the security model.

Private Secure Environment

A Private Secure Environment (or PSE) is a somewhat grand name for a keystore[1]. Its function is to provide a secure store containing *labelled public/private key pairs.*

Keys are used to identify particular roles held by a particular E-speak entity. By using these keys to sign certificates, an entity can prove possession of the key-pair and thus authenticate the certificate i.e. showing that who originated it and that it was not modified. Clearly, e-speak entities need at least one key-pair and in general can have more than one, each with their own key-pair. The PSE provides a secure means of storing this mapping from labels to key-pairs.

As mentioned above, the other main function of a PSE is to use key-pairs for digitally signing certificates[2]. The idea is that the private key in some key pair is used to construct signatures of certificates (i.e. signing), and the key-pair's public key is used to verify these signatures.

Accordingly, any entity needing to verify a signature naturally needs the corresponding public key. Fortunately, the particular certificate format used by e-speak generally includes the public-key of the issuer signing the certificate, as well as that of the subject.

A general design goal of the PSE is to prevent unnecessary exposure of the private part of a key pair. For this reason, there is no access method provided by the PSE that *directly* exposes private keys as data. All access to private key data is deliberately made indirect and encapsulated. Thus the PSE's API only needs to provide a way to sign particular certificates, via some labelled key-pair within the current PSE.

1 The Private Secure Environment was originally envisaged as a secure data base containing private/public keys, certificates, trust assumptions and policy-decision support. However, it became evident that a minimalist approach leads to a less complex and more maintainable design.

2 Just like in real life, it is important to understand what is signed on your behalf. Do not digitally sign arbitrary pieces of data. Fortunately, e-speak certificates are sufficiently structured and declarative that their meaning is sufficiently well-defined and unambiguous. Even so, any certificates that an entity signs represents authorisations for other entities to use or to do something with – if you don't want to authorise that something, then don't sign it's certificate.

The PSE itself can be stored as a binary file[3] in your local file system. This data is encrypted and a passphrase is required to lock/unlock the data it contains.

Delegation and Trust Assumptions

All e-speak entities can potentially be consumers (or users) of certificates and simultaneously, providers (or issuers) of certificates. For this reason, the security model needs to be symmetric and to treat parties uniformly.

Certificates can be *delegated* – this means that a sequence of delegated certificates can be processed and combined together (i.e. chaining) to produce a resulting authorization supported by all of the contributing certificates. The validation of a chain of delegated certificates must be rooted in something that the *verifying* entity trusts implicitly. This implies that all entities must have their own set of trusted certificates within which all security verifications that they authorize are *grounded*. This set of particularly trusted are known as Trust Assumptions. Each verifying entity selects or defines their own set of Trust Assumption certificates for this express purpose.

Because each verifier's Trust Assumption certificates represent the "roots" of secure access control, their entire function is to base all authorization decisions derived from them. For this reason, in each Trust Assumption, the delegation flag is always assumed to be true and, moreover, the validity period they contain is *ignored*. In other words, any available signed certificate can be used as a Trust Assumption, irrespective of the values of the delegation flag or the validity period it has.

This has the advantage that any certificate specifically created for use as a Trust Assumption certificate could have delegation set to false and have an ineffective validity period e.g. set the not-after date to be some date-time in the very near future, which expires very soon after it is signed. This means that although the certificate could be signed and then used as a Trust Assumption by its creator, it cannot be effectively used as a certificate by anyone else.

The PSE does not need to provide special support per se for Trust Assumptions – they are after all just a particular collection of (signed) certificates.

3 The typical, default name for a PSE data file is securestore.txt.

The PSE Manager

The PSE Manager is a GUI tool for setting up and managing PSE's and also text files containing certificates. The main tasks that this tool supports are:

- Creation of PSE's, storing and retrieving them to/from binary files.
- Passphrase management for PSE's.
- Generating key-pairs and assigning labels to them.
- Creating and editing individual certificates.
- Signing and validating individual certificates.
- Saving and retrieving collections of certificates to/from text files.

