Documentation

Search All 1.x Documentation

Welcome to the CleanSpeak 1.x Documentation home. If you're looking for documentation for another version, please visit the CleanSpeak Documentation Home.

Technical Guide

Welcome to the CleanSpeak 1.x technical documentation. You can use the table of contents on the left to navigate to the page you want to read, or use the links at the top and bottom to traverse through the documentation.

Overview

This chapter gives you a brief overview of the layout of the Clean Speak™ system and what is provided to you. It also tells you things you might need to build and how the various integrations work.

This diagram gives you a general description of a the possible ways that Clean Speak™ is setup and integrated with your applications. As you can see near the top are various applications that you want to integrate with Clean Speak™. These applications are using the Clean Speak™ WebService for either filtering or moderation (depending on which products you need and have licensed). These applications communicate with the WebService via XML over HTTP.

The WebService loads and stores data from a relational database using SQL. This database is setup and configured by following the instructions in the Installation section of these docs. If you are using the Real-Time Filter product, the WebService will load the white and black lists it uses to filter from the database. If you are using the Moderator product, the items you send to the WebService to be moderated are stored in the database.

An optional integration method is to install the Java API inside your application. The server on the right side of the diagram contains the Java API inside it. This integration method is also covered in these docs.

The Management Interface is the web application that is used to update the white and black lists used by the Real-Time Filter as well as to moderate content sent to the Moderator. This interface runs separately from the WebService and communicates with the database to perform these tasks.

You'll notice that both the Management Interface and the WebService use the same database. This allows the Management Interface to make changes to the white and black lists and then for the WebService to query the database for those changes. This process occurs using a feature we call "Notifications". The Management Interface can send out notifications to the WebService or any of your servers to tell them to reload their data from the database. These are simple messages sent from the Management Interface over HTTP.

In the Java API integration part of this diagram, you'll notice there is a box named Listener. This is a component that you might need to write if you want your application to be able to reload the receive notifications from the Management Interface and reload the Java API. Some general guidelines and examples of writing a notification service are provided in the Java API section of these docs.

The last piece of this diagram is the HTTP moderation notifications. These are messages that are sent by the Management Interface back into your applications after content has been moderated using the Moderator product. These messages are XML sent over HTTP and contain information about the moderation results. If you are using the Moderator product, you will need to write a service that can accept these messages and update your records accordingly.

Next Steps

In the next chapters we will cover all of the system and software requirements for Clean Speak™, detailed installation instructions, configuration, how to run Clean Speak™, and how to use Clean Speak™.

Important Notices

Version 1.0

For this version the two WebService applications have been collapsed into a single application and renamed to the Clean Speak™ WebService. This has major impacts for existing customers because it changes the directory structure of the installation. It also makes an upgrade from any previous version to this version impossible for the WebService applications. Upgrades after this version will return to normal.

If you are running an old version of either of the WebService applications, you will need to follow these steps to upgrade to the new single WebService:

1. Install the new WebService application
2. Copy the server.xml file from the old installation to the new installation
3. Copy the init.d-overrides.sh (if you have this file) from the old installation to the new installation
4. (Optional) Copy the log files from the old installation to the new installation
5. Un-install the old installation

Additionally, if you are attempting to upgrade using the RPM bundle, you will receive an error. This is because RPM doesn't understand the RC versioning scheme that Clean Speak has been using and therefore believes that the RC versions are newer than the 1.0 version. Therefore, you will need to un-install the older version of the Management Interface and re-install it.

If you need any assistance with this upgrade, please contact Inversoft for support.

Release Notes

Version 1.3.3

- Removed "All" option on whitelist index page (if you clicked the "All" option in the pagination on the whitelist of Global, it would crash Clean Speak since there are 140,000+ words on Global's whitelist)
- Changed the "Variations" field on the blacklist entry add/edit form to be a textarea
- Fixed a bug with the Import feature where the definition wasn't being imported properly
- Fixed a bug with the Export feature where if the definition was null it would put the word "null" into the exported XML. It now outputs an empty String in the CDATA block instead

Version 1.3.2

- Fixed issue in JSON response from the WebService where a comma was missing

Version 1.3.1

- Added initial conjugation support for Italian, German and Polish
- Fixed minor issues in French and Spanish conjugations

Version 1.3

- Added initial conjugation support for French and Spanish
- Added locale to WebService response XML and JSON messages
- Added import ability to the management interface for importing XML files
- Fixed an approval bug that prevented "Revert All" from working properly
- Minor bug fixes to the display, sorting, etc.

Version 1.2.5

- Minor display issues in the Management Interface
- Issues with HTML characters on forms in the Management Interface

- Changed URLOptions to use domains as the key of the quality Map
- Pagination issues in some Management Interface pages
- Added support for Unicode characters (accented letters) as replacements
- Moderation notification failures when batch size is set
- Better separator handling (s....h...i...t)
- Ability to add leet speak to the black list (@$$)
- Changed handling of top-level domains that are dictionary words (net, it, etc) to produce better quality scores in the URL filter
- Fixed issue with URL filter where it wasn't finding URLs such as com.com and net.net
- Removed slashes from being replacements for i (/ != i)
- Fixed issue where the URL filter wasn't finding URLs that had a period after them (www.example.com.)

Version 1.2.4

- Fixed database connection issues with the WebService and the moderation system (this is a high priority update)

Version 1.2.3

- Fixed problem with moderation system displaying items multiple times in the moderation queue if the item had multiple attributes

Version 1.2.2

- Updated the email filter to use the URL filter handling
- Updated the XML schema for the WebService (this is deprecated but needed updates)

Version 1.2.1

- Fixed bugs with the URL filter
- Fixed issue in the Management Interface with some browsers handling non ASCII characters. This allows full unicode support via the Management Interface for non-English languages

Version 1.2

- Added the new REST interface to the WebService and deprecated the old XML interface
- Added exact match feature to the filter
- Fixed the phone number filter so it can be accessed from the WebService
- Fixed the URL filter so it can be accessed from the WebService
- Updated and re-organized the menus inside the Clean Speak™ Management Interface
- Fixed various customer reported filter issues
- Initial preview release of the Clean Speak™ Monitor is now available

Version 1.1

- Added phone number filter
- Added help messages to blacklist form for each field
- Added single test page to verifications
- Added count to moderation queue
- Added new search criteria for the audit logs including user and date
- Added moderation search to assist in finding items
- Added a new WebService to manage user accounts for the Management Interface
- Added an upgrade notice to the Management Interface
- Added authentication support to the WebService that is configurable in the Management Interface
- Added dates and other criteria to the moderation archive search
- Fixed search forms to ignore case and provide better results
- Fixed pagination to be simpler to use for large lists
- Fixed Real-Time Filter issue where the WebService was filtering all types regardless of the request
- Fixed Real-Time Filter response XML schema to work with XJC
- Fixed blacklist form conjugation preview so that it isn't automatic
- Fixed various Real-Time Filter bugs to increase accuracy

Version 1.0.3

- Added better support for sorting in verification cases

Version 1.0.2

- Fixed problem with shutdown of the WebService
- Fixed problem in Management Interface with search returning incorrect results
- Fixed problem in Management Interface with errors when clicking links in search results

Version 1.0.1

- Fixed problem with license files and the Monitor product

Version 1.0

- Collapsed the two WebService applications into a single application (see Important Notices for details)
- Added the ability to filter URLs using the Real-Time Filter
- Reduced memory footprint of the Real-Time Filter
- Fixed various bugs with multiple separators (s......h__________i________t)
- Changed the URI for the /monitor action in the WebService to /metrics so that it doesn't conflict with the new Monitor Product

Requirements

There are a number of requirements to use Clean Speak™. There are also a number of technologies that are required depending on how you want to use Clean Speak™. Installation of all of these components (except the operating system) are covered in the Installation chapter. The required technologies are:

Operating System

Clean Speak is capable of running on most modern operating systems. Below is a list of the supported operating systems.

- Fedora 10 or higher
- Redhat ES 5 or higher
- CentOS 5.2 or higher
- Debian 5 or higher
- Ubuntu 8 or higher
- Windows XP or higher
- Windows Server 2003 or higher
- Solaris 10 or higher
- Open Solaris 2008.05 or higher
- Mac OS X 10.5.4 or higher (64-bit Intel only)
- Mac OS X-Server 10.5.4 or higher (64-bit Intel only)

Many other operating systems are capable of running the Clean Speak™, but are not officially supported. Any operating system capable of running Java version 1.6 or higher should be capable of running Clean Speak™.

Java

You must have Java installed to use Clean Speak™. Here are a list of the acceptable versions of Java.

- JDK 6.0 or higher from Sun
- OpenJDK 6.0 or higher
- JDK 6.0 or higher from Apple

JEE Server for WebService and Management Interface

In order to use the Management Interface or WebService, you must have a JEE server installed. The officially supported JEE server is:

- Bundled Tomcat version 6.0.20

The Clean Speak™ Management Interface and WebService are capable of running on any JEE-compliant web server. Inversoft only officially supports the bundled Tomcat releases. If you are interested in using another JEE web server, Inversoft recommends Oracle's WebLogic or Sun's GlassFish. However, we do not support these web servers.

NOTE: Versions of Tomcat that are available for Red Hat and Debian systems via Yum or Aptitude are not valid Tomcat releases. Both of these distributions have made changes to Tomcat that are non-standard. For this reason, Inversoft does not support these versions of Tomcat.

Database

There are a couple of different options for the database setup to use with Clean Speak™. Below are the types of databases that are currently supported.

- No database
- MySQL 5.0 or higher
- PostgreSQL 8.4 or higher

If you don't intend to use the Management Interface and want to only use the Clean Speak™ Real-Time Filter (either via the Java API or the WebService) using the XML database file and not a relational database then you don't need a relational database installed. If you are using the Management Interface, Moderator, or Real-Time Filter using a relational database, then you will need one of the supported relational databases installed.

Installation

This chapter provides instructions on installing all the necessary software required to run Clean Speak™.

In most cases you will need to install the Management Interface and the WebService to use Clean Speak™. The installation instruction for those components are:

- WebService
- Management Interface

Regardless of which product you are installing, you will need to minimally install Java and potentially install a database first. The documents that cover the installation of Java and a database are:

- Java
- Database

After you have installed Java, optionally a database, and the product(s), you might also need to install the MySQL connector. Here is the document that covers the installation of the MySQL connector:

- MySQL Connector

Lastly, you will need to install your license file. You can generate a license file from the My Account section of the website. Save this file and then follow the instructions in this document to install it:

- License File

Java

Installation - Java

The first thing you need to install to run Clean Speak™ is Java 6.0 or higher. Some operating systems, such as Linux, make installing Java simple using the package management system that is part of the operating system. Other operating systems, such as Mac OS X, come with Java pre-installed. For all other operating systems you will need to download Java from Sun/Oracle here:

http://java.sun.com/javase/downloads/

Java can be installed anywhere on your system as long as it is accessible and executable by the user that will be running Clean Speak™.

Windows

Depending on the method you use to run the WebService and Management Interface, you might need to set an environment variable name **JAVA_HOME** and set its value to the location you installed Java (usually something like c:\program files\java\jdk1.6.0_24)

On some Windows systems there is a problem with the way that Java is installed and the files that Tomcat requires. Copy the file msvcr71.dll from the Java bin directory (commonly located at c:\program files\java\jdk1.6.0_24\bin) to the c:\windows\system32 directory (different versions of Windows might have different paths to the system library directory and you should consult the documentation for your version of Windows to determine where the system library directory is). If the Windows system library directory already contains the file, you do not need to copy it over.

Debian

If you are running a Debian system, including Ubuntu, you can install Java using Aptitude by executing this command:

```
$ sudo apt-get install openjdk-6-jdk
```

Redhat

If you are running a Red Hat system, including Fedora, Red Hat EL, or CentOS, you can install Java using Yum by executing this command:

```
$ sudo yum install java-1.6.0-openjdk
```

Mac OS X

The Apple OS X operating system includes Java version 1.6.0. This version of Java is located in the directory:

```
/System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home
```

Database

Installation - Database

To use the Clean Speak™ Management Interface, Moderator, or Real-Time Filter using a database you will need to setup a database that will store all of the data used by Clean Speak™. Inversoft provides the necessary files for the MySQL and PostgreSQL relational databases.

First, you need to install MySQL version 5.0 or later or PostgreSQL version 8.0 or later (we recommend version 8.3 or 8.4). The database can be installed anywhere you want. Many operating systems, such as Linux, make installed MySQL or PostgreSQL simple using the package management system that is part of the operating system. Other operating systems require that you download and install one of these databases. Here are the location of MySQL and PostgreSQL:

- http://www.postgresql.org/download/
- http://www.mysql.com/downloads/

Debian

If you are running a Debian system, including Ubuntu, you can install MySQL or PostgreSQL using Aptitude by executing one of these commands:

```
# For MySQL
$ sudo apt-get install mysql-server

# For PostgreSQL
$ sudo apt-get install postgresql
```

Redhat

If you are running a Red Hat system, including Fedora, Red Hat EL, or CentOS, you can install MySQL or PostgreSQL using Yum by executing one of these commands:

```
# For MySQL
$ sudo yum install mysql-server mysql

# For PostgreSQL
$ sudo yum install postgresql-server postgresql
```

Database Creation

If you have not already created the Clean Speak™ database instance and tables, you will need to do that now. If you have already created the database, you can skip this section.

To create a new database, execute one of these commands:

```
# For MySQL
$ mysql --default-character-set=utf8 -u<username> -e "create database cleanspeak character set = 'utf8'
collate = 'utf8_bin';"

# For PostgreSQL
$ psql -U<username> -c "CREATE DATABASE cleanspeak ENCODING 'UTF-8' LC_CTYPE 'en_US.UTF-8' LC_COLLATE
'en_US.UTF-8' TEMPLATE template0"
```

The username must be either the root user or a user that has privileges to create databases. For MySQL, this is generally a user named 'root'. For PostgreSQL, this is generally a user named 'postgres'.

You will also want to create a user that only has access to this database in order to ensure proper security. Create this user by executing these commands:

```
# For MySQL

$ mysql --default-character-set=utf8 -u<username> -e "grant all on cleanspeak.* to 'cleanspeak'@'localhost'
identified by '<password>'" cleanspeak

# For PostgreSQL
$ psql -U<username> -c "CREATE ROLE cleanspeak WITH LOGIN PASSWORD '<password>'; GRANT ALL PRIVILEGES ON
DATABASE cleanspeak TO cleanspeak; ALTER DATABASE cleanspeak OWNER TO cleanspeak;"
```

By default, the application is configured to connect to the database named cleanspeak on localhost with the user name cleanspeak and the password cleanspeak. For development and testing, you can use these defaults; however, we recommend a separate database server and a more secure password for production systems.

Table Creation

Once you have created the database, you need to create the necessary database tables and load the initial data into the database. To create the tables you will need to download the cleanspeak-database-schema ZIP file. This contains the database script that creates the necessary tables. Once you have downloaded this file and extracted it, execute this command:

```
# For MySQL
$ mysql --default-character-set=utf8 -u<username> cleanspeak < cleanspeak-mysql.sql

# For PostgreSQL
$ psql -U<username> cleanspeak < cleanspeak-postgresql.sql
```

Real-Time Filter Data

In order for the Clean Speak™ Real-Time Filter to work properly, you need to insert the initial data into the database. This data includes the profanity and other words and phrases that the filter uses. In order to load this data, you must first download the cleanspeak-database-real-time-filter ZIP file. Once you have downloaded this file and extracted it, execute this command:

```
# For MySQL
$ mysql --default-character-set=utf8 -u<username> cleanspeak < cleanspeak-real-time-filter.sql

# For PostgreSQL
$ psql -U<username> cleanspeak < cleanspeak-real-time-filter.sql
```

MySQL BLOB Configuration

If you are using the Clean Speak™ Monitor product and MySQL you will need to ensure that your MySQL server is configured properly to allow large BLOBs to be sent to the database. This configuration parameter is set in my.cnf configuration file and is called**max_allowed_packet**. This setting should be set to the maximum allowed value of 1GB like this:

```
max_allowed_packet=1G
```

Java API

Installation - Java API

Download and unzip the Clean Speak™ Real-Time Filter archive somewhere on your hard drive. This bundle contains a single directory. Inside this directory there will be three sub-directories as follows:

```
CLEANSPEAK_HOME
      |
      |- lib
      |
      |- javadoc
      |
      |- samples
```

The lib directory contains the filter JAR file and all of the dependencies that are required in order to use the filter. The javadoc directory contains the JavaDoc documentation for the filter that can be used to help with integrating the filter into your application. The samples directory contains a few examples that illustrated how to use the filter.

Adding to Your Application

The last step required before you can use the Java API is to add the JAR files into your application's classpath. If you are building a JEE (formerly known as J2EE) web application, copy all of the JAR files from the lib directory into the WEB-INF/lib directory of your application. If you are building a JEE (formerly known as J2EE) enterprise application or EJB, copy all of the JAR files from the lib directory into your EAR or EJB classpath. Different JEE containers have different methods of adding JAR files to an enterprise application and EJB classpath. Consult your container's documentation for more information.