The main purpose served by this tool is to provide a bootstrap that enables PSE's and certificates to be deployed and created. Particular end-user services implemented using e-speak security needs to deploy their own security model tailored to their own needs and requirements. It is envisaged that there could be many tools such as this one – this is but one example of such a tool. There is an e-speak API for managing and manipulating PSE's in a programmatic manner.

The basic security model requires that all users of e-speak enabled services need to have their own PSE to store their own keys, plus their own collections of service-specific certificates issued by those services against some of the user's own public-keys. These certificates do not need to be kept secret (although the user may want to do so for privacy reasons) and can be kept in a public place, even accessible by URL.

The following sections illustrate some of the ways that the PSE Manager can be used to manage PSE's and certificate files. It is not an exhaustive guide or manual – but then ToolTip help has also been provided for most GUI controls.

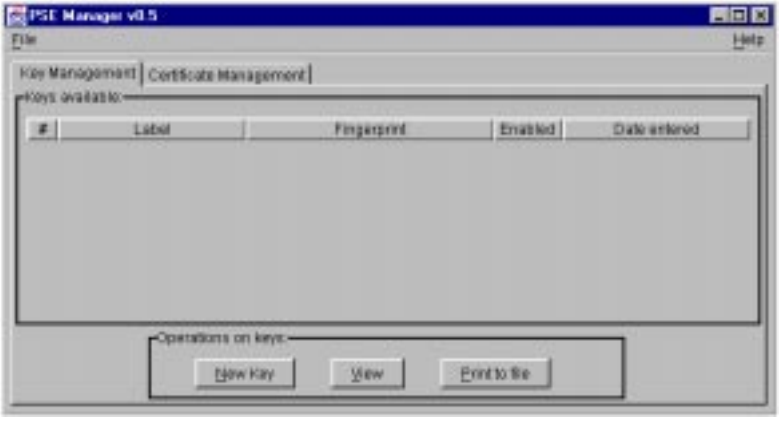
Starting the PSE Manager

You can invoke the PSE Manager using the following Java command line:

```
java net.espeak.security.pse.manager.Run
```

This can be invoked from a shell script.

After it is started, the PSE Manager consists of two panes – one for Key Management and the other for Certificate Management. Here, the Key Management pane is selected.



Creating a PSE

Using the PSE Manager to create a PSE is straightforward — the following steps show how to do this.

1. Make sure that the “Key Management” pane is selected – this is the default.
2. Select the File menu and then the New menu-item.
3. A File dialog box appears. Type the name of a new PSE file[4].

4 Naturally, choosing the name of an existing PSE is also permitted

4 After specifying a PSE filename, another dialog box appears. Specify a passphrase. This passphrase forms an encryption key that encodes the PSE data when stored. The effect of this encoding is to lock the PSE data making it only accessible to those knowing the passphrase.

When you are done with these steps. an empty PSE object is created and initially saved. Note that the application title bar now contains the name of the PSE object.

The next step is to generate and add keys.

Generating Key Pairs

After we have a PSE object, we can generate key pairs and add them into it. The following steps show how to do this.

1 As before, make sure the “Key Management” pane is selected.

2 Click the New Key button.
If this is the first time that key generation has been requested during this session, then there is a short delay while the key generation system is initialised. At the present time, only the current default type of key is generated[5].

3 After key generation has initialized, a dialog appears inviting you to type a new label for the next key. After typing your new label, press OK to generate the key, or Cancel to end key generation.

NOTE: All labels have to be unique. Key-pair labels are used within Certificate Management as a convenient way of selecting the issuer and subject Principals for certificates.

Furthermore, there are no methods provided for either exporting or importing key-pairs. Such a facility could potentially allow key-pairs to be isolated and analyses as data objects. Clearly, importing a key-pair from some other source has obvious security implications (e.g. the private part of an imported key pair can be known to others or in some other way compromised).