For other applications, you can either specify each JAR file from the lib directory as part of the applications classpath, or you can copy these JAR files to the correct directory for your application. Some frameworks provide a special directory that you add JAR files to. Consult the necessary documentation for your frameworks or application to determine where to place the Java API JAR files.

WebService

Installation - WebService

Before you install the WebService, you must first install a Java and a database. Here are the documents that cover the installation of each:

- Java
- Database

Bundles

There are a number of different ways to install the Clean Speak™ WebService depending on the operating system and the JEE web server you will be using. Here are the various bundles and the operating system and web servers it works with:

- ZIP file
    - cleanspeak-webservice-<version>.zip
    - This bundle can be used on any operating system, but is primarily for Windows
    - Includes Tomcat version 6.0.20
- Debian package
    - cleanspeak-webservice-<version>.deb
    - This bundle can be used on any operating system that supports the Debian package management system (dpkg)
    - Includes Tomcat version 6.0.20
- RPM package
    - cleanspeak-webservice-<version>.rpm
    - This bundle can be used on any operating system that supports the Redhat Package Management system (rpm)
    - Includes Tomcat version 6.0.20
- (Experts Only) Webapp only ZIP file
    - cleanspeak-webservice-webapp-<version>.zip
    - This bundle can be used on any operating system
    - Does not include a web server and requires advanced configuration and setup. Installation of this bundle is not covered in this document

Inversoft highly recommends using the Debian, RPM, or ZIP installation bundles. The other bundles are for advanced users and system administrators that feel comfortable with Java, Tomcat, and database setup and configuration.

Windows

To install on Windows XP or higher use the ZIP bundle. Extract the ZIP file anywhere on the file system. Remember where you extract the files. This location will be referred to as CLEANSPEAK_HOME. We suggest extracting this file to a directory such as **c:\Inversoft**on Windows.

If you want to run the WebService as a Windows Service you can install it as a Windows service using the service.bat script that ships with Tomcat. Here is how you execute that script from the command-line to install the WebService as a Windows service:

```
C:\Inversoft\cleanspeak-webservice\apache-tomcat-6.0.20\bin>service.bat install CleanSpeakWebService
```

The last parameter to this script is the name of the service. You can use any name as long as it doesn't contain any periods ('.'), underscores ('_') or spaces (' ').

Redhat

To install on a Red Hat system, including Fedora or CentOS, use the RPM bundle. Execute this command to install the RPM (replace <version> with the correct version number):

```
$ sudo rpm -i cleanspeak-webservice-<version>.rpm
```

This installation process places the Clean Speak™ WebService, including Tomcat 6.0.20, in the directory /usr/local/inversoft/cleanspeak-webservice. This location will be referred to as CLEANSPEAK_HOME. In addition to the files in this directory, the RPM also installs an init script named /etc/init.d/cleanspeak-webservice. This script will be used to start and stop the application once it has been configured.

Debian

To install on a Debian system, including Ubuntu, use the DEB bundle. Execute this command to install the DEB (replace <version> with the correct version number):

```
$ sudo dpkg -i cleanspeak-webservice-<version>.deb
```

This installation process places the Clean Speak™ WebService, including Tomcat 6.0.20, in the directory /usr/local/inversoft/cleanspeak-webservice. This location will be referred to CLEANSPEAK_HOME for the rest of this document. In addition to the files in this directory, the RPM also installs an init script named /etc/init.d/cleanspeak-webservice. This script will be used to start and stop the application once it has been configured.

Sudo

On Linux systems, the init script uses sudo to run Clean Speak as the cleanspeak user. Therefore, you must have sudo installed and must grant sudo privileges to the root user.

Next Steps

To complete the installation of the WebService, you will need to install a license file and optionally the MySQL connector (if you are using MySQL as your database). Here are the documents for those installation steps:

- Install the MySQL Connector
- Install your License File

After you have installed Java, a database, the Clean Speak™ WebService, the MySQL connector, and your license file, you are ready to configure the WebService. Here are the documents that cover the steps to configure the product:

- Common Configuration
- WebService Configuration

Management Interface

Installation - Management Interface

Before you install the Management Interface, you must first install a Java and a database. Here are the documents that cover the installation of each:

- Java
- Database

Bundles

There are a number of different ways to install the Clean Speak™ Management Interface depending on the operating system and the JEE web server you will be using. Here are the various bundles and the operating system and web servers it works with:

- ZIP file
    - cleanspeak-management-interface-<version>.zip
    - This bundle can be used on any operating system, but is primarily for Windows
    - Includes Tomcat version 6.0.20
- Debian package
    - cleanspeak-management-interface-<version>.deb
    - This bundle can be used on any operating system that supports the Debian package management system (dpkg)
    - Includes Tomcat version 6.0.20
- RPM package
    - cleanspeak-management-interface-<version>.rpm
    - This bundle can be used on any operating system that supports the Redhat Package Management system (rpm)
    - Includes Tomcat version 6.0.20
- (Experts Only) Webapp only ZIP file
    - cleanspeak-management-interface-webapp-<version>.zip
    - This bundle can be used on any operating system
    - Does not include a web server and requires advanced configuration and setup. Installation of this bundle is not covered in this document

Inversoft highly recommends using the Debian, RPM, or ZIP installation bundles. The other bundles are for advanced users and system administrators that feel comfortable with Java, Tomcat, and database setup and configuration.

Windows

To install on Windows XP or higher use the ZIP bundle. Extract the ZIP file anywhere on the file system. Remember where you extract the files. This location will be referred to as CLEANSPEAK_HOME. We suggest extracting this file to a directory such as**c:\Inversoft** on Windows.

If you want to run the Management Interface as a Windows Service you can install it as a Windows service using the service.bat script that ships with Tomcat. Here is how you execute that script from the command-line to install the Management Interface as a Windows service:

```
C:\Inversoft\cleanspeak-management-interface\apache-tomcat-6.0.20\bin>service.bat install
CleanSpeakManagementInterface
```

The last parameter to this script is the name of the service. You can use any name as long as it doesn't contain any periods ('.'), underscores ('_') or spaces (' ').

Redhat

To install on a Red Hat system, including Fedora or CentOS, use the RPM bundle. Execute this command to install the RPM (replace <version> with the correct version number):

```
$ sudo rpm -i cleanspeak-management-interface-<version>.rpm
```

This installation process places the Clean Speak™ Management Interface, including Tomcat 6.0.20, in the directory /usr/local/inversoft/cleanspeak-management-interface. This location will be referred to as CLEANSPEAK_HOME. In addition to the files in this directory, the RPM also installs an init script named /etc/init.d/cleanspeak-management-interface. This script will be used to start and stop the application once it has been configured.

Debian

To install on a Debian system, including Ubuntu, use the DEB bundle. Execute this command to install the DEB (replace <version> with the correct version number):

```
$ sudo dpkg -i cleanspeak-filter-management-interface-<version>.deb
```

This installation process places the Clean Speak™ Management Interface, including Tomcat 6.0.20, in the directory /usr/local/inversoft/cleanspeak-management-interface. This location will be referred to CLEANSPEAK_HOME for the rest of this document. In addition to the files in this directory, the RPM also installs an init script named /etc/init.d/cleanspeak-management-interface. This script will be used to start and stop the application once it has been configured.

Sudo

On Linux systems, the init script uses sudo to run Clean Speak as the cleanspeak user. Therefore, you must have sudo installed and must grant sudo privileges to the root user.

Next Steps

To complete the installation of the Management Interface, you will need to install a license file and optionally the MySQL connector (if you are using MySQL as your database). Here are the documents for those installation steps:

- Install the MySQL Connector
- Install your License File

After you have installed Java, a database, the Clean Speak™ Management Interface, the MySQL connector, and your license file, you are ready to configure the Management Interface. Here are the documents that cover the steps to configure the product:

- Common Configuration
- Management Interface Configuration

MySQL Connector

Installation - MySQL Connector

In order to hook the application up to a MySQL database, you need to download and install the MySQL connector for Java. Unfortunately, MySQL is licensed under the GPL license, which prevents us from shipping this library inside our bundle. Therefore, you need to download the library by opening a browser and clicking this URL:

http://mirrors.ibiblio.org/pub/mirrors/maven2/mysql/mysql-connector-java/5.1.9/mysql-connector-java-5.1.9.jar

Save this file to the CLEANSPEAK_HOME/apache-tomcat-6.0.20-config/lib directory.

License File

Installation - License

You need to install your license file before the products will run properly. You can generate a license the from the My Account section of the website. Ensure that the license file is named *cleanspeak.license*. Copy this file into the *CLEANSPEAK_HOME/web/WEB-INF/classes* directory (this directory is the same for both products).

Java API

In order to use the Java API, your license file should be placed into your applications classpath. This is not required, but is the simplest method of installing the license file. If you want to store the license file somewhere else on the file system, you will need to pass the location of the license file to the Licensing API. This process is described in the Java API usage section. For now, we'll assume you are installing the license file into the classpath.

If you are building a JEE (formerly known as J2EE) web application place the license file into the WEB-INF/classes directory of your application.

If you are building a JEE (formerly known as J2EE) enterprise application or EJB, place the license file into your EJB JAR file or into a container specific directory. For example, WebLogic uses a directory named APP-INF/classes inside the EAR for these types of files. Different JEE containers have different methods of adding files to the classpath of an enterprise application. Consult your container's documentation for more information.

For other types of applications, add the directory that contains the license file to the applications classpath. This is generally accomplished using the classpath parameter to the java command like this:

```
# Unix systems
java -cp /path/to/directory/that/contains/license/file

# Windows systems
java -cp c:\path\to\directory\that\contains\license\file
```

Be sure to include the directory that contains the license file and not the file itself in the classpath passed to the java executable.

Caveats

The license file is a binary file and therefore you should be careful not to treat it as a text file within your source control system or via email as it will become corrupt.

Other JEE Web Servers

Installation - Other Web Server

All of the bundles (except the webapp only bundle) come with Tomcat 6.0.20. This is the preferred web server. In some cases, you might need to run a web server other than Tomcat. If this is the case, we assume that you have a good understanding of Java and JEE. Therefore, we do not cover this installation process explicitly. However, here are some tips to get things working properly.

- The server must support JEE 1.4 or higher. If you find a server that specifies versions of the Servlet and JSP specifications here are the versions of those specifications that the server must conform to:
    - Servlet: 2.4 or higher
    - JSP: 2.0 or higher
- Ensure you have a JDBC DataSource (sometimes called a connection pool) in the global JNDI tree under one of these names
    - For the WebService - **java:comp/env/jdbc/cleanspeak-webservice**
    - For the Management Interface - **java:comp/env/jdbc/cleanspeak-management-interface**
- Ensure you have the MySQL or PostgreSQL driver JAR file installed in the necessary location for your JEE web server to properly create a global DataSource in the JNDI tree
- You might need to edit the web/WEB-INF/web.xml file in the Clean Speak™ WebService or Management Interface bundle to properly configure a reference to the JDBC DataSource in the JNDI tree (this will depend on your JEE web server's requirements)
- Ensure you have a JavaMail session in the global JNDI tree under the name java:comp/env/mail/Session
- Ensure you have the JavaMail and Java Activation Framework JAR files in the necessary location for your JEE web server to create a global JavaMail session in the JNDI tree
- Configure the JEE web server to listen on an open port (generally JEE web servers listen on port 8080. You might need to use the JSVC or sudo program or run the JEE web server as a privileged user in order to bind ports below 1024)

The PostgreSQL, JavaMail and Activation JAR files listed above are located in CLEANSPEAK_WEBSERVICE_HOME/lib or CLEANSPEAK_MI_HOME/lib directory. You will need to download MySQL JAR manually and install it. You can download the MySQL JAR file from here:

http://mirrors.ibiblio.org/pub/mirrors/maven2/mysql/mysql-connector-java/5.1.9/mysql-connector-java-5.1.9.jar

Tomcat example

As a reference to guide you in configuring other JEE web servers, here is the Tomcat configuration that Clean Speak™ uses by default:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  This file contains a default server configuration. Please consult the Tomcat
  documentation for information about modifying this file.
-->
<Server port="8010" shutdown="SHUTDOWN">

  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JasperListener" />
  <Listener className="org.apache.catalina.mbeans.ServerLifecycleListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />

  <Service name="Catalina">

    <!--
      This is the main port that Tomcat listens on. If you need to change the port, do it
      here. You might also need to change the port specified on the root element of this
      XML file as well.
    -->
    <Connector port="8011" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />

    <!--
      Uncomment this block if you want to use Apache in front of the webservice
    -->
    <!--
    <Connector port="8009" enableLookups="false" redirectPort="8443" protocol="AJP/1.3" />
    -->

    <Engine name="Catalina" defaultHost="localhost">
      <Host name="localhost" appBase="" unpackWARs="true" autoDeploy="true" xmlValidation="false"
xmlNamespaceAware="false">
        <Context path="" docBase="../web">
          <!--
            Update this to point to your database.
          -->
          <!-- MySQL version -->
          <Resource name="jdbc/cleanspeak-management-interface" auth="Container"
type="javax.sql.DataSource"
                    maxActive="20" validationQuery="select 1"
                    username="cleanspeak" password="cleanspeak" driverClassName="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost:3306/cleanspeak?relaxAutoCommit=true&characterEncoding=utf8"/>

          <!-- PostgreSQL version -->
          <!--
          <Resource name="jdbc/cleanspeak-management-interface" auth="Container"
type="javax.sql.DataSource"
                    maxActive="20" validationQuery="select 1"
                    username="cleanspeak" password="cleanspeak" driverClassName="org.postgresql.Driver"
                    url="jdbc:postgresql://localhost:5432/cleanspeak"/>
          -->

          <!--
            Update this to point to your SMTP server.
          -->
          <Resource name="mail/Session" auth="Container" type="javax.mail.Session"
mail.smtp.host="localhost" />
        </Context>
      </Host>
    </Engine>
  </Service>
</Server>
```

Configuration

This chapter covers the configuration required to run the Clean Speak™ WebService and Management Interface.

- Common Configuration
- WebService Configuration

- Management Interface Configuration

Common Configuration

Configuration - Common

There are a couple of common steps that you need to take to configure both the Clean Speak™ WebService and Management Interface applications. These are described here. The variable CLEANSPEAK_HOME refers to the installation directory of either of the two applications.

Windows

In order to run Clean Speak on a Windows server, you need to ensure that it uses the version of Tomcat that ships with Clean Speak. Therefore, you need to remove any Tomcat environment variables that might have been previously setup on the server. These variables is generally named **CATALINA_HOME** and **CATALINA_BASE**.

All Operating Systems

Copy the file CLEANSPEAK_HOME/apache-tomcat-6.0.20/conf/server.xml-example to CLEANSPEAK_HOME/apache-tomcat-6.0.20/conf/server.xml.

JVM Settings

On Linux systems that are using the init.d scripts that come with Clean Speak™ to start and stop the application, if you need to override any of the JVM configuration parameters you will need to copy the file **CLEANSPEAK_HOME/apache-tomcat-6.0.20/bin/init.d-overrides.sh-example** to **CLEANSPEAK_HOME/apache-tomcat-6.0.20/bin/init.d-overrides.sh**.

Memory Settings

Inside this file, you will find a number of different environmental variables that you can modify. If you need to change the memory settings for the JVM, uncomment the JAVA_OPTS variable and modify the **-Xmx512M** setting. This controls the maximum amount of memory, in megabytes, to give the JVM. The default value is 512 Megabytes. You can set this value to anything that your server can comfortably handle. If you need assistance setting this parameter, contact Inversoft for support.

WebService Configuration

Configuration - WebService

Before you configure the WebService, you must first perform the common configuration steps. Here are the documents that cover those steps:

- Common Configuration

Once you have finished the common configuration, you can setup your database connection for the WebService. If you are using the WebService without a database connection and instead using the XML file, you can skip this section.

Using a Relational Database

Open the server.xml (located at CLEANSPEAK_HOME/apache-tomcat-6.0.20/config) file in a text editor and scroll to the bottom. If you are using PostgreSQL, comment out the MySQL Resource definition and uncomment the PostgreSQL Resource definition. Next, edit the Resource definition so that the username, password, and url attributes are correct. The url attribute needs to point to the correct server that your database is running on. Here is an example Resource element for a PostgreSQL database:

```
<Resource name="jdbc/cleanspeak-webservice" auth="Container"
          type="javax.sql.DataSource" maxActive="15"
validationQuery="select 1"
          username="cleanspeak" password="cleanspeak"
          driverClassName="org.postgresql.Driver"
          url="jdbc:postgresql://databases.example.com:5432/cleanspeak"/>
```

Using the XML Database

If you comment out the MySQL and PostgreSQL Resource elements defined above, this will tell Clean Speak™ not to use a Relational Database. Instead, it will use an XML file to load the Real-Time Filter white and black lists from. The XML file is included in the Real-Time Filter Database download located in the Downloads section of the website. Once you download that bundle, it will contain a file named **cleanspeak-real-time-filter.xml**. You will need to copy this file to the **CLEANSPEAK_HOME/web/WEB-INF/classes** directory and rename it to **cleanspeak-database.xml**.

You can change the default location and name that Clean Speak™ uses to load the XML database from. This is accomplished by copying the **CLEANSPEAK_HOME/web/WEB-INF/config/config-production.xml-example** file to**CLEANSPEAK_HOME/web/WEB-INF/config/config-production.xml** and editing this file in a text editor. Uncomment the**<database>** element and put the full path to the file you would like to use. This step is not required unless you are changing the location or the name of the file.

Using this type of configuration will prevent the Management Interface from sending notifications to the WebService to tell it to load new words to be filtered from the database. Instead, you will need to either edit the XML file by hand or export to XML from the Management Interface, copy the new XML file to the WebService and restart the WebService application.

Management Interface Configuration

Configuration - Management Interface

Before you configure the Management Interface, you must first perform the common configuration steps. Here are the documents that cover those steps:

- Common Configuration

Once you have finished the common configuration, you can setup your database connection for the Management Interface.

Datasource

Open the server.xml (located at CLEANSPEAK_HOME/apache-tomcat-6.0.20/config) file in a text editor and scroll to the bottom. If you are using PostgreSQL, comment out the MySQL Resource definition and uncomment the PostgreSQL Resource definition. Next, edit the Resource definition so that the username, password, and url attributes are correct. The url attribute needs to point to the correct server that your database is running on. Here is an example Resource element for a PostgreSQL database:

```
<Resource name="jdbc/cleanspeak-management-interface" auth="Container"
          type="javax.sql.DataSource" maxActive="15"
validationQuery="select 1"
          username="cleanspeak" password="cleanspeak"
          driverClassName="org.postgresql.Driver"
          url="jdbc:postgresql://databases.example.com:5432/cleanspeak"/>
```

Running

This chapter covers how to start and stop the Clean Speak™ WebService and Management Interface applications.

- Running the Webservice
- Running the Management Interface

Running the Webservice

Below are instructions for starting and stopping the WebService on both Windows and Linux.

Windows

To start the Clean Speak™ WebService on Windows when it is not installed as a Windows Service, execute the startup.bat script located in the CLEANSPEAK_HOME/apache-tomcat-6.0.20/bin directory using the command-line like this:

```
> startup.bat
```

To stop the Clean Speak™ WebService on Windows, execute the shutdown.bat script like this:

```
> shutdown.bat
```

You can also start and stop Clean Speak™ by double clicking the these scripts from Windows Explorer.

Windows Service

If you installed the Clean Speak™ WebService as a Windows Service, you can configure it to automatically be started when Windows boots up. This configuration is part of the Windows Services Administration tool. Consult Windows help to learn how to access this tool. You can also manually start and stop the Clean Speak™ products from this tool as well.

Linux (RPM and Deb bundles)

These packages both include an init script that is used to control the application as well as start the application when the server is booted up. To start the Clean Speak™ WebService execute this command:

```
$ sudo /etc/init.d/cleanspeak-management-interface start
```

To stop the Clean Speak™ WebService execute this command:

```
$ sudo /etc/init.d/cleanspeak-management-interface stop
```

Linux

To start the Clean Speak™ WebService on Linux when it is not installed using the RPM or Debian packages, execute the startup.sh script located in the CLEANSPEAK_HOME/apache-tomcat-6.0.20/bin directory like this:

```
$ ./startup.sh
```

To stop the Clean Speak™ WebService on Linux, execute the shutdown.sh script like this:

```
$ ./shutdown.sh
```

Once you have the WebService up and running, you can do a number of different things with it:

- Verify it is working with the **/test** page
- Look at the performance metrics with the **/metrics** page
- Send requests to the WebService

Testing

To test that the WebService is running correctly, open this URL in your browser:

http://localhost:8001/test

This page contains a form that allows you to test the WebService to ensure it is working properly.

Metrics

Each WebService instance provides a page that displays the current performance metrics and some historical data. You can view this information by opening this URL in your browser:

http://localhost:8001/metrics

Running the Management Interface

Below are instructions for starting and stopping the Management Interface on both Windows and Linux.

Windows

To start the Clean Speak™ Management Interface on Windows when it is not installed as a Windows Service, execute the startup.bat script located in the CLEANSPEAK_HOME/apache-tomcat-6.0.20/bin directory using the command-line like this:

```
> startup.bat
```

To stop the Clean Speak™ Management Interface on Windows, execute the shutdown.bat script like this:

```
> shutdown.bat
```

You can also start and stop the Clean Speak™ Management Interface by double clicking the these scripts from Windows Explorer.

Windows Service

If you installed the Clean Speak™ Management Interface as a Windows Service, you can configure it to automatically be started when Windows boots up. This configuration is part of the Windows Services Administration tool. Consult Windows help to learn how to access this tool. You can also manually start and stop the Clean Speak™ products from this tool as well.

Linux (RPM and Deb bundles)

These packages both include an init script that is used to control the application as well as start the application when the server is booted up. To start the Clean Speak™ Management Interface execute this command:

```
$ sudo /etc/init.d/cleanspeak-management-interface start
```

To stop the Clean Speak™ Management Interface execute this command:

```
$ sudo /etc/init.d/cleanspeak-management-interface stop
```

Linux

To start the Clean Speak™ Management Interface on Linux when it is not installed using the RPM or Debian packages, execute the startup.sh script located in the CLEANSPEAK_HOME/apache-tomcat-6.0.20/bin directory like this:

```
$ ./startup.sh
```

To stop the Clean Speak™ Management Interface on Linux, execute the shutdown.sh script like this:

```
$ ./shutdown.sh
```

Testing

Once you have started the Management Interface, you can test that it is working properly by opening this URL in your browser:

http://localhost:8011/

You might need to modify the port if you changed the port Tomcat listens on in the Tomcat configuration file.

WebServices

This chapter covers the WebServices APIs that are provided by Clean Speak™. All of these WebServices are accessed through the Clean Speak™ WebService application. This section assumes that you have that application correctly installed and running.

- Quick Start Guide
- REST Overview
- Content Handling
- User Management
- Legacy Real-Time Filter
- Legacy Moderator

Quick Start Guide

WebService - Quick Start Guide

This document covers how to quickly get started using the Clean Speak™ Real-Time Filter WebService.

To get started using the WebService, you will need to send an HTTP request with the correct parameters. We'll cover two simple requests to the Real-Time Filter WebService using Java and PHP.

The Java examples only use classes from the JDK. We recommend using libraries for connecting to the WebService rather than JDK classes to help simplify your development. Here are some suggested libraries:

- Apache HTTP Component Client for making the HTTP request
- JDOM for parsing the XML
- DOM4J for parsing the XML
- JSON Simple for parsing the JSON

Java Locate Operation (using XML)

This example code illustrates a locate operation in Java that returns XML results.

```
URL url = new URL("http://localhost:8001/content/handle.xml");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
urlConnection.setDoInput(true);
urlConnection.setDoOutput(true);
urlConnection.setRequestMethod("POST");
urlConnection.addRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
urlConnection.connect();

OutputStream out = urlConnection.getOutputStream();
OutputStreamWriter writer = new OutputStreamWriter(out);
writer.append("request.content=").append(URLEncoder.encode("Some content to
filter", "UTF-8")).append("&");
writer.append("request.filter.operation=locate").append("&");
writer.append("request.filter.enabled=true").append("&");
writer.append("request.filter.cleanspeakdb.enabled=true");
writer.flush();
writer.close();

InputStream inputStream = urlConnection.getInputStream();
Document dom =
DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(inputStrea
m);
Element filter = (Element)
dom.getDocumentElement().getElementsByTagName("filter").item(0);
NodeList matches = filter.getElementsByTagName("match");
for (int i = 0; i < matches.getLength(); i++) {
    Element match = (Element) matches.item(i);
    System.out.println("Found match at " + match.getAttribute("start") + "
" + match.getAttribute("length"));
}
inputStream.close();
```

Java Replace Operation (using XML)

This example code illustrates a replace operation in Java that returns XML results.

```
URL url = new URL("http://localhost:8001/content/handle.xml");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
urlConnection.setDoInput(true);
urlConnection.setDoOutput(true);
urlConnection.setRequestMethod("POST");
urlConnection.addRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
urlConnection.connect();

OutputStream out = urlConnection.getOutputStream();
OutputStreamWriter writer = new OutputStreamWriter(out);
writer.append("request.content=").append(URLEncoder.encode("Some content to
filter", "UTF-8")).append("&");
writer.append("request.filter.operation=replace").append("&");
writer.append("request.filter.enabled=true").append("&");
writer.append("request.filter.cleanspeakdb.enabled=true");
writer.flush();
writer.close();

InputStream inputStream = urlConnection.getInputStream();
Document dom =
DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(inputStrea
m);
Element filter = (Element)
dom.getDocumentElement().getElementsByTagName("filter").item(0);
Element replacement = (Element)
filter.getElementsByTagName("replacement").item(0);
System.out.println("Replacement is " + replacement.getTextContent());
inputStream.close();
```

PHP Locate Operation (using XML)

This example code illustrates a locate operation in PHP that returns XML results.

```
$data = "request.content=" . urlencode("Some content to filter") . "&" .
  "request.filter.operation=locate&" .
  "request.filter.enabled=true&" .
  "request.filter.cleanspeakdb.enabled=true";

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, "http://localhost:8001/content/handle.xml");
curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type:
application/x-www-form-urlencoded'));
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $data);

$response = curl_exec($ch);
curl_close($ch);

$parser = xml_parser_create();
xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, 0);
xml_parse_into_struct($parser, trim($response), $structure, $index);
xml_parser_free($parser);
foreach ($structure as $s) {
  if ($s['tag'] == "match") {
    print("Found match at " . $s['attributes']['start'] . " " .
$s['attributes']['length'] . "\n");
  }
}
```

PHP Replace Operation (using XML)

This example code illustrates a replace operation in PHP that returns XML results.

```
$data = "request.content=" . urlencode("Some content to filter") . "&" .
  "request.filter.operation=replace&" .
  "request.filter.enabled=true&" .
  "request.filter.cleanspeakdb.enabled=true";

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, "http://localhost:8001/content/handle.xml");
curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type:
application/x-www-form-urlencoded'));
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $data);

$response = curl_exec($ch);
curl_close($ch);

$parser = xml_parser_create();
xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, 0);
xml_parse_into_struct($parser, trim($response), $structure, $index);
xml_parser_free($parser);
foreach ($structure as $s) {
  if ($s['tag'] == "replacement") {
    print("Replacement is " . $s['value'] . "\n");
  }
}
```

Python Match Operation (using JSON via Django)

```
import httplib
import urllib

from django.utils import simplejson

class ApiKeyError(Exception):
    """Raised when the WebService API returns a 401 status code.

    The API key set via set_api_key, or otherwise, is invalid and must be
    rectified. No further information is available on the error.
    """
    pass


class UnknownResponseError(Exception):
    """Raised when the WebService API returns a status code other than 200,
400
    or 401.
    """
    pass
```

```
class ResponseData(object):
    """Simple class returned by send_request, containing the success state
and
    JSON data of the response.
    """
    def __init__(self, success, data):
        self.success = success
        self.data = data


class WebService(object):
    def __init__(self, api_key):
        """Initialize the WebService object (must provide an API key)

        Note: the API key is not tested until send_request.
        """
        self.set_api_key(api_key)
        self.post_data = {}

    def set_api_key(self, api_key):
        """Set the API key"""
        self.api_key = api_key

    def set_param(self, key, value):
        """Set a HTTP POST parameter. All keys are prepended with
'request.'"""
        if isinstance(value, bool):
            value = str(value).lower()

        self.post_data['request.' + key] = value

    def send_request(self, param_data=None):
        """Send the POST request to the Inversoft API server.

        If param_data is None, the POST data set by set_param is used.
        Otherwise, param_data completely overrides other parameters set.

        ApiKeyError and UnknownResponseError may be raised by this method.

        Returns a ResponseData instance, which contains the success state
and
        JSON data from the response.
        """
        if param_data is None:
            param_data = self.post_data

        params = urllib.urlencode(param_data)

        headers = {
            'Content-Type': 'application/x-www-form-urlencoded',
            'Authentication': self.api_key,
        }
        conn = httplib.HTTPConnection('api.inversoft.com', 8001)
        conn.request('POST', '/content/handle.js', params, headers)

        response = conn.getresponse()

        if response.status == 401:
            raise ApiKeyError()
        elif response.status not in (200, 400):
            raise UnknownResponseError('Received status code %s (%s);
expected 200, 400 or 401' % (response.status, response.reason))

        data = response.read()
        conn.close()

        return ResponseData(response.status == 200, simplejson.loads(data))


def example():
    """Example WebService API usage."""
    # Create a WebService instance and populate the parameters
    cs = WebService('api_key_here')
    cs.set_param('content', text)
    cs.set_param('filter.enabled', True)
    cs.set_param('filter.cleanspeakdb.enabled', True)
    cs.set_param('filter.operation', 'match')
    response = cs.send_request()

    if not response.success:
        # Print some error messages here
        return
    # Deserialized JSON data
    print response.data
```

REST Overview

REST WebServices

This document covers how Clean Speak™ defines REST WebServices.

HTTP Methods

Each operation for a REST WebService URL is based on the HTTP method that is used. Here are the methods that the majority of the Clean Speak™ WebServices support:

Method	Description
GET	This retrieves an object.
POST	This creates an object.
PUT	This updates an object.
DELETE	This deletes an object.

Request Parameters

Request parameters are sent to the REST WebServices using the HTTP standard that most browsers conform to. Each HTTP method receives parameters differently in order to conform to this standard. Here are each of the HTTP methods and how they receive parameters:

Method	Parameters

GET	Parameters for a GET request are passed as URL parameters within the HTTP header. These parameters are define as part of the URL specification that HTTP uses. That specification is located at http://www.ietf.org/rfc/rfc2396.txt.
	
	Depending on the WebService, most GET requests can take an ID parameters. This ID parameter can be passed as a URL parameter or part of the URI. Look at the documentation page for the specific WebService to determine if it contains an ID parameter.
	
	Here are some examples of GET URLs:
	
	`http://localhost:8001/example/1`
	`http://localhost:8001/example?id=1`
	`http://localhost:8001/example?search=some+search+string`

POST	Parameters for a POST request can be passed in the HTTP message-body (also known as the entity-body) or as URL parameters. The ID parameter might also be passed as part of the URI, depending on the specific WebService you are calling. The HTTP message-body is defined in the HTTP specification here:http://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html#sec4.3 The format of the message-body is specified by the HTTP Transfer-Encoding, which is specified by the **Content-Type**HTTP header. To use the message-body mechanism to send in the data you must set the **Content-Type** header to**application/x-www-form-urlencoded**. If you want more information on the **application/x-www-form-urlencoded** Transer-Encoding it is defined as part of the HTML specification here: http://www.w3.org/TR/html401/interact/forms.html. Here is an example of HTTP bodies for POST requests: `request.content=testing&request.filter.enabled=true&request.filter.cleanspeakdb.enabled=true`

PUT	Parameters for a PUT request are the same as the POST request for consistency, even though the HTML specification doesn't cover the PUT method.
DELETE	Parameters for a DELETE request are are the same as the GET request. Similarly, the ID parameter can be set as a URL parameter or part of the URI.

Example Code

Here are some examples of accessing a WebService using each of the above methods. These examples are all in Java, but other languages have similar APIs.

GET Example

```
URL url = new URL("http://localhost:8001/example/1");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
urlConnection.setDoInput(true);
urlConnection.setRequestMethod("GET");
urlConnection.connect();
```

POST Example

```
URL url = new URL("http://localhost:8001/example");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
urlConnection.setDoInput(true);
urlConnection.setDoOutput(true);
urlConnection.setRequestMethod("POST");
urlConnection.addRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
urlConnection.connect();

OutputStream out = urlConnection.getOutputStream();
OutputStreamWriter writer = new OutputStreamWriter(out);
writer.append("request.content=").append(URLEncoder.encode("Some content",
"UTF-8")).append("&");
writer.append("request.filter.enabled=true").append("&");
writer.append("request.filter.cleanspeakdb.enabled=true");
writer.flush();
```

PUT Example

```
URL url = new URL("http://localhost:8001/example/1");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
urlConnection.setDoInput(true);
urlConnection.setDoOutput(true);
urlConnection.setRequestMethod("PUT);
urlConnection.addRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
urlConnection.connect();

OutputStream out = urlConnection.getOutputStream();
OutputStreamWriter writer = new OutputStreamWriter(out);
writer.append("someParameter=").append(URLEncoder.encode("Parameter value",
"UTF-8")).append("&");
writer.append("anotherParameter=").append(URLEncoder.encode("Some other
value", "UTF-8"));
writer.flush();
```

DELETE Example

```
URL url = new URL("http://localhost:8001/example/1");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
urlConnection.setDoInput(true);
urlConnection.setRequestMethod("DELETE");
urlConnection.connect();
```

Authentication

If you are using the Installable version of Clean Speak™ and have enabled WebService authentication through the Management Interface, you will need to pass in authentication key you setup in the Management Interface using the HTTP header named**Authentication** to the WebService.

Here is a Java sample that illustrates how this header is passed to the WebService:

```
URL url = new URL("http://localhost:8001/example");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
urlConnection.addRequestProperty("Authentication",
"your-authentication-key");
...
```

Content Handling

WebService - Content Handling

This WebService is the main access point for Clean Speak™ and provides all of these content handling functions:

- Real-Time Filtering
- Content Moderation
- Content Monitoring and Analysis

REST WebService

This WebService is a RESTful WebService. For more information on how Clean Speak™ defines REST WebServices, consult the REST Overview document.

URLs

The URL for this WebService determines the response type. Here are the URLs and their associated response types:

URL	Response Type
http://localhost:8001/content/handle.js	JSON response
http://localhost:8001/content/handle.xml	XML response
http://localhost:8001/content/handle	XML response

Response codes

You can determine if the request was successful or not based on the response code. Here are the response codes and their meanings:

Code	Description
200	The request was successful.
400	The request as invalid and malformed. The response will contain a JSON or XML message with the specific errors
401	WebService authentication is enabled and you did not supply a valid Authentication header. The response will be empty.

Authentication

If you have enabled WebService authentication through the Management Interface, you will need to pass in the HTTP header named**Authentication** to the WebService. This header should be set to the same value as the authentication key setup in the Management Interface.

Operations

Below are the HTTP methods, the operation they perform, the request information, successful response message format, and error response format.

Method	Description	Request	Success Response	Error Response
POST	This method handles all requests to this WebService. It can perform filtering, moderation and/or monitoring.	Request Parameters	• JSON Response • XML Response	Standard Error Response

Request Parameters

The POST operation for the content handling WebService takes a number of parameters that control how the request is processed. If you are using the Installable version of Clean Speak™, there are a number of extra parameters available to you that allow you to use the Moderation and Monitoring feautres of Clean Speak™. If you are using the Hosted version of Clean Speak™ you can only use the Real-Time Filter feature of the WebService.

Real-Time Filter

To tell the WebService to send the content to the Real-Time Filter, which we refer to as a Filter request, you must supply the following parameter and value:

```
request.filter.enabled=true
```

If you specify that the WebService should run the Real-Time Filter, you will also need to provide two additional pieces of information to the WebService. The first informs the WebService the Real-Time Filtering operation to perform and the second tells it the filters to apply.

There are three filtering operations that the Real-Time Filter can preform. These are:

- locate
- match
- replace

The locate operation tells the filter find and return the location of any matches it finds in the content. The response from the WebService (described later) will contain these results. The locate operation doesn't take any additional attributes, but you must tell the filter the types of content to locate using a number of elements. These are described below.

The match operation tells the filter find and return whether or not there were any results. This result is a simple true or false response from the WebService (described later). The match operation doesn't take any additional attributes, but you must tell the filter the types of content to locate using a number of elements. These are described below.

The replace operation tells the filter find and replace all matches inside the String with either a character (such as *) or another String (such as "censored"). The replace result is returned from the WebService as a String (described later). The replace operation has two additional parameters that control how the matches are replaced. These parameters are named replaceChar and replaceString (described below). You can specify either or these parameter or none. The default replacement is the '*' character.

There are currently six filters you can apply to the content with the Real-Time Filter. These are the filters that are available:

- Clean Speak™ Database - This tells the Real-Time Filter to filter all of the words and phrases in the database. For the Hosted version of Clean Speak™, the database contains 600+ of the most common profanity, racial slurs, etc. To turn this filter on, specify the parameter **request.filter.cleanspeakdb.enabled=true**
- Email - This tells the Real-Time Filter to filter email addresses. The filter will attempt to find anything that looks like a valid email address inside the content. To turn this filter on, specify the parameter **request.filter.emails.enabled=true**
- Phone Numbers - This tells the Real-Time Filter to filter phone numbers. The filter will attempt to find anything that looks like a valid phone number inside the content. To turn this filter on, specify the parameter**request.filter.phone_numbers.enabled=true**
- URLs - This tells the Real-Time Filter to filter URLs. The filter will attempt to find anything that looks like a valid URL inside the content. To turn this filter on, specify the parameter **request.filter.urls.enabled=true**
- Words - This tells the Real-Time Filter to filter out specific words. The words to filter are specified using an additional parameter. To turn this filter on, specify the parameter **request.filter.words.enabled=true**

Moderator

To tell the WebService to send the content to the Moderator, which we refer to as a Moderate request, you must be using the Installable version of Clean Speak™ and you must supply the following parameter and value:

```
request.moderator.enabled=true
```

The Moderator does not require any additional parameters and all of its operations are configured through the Clean Speak™ Management Interface. Remember that you must be using the Installable version of Clean Speak™ to use the Moderator.

NOTE: Dates are always specified using the ISO 8601 format (i.e. 2010-10-20T04:12:34.234Z).

Name	Description	Type	Required?
request.content	The content that will be filtered, moderated and/or monitored.	String	Yes
request.createDate	The date that the content was created.	Date	Yes for Monitor requests
request.context	The context that the request was generated from (i.e. the game or application that generated the content). This must be a valid context configured via the Clean Speak™ Management Interface.	String	Yes for Moderate and Monitor requests
request.location	The location that the content was generated. Often this is a game specific location ID or a topic ID for forums. This location is used to determine conversations within a game or application.	String	No
request.receiverID	The receiver of the content (i.e. chat message or private message). You should only specify this parameter when the content was only viewable by a single person. If multiple people were able to see the content, this should not be provided.	String	No
request.senderID	The sender (generator) of the content.	String	Yes for Monitor requests
request.type	The type of the content. Currently, the Clean Speak™ Moderator supports these values: **text** and **image**. The Clean Speak™ Monitor currently supports these values**chat**, **post** and **text**.	String	Yes for Moderate and Monitor requests

request.uid	A unique identifier for the content. This UID is used by the notification system when content is moderated or actioned within Clean Speak™. This UID must be unique across all contexts.	String	Yes for Moderate requests
request.attributes[{index}].name	The name of an attribute that is associated with the content. Attributes can have arbitrary names and values. The {index} should start at 0 and increment by one for each attribute. For example, if you have 3 attributes, you would specify the parameters: • request.attributes[0].name • request.attributes[0].value • request.attributes[1].name • request.attributes[1].value • request.attributes[2].name • request.attributes[2].value	String	No
request.attributes[{index}].value	The value of an attribute that is associated with the content.	String	No
request.filter.enabled	Informs Clean Speak™ that filtering should be performed on the content and any results should be sent back in the response. The filtering is controlled by the parameters below. Each of the parameters controls how the filter attempts to locate specific content types. The types of content that the filter can search for are: • Words and Phrases from the Clean Speak™ Database • Emails • Arbitrary words and phrases • Arbitrary characters • Phone numbers • URLs	boolean	No
request.filter.operation	The operation performed by the Clean Speak™ Real-Time Filter. The possible values are: • match - This operation determines if the content contains any blacklisted words or phrases. The response will contain a true/false result. • locate - This operation locates all blacklisted words or phrases. The response will contain a list of all of the words and phrases that were found, where they were found, and additional information for each. • replace - This operation replaces all blacklisted words or phrases with a character or string. By default the words and phrases are replaced with asterisks. You can control the replacement character or string using additional request parameters described below.	String	Yes if the Real-Time Filter is enabled for the request
request.filter.replaceChar	The character that is used during a replace operation.	char	No
request.filter.replaceString	The String that is used during a replace operation.	String	No
request.filter.characters.enabled	Informs the Clean Speak™ Real-Time Filter to filter a set of arbitrary characters. The characters are supplied using the parameter below.	boolean	No
request.filter.characters.character	The character that the Clean Speak™ Real-Time Filter should search for. You can specify multiple characters by supplying this parameter multiple times.	char	Yes if the characters filter is enabled
request.filter.cleanspeakdb.enabled	Informs the Clean Speak™ Real-Time Filter to filter words and phrases from the Blacklist stored inside the Clean Speak™ database tables or that were loaded from the Clean Speak™ XML database.	boolean	No
request.filter.cleanspeakdb.category	Controls which words and phrases from the Clean Speak™ database are searched for. Only words and phrases on the Blacklist whose category is set to this value are searched for. You can specify multiple categories by supplying this parameter multiple times. The initial set of categories are: • Slang - This category contains all Slang, Swear, and other offensive language • Racial - This category contains racial slurs • Youth - This category contains words that are considered inappropriate for younger audiences • Drug - This category contains drug terms • Alcohol - This category contains alcohol terms • Gang - This category contains gang related terms • Religious - This category contains terms that might be considered offensive by certain religious groups • AsciiArt - This category contains offensive pictures formed by combining ascii characters	String	No
request.filter.cleanspeakdb.locale	Controls which words and phrases from the Clean Speak™ database are searched for. Only words and phrases on the Blacklist whose locale is set to this value are searched for. You can specify multiple locales by supplying this parameter multiple times.	String	No

request.filter.cleanspeakdb.severity	Controls which words and phrases from the Clean Speak™ database are searched for. Only words and phrases on the Blacklist whose severity setting is equal to or more severe than this value are searched for. For example, if you specify**high** the filter will search for words and phrases marked as**high** or **severe**. The list of severities are: • mild - Mild and generally not offensive to most adults and children. Some might feel that these terms should not be used in by children or when directed at someone else • medium - Generally not considered offensive when used in friendly company, but still used in polite company or in a professional setting • high - Most people would consider these offensive and they are generally not used in polite company or professional settings • severe - These are always considered offensive and used rarely	String	No
request.filter.emails.enabled	Informs the Clean Speak™ Real-Time Filter to filter emails.	boolean	No
request.filter.emails.strict	Informs the Clean Speak™ Real-Time Filter that any instances of the '@' character should be considered an email address. If this is false, the Clean Speak™ Real-Time Filter will the standard search mechanism for locating emails. This defaults to false.	boolean	No
request.filter.phone_numbers.enabled	Informs the Clean Speak™ Real-Time Filter to filter phone numbers. This attempts to identify sequences of numbers or words that are numbers (English only currently) that look like phone numbers. By default, the phone number filter is aggressive in looking for matches. However, it also provides a quality value based on how good each match looks. The quality score is a double value between 0.0 and 1.0. You can use the parameters below to control the scoring mechanism and control the length criteria for the phone number filter.	boolean	No
request.filter.phone_numbers.maximumMatchLength	The maximum number of digits or words that the phone number can have to be considered a valid match. This defaults to 20	int	No
request.filter.phone_numbers.minimumMatchLength	The minimum number of digits or words that the phone number must have to be considered a valid match. This defaults to 6	int	No
request.filter.phone_numbers.separatorPenalty	The penalty (subtraction) that the Clean Speak™ Real-Time Filter applies when it encounters a separator besides a dash or parenthesis in a phone number. For example,**303;555;1234** contains two alternate separators. This defaults to -0.05	double	No
request.filter.phone_numbers.spacePenalty	The penalty (subtraction) that the Clean Speak™ Real-Time Filter applies when it encounters a space in a phone number. For example, **303 555 1234** contains two spaces. This defaults to -0.05	double	No
request.filter.phone_numbers.wordPenalty	The penalty (subtraction) that the Clean Speak™ Real-Time Filter applies when it encounters a word in a phone number rather than a digit. For example, **three zero three 555 1234** contains three words. This defaults to -0.03	double	No
request.filter.urls.enabled	Informs the Clean Speak™ Real-Time Filter to filter URLs. This attempts to identify sequences of characters or digits that constitute a URL. Since this is often extremely difficult to do accurately, the filter provides a quality value based on how good each match looks. The quality score is a double value between 0.0 and 1.0.	boolean	No
request.filter.urls.maximumMatchLength	The maximum length for a URL to be considered a valid match. This defaults to 50	int	No
request.filter.urls.spacePenalty	The penalty (subtraction) that the Clean Speak™ Real-Time Filter applies when it encounters a space in a URL. For example, **www inversoft com** contains three spaces. This defaults to -0.05	double	No
request.filter.words.enabled	Informs the Clean Speak™ Real-Time Filter to filter a set of arbitrary words. The words are supplied using the parameter below.	boolean	No
request.filter.words.word	The word that the Clean Speak™ Real-Time Filter should search for. You can specify multiple words by supplying this parameter multiple times.	String	Yes if the word filter is enabled
request.moderate.enabled	Informs Clean Speak™ that the content should be handled by the Moderator system and possibly added to the moderation queue (depending on the configuration).	boolean	No

JSON Response

This JSON response contains the result of the operations specified in the request. Each operation will place a set of results into the response directly inside the main JSON object. Here is an example of the JSON response for a filter request:

```
{
  "filter": {
    "matched": true,
    "matches": [
      { "start": 0, "length": 6, "matched": "fucker", "type":
"cleanspeakdb",
        "root": "fuck", "category": "Slang", "severity": "severe" }
    ]
  }
}
```

Here is an example of the JSON response for a moderate request:

```
{
  "moderate": {
    "result": "PENDING_MODERATION"
  }
}
```

The JSON keys and values are described in this table:

Key	Value Description
filter	The filter object that contains the result from a filter request.
filter.matched	A boolean that defines if the filter found any results or not.
filter.matches	A list of matches that the filter found if the operation is set to **locate**.
filter.matches.category	If the match is of the type **cleanspeakdb** then this will contain the category of the match.
filter.matches.length	The length of a single match.
filter.matches.matched	If the match is of the type **cleanspeakdb** then this will contain the conjugation or variation that the filter matched on.
filter.matches.root	If the match is of the type **cleanspeakdb** then this will contain the root word of the match.
filter.matches.severity	If the match is of the type **cleanspeakdb** then this will contain the severity of the match.
filter.matches.start	The start index of a single match.
filter.matches.type	The type of the match. This will correspond to the filter types that were enabled and configured in the request. The current filters that the Clean Speak™ Real-Time Filter supports are: • characters • cleanspeakdb • emails • phone_numbers • urls • words
filter.replacement	The content with all matches replaced if the operation is set to **replace**.
moderate	The moderate object that contains the result from a moderate request.

moderate.result	The result of the moderation of the content. In most cases, the content will be added to the moderation queue but it is also possible to have the content automatically approved or rejected, approved or rejected outright, or rejected because the UID has already been submitted. The possible values are: • PENDING_MODERATION - This indicates that the content was added to the moderation queue. • APPROVED - This indicates that the content was approved. This generally means that the content was on the whitelist. • REJECTED - This indicates that the content was rejected. This generally means that the content was on the blacklist. • AUTO_APPROVED - This indicates that the content was automatically approved. This is a configuration option that can be set in the Clean Speak™ Management Interface. • AUTO_REJECTED - This indicates that the content was automatically rejected. This is a configuration option that can be set in the Clean Speak™ Management Interface. • DUPLICATE - This indicates that a content item with the same UID was previously submitted.

XML Response

The XML response from the WebService is nearly identical to the JSON response. The installation bundles also include a full XML schema file for the XML response. Here the possible XML elements and attributes. You can refer to the descriptions above for each:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <filter matched="true">
    <match start="0" length="6" matched="fucker" root="fuck"
category="Slang" severity="severe"/>
    <replacement><![CDATA[******]]>
  </filter>
  <moderate result="PENDING_MODERATION"/>
</response>
```

Example Code

Here is a simple example code using Java to access the WebService and request a Real-Time Filter replace operation. This will turn on the Real-Time Filter, tell it to perform a replace operation, and apply the cleanspeakdb and url filters:

```
URL url = new URL("http://localhost:8001/example");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
urlConnection.setDoInput(true);
urlConnection.setDoOutput(true);
urlConnection.setRequestMethod("POST");
urlConnection.addRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
urlConnection.connect();

OutputStream out = urlConnection.getOutputStream();
OutputStreamWriter writer = new OutputStreamWriter(out);
writer.append("request.content=").append(URLEncoder.encode("Some content",
"UTF-8")).append("&");
writer.append("request.filter.operation=replace").append("&");
writer.append("request.filter.enabled=true").append("&");
writer.append("request.filter.cleanspeakdb.enabled=true").append("&");
writer.append("request.filter.urls.enabled=true");
writer.flush();

InputStream in = urlConnection.getInputStream();
// Process the input stream for the XML or JSON here
```

User Management

System WebService - User

You can use the WebService to access the user management functions of Clean Speak™ The available operations are:

- Retrieve a user's information
- Create a new user
- Modify an existing user
- Delete a user

REST WebService

This WebService is a RESTful WebService. For more information on how Clean Speak™ defines REST WebServices, consult the REST Overview document.