Saving a PSE

You can save a PSE to disk as follows:

1 As before, check that the “Key Management” pane is selected.

2 On the File menu, click Save or the Save As.

- **Save**: In this case, if there is some change to save, the PSE is saved with the current PSE filename using the current passphrase.
- **Save As**: In this case, a File dialog appears, allowing selection of a PSE filename to save the PSE to using the current passphrase.

With either the **Save** or **Save As** alternatives, a dialog appears for you to confirm saving the PSE using the current passphrase.

The current passphrase for the PSE can be changed using the Passphrase menu item from the File menu.

Numbered backup files are made (to a maximum depth of 10) whenever a PSE file is overwritten.

---

5 At present, the type of key-pair generated depends upon the current default public-key algorithm. In this release, this default algorithm is ElGamal. Future releases will add support for more algorithms, such as RSA.

Known problems with Key Management

Do not attempt to open another PSE after closing a PSE (or to reload the PSE you have just closed). Instead exit and restart the tool.

Certificates

There are two basic types of certificate: Attribute certificates and Name certificates. As already outlined, attribute certificates act like “tickets”, licensing the verifier to grant permission to particular authenticated entities for accessing some resource. Attribute certificates can be chained together via delegation to create indirect authorizations, often based upon roles that entities possess.

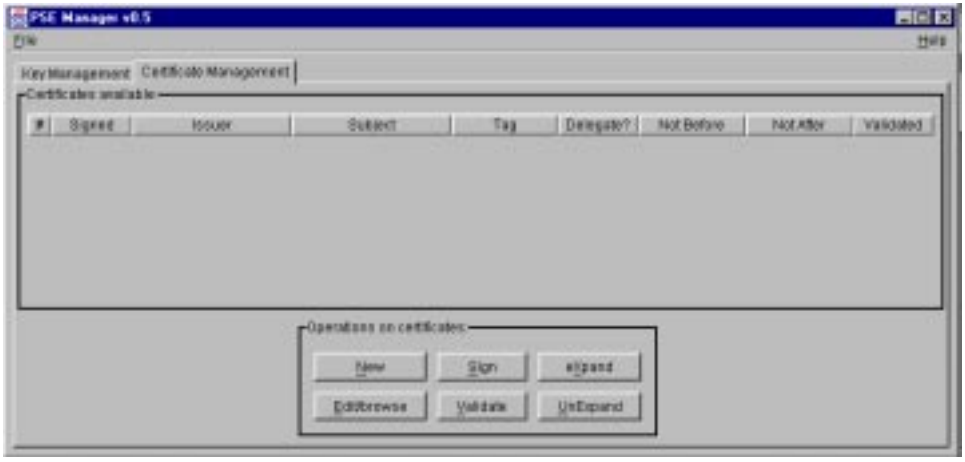
Name certificates are more specialized and essentially allow symbolic names to be associated with a Principal[6] – they act rather like rewrite rules, replacing symbolic names by particular Principals. Name certificates can be used to create *groups* of users, so that each member of the group inherits all the permissions granted to that group as a whole. With respect to authorizations, Name certificates behave in a similar way to Attribute certificates with delegation true and a default `allow everything` attribute (i.e. “*”).

Using the PSE Manager for Certificate Management

As noted earlier, the PSE is essentially a secure key-store, and does not itself contain certificate data. In order to use key-pairs held within a PSE for signing and verification purposes, the PSE Manager provides functions for loading, processing and saving certificate text files.

We can switch from managing keys to managing certificates within the PSE Manager by selecting the “Certificate Management” pane. This in turn configures the File menu to provide options that process certificate text files, instead of PSE data files.

6 The term “Principal” means the Public Key. You might find it helpful to think of it as the entity to whom the public-private key-pair belongs.