URLs

The URL for this WebService determines the response type. Here are the URLs and their associated response types:

URL	Response Type
http://localhost:8001/system/user.js	JSON response
http://localhost:8001/system/user.xml	XML response
http://localhost:8001/system/user	XML response

Response codes

You can determine if the request was successful or not based on the response code. Here are the response codes and their meanings:

Code	Description

200	The request was successful.
400	The request as invalid and malformed. The response will contain a JSON or XML message with the specific errors
401	WebService authentication is enabled and you did not supply a valid Authentication header. The response will be empty
404	The ID is invalid. The response will be empty

Authentication

If you have enabled WebService authentication through the Management Interface, you will need to pass in the HTTP header named**Authentication** to the WebService. This header should be set to the same value as the authentication key setup in the Management Interface.

Operations

Below are the HTTP methods, the operation they perform, the request information, successful response message format, and error response format.

Method	Description	Request	Success Response	Error Response
GET	This retrieves a user's information	ID Request Parameters	• Render User JSON Response • Render User XML Response	Standard Error Response
POST	This creates a new user	User Request Parameters	• Render User JSON Response • Render User XML Response	Standard Error Response
PUT	This updates an existing user	User Request Parameters	No response message.	Standard Error Response
DELETE	This deletes an existing user	ID Request Parameters	No response message.	Standard Error Response

ID Request Parameters

The GET and DELETE operations use a single parameter to identify the user to be retrieved or deleted. Here is that parameter:

Name	Description	Type	Required?
id	The ID of the user to retrieve or delete	Integer	Yes

You can specify the ID parameter as a URL parameter for the GET and DELETE operations, or you can specify it via the URI. Here are examples of each of those URLs:

```
http://localhost:8001/system/user?id=2
http://localhost:8001/system/user.js?id=2
http://localhost:8001/system/user.xml?id=2
http://localhost:8001/system/user/2
http://localhost:8001/system/user/2.js
http://localhost:8001/system/user/2.xml
```

User Request Parameters

The POST and PUT operations use a number of different parameters to control the user information that is used to create or update the user. You can specify these parameters as URL parameters or in the HTTP body using the x-www-form-urlencoded transfer encoding. Additionally, you can specify the ID parameter on the URI in the same you can for the GET or DELETE operations. Here are those parameters:

Name	Description	Type	Required?
id	This specifies the user to update or the ID to assign to a new user. If you pass this in to a POST request you must ensure that the ID given will not cause unique key violations.	Integer	Yes for GET, PUT and DELETE. No for POST
user.email	The email (or username) for the user	String	Yes
user.password	The password for the user. If the request is a PUT and this is not specified, the user's current password will remain in effect	String	Yes for POST. No for PUT
roles	The role(s) for the user. You can specify this parameter multiple times, once for each role. If this parameter is not specified, the user will have the admin role and have access to everything.	String	No

Render JSON Response

This JSON response contains the information about the user. This is used for GET and POST requests. The reason this is used for POST is to provide the ID of the user back to the caller.

```
{
  "user": {
    "id": 42,
    "email": "example@test.com"
    "roles": [
      {"id": 3, "name": "filter"}
      {"id": 5, "name": "moderator"}
    ]
  }
}
```

The JSON keys and values are described in this table:

Key	Value Description
user	The value for this key is a JSON object where the keys are the user fields and the values for each field.
user.id	The value for this key is an integer containing the user's ID.
user.email	The value for this key is a String containing the user's email or username.
user.roles	The value for this key is a JSON array that contains multiple JSON objects, one for each role that the user has. The object contains two properties, one for the ID of the role and one for the name.

Render XML Response

This XML response contains the information about the user. This is used for GET and POST requests. The reason this is used for POST is to provide the ID of the user back to the caller.

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <user id="42" email="test@example.com">
    <roles id="3" name="filter"/>
    <roles id="5" name="moderator"/>
  </user>
</response>
```

The XML elements and values are described in this table:

Element	Value Description
user	This element contains elements for the roles and contexts of the user. It has two attributes for the user's fields. The id attribute contains the ID for the user and the email attribute contains the email of the user.
roles	This element contains the role information for a role the user has. This includes the ID and name of the role. There will be multiple roles elements, one for each role the user has.

Legacy Real-Time Filter

Real-Time Filter - Legacy Real-Time Filter WebService

NOTE: This WebService interface is deprecated. Please use the new Content Handling REST interface for filtering, moderation and monitoring requests.

Filtering

To use the WebService for filtering requests, you will need to write some code that will send requests to this URL:

http://localhost:8001/filter

You might need to modify the port if you changed the port Tomcat listens on in the Tomcat configuration file.

The way that you send and receive requests from the WebService is using either XML or Google's Protocol Buffers. XML is generally the simplest and most widely support method for using the WebService. However, if you want the absolute fastest performance possible, Google's Protocol Buffers uses a highly efficient binary format that can reduce the overhead associated with XML creation and parsing.

This document only covers using XML, but you can also refer to the Protocol Buffer definition that comes with the WebService if you want to use that method. To use Protocol Buffers, you will need to use an HTTP POST request with the Content-Type header set to application/vnd.google.protobuf and place the Protocol Buffer byte stream into the HTTP request body.

Authentication

If you have enabled WebService authentication through the Management Interface, you will need to pass in the HTTP header named**Authentication** to the WebService. This header should be set to the same value as the authentication key setup in the Management Interface.

Requests

In order to send a request to the WebService, you need to initiate an HTTP request to the server. This HTTP request will use the URL as described above. The WebService only accepts HTTP POST requests where the request body is an XML message. This is commonly referred to as XML over HTTP.

The XML message sent to the WebService will depend on your specific filtering needs. The root element of the XML message is named request and must have the XML namespace of http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0. Here is an example of this root element:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
</request>
```

Contexts

The root element takes a single attribute, which can be used to control the filter. This attribute is named **context** and it determines which context is to process the request. Contexts are the way that Clean Speak™ allows you to control the set of words and phrases that are filtered. If you are using the Clean Speak™ Management Interface you can create new contexts and manage the words that are part them. You can also control a context's whitelist, which are the words that should be allowed through the filter.

To specify a different context for a WebService request, specify the name of the context on the root element. Here is an example:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0"
         context="My Custom Context">
</request>
```

This request will use all of the words and the whitelist from the "My Custom Context". It will also use all of the words and the whitelist from the "Global" context. All contexts form a hierarchy and inherit all of the information from their parent. Currently, all new contexts inherit from the "Global" context. In the future, Clean Speak™ might support more complex hierarchies between contexts.

The String

Within this element you must pass the String to be filtered using an element named in. This element's body contains the String. You can use CDATA blocks if you want, but they are not required. Here is an example:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
  <in>Hey, let's filter this message</in>
</request>
```

The Operation

Next, you need to tell the WebService what type of operation to perform with the result of the filtering process. There are currently three types of operations:

- locate
- match
- replace

Locate

The locate operation tells the filter find and return the location of any matches it finds in the String. The response from the WebService (described later) will contain these results.

The locate operation doesn't take any additional attributes, but you must tell the filter the types of content to locate using a number of elements. These are described below.

Here is an example:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
  <in>Hey, let's filter this message</in>
  <locate>
    ...
  </locate>
</request>
```

Match

The match operation tells the filter find and return whether or not there were any results. This result is a simple true or false response from the WebService (described later).

The match operation doesn't take any additional attributes, but you must tell the filter the types of content to locate using a number of elements. These are described below.

Here is an example:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
  <in>Hey, let's filter this message</in>
  <match>
    ...
  </match>
</request>
```

Replace

The replace operation tells the filter find and replace all matches inside the String with either a character (such as *) or another String (such as "censored"). The replace result is returned from the WebService as a String (described later).

The replace operation takes two additional attributes, which control how the matches are replaced. These attributes are named withString and withCharacter. You can specify either or these attributes or none. The default replacement is the '*' character. In addition to these attributes, you must tell the filter the types of content to replace using a number of elements. These are described in section 4.1.3.

Here is an example:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
  <in>Hey, let's filter this message</in>
  <replace>
    ...
  </replace>
</request>
```

Content Types

Next, you need to tell the WebService exactly what you want filtered. There are a variety of different types of content that can be filtered. The types currently supported by the filter are:

- Clean Speak™ Database
- Emails
- Words
- Characters
- URLs

Clean Speak™ Database

This type tells the filter to match all of the words that are specified in the Clean Speak™ Database. The Clean Speak™ Database will either be the XML file that ships with the Filter WebService or the relational database setup above.

You can control how the WebService matches profanity using attributes of the cleanspeakdb element. These attributes are:

- categories - This contains a space separated list of categories to match. The categories are:
    - Slang - This category contains all Slang, Swear, and other offensive language
    - Racial - This category contains racial slurs
    - Youth - This category contains words that are considered inappropriate for younger audiences
    - Drug - This category contains drug terms
    - Alcohol - This category contains alcohol terms
    - Gang - This category contains gang related terms
    - Religious - This category contains terms that might be considered offensive by certain religious groups
    - AsciiArt - This category contains offensive pictures formed by combining ascii characters
- severity - This controls how severe the matches are. The severity levels are:
    - none - Not severe at all
    - mild - Mild and generally not offensive to most adults and children. Some might feel that these terms should not be used in by children or when directed at someone else
    - medium - Generally not considered offensive when used in friendly company, but still used in polite company or in a professional setting
    - high - Most people would consider these offensive and they are generally not used in polite company or professional settings
    - severe - These are always considered offensive and used rarely
- locales - This contains a space separated list of languages and optionally country codes, which control the languages and regions of the matches. Currently, only American English and general Spanish terms are provided by Inversoft, but British English, Mexican/Cuban/Peruvian Spanish, French, Italian, German, and Russian are being added. Each entry in this list must start with an ISO 639 language code and be optionally followed by an underscore and a 2-digit ISO 3166 country code.
    - For example, en, en_US, en_GB, es, es_MX, etc.

Here is an example:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
  <in>Hey, let's filter this message</in>
  <locate>
    <cleanspeakdb categories="Slang Racial" severity="medium" locales="en
es_MX"/>
  </locate>
</request>
```

Emails

The **emails** type tells the WebService to match any emails. There is a single option that can be passed to the emails type, which will control how strict this filter will behave. This attribute is:

- strict - This is a Boolean value. When it is set to true the WebService will locate any instances of the '@' character and assume that they form an email. If this is false, the WebService will use a more intelligent search mechanism for locating emails.

Here is an example:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
  <in>Hey, let's filter this message</in>
  <locate>
    <emails strict="true"/>
  </locate>
</request>
```

Words

The **words** type allows you to tell the WebService to match an arbitrary set of words. The WebService will perform the same matching logic that it uses to find profanity, including looking for character replacements, leet speak, separators, spaces, and more.

This type takes a single attribute, which controls the list of words that the WebService will look for. This attribute is:

- words - This attribute is a space separated list of words that the WebService will look for. This list can contain phrases, just replace spaces in the phrase with dashes
    - For example: "this-would-be-a-phrase word1 word2"

Here is an example:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
  <in>Hey, let's filter this message</in>
  <locate>
    <words words="frank fred you-are-cool"/>
  </locate>
</request>
```

Characters

The **characters** type allows you to tell the WebService to match any arbitrary set of characters. Often people use this to disallow dashes, periods, numbers, and other characters that are often used to transmit personal information such as phone numbers and addresses.

This type takes a single attribute, which controls the list of characters that the WebService will look for. This attribute is:

- characters - This attribute is a space separated list of single characters that the WebService will look for.

Here is an example:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
  <in>Hey, let's filter this message</in>
  <locate>
    <characters characters="1 2 3 4 @ $ #"/>
  </locate>
</request>
```

URLs

The **urls** type allows you to tell the WebService to match URLs. This filter type will attempt to find any pattern of characters that resemble a URL. Because this is extremely difficult to do with 100% accuracy, this filter type will return a quality score in the response that indicates how accurate it feels each match is.

Here is an example:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
  <in>Hey, let's filter this message</in>
  <locate>
    <urls/>
  </locate>
</request>
```

The Response

The responses from the WebService are returned as XML messages within the HTTP response body. This XML will vary depending on the request that you sent to the WebService.

All of the response XML messages all start with an element named response and have a XML namespace of http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0. Here is an example of this XML element:

```
<response
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
</response>
```

If there was an error during the filtering, the WebService will send back an XML message without any child elements, but it will add two attributes to the root element. These attributes are:

- error - This attribute contains an error code that describes the error
- errorMessage - This attribute contains a descriptive message of the error.

Here is an example:

```
<response
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0"
          error="invalid-request" errorMessage="Something wicked this way
comes">
</response>
```

For responses without any errors, the response element will contain one of these different elements based on the operation you specified in the request. These elements are:

- located
- matched
- replaced

Located

This element contains the location of a single match. There will be multiple instances of this element for multiple matches. This element contains different information based on the type of match that was found. The attributes of this element are:

- type - The type of the match. This is always one of the types from the request (cleanspeakdb, words, characters, emails, etc)
- start - The offset from the beginning of the String sent in the request to the WebService
- length - The length is the total length of the match
- word - If the type is 'profanity' this will be the word as it was matched in the String
- root - If the type is 'profanity' this will be the root word matched. For example, if 'fuckin' is matched the root word will be 'fuck'
- category - If the type is 'profanity' this will be the category of the word matched (Slang, Racial, etc)

- severity - If the type is 'profanity' this will be the severity of the word matched (high, severe, etc)

Here is an example:

```
<response
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
  <located type="email" start="10" length="20"/>
  <located type="cleanspeakdb" start="62" length="6" word="fuckin"
root="fuck"
           category="Slang" severity="severe"/>
</response>
```

Matched

This element contains a simple Boolean that indicates whether or not a match was found by the WebService. Here is an example:

```
<response
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
  <matched>true</matched>
</response>
```

Replaced

This element contains a String, which has all of the matches found by the WebService replaced. It also contains an attribute named madeReplacements that denotes if any replacements were made. Here is an example:

```
<response
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0">
  <replaced madeReplacements="true">The guy is a ****** and his email is
************************</replaced>
</response>
```

Java Example

If you are using Java to call the WebService, there are a variety of ways to accomplish this from using the URLConnection class and a SAX XML parser to using a library such as Apache Commons HTTP client and JAXB. There are many articles and websites online that you can find information about invoking XML over HTTP WebServices using Java.

To help get you started, we've provided a simple example that uses URLConnection and DOM to use the WebService:

```
URL url = new URL("http://localhost:8001/filter");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
urlConnection.setDoInput(true);
urlConnection.setDoOutput(true);
urlConnection.setRequestMethod("POST");
urlConnection.addRequestProperty("Content-Type", "text/xml");
urlConnection.connect();

String xml =
  "<request
xmlns=\"http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0\
">\n" +
  "  <in>Fuck you</in>\n" +
  "  <match>\n" +
  "    <cleanspeakdb severity=\"medium\" categories=\"Slang Youth\"/>\n" +
  "  </match>\n" +
  "</request>";
OutputStream output = urlConnection.getOutputStream();
output.write(xml.getBytes("UTF-8"));
output.flush();

int code = urlConnection.getResponseCode();
InputStream is;
if (code != 200) {
  is = urlConnection.getErrorStream();
} else {
  is = urlConnection.getInputStream();
}

DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document document = builder.parse(is);
Element responseElement = document.getDocumentElement();
String error = responseElement.getAttribute("error");
if (error != null && error.length() > 0) {
  System.out.println("There was an error");
} else {
  Element matchedElement = (Element)
      responseElement.getElementsByTagName("matched").item(0);
  boolean matched = Boolean.parseBoolean(matchedElement.getTextContent());
  if (matched) {
    System.out.println("Content found");
  } else {
    System.out.println("Content not found");
  }
}
```

Ruby Example

Here is some sample Ruby code that uses REXML and the Net classes to contact the WebService:

```
require 'net/http'
require 'rexml/document'

request =
  "<request
xmlns=\"http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0\
">\n" +
  "  <in>Fuck you</in>\n" +
  "  <match>\n" +
  "    <cleanspeakdb severity=\"medium\" categories=\"Slang Youth\"/>\n" +
  "  </match>\n" +
  "</request>"

post = Net::HTTP::Post.new("/filter")
post.content_type = "text/xml"
post.body = request
http = Net::HTTP.new("localhost", 8001)
httpResponse = http.request(post)

document = REXML::Document.new(httpResponse.body)
response = document.root
if response.attributes['error']
  puts "There was an error [" + response.attributes['error'] + "]"
else
  message = response.elements['matched'].text
  if  message == 'true'
    puts "Content found"
  else
    puts "Content not found"
  end
end
```

PHP Example

Here is some example PHP code that uses the CURL library. This code came from a client and has not yet been verified by Inversoft. If you find problems with this code, please contact Inversoft support for assistance.

```
<?php
$xml =
  "<request
xmlns=\"http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0\
">" .
  "  <in>Fuck you</in>" .
  "  <match>" .
  "    <cleanspeakdb severity=\"medium\" categories=\"Slang Youth\"/>".
  "  </match>" .
  "</request>";

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, "http://localhost:8001/filter");
curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: text/xml'));
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, utf8_encode($xml));

$response = curl_exec($ch);
curl_close($ch);

$parser = xml_parser_create();
xml_parser_set_option($parser, XML_OPTION_TARGET_ENCODING, "UTF-8");
xml_parser_set_option($parser, XML_OPTION_SKIP_WHITE, 1);
xml_parse_into_struct($parser, $response, $structure, $index);
xml_parser_free($parser);
foreach ($structure as $s) {
  if ($s['tag'] == "response" && $s['attributes']['error']) {
    print("There was an error [" . $s['attributes']['error'] . "]");
  } else if ($s['tag'] == "matched") {
    if ($s['value'] == "true") {
      print("Content found");
    } else {
      print("Content not found");
    }
  }
}
?>
```

Legacy Moderator

Moderator - WebService

NOTE: This WebService interface is deprecated. Please use the new Content Handling REST interface for filtering, moderation and monitoring requests.