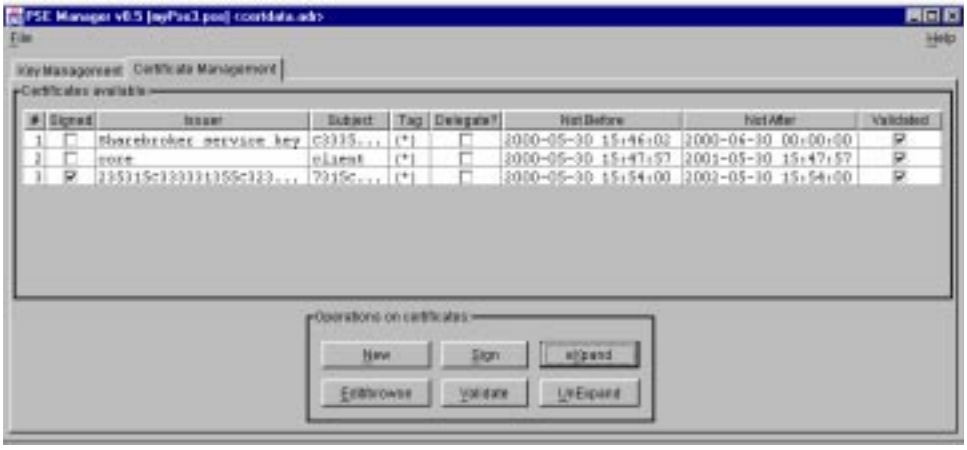
As can be seen, there are 6 certificate processing functions available:

1. **New** : Invokes the certificate editor to create a new certificate.
2. **Edit/browse** : Invokes the certificate editor on selected certificates.
3. **Sign** : This operation tries to sign selected certificates.
4. **Validate** : This operation checks that the selected certificates are well-formed, their validity period is meaningful and, if signed, checks that the signature is valid.
5. **Expand** : This expands any PSE key-labels used in either the issuer or subject of selected certificates.
6. **UnExpand** : (Inverse to **Expand**.) This matches any literal public-keys against entries in the current PSE and returns their labels.

Using PSE key-labels as symbolic Principals

The use of PSE key-pair labels provides a useful and convenient way of referring

to key-pairs defined in the current PSE as shown below:



This shows three certificates, one of which has been signed. All of these certificates are valid in the sense described above. Because PSE key-labels are purely locally bound names, they cannot occur within signed certificates – any such labels must be expanded to literal Principals (i.e. public keys). As a convenience, the signing operation attempts to expand any symbolic Principals (i.e. PSE key-labels) before actually signing the certificate, and fails if there are any symbolic Principals remaining.

Partial certificates can be constructed that contain arbitrary key-labels, not only those in the particular PSE that happens to be loaded at the time. In fact, certificate management does not require a PSE to be loaded at all – and by switching between key and certificate management, different PSE's can be loaded, thus conveniently allowing different sets of key-pairs and their labels to be used in building up a particular set of certificates.

Creating, Editing and Browsing Certificates

The Certificate editor pane is used for creating, editing and browsing certificates and is invoked from the **New** and **Edit/Browse** buttons. The **New** button invokes the editor on a fresh entry, which if accepted, is added to the end of the certificate list. The **Edit/Browse** button invokes the editor on all of the selected certificates.[7]

The Certificate editor consists of 4 panels:

1. Certificate type: Name or Attribute (Name and Attribute certificates are explained in the E-speak Architecture Specification Chapter on Access Control)

2. Specification of Issuer and Subject Principals, using symbolic Principals such as PSE key-labels, fingerprints of known keys and also arbitrary key-labels added manually.

3. Specification of the authorization attribute (or `tag') in the S-expression format as defined by SPKI. However, for e-speak enabled services, the authorization expression is generally of this simplified form[8]:

   ```
   (net.espeak.method (<interfaces>) (<methods>))
   ```

   In addition, the tag can be specified from Service Metadata descriptions (see "Service Metadata and tag files" on page 130 for more details). The delegation flag can also be specified.

4. Validity period: The period is specified by a Not Before date-time and a Not After date-time value. These UTC date-time specifications have the following form:

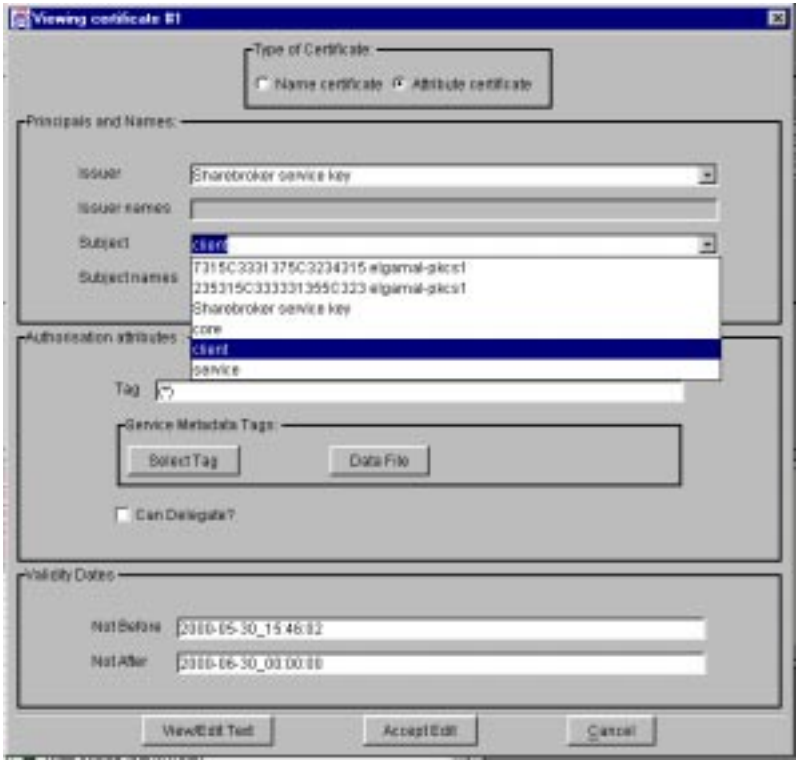
   ```
   yyyy-MM-dd_HH:mm:ss
   ```

   where `yyyy` = year, `MM` = month, `dd` = day, `HH` = hour, `mm` = minutes, `ss` = seconds.

7 The sole certificate in a certificate list of length one is implicitly selected.

8 The attributes/tags used by e-speak enabled services can be more complex than this – see the E-speak Architecture Specification Chapter on Access Control for full details. Additionally, the standard atom: `net.espeak.method` is generally suppressed when displaying standard tags.

The Certificate editor is shown below:



Know problems with certificate management

- Do not try to sign a certificate with the issuer field expanded. When the PSE manager signs a certificate it tries to expand the issuer field and fails if it is already expanded. Note that the subject field should only be expanded if the key is not available in the currently open PSE (as in the case of when Bob signs a certificate from Alice in “Certificates generated by Alice and Bob” on page 128).
- Expanding or collapsing a signed certificate destroys the signature on that certificate.

Using PSE Manager as an Attribute Certificate Issuer

We now give a short scenario to show how the features of a tool like PSE Manager can be used to issue certificates where two separate entities are involved. The objective of the scenario is to show how PSE Manager can facilitate communication of public key information from one entity to another.