Moderation

To use the WebService for moderation requests, you will need to write some code that will send requests to this URL:

http://localhost:8001/moderate

You might need to modify the port if you changed the port Tomcat listens on in the Tomcat configuration file.

The way that you send and receive requests from the WebService is using either XML or Google's Protocol Buffers. XML is generally the simplest and most widely support method for using the WebService. However, if you want the absolute fastest performance possible, Google's Protocol Buffers uses a highly efficient binary format that can reduce the overhead associated with XML creation and parsing.

This document only covers using XML, but you can also refer to the Protocol Buffer definition that comes with the WebService if you want to use that method. To use Protocol Buffers, you will need to use an HTTP POST request with the Content-Type header set to application/vnd.google.protobuf and place the Protocol Buffer byte stream into the HTTP request body.

Configuration

Before the Clean Speak™ WebService will accept moderation requests, you must first log into the Clean Speak™ Management Interface and create a Context for the application that will be sending moderation requests to the WebService. Once the Context has been created and approved, you set the Moderator configuration for this Context (under the Moderator > Configuration menu item). Once this configuration has been setup, the WebService will accept moderation requests for that context.

Authentication

If you have enabled WebService authentication through the Management Interface, you will need to pass in the HTTP header named**Authentication** to the WebService. This header should be set to the same value as the authentication key setup in the Management Interface.

Requests

In order to send a request to the WebService, you need to initiate an HTTP request to the server. This HTTP request will use the URL as described above. The WebService only accepts HTTP POST requests where the request body is an XML message. This is commonly referred to as XML over HTTP.

The XML message sent to the WebService will depend on your specific filtering needs. The root element of the XML message is named request and must have the XML namespace of http://www.inversoft.com/schemas/cleanspeak/moderator-webservice/1.0. Here is an example of this root element:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/moderator-webservice/1.
0">
</request>
```

Contexts

The root element takes a single attribute, which is used to control how moderation is handled. This attribute is named context and it must be the same context that you configured via the Management Interface above. Here is an example:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/moderator-webservice/1.
0"
         context="My Custom Context">
</request>
```

This request will use the moderation configuration you specified for the context named "My Custom Context" via the Management Interface.

Inside the request element you can specify any number of item elements. Each item is an item that is to be moderated. The item element contains two attributes and two elements (one is required the other optional). The attributes on the item element are:

- uid - This attribute must contain a unique identifier for the item. Each item sent for moderation must have a unique identifier. If you send two items in for moderation with the same unique identifier or the same item twice, the Moderator WebService will return an error.
- type - This attribute specifies the type of the item. Currently, the types that the Clean Speak™ Moderator supports are:
    - username
    - image
    - text

The child elements for the item element are:

- content - This element contains the content to be moderated and can be escaped or contained in a CDATA block. The content can be a username, text, or URL to an image (audio and video coming soon).
- attribute - This element contains a key-value pair attribute, which is store along with the item during moderation.

Here is an example:

```
<request
xmlns="http://www.inversoft.com/schemas/cleanspeak/moderator-webservice/1.
0"
         context="My Custom Context">
  <item uid="unique-id-1" type="username">
    <content>johnny1234</content>
  </item>
  <item uid="unique-id-2" type="image">
    <content>http://example.com/images/foo.gif</content>
    <attribute key="image-type" value="gif"/>
  </item>
  <item uid="unique-id-2" type="text">
    <content></content>
  </item>
</request>
```

Response

The responses from the WebService are returned as XML messages within the HTTP response body. This XML will vary depending on the request that you sent to the WebService.

All of the response XML messages all start with an element named response and have a XML namespace of http://www.inversoft.com/schemas/cleanspeak/moderator-webservice/1.0. Here is an example of this XML element:

```
<response
xmlns="http://www.inversoft.com/schemas/cleanspeak/moderator-webservice/1.
0">
</response>
```

If there was an error during the moderation, the WebService will send back an XML message without any child elements, but it will add two attributes to the root element. These attributes are:

- error - This attribute contains an error code that describes the error
- errorMessage - This attribute contains a descriptive message of the error

Here is an example:

```
<response
xmlns="http://www.inversoft.com/schemas/cleanspeak/filter-webservice/1.0"
          error="invalid-request" errorMessage="Something wicked this way
comes">
</response>
```

Item

For responses without any errors, the response element will contain one or more item elements, one for each item sent in the request. The item element in the response contains two attributes:

- uid - The unique identifier of the item
- result - The result for the item. The possible results are:
    - APPROVED - The item was approved because the moderation system determined it was identical to a previously approved or whitelisted item (generally only for usernames where the username is on the whiltelist)
    - AUTO_APPROVED - The item was automatically approved because the configuration for the context was set to auto-approved items

- REJECTED - The item was rejected. This occurs when the moderation system determines that the item was blacklisted, identical to a previously rejected item or can otherwise be immediately rejected. This occurs for usernames where the username is on the blacklist or when the context is configured to run the Real-Time Filter and the filter finds profanity inside the content of the item.
- MODERATED - The item was added to the moderation queue and is awaiting moderation.

Java Example

If you are using Java to call the WebService, there are a variety of ways to accomplish this from using the URLConnection class and a SAX XML parser to using a library such as Apache Commons HTTP client and JAXB. There are many articles and websites online that you can find information about invoking XML over HTTP WebServices using Java.

To help get you started, we've provided a simple example that uses URLConnection and DOM to use the WebService:

```
URL url = new URL("http://localhost:8020/moderate");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
urlConnection.setDoInput(true);
urlConnection.setDoOutput(true);
urlConnection.setRequestMethod("POST");
urlConnection.addRequestProperty("Content-Type", "text/xml");
urlConnection.connect();

String xml =
  "<request
xmlns=\"http://www.inversoft.com/schemas/cleanspeak/moderator-webservice/1
.0\">" +
  "  <item uid=\"unique-id-1\" type=\"username\">" +
  "    <content>franky</content>" +
  "  </item>" +
  "</request>";
OutputStream output = urlConnection.getOutputStream();
output.write(xml.getBytes("UTF-8"));
output.flush();

int code = urlConnection.getResponseCode();
InputStream is;
if (code != 200) {
  is = urlConnection.getErrorStream();
} else {
  is = urlConnection.getInputStream();
}

DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document document = builder.parse(is);
Element responseElement = document.getDocumentElement();
String error = responseElement.getAttribute("error");
if (error != null && error.length() > 0) {
  System.out.println("There was an error");
} else {
  // Response items handled here
  ...
}
```

Ruby Example

Here is some sample Ruby code that uses REXML and the Net classes to contact the WebService:

```
require 'net/http'
require 'rexml/document'

request =
  "<request
xmlns=\"http://www.inversoft.com/schemas/cleanspeak/moderator-webservice/1
.0\">" +
  "  <item uid=\"unique-id-1\" type=\"username\">" +
  "    <content>franky</content>" +
  "  </item>" +
  "</request>";

post = Net::HTTP::Post.new("/filter")
post.content_type = "text/xml"
post.body = request
http = Net::HTTP.new("localhost", 8080)
httpResponse = http.request(post)

document = REXML::Document.new(httpResponse.body)
response = document.root
if response.attributes['error']
  puts "There was an error [" + response.attributes['error'] + "]"
else
  // Response items handled here
  ...
end
```

PHP Example

Here is some example PHP code that uses the CURL library. This code came from a client and has not yet been verified by Inversoft. If you find problems with this code, please contact Inversoft support for assistance.

```
<?php
$xml =
  "<request
xmlns=\"http://www.inversoft.com/schemas/cleanspeak/moderator-webservice/1
.0\">" +
  "  <item uid=\"unique-id-1\" type=\"username\">" +
  "    <content>franky</content>" +
  "  </item>" +
  "</request>";

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, "http://localhost:8080/filter");
curl_setopt($ch, CURLOPT_HTTPHEADERS, array('Content-Type: text/xml'));
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, utf8_encode($xml));

$response = curl_exec($ch);
curl_close($ch);

$parser = xml_parser_create();
xml_parse_into_struct($parser, trim($response), $structure, $index);
xml_parser_free($parser);
foreach ($structure as $s) {
  if ($s['tag'] == "response" && $s['attributes']['error']) {
    print("There was an error [" . $s['attributes']['error'] . "]");
  } else {
    // Response items handled here
    ...
  }
}
?>
```

Miscellaneous

This chapter contains miscellaneous information that isn't covered in other chapters.

- Real-Time Filter Java API
- Moderator Notifications

Real-Time Filter Java API

Real-Time Filter - Java API

License

The first thing you need to do before you can use the Clean Speak™ Real-Time Filter Java API is to load your license file. You can place the license file either in the classpath as described in the license installation instructions. You only need to initialize the load the license file once per JVM instance. Therefore, we recommend putting the license initialization code into a static block constructor or a JEE Servlet Context listener.

Here is an example of using a static block constructor to load a license file:

```
import com.inversoft.cleanspeak.license.License;
...
public class MyClass {
    static {
        License.initialize("cleanspeak.license");
    }
    ...
}
```

This code tells the license system that your license file is located in the classpath or the current directory in a file named cleanspeak.license. This API accepts both file paths and classpath locations.

Using the Filter

The main class used for filtering is named ContentFilter. This class requires a number of constructor parameters in order to be properly created. Therefore, it is generally easiest to use the ContentFilterFactory to create an instance of the ContentFilter. The simplest way to use the factory is like this:

```
import com.inversoft.cleanspeak.filter.content.ContentFilter;
import com.inversoft.cleanspeak.filter.content.ContentFilterFactory;

...

ContentFilter filter = ContentFilterFactory.newFilter(...);
```

There are a number of different static methods on the ContentFilterFactory class and the correct method will depend on the database you will be using.

Using the XML Database

The ContentFilterFactory has a method that takes a few parameters and allows you to use the XML Database file. The parameters to the Java API method are the severity rating and the location of the Clean Speak™ Database.

The severity rating is a measure of how offensive a particular word or phrase is. These ratings range from none, meaning not offensive at all, to severe, meaning highly offensive and profane.

The severity rating that is passed into the ContentFilterFactory determines which entries are loaded from the XML Database file and cached in memory. If your application won't be filtering any entries in the database with a severity rating below medium, you should pass medium to the ContentFilterFactory. This will reduce the number of entries loaded from the XML database and reduce the memory footprint and increase the speed of the filter.

The location of the Clean Speak XML Real-Time Database file is the second argument passed to the ContentFilterFactory. This location can be a relative to the current directory used to start the application or an absolute location on the file system.

Here is how a typical call to the ContentFilterFactory looks:

```
import com.inversoft.cleanspeak.domain.SeverityType;
import com.inversoft.cleanspeak.filter.content.ContentFilter;
import com.inversoft.cleanspeak.filter.content.ContentFilterFactory;

...

ContentFilter filter = ContentFilterFactory.newFilter(SeverityType.medium,
    "/var/Inversoft/cleanspeak/cleanspeak-database-1.0.xml");
```

Using ContextLoaders

The ContentFilterFactory has another method that takes a ContextLoader. This method uses the ContextLoader passed in to load a specific context from a database and creates the ContentFilter for that context. The name of the specific context to load from the database is the second parameter to this method.

Contexts

Before we illustrate how to use this method, let's briefly cover contexts. Contexts are the way that Clean Speak™ groups words and phrases in the database. For game companies, contexts are generally games. This allows each game to have a different set of words and phrase that should be filtered. Contexts have a number of other features that are used by the filter during processing including a whitelist.

In most situations, there is a context named "Global" that contains the words and phrases that should be filtered for all games. When new contexts are created, they are always children of the "Global" context. This means that if you create a filter for a context other than the "Global" context, all words and phrases in the "Global" context as well as those in the child context are filtered.

The Clean Speak™ Real-Time Filter Java API comes with a variety of different ContextLoader implementations that you can use to create your ContextFilter. Each ContextLoader implementation requires different parameters in order to be created.

JPA ContextLoader

The Java JPA ContextLoader requires an EntityManager instance be passed to the constructor. This EntityManager must be created from a persistence-unit that is configured with the JPA classes from the Clean Speak™ Domain JAR file. Here is an example persistence.xml file that contains these classes:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
             http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
             version="1.0">
  <persistence-unit name="punit">

<non-jta-data-source>java:comp/env/jdbc/cleanspeak</non-jta-data-source>
    <class>com.inversoft.cleanspeak.domain.jpa.JPAContext</class>
    <class>com.inversoft.cleanspeak.domain.jpa.JPAEntry</class>
    <class>com.inversoft.cleanspeak.domain.jpa.JPAWhitelistEntry</class>
    <class>com.inversoft.cleanspeak.domain.jpa.JPAWhitelistWord</class>
    <properties>
      <property name="hibernate.dialect"
                value="org.hibernate.dialect.MySQL5InnoDBDialect"/>
    </properties>
  </persistence-unit>
</persistence>
```

You can then use the standard JPA classes to create an instance of the EntityManager interface and pass this to the JPAContextLoader. Here is an example of creating a ContentFilter using JPA:

```
EntityManager entityManager = ...;
JPAContextLoader loader = new JPAContextLoader(entityManager);
ContentFilter filter = ContentFilterFactory.newFilter(loader, "Global");
```

DAO ContextLoader (JDBC)

Another ContextLoader implementation for Java is the DAO implementation. This implementation requires that you pass in a DAO service to the constructor. The DAO service is used to hook up to JDBC as well as provide a service that generates the appropriate SQL for the database. The default implementations of all of these interfaces will use SQL-99 compliant SQL, which should work fine for almost all relational databases. Here is an example of using this implementation to create a ContentFilter:

```
Connection connection = ...;
DAOContextLoader loader = new DAOContextLoader(new
JDBCDomainDAO(connection, new SQL99StatementGenerator()));
ContentFilter filter = ContentFilterFactory.newFilter(loader, "Global");
```

SAX ContextLoader (XML)

Another ContextLoader implementation for Java is the SAX implementation. This is the implementation that is used from some of the ContentFilterFactory methods. This implementation requires that you pass in the location of the XML file to load the data from. It uses SAX parser to parse the XML file. Here is an example of using this implementation to create a ContentFilter:

```
Connection connection = ...;
SAXContextLoader loader = new
SAXContextLoader(asList("cleanspeak-database.xml"), SeverityType.mild,
    this.getClass().getClassLoader());
ContentFilter filter = ContentFilterFactory.newFilter(loader, "Global");
```

Filtering

Once you have an instance of the ContentFilter you can send it filtering requests. The ContentFilter class uses the builder pattern to create a filtering DSL (domain specific language). Here is an example of using the filter to replace profanity inside a String:

```
String result = filter.in("This might contain profanity").
      replace(ProcessorType.cleanspeakdb).
      go();
```

The result of this call will be the String passed to the in method with all profanity replaced with asterisks. If we dissect this call, you will notice that the first method invoked is named **in**. This method takes the String to be filtered.

The next method determines the operation that the filter should perform. In this case the operation is replace, which replaces any matches in the String with characters or other Strings. The replace method takes a list of Strings or ProcessorType enumeration values. This list identifies the types of content being replaced. In this case, words and phrases from the Clean Speak™ Database are being replaced.

The last method in the call is named **go**. This signals to the filter that everything has been setup and it should perform the requested operation.

As you can see, this code reads somewhat like English. If you say it out loud it is:

"Filter in the String 'This might contain profanity' and replace Clean Speak Real-Time Database entries. Go!"

Operations

The filtering DSL provides three different types of operations that can be performed on a request. These operations are:

- match
  - This looks for matches in the String. These matches might be profanity or other type of content that you want to locate.
- find
  - This locates all of the matches in the String and returns them in a List. These matches might be profanity or other type of content that you want to locate. Each result contains the start and end index of the match as well as the information about the match. In the case of profanity, the information is the profanity word matched, a definition, parts of speech data and a locale.
- replace
  - This finds and then replaces all matches in the String. Matches can be replaced with asterisks, any other character, or entire words.

Options

The filtering DSL allows for options to be specified for each filtering request. Options are specified in one of two ways. The first way is via additional methods for specific operations, such as replace. The second way is via Option classes.

The first method uses additional methods on the API in order to control how the specific operations are performed. This is currently only applicable to the replace operation. The options you can specify for the replace operation are the replacement character or String. Here are some examples:

```
// This replaces with the character ! rather than *
String result = filter.in("This might contain profanity").
    replace(ProcessorType.cleanspeakdb).
    withCharacter("!").
    go();
```

```
// This replaces with the String "(censored)" rather than the character *
String result = filter.in("This might contain profanity").
    replace(ProcessorType.cleanspeakdb).
    withString("(censored)").
    go();
```

The second method provides a way for options to be specified for the type of content that is being filtered. In our examples we are filtering words and phrases from the Clean Speak™ Database. We might want to control the categories of words and phrases that should be filtered. We do this by supplying a CleanSpeakDBOptions object to the filter like this:

```
String result = filter.in("This might contain profanity").
    replace(ProcessorType.cleanspeakdb).
    withOptionsFor(ProcessorType.cleanspeakdb,
        new CleanSpeakDBOptions(SeverityType.high, "Slang", "Racial")).
    go();
```

This call replaces all words and phrases from the Clean Speak™ Database with a severity rating of high or higher in the categories Slang and Racial. The CleanSpeakDBOptions class takes a number of different parameters that control the filtering process. Consult the JavaDoc for that class for more information.

Advanced Filtering

The ContentFilter API can perform any number of different types of content filtering concurrently. This is accomplished by passing one or more processor types to the operation method like this:

```
String result = filter.in("This might contain profanity").
    replace(ProcessorType.cleanspeakdb, ProcessorType.words,
ProcessorType.email).
    ...
    go();
```

There are currently 5 different content filters that come with the Clean Speak™. The current filters are:

- cleanspeakdb
  - This processor filters all of the words in the Clean Speak™ Database.
- words
  - This processor filters an arbitrary set of Strings.
- characters
  - This processor filters an arbitrary set of characters.
- emails
  - This processor filters email addresses.
- urls

    - This processor filters URLs.

We have already covered the cleanspeakdb processor and filtering words and phrases from the Clean Speak™ Database. The word, character, and email filters work in much the same way. The main difference is that these processors have different option classes and sometimes require that the option classes be passed to the ContentFilter. The option classes for these processors are:

- WordOptions
- CharacterOptions
- EmailOptions
- URLOptions

These classes are used in the same manner that the CleanSpeakDBOptions class is used. Here are some examples:

This example uses the words filter to replace all occurrences of the Strings "foo" and "bar".

```
String result = filter.in("This might contain profanity and foo and bar").
    replace(ProcessorType.words).
    withOptionsFor(ProcessorType.words, new WordOptions("foo", "bar")).
    go();
```

This example uses the characters filter to replace all occurrences of the characters $ and &.

```
String result = filter.in("This might contain profanity and $ and &").
    replace(ProcessorType.characters).
    withOptionsFor(ProcessorType.characters, new CharacterOptions('$',
'&')).
    go();
```

This example uses the emails filter to replace all email addresses.

```
String result = filter.in("My email is joe@example.com").
    replace(ProcessorType.emails).
    go();
```

Updating the Filter

If you are using the Clean Speak™ Management Interface, you can make updates to the Clean Speak™ Database and then notify your filters instances from that interface. The Clean Speak™ Management Interface contains a section where you can add servers to be notified. After you have made changes to the database, you can use the server notification tool inside the Management Interface to send notification messages to your filter instances.

The Clean Speak™ Management Interface uses simple HTTP messages to notify filter instances that there might be updates. The HTTP message is an empty HTTP POST request. The Management Interface assumes that the response to this HTTP message is a valid HTTP status code. A status code of 200 indicates that the filter instance received the message successfully and has reloaded the filter. Any other code is interpreted as a failure.

If you want to support updates inside you application, you will need to write an HTTP listener that will listen for the notification messages from the Management Interface. The method for writing this listener is dependent on your environment, software, frameworks and systems. Therefore, we don't provide details on implementing listeners. We do offer some general guidelines you should follow when writing your listener:

- Ensure your listener correctly implements or uses the HTTP protocol
- Most listeners should handle requests in a separate thread
    - This thread should receive the message and create a new ContentFilter instance
    - Once the new ContentFilter has been created, you can swap it with the old one. This might require some minimal synchronization or the use of a volatile field
- Listeners should use the needsReload method on the ContextLoader implementations to determine if the context they are using needs to be reloaded

Here is a simple listener written in Java using a Servlet:

```
public class UpdateServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse
response) {
        ContextLoader loader = ...;

        // In most cases you will use a singleton or something like that to
store
        // the ContentFilter. This singleton can also store the last load
time
        long lastLoad = ContentFilterSingleton.getLastLoadTime();
        if (loader.needsReload("Global", lastLoad) {
            ContentFilter filter = ContentFilterFactory.newFilter(loader,
"Global");
            ContentFilterSingleton.setContentFilter(filter);
        }

        response.setStatus(200);
    }
}
```

In this code, the process of retrieving the ContextLoader has been left out. A few things to keep in mind about ContextLoader's:

- You should not cache ContextLoader's. They are design to be instantiated, used, and then released each request.
- Each request should create a new ContextLoader.
- Each request should fetch a Connection to the database or use a new EntityManager. This Connection or EntityManager should be passed to the new ContextLoader.

A typical block of code that creates a new ContextLoader might look something like this:

```
InitialContext context = new InitialContext();
DataSource dataSource = (DataSource)
context.lookup("java:comp/env/jdbc/DefaultDS");
Connection connection = dataSource.getConnection();
ContextLoader loader = new DAOContextLoader(new JDBCDomainDAO(connection,
new SQL99StatementGenerator()));
```

Moderator Notifications

Moderator - Notifications

The Clean Speak™ Moderator system is designed to send notifications back into your application after items have been moderated. The location that notifications are sent is configured via the Management Interface inside the Moderator Configuration settings for each Context. The location must be a standard HTTP URL that points to a WebService capable of handling XML-over-HTTP in the same fashion that the Moderator WebService does.

You will need to create a WebService inside you application that handles these notifications. This WebService should receive HTTP POST requests with a Content-Type header of text/xml. The XML that is sent to this WebService from the Clean Speak™ Moderator system is as follows.

Notification

The root element of the notification XML is named notification. This element has an XML namespace of http://www.inversoft.com/schemas/cleanspeak/moderator-notification/1.0. Here is an example:

```
<notification
xmlns="http://www.inversoft.com/schemas/cleanspeak/moderator-notification/
1.0">
</notification>
```

Item

The notification element contains child elements for each item that has been successfully moderated. Each of these item elements contains the unique identifier for the item and the result of the moderation. These are specified using these attributes:

- uid - The unique identifier of the item
- result - The result of the moderation for the item. The possible values are:
    - APPROVED - The item was approved
    - REJECTED - The item was rejected

The item element also contains child elements for the attributes that were sent to the Moderator WebService request. Here is an example of the item elements:

```
<notification
xmlns="http://www.inversoft.com/schemas/cleanspeak/moderator-notification/
1.0">
  <item uid="unique-id-1" result="APPROVED"/>
  <item uid="unique-id-2" result="REJECTED">
    <attribute key="image-type" value="gif"/>
  </item>
</notification>
```

Responses

Your notification WebService must consume the XML described above and send back the appropriate response. The response that your notification WebService must send back to the Clean Speak™ Moderator System is using the HTTP response status code header. If your application successfully handled the notification, it should respond with a HTTP status code of 200. If there was a failure, it should respond with a status code of 500.

Caveats

There are few caveats regarding the notification process. It is possible that the Moderator System might send multiple notifications for the same item with different results. This can occur when an administrator overrides a previous moderation for an item.

Updates and Patches

Updates and Patches

Periodically, Inversoft releases updates for the Clean Speak™. You might also require a patch if you are running into problems and have contacted Inversoft Support for assistance. In these cases, you will need to update the application.

Linux

Updating your application is easy if you installed using the RPM or Debian packages. All you need to do is to issue an update command to the dpkg or rpm program and specify the new package file. Here is an example:

```
# For the Management Interface on Debian
$ dpkg -i cleanspeak-management-interface-<version>.deb