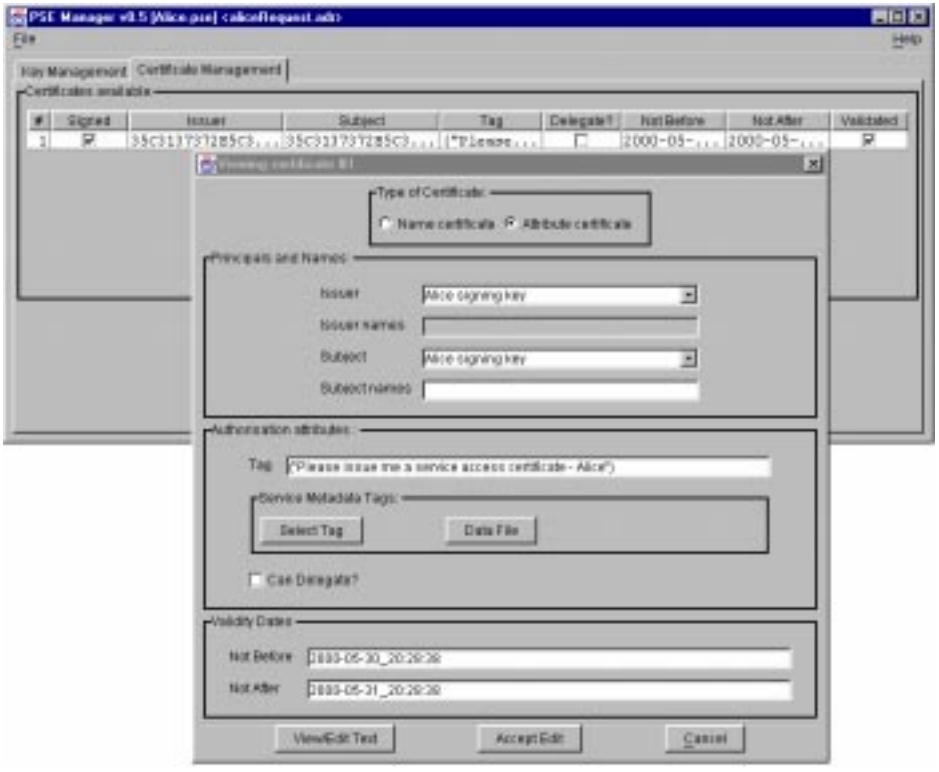
NOTE: The procedure discussed below is not secure on its own – it merely serves to show how public-key information can be exchanged using the features provided by PSE Manager.

Imagine that Alice wants to sign-up to the fabulously chic e-speak enabled service, Gadgets-R-Us, that is administered by Bob. However, there is a just one small problem – Alice needs a Service Access Certificate issued by Bob that uses one of Alice's public-keys as the certificates' subject Principal.

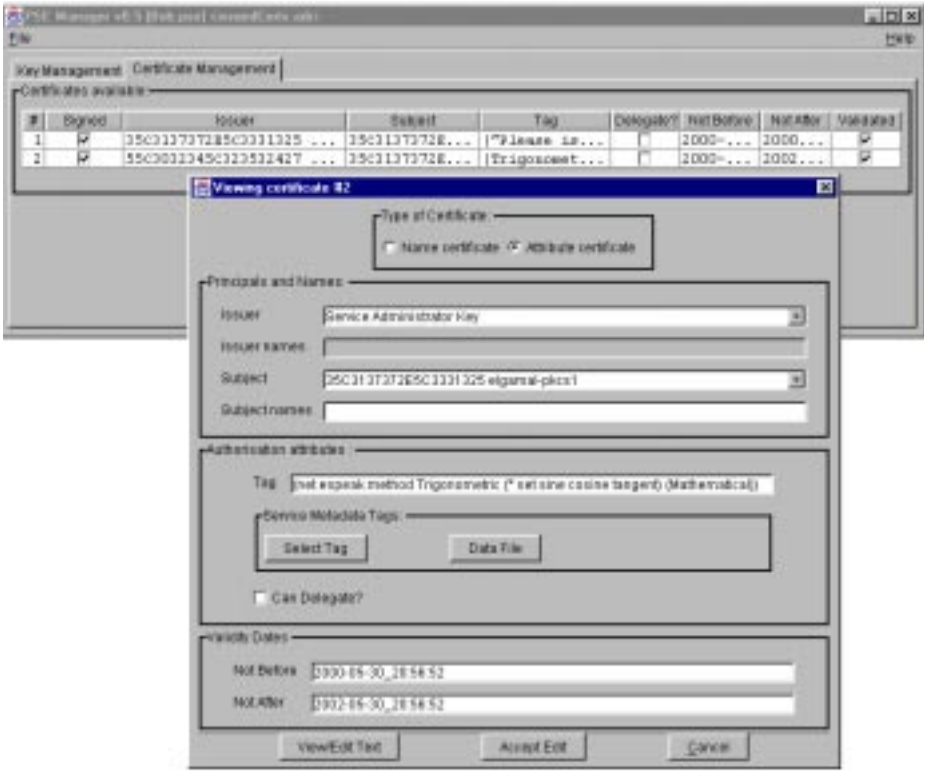
Fortunately, both Alice and Bob can each use PSE Manager with their own PSE's to achieve this happy situation. This dialogue just consists of two steps.