# For the Management Interface on Redhat
$ rpm -U cleanspeak-management-interface-<version>.rpm
```

Windows

On Windows, the steps required are different. Follow these steps to upgrade your version on Windows:

1. Download the webapp only bundle
2. Shut-down your Tomcat or your JEE server
3. Rename the CLEANSPEAK_HOME/web directory to something meaningful like web-2009-06-07 (this is helpful in case you need to revert to this version of the application)
4. Extract the update bundle to the CLEANSPEAK_HOME directory. This will place a new directory called web in the directory
5. Restart Tomcat

Database

Depending on your current version and the new version you will be updating to you might need to execute one or more SQL scripts to update your database. These scripts are listed on the Downloads page under the Upgrade Downloads section. To determine which scripts you might need to run, determine the current version you are running. Based on that version number, run all of the scripts where the first version is your version number or higher. Continue running each script until you reach a script where the second number is higher than the version you are updating to.

Here is the types of versions that Clean Speak™ has:

1. 1.0 is a major version
2. 1.2 is a minor version
3. 1.2.1 is a patch version
4. 1.4-RC1 is a preview version

The ordering of the versions is as follows:

1. Major Preview Version
2. Major Version
3. Patch Version(s)
4. Minor Preview Version
5. Minor Version
6. Patch Version(s)

For example, here are a number of versions in order:

1. 1.0-RC1
2. 1.0-RC2
3. 1.0-RC3
4. 1.0
5. 1.0.1
6. 1.0.2
7. 1.1-RC1
8. 1.1-RC2
9. 1.1-RC3
10. 1.1-RC4
11. 1.1
12. 1.1.1
13. 1.1.2
14. 1.1.3
15. 1.1.4

Not all releases include preview versions, but all preview versions will use the RC suffix followed by a number. RC stands for Release Candidate.

Migrating Between Environments

In some cases, you might have multiple installations of Clean Speak in different environments and migrate between environments. Here's a common example:

1. Update Clean Speak from 1.0.1 to 1.0.2 in your staging environment
2. Edit various white and black lists using the Management Interface in your staging environment
3. Test your changes inside your game or application in your staging environment
4. Update Clean Speak from 1.0.1 to 1.0.2 in your production environment
5. Migrate your database changes from your staging to production environment

These steps are all very straight forward, except for the last step. This step can be difficult if you are using both the Real-Time Filter and the Moderator products. The reason is that you don't want to migrate the entire staging database from staging to production, because this will end up clobbering all of the content handled by the Moderator product. Instead, you need to do a partial migration of the database between environments.

The process for a partial migration of the database for the Real-Time Filter is as follows:

1. Backup your production Clean Speak database entirely
2. Turn off foreign key constraints in your database

3. Drop these tables in your production database (in this order)
    - filter_blacklist_entry_modifications
    - filter_excluded_entry_modifications
    - filter_whitelist_word_modification
    - filter_excluded_entries
    - filter_blacklist_entries
    - filter_whitelist_words
    - filter_verification_cases
    - context_modifications

Troubleshooting

If any problems arise or if you are unable to access the WebService or the Management Interface, consult the log files located in the Tomcat installation directory under the CATALINA_HOME/logs directory. In most cases all errors will be written out to the catalina.out log file. You can contact Inversoft support to help troubleshoot any possible errors that might exist in this file.

User Guide

Welcome to the CleanSpeak 1.x user manual. You can use the table of contents on the left to navigate to the page you want to read, or use the links at the top and bottom to traverse through the documentation.

User Getting Started

Getting Started

To begin using the Clean Speak™ Management Interface, open your browser and type in the address for the application. If you are running everything on your local computer, the address should be:

http://localhost:8011

The default username and password are:

U: admin@inversoft.com
P: password

After you login into the application, you can navigate around using the menu at the top. To learn about the specific features of the application, use the context help windows that are located throughout the application. By clicking on the Help button the help window for that page will be displayed. The help button looks this:

Notice the question mark on the right hand side. This is the help button. It will pop up the help window like this:

You can use the scroll bar on the right side of the window to view the help window contents. When you are finished reading, click the X at the top of the window or hit the Escape key.

Contexts

A Context is generally the same as an application or a game. It is the core concept for Clean Speak™ to determine how to handle Real-Time Filtering and Moderation. For the Real-Time Filter, each Context has its own white and black lists. For the Moderator, each Context has its own moderation queue.

An example of a situation where you might want to create a new Context for the Real-Time Filter is that you have an application/game where the word "ghoul" isn't allowed. However, it all other applications it is allowed. For this situation, you should create a new Context for this application and add the word "ghoul" to the blacklist.

If you are using the Moderator, you must create a Context for each application that will have content moderated. The **Global**(described below) does not have a moderation queue.

Hierarchy

Contexts form a hierarchy. The root of the hierarchy is always the Context named **Global**. For Real-Time Filtering, this Context should contain the blacklisted and whitelisted words that should be used across all Contexts (i.e. across all of your applications). When you create a new Context, it is always a child of the **Global** Context. This means that the new Context will use all of the blacklist and whitelist words from the **Global** Context in addition to any new blacklist and whitelist words you add to the new Context.

For Moderation, Contexts sometimes use this hierarchy. Specifically, the username white and black lists use the Context hierarchy during processing.

Here is a diagram that illustrates this relationship:

Caveats

A few things to keep in mind regarding Contexts.

- You cannot edit the **Global** Context. The name of this Context must always be **Global** and should not be modified in anyway.
- If you Un-delete a Context and then delete it, it will immediate disappear from the listing table and be put back into the Deleted state.
- You cannot delete the the **Global** Context. Since this Context is the root of the hierarchy, the **Global** Context cannot be deleted.
- If you accidentally deleted a Context that you did not intend to delete, you can un-delete it easily by clicking the **VIEW DELETED** button. This will take you to a page that lists out the deleted Contexts and allows you to un-delete them.

Filtering

This chapter provides information about using the Clean Speak™ Management Interface to manage the Clean Speak™ Real-Time Filter.

Filtering Overview

The Clean Speak™ Management Interface provides a set of tools that allow you to manage the Real-Time Filter's white and black lists. These lists control how the filter works and what it will filter.

Each Context has a separate white and black list. Additionally, the **Global** Context also has a white and black list. The white and black lists for Context's other than **Global** inherit everything from the white and black lists from the **Global** Context.

The best way to think about this concept is that if you have multiple applications the might potentially have different sets of words and phrases to be filtered, you will want to put any words that should be filtered in all the applications into the **Global** Context. Then you can add the specific words that should be filtered only for specific Contexts into the Context's black list.

White and Black Lists

The Clean Speak™ Real-Time Filter works using a white and black list. This allows the filter to reduce false positives by ensuring that any matches it finds are not on the white list.

NOTE: The white list overrides the black list in all cases. If you have the word "fuck" on your whitelist and on your blacklist, it will NOT be filtered.

Here is a short description of how the filter works:

1. The filter looks for any words or phrases on the blacklist
2. If a match is found the filter performs these checks:
   a. If the match is NOT embedded (meaning that is has whitespace next to it rather than additional characters), the match is approved
   b. If the match is embedded and the blacklist entry is NOT marked as embeddable or distinguishable, the match is thrown out
   c. If the match is embedded and the blacklist entry is marked as embeddable or distinguishable, the filter checks if full word surrounding the match is on the white list. For example, if "ass" is on the blacklist and the filter is sent "I assume you are correct", it will find a match for "ass" at the word "assume". It checks if the full word next to the match is on the whitelist. In this case the word "assume" is on the whitelist and the filter throws out the match. This is called the *whitelist check*
   d. If the match is embedded, the blacklist entry is marked as distinguishable, and the match passed the *whitelist check*, the match is approved
   e. If the match is embedded, the blacklist entry is marked as embeddable, and the match passed the *whitelist check*, the filter looks at the words on either side of the match to see if they are on the whitelist. This helps prevent false positives for words like "asshanti" but allows creative combinations such as "twatbag" to be found.

As you can see, the filter uses both the white and black lists heavily in order to produce the best results. It is important to ensure that you are managing both lists accurately in order for the filter to work properly. These concepts are covered in more detail in the next section when we talk about each of the proeprties associated with a blacklist entry.

Workflow

The Real-Time Filter is designed to be used in a workflow. This workflow allows you to make modifications, ensure that they work, approve your changes, and update your application to use the changes. Here are the main steps for the workflow and an overview of each one. The rest of the manual pages in this section go into additional detail about each step of the workflow.

1. **Make changes to a Context**

   This step generally involves creating a new Context or editing an existing Context. Within the new or existing Context, you will be adding, editing or deleting entries from the white and black lists. Any changes you make will be in a pending state and will not be used by any of the filters until they are approved.

2. **Test the changes**

   Once you have made the changes to the white and black lists, you should test your changes to ensure that they do not cause the filter to stop working properly. This can be accomplished by using the Verification tool. This tool allows you to run the filter against a set of test cases and quickly verify that the filter is working properly.

   You can also manage the test cases that the Verification tool runs. This will allow you to add new test cases based on the changes you have made.

3. **Approve the changes**

   After you have tested your changes, you can approve or reject them. Any change you made can be rejected individually. This allows you to remove changes that ended up not working as you thought. Once you have the filter working properly and are satisfied with your changes, you can approve all of the changes together. Both the approval and rejection of changes is performed via the Approvals tool.

4. **Notify the filters**

   Once you have approved all of your changes, you can send out notifications to the filters that the database has been updated. You send out notifications using the Notify Servers tool. This tool allows you to manage the list of servers to be notified and send the notifications.

Filtering a Word

One of the main functions of the Clean Speak™ Management Interface is to add a new word (or phrase) that you want to be filtered. This document covers that process.

Context

The first step is to select the Context that you want to add the new word to. Depending on the Context you add the word to, the Real-Time Filter will filter the new word for a single Context or for all Contexts. If you add the new word to the **Global** Context, it will be filtered for all Contexts. If you add it to a different Context, for example a new Context you created call "Forums", it will only be filtered for the Forums Context.

When you click on the Filter > Manage Lists menu option you will be shown the list of Contexts that you can add to new word to. Click on one of the **BLACKLIST** buttons corresponding to the Context you want to add the new word to. This will bring you to a display that lists out all of the current blacklisted words for that Context. Next, click on the **ADD** button to add a new word.

Adding the Word

After you click the **ADD** button, you will be brought to a page that contains three columns of information. The first column contains a number of different form fields that you will use to define the new word to be filtered. The second column contains help information about each field. The third column displays a preview of the conjugations of the word that the filter generates automatically.

First, you'll need to understand how the filter works with each of the form fields to make your life simpler. Although it might appear at first that the more complex the form the longer it will take to add your word, but each of the fields on this form have very specific meanings to the filter and they will help reduce your work and also reduce false-positives.

Here are the fields and their meanings:

Field Name	Description
Word	This property defines the root word or phrase to be blacklisted. The blacklist does not need to contain all of the conjugations or variations of a specific word. Instead, it only needs to contain the root word and the filter will handle the conjugation of that word. **NOTE**: If you specify a phrase in this field and save the phrase, all of the spaces in the phrase will be replaced with dashes (-).

Category	This property defines the general category that the word belongs to. Categories help better separate words and are also used by the filter to determine which words from the blacklist to look for. The default set of categories includes: • AsciiArt - Entries that depict sexually explicit or offensive images using characters • Alcohol - Entries pertaining to alcohol • Drug - Entries pertaining to drugs • Gang - Entries that are gang terms • Racial - Entries that are racial slurs • Religious - Entries that are possibly religiously offensive • Slang - General category for most slang including most profanity and offensive words/phrases • Youth - General category for words that are youth sensitive but not necessarily offensive for adults Categories are completely arbitrary and you can use any name you want. The name should not contain spaces and should be concise. We suggest using the Categories above for most of your words and only adding new Categories if none of the existing Categories fits the new word.
Severity	This property denotes how severe the entry is. The higher the severity the more offensive the entry is. This is mostly subjective, but here are some general guidelines for setting the severity of words. • mild - The entry is generally not offensive in most situations • medium - The entry is sometimes offensive but generally okay to use in professional situations • high - The entry is always offensive but sometimes still used in professional situations • severe - The entry is always offensive and never used in professional situations

Locale	This property defines the language of the entry. In most cases this is simply the ISO 639-2 2-digit language code of the entry. However, you can also specify a specific dialect of the language by combining the ISO 3310 country code to form a standard locale definition. The standard locale definition format is like this:
	
	`[language]_[country]`
	
	Here are a few examples:
	
	`en = English`
	`en_US = American English`
	`en_GB = Great Britian English`
	`fr = French`
	`fr_CA = Canadian French`

Variations	This property defines any variations of the word that you also want filtered. This might include irregular conjugations or other words with the same meaning. Each variation should be separated by a space. If you want to add phrases to this field, you must replace spaces with dashes (-). For example, if you want to add the phrase **dumb luck** as a variation, you would add **dumb-luck** to this field.
Definition	This property is optional and allows you to specify a definition for the word.
Adjective	This property defines whether or not the word is an adjective. This property helps during the conjugation process. Adjectives are conjugated by adding different adjective endings. They are also transformed into adverbs by adding adverb endings. In some cases, the root word is actually a noun and will be conjugated into an adjective. Therefore, it is important to determine what the root word is and mark only the correct properties for that word. For example, you should not add the word **muddy** but should instead add the word **mud** and mark it is a noun. The filter will conjugate the word into **muddy**. However, the word **fugly** is an adjective and never a noun.
Adverb	This property defines whether or not the word is an adverb. This property is used during the conjugation process. Adverbs are conjugated by adding different adverb endings. They are also transformed into adjectives by adding adjective endings. In some cases, the root word is actually a noun and will be conjugated into an adverb. Therefore, it is important to determine what the root word is and mark only the correct properties for that word.
Noun	This property defines whether or not the word is a noun. This property is used during the conjugation process. Nouns are conjugated by adding different noun endings. They are also transformed into adverbs and adjectives by adding both the adverb and adjective endings.
Verb	This property defines whether or not the word is an verb. This property is used during the conjugation process. Verbs are conjugated by adding different verb endings. They are also transformed into adverbs and adjectives by adding both the adverb and adjective endings.

Collapsible	This property determines if double letters within the word can be collapsed into single letters. For example, the word**blueball** can be collapsed to **bluebal** and still retain its meaning. However, the word **tool** cannot be collapsed into the word **tol** because it is no longer the same word. This property also determines if the word can have the characters **ck** collapsed into a **k**. For example, the word **dick**would be collapsed to **dik**.

Distinguishable	This property defines whether or not the entry can be found when it is embedded next to other words. For example, the word **fuck** can generally be distinguished when it is embedded like this:
	
	`foofuck`
	`blahfuckblah`
	
	On the flip side, the word **ass** is almost never distinguishable. Here are some examples:
	
	`thassten`
	`asshanti`
	
	When entries are marked as distinguishable, if the filter finds the word anywhere in the text being filtered, it will mark that word as a match. The only case where this isn't true is if the word is embedded and the full word is on the whitelist.
	
	If you select the distinguishable option, the word will also be marked as embeddable and the embeddable option will be hidden. This is because the distinguishable and embeddable options are directly linked to each other. A word cannot be distinguishable without being embeddable.

Embeddable	This property defines whether or not the filter will locate the word when it is next to other words. When entries are marked as embeddable and the filter finds the word next to another word, a few additional checks are performed. The checks are:
	
	• Check if the entire match is on the whitelist and if so, reject the match
	• Check if the words next to the match are on the whitelist and if they are, accept the match
	
	Here are a few examples of embedded entries and how the filter treats them:
	
	`"I `**`ass`**`ume you are right"`
	
	Here the filter found the word **ass**, but it is next to another word. The filter first checks if the entire match is on the whitelist. In this case, the word **assume** is on the whitelist and therefore the filter will ignore this match.
	
	`"You are an `**`ass`**`jockey"`
	
	Here the filter found the word **ass**, but it is next to another word. The filter first checks if the entire match is on the whitelist. In this case, the word **assjockey** is not on the whitelist. Next, the filter checks if the word next to the match is on the whitelist. Since the word **jockey** is on the whitelist the filter will assume that this is in fact a match.
	
	`"My username is r`**`ass`**`123"`
	
	Here the filter found the word **ass**, but it is next to other words. The filter first checks if the entire match is on the whitelist. In this case, the word **rass123** is not on the whitelist. Next, the filter checks if the word next to the match is on the whitelist. Neither the word **r** or **123** are on the whitelist. Therefore, the filter assumes that this is not a match.

Conjugate	This property determines if the filter will conjugate the word based on the POS (parts of speech) properties and also look for the conjugations. When this property is set, you must also set one of the POS properties and the locale property. Currently, the only conjugation that is supported is English and therefore the locale must contain the language code **en**. If you don't set the locale and the POS properties correctly, no conjugation will occur. You can specify a locale for a specific region such as Great Britain using a locale like **en_GB** and the filter will still conjugate the word since the language is English.

Example

Let's assume for this example that you want to filter the word *run* in your application. Here is how you should fill out this form to ensure that this word and all of its variations are filtered:

1. Begin by adding the root word in the form field labeled **Word**. In this case you will type *run* into this field
2. Determine the severity of the word based on the suggestions above. In this case, set the severity to *high* using the select box labeled **Severity**
3. Assign the word to a Category in the field labeled **Category**. If you start typing or hit the down arrow the list of current Categories will appear. For our example, hit the letter "S" and then use the arrow keys to select the category *Slang* from the list of suggestions.
4. Add any variations of the word into the field labeled **Variations**. This should include irregular conjugations (swim and swam) or other words with the same meaning. For our example, the word *run* has an irregular conjugation, you will want to add the word*ran* as a variation

5. Specify the locale of the word in the field labeled **Locale**. For our example, use the locale *en* to mark the word as English
6. Optionally specify a description of the word if it isn't obvious. This word is quite obvious, but go ahead and type in a simple definition.
7. Determine the parts of speech of the word (adjective, adverb, noun, verb). For each part of speech, select the checkbox associated with it. For our example, select the **Noun** and **Verb** checkboxes
8. Specify if the word is embeddable or distinguishable or neither. For our example, we want to ensure that the word *run* is found for cases such as *runfast*. But we want to ensure it isn't found in cases such as the username *drunster11*. Therefore, we'll mark the word as *Embed dable* but not *Distinguishable*. If we mark it as distinguishable, then the filter will mark *drunster11* as a match. However, it would not mark the word *trunk* as a match since that word is on the whitelist.
9. Specify if the word is collapsible. Since our example doesn't have any double letters of phonetic double letters (like ck), this checkbox doesn't apply and we can leave it unchecked.
10. Specify if you want the word conjugated and preview the conjugations. For our example, we definitely want the word to be conjugated so select this option. If we didn't select this option, the filter would not find any of the conjugations such as *running*,*runs*, *runner*, *etc.*. This is also a major time saver. If you had to add all of the different conjugations by hand, you could spend hours adding each one and still not cover every case.

Finally, click the **SAVE** button to save the new word.

Whitelist Warning

The word you are adding might exist on the whitelist. If this occurs you will be presented with a page that prompts you if you want to remove the words from the whitelist. You can either click **SAVE** button to remove the listed words from the whitelist, click **BACK** to edit your new word, or click **CANCEL** to throw out the new word.

Pending

After you save your new word, it will be in a pending state. This means that your application will not filter this word, even if you restart Clean Speak™. This feature allows you to make multiple changes to your white and black lists before the filter starts using them. Once you have completed all of your changes, you will approve them and then send out a notification.

Next Steps

Next, you will verify your changes using the Verification Tool. Click here to read about that tool.

Verifying Your Changes

After you have added new words to your lists, edited existing words, or deleted words that you no longer want white or black listed, you should verify that your changes will work as expected. The Clean Speak™ Management Interface provides a tool for this called the Verification Tool. Click on the Filter > Verification menu to access this tool.

Verification Cases

The first step to verify that your white and black lists are working is to add Verification Cases for your changes. The Verification Cases are divided into 4 different groups. Here are the groups and how you should use them.

Name	Description

Word Cases That Should Match	This group should contain single words or short phrases that the filter should find matches in. For example
	`fuck`
	`this shit sucks`

Word Cases That Should Not Match	This group should contain single words or short phrases that the filter should not find matches in. In most cases, these are words or phrases that include things from the blacklist or look similar to black listed words that you don't want the filter to match. For example
	`asshanti`
	`bob`

Text Cases That Should Match	This group should contain longer text that the filter should find matches in.
Text Cases That Should Not Match	This group should contain longer text that the filter should not find matches in.

Verification

Once you have setup the Verification Cases you can run the Verification tool by clicking on **Run Verification**. From here, you need to select the Context that you want to run the verification for and the severity level. The severity level you select will instruct the filter to attempt to filter all of the words on the blacklist whose severity level is the same or higher than the one you select.

NOTE: This process might take a couple of seconds to run. This is because the filter needs to be created to run the Verification with. This means that the entire database needs to be loaded, which can take a couple of seconds to complete. This is not the speed of the filter. You can see how fast the filter ran when the results are displayed.

Single Verification

You can also click the **Run Single Verification** link to run a simple test against the filter. This will display a form where you can type in any text and the run the filter against that text.

Next Steps

Next, you will approve your changes using the Approval Tool. Click here to read about that tool.

Approving Your Changes

Once you have made all the modifications you want, verified that they work as expected using the Verification Tool, you now need to approve all your changes. This is done using the Approval Tool. Click the System > Approvals menu item to use the Approval Tool.

Changes

The Approval Tool displays all of the changes you have made to the system, including white and black list changes and Context changes. All the changes for a single Context are grouped together underneath a heading for that Context. From this display, you can revert any specific change or revert all the changes for a single Context.

After you have reviewed all of the modifications and reverted anything you no longer needed, you can click the **Approve All** button at the top to approve all of the changes.

Next Steps

Next, you will push out your changes using the Notification Tool. Click here to read about that tool.

Pushing Your Changes

Once you have approved all of your changes, you will send out notifications to the Clean Speak™ WebService servers or any custom filtering servers you have built. This is done using the Notification Tool.

Servers

The first thing you need to do to send out notifications is to configure your server URLs. These URLs are the location that the Clean Speak™ Management Interface will send notifications to. To manage the servers that notifications are sent to, click the Filter > Servers menu item.

If you are using the Clean Speak™ WebService, the URL is based on the URL that your application uses to filter. Here are some examples of WebService filter URLs and how you can determine the update URLs for each one:

Filter URL	Update URL
http://localhost:8001/filter	http://localhost:8001/update
http://192.168.0.100:8001/filter	http://192.168.0.100:8001/update
http://cleanspeak01.internal.mycompany.com:8001/filter	http://cleanspeak01.internal.mycompany.com:8001/update

As you can see, the filter URL is changed so that the suffix */filter* becomes */update*.

If you are unsure about the URL to use, contact the developers who installed and integrated Clean Speak™ into your application. If this was Inversoft, contact Inversoft support for assistance.

Notifications

Once you have your servers setup, you can click the Filter > Notify Servers menu item to send out the notifications. From here, you select the servers you want to notify and then click the **NOTIFY** button. This process might take a couple of seconds to complete. Once it has completed, you will be shown a table of results that lists whether or not each server was successfully notified or not.

Failures

If for any reason a notification failed, please inspect the Tomcat logs for the WebService server that failed. This log might contain error messages related to the failure. You can also inspect the Tomcat logs for the Management Interface. These might also contain errors pertaining to the failure. If you cannot find the root cause of the problem yourself, contact Inversoft support for assistance.

Importing Lists

If you have purchased a premium list from Inversoft or one of our partners, or have simply created your own list and need to migrate it to a different database, you can import lists into Clean Speak easily using the Import Tool.

First, click on the Filter > Import menu option. From this screen, you will need to select the Context you want to import the list into. In most cases, you'll import the list directly into Global. However, you might want to create a separate Context to import a list into. If you need to create a new Context, click on the System > Contexts menu option before using the Import Tool.

Once you have selected the Context to import into, next select the Clean Speak XML file you want to import. Clean Speak XML files are generally provided by Inversoft or can be created directly from Clean Speak using the Export Tool.

Once you have clicked the Import button and the import is complete, all of the data from the imported list is placed into a pending state and will need to be approved. You can use the Approval Tool to approve or reject the changes made during the import by clicking on the System > Approvals menu option. The Approval screen should show all of the data from the imported list and any pendings changes you made using the Clean Speak interface.

Multilingual Filtering

Clean Speak™ currently supports basic filtering in any language. If you add a word or phrase in a language other than English to Clean Speak™ using the Management Interface, the Clean Speak™ Real-Time Filter will find those words and phrase.

Additionally, Clean Speak™ is capable of conjugating in French and Spanish. It however does not currently support embedding in those languages. Therefore, when you add a new word in French or Spanish using the Management Interface, the "Embeddable", "Collapsible" and "Distinguishable" checkboxes will be hidden.

To add a new word or phrase to your blacklist, click on the Filter > Manage Lists menu option and then click on the blacklist you want to edit. Next, click the ADD ENTRY button and add the information about your new word or phrase to be blacklisted. When you specify the Locale of the entry, make sure that it is the correct ISO language code for the language the word is in.

For example, if you are adding the word "merde", you will want to set the Locale to be "fr" for French.