1. Alice uses PSE Manager, loaded with her own PSE, to construct a certificate and selects the label of one of her Principals for use as both the Issuer and Subject Principal. The other fields can be set arbitrarily, but validly. For example, the tag field can consist of a string requesting Bob to construct a service access certificate. The resulting certificate is signed by Alice, saved into a text file (called here aliceRequest.adr) and sent to Bob by, for example, conventional e-mail.

2. Bob now uses PSE Manager with his own PSE, and imports Alice's self-signed certificate as a text file edited out of his e-mail.



At this stage, Bob should consider whether the certificate really was sent by Alice, whoever she is. There are some checks that Bob can do here – such as check that the fingerprint associated with the Issuer and Subject Principal are identical. Bob could do some checks offline for himself, perhaps by direct contact with Alice. Also, Alice could have offered other additional authentication evidence that Bob could do some checks against. Anyhow, lets assume that Bob satisfies himself that Alice should be issued with a Service Access Certificate for Gadgets-R-Us.



Using the PSE Manager, Bob can construct a fresh attribute certificate containing Alice's Principal as Subject. Bob adds appropriate attributes, validity period, delegation rights and uses one of his own key-pairs as the Issuer. The resulting certificate is then signed by Bob, saved to a text file (called here `issuedCerts.adr`) and then sent back to Alice, again perhaps using conventional e-mail.

At the end of all this, Alice now possesses a Service Access Certificate that she can use to access Bob's Gadgets-R-Us service, by using the SLS secure session protocol provided within e-speak security.

Lightly edited versions of the text files, `aliceRequest.adr` and `issuedCerts.adr` produced by Alice and Bob can be found in the following section.

Certificates generated by Alice and Bob

This section contains lightly edited version of the text files, `aliceRequest.adr` and `issuedCerts.adr`, as generated by PSE Manager in the above example:

`aliceRequest.adr`

```
!!! Certificate Data File : C:\ … \aliceRequest.adr
!!! Written by PSE Manager v0.5 : $Revision: 1.1.2.2 $ $Date: 2000/05/12 14:12:24 $
!!! Dated 2000-05-30_20:32:58

((signed (cert
   (issuer (public-key elgamal-pkcs1 "\003\017v\245\235b\004 … \177.\312\357"))
   (subject (public-key elgamal-pkcs1 "\003\017v\245\235b\004 … \177.\312\357"))
   (tag ("Please issue me a service access certificate - Alice"))
   (not-before 2000-05-30_20:28:38)
   (not-after 2000-05-31_20:28:38)
   ) (signature (hash SHA-1 "a\210i\330\263o\303\336j;q\000\037 …")
               (public-key elgamal-pkcs1 "\003\017v\245\235b\004 … \177.\312\357")
                    "\003\017l\314\254\227 …"))


)

!!! Summary:
!!! ========
!!! Signed objects      = 1
!!! Unsigned objects    = 0
!!! Name Certificates   = 0
!!! Attr. Certificates  = 0
```

`issuedCerts.adr`

```
!!! Certificate Data File : C:\ … \issuedCerts.adr
!!! Written by PSE Manager v0.5 : $Revision: 1.1.2.2 $ $Date: 2000/05/12 14:12:24 $
!!! Dated 2000-05-30_21:00:21


((signed (cert
   (issuer (public-key elgamal-pkcs1 "\003\017v\245\235b\004 … \177.\312\357"))
   (subject (public-key elgamal-pkcs1 "\003\017v\245\235b\004 … \177.\312\357"))
   (tag ("Please issue me a service access certificate - Alice"))
   (not-before 2000-05-30_20:28:38)
```

```
   (not-after 2000-05-31_20:28:38)
   ) (signature (hash SHA-1 "a\210i\330\263o\303\336j;q\000\037 …")
               (public-key elgamal-pkcs1 "\003\017v\245\235b\004 … \177.\312\357")
                    "\003\017l\314\254\227 …"))


(signed (cert
   (issuer (public-key elgamal-pkcs1 "\003\017v\245\235b\004 … E\024\252B{ [e"))
   (subject (public-key elgamal-pkcs1 "\003\017v\245\235b\004 … \177.\312\357"))
   (tag (net.espeak.method Trigonometric (* set sine cosine tangent)
(Mathematical)))
   (not-before 2000-05-30_20:56:52)
   (not-after 2002-05-30_20:56:52)
   ) (signature (hash SHA-1 " \b\253\225\037W\"0\351\352 …")
                (public-key elgamal-pkcs1 "\003\017v\245\235b\004 … E\024\252B{
[e")
                   "\003\017PZ\031\235g!a …"))

)

!!! Summary:
!!! ========
!!! Signed objects      = 2
!!! Unsigned objects    = 0
!!! Name Certificates   = 0
!!! Attr. Certificates  = 0
```

Service Metadata and tag files

The Certificate editor permits Authorisation Attributes (or `tags') to either be edited textually or to be structurally edited using schematic data specified using Service Metadata tag files.

The purpose of these Service Metadata text files is to simply specify the interfaces that a particular service provides to its users and also the methods that each of these interfaces provide. Here is a simple example of a tag file (called `serviceMetadata.dat`):

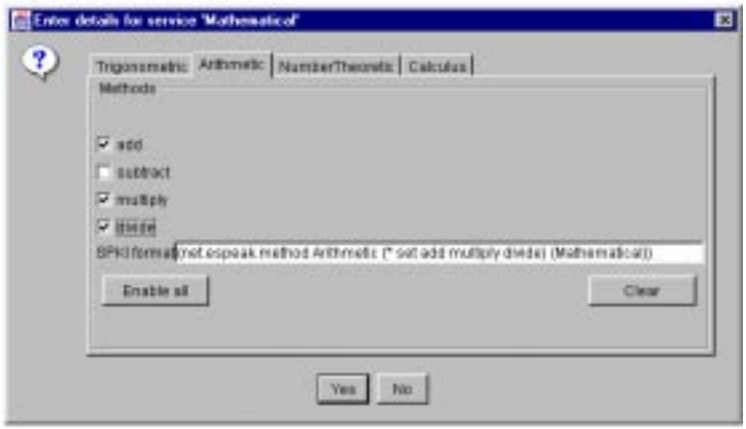
```
SERVICENAMEMathematical
Arithmeticadd:subtract:multiply:divide
Calculusdifferentiate:integrate
Trigonometricsine:cosine:tangent:secant:cosec:cotangent
NumberTheoretichcf:lcd:nextPrime
```

NOTE: SERVICENAME is what is called the serviceID in the E-speak Architecture Specification and J-ESI Programmer's Guide.

The first line specifies the serviceID (i.e. `Mathematical`), and subsequent lines specify particular interfaces (i.e. `Arithmetic, Calculus, Trigonometric, NumberTheoretic`). The remainder of each line specifies the names of public methods that each interface supports, separated by colons. So, the above specifies that the `Calculus` interface can support the two methods:

`differentiate and integrate`

This metadata is used to derive a simple structural GUI dialog which eases the entry of structured tag data. For example, using the tag file `serviceMetadata.dat` defined above, here is an example use of the structural dialog that is generated:



By clicking the Yes button, the following tag is created:

```
(net.espeak.method Arithmetic (* set add multiply divide) (Mathematical))
```

Compound tag permissions for several interfaces can be generated by visiting the tab pane corresponding to each interface required, and then selecting the particular methods required. The resulting composite tag expression is constructed from the selections made on each interface.

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W