Code Analyzer

Version 1.12

[User Guide]

Kieran Greer,
Email: help@distributedcomputingsystems.co.uk.
http://distributedcomputingsystems.co.uk/codeanalyzer.html

Table of Contents

1 Introduction

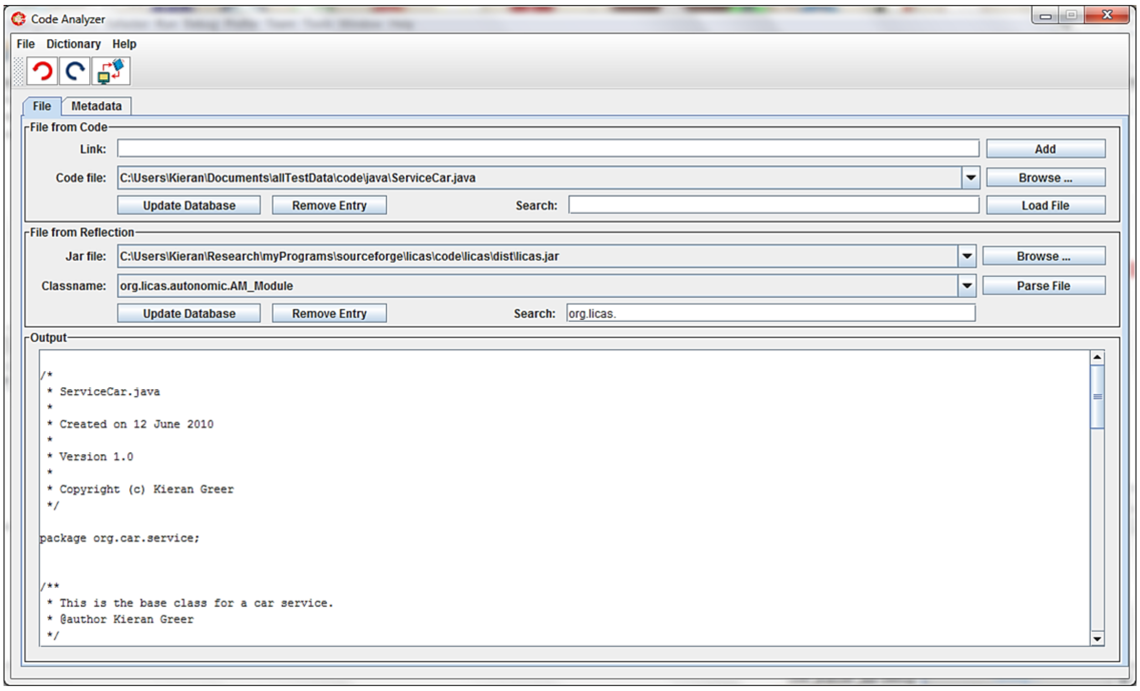
This document describes the code analyzer and semantic mapping application. This application is able to read a source code file and parse it, to extract the method descriptions. This information is then used to allow the user to enhance these method descriptions with standard definitions of words from online dictionaries. This new metadata can be used in two different ways. If you have a web service, you can automatically generate enhanced descriptions of the main concepts or terms in the service interface. The methods exposed as the interface over the Internet can be added to a script with the enhanced definitions. The method name, return type and parameter descriptions can all be updated with additional definitions or keywords, that can then be read and understood by another program or human user. While some knowledge and skill is required to determine what keywords and definitions to use, this application largely automates the process, making it much easier for the writer of the service to do this. One advantage of this mechanism is the fact that the descriptions are based on standard definitions retrieved from online dictionaries. These can even be checked if the web address is known. This is therefore a ready-made standard for defining your web service interface that can be utilised relatively easily. The application can also return definitions in different languages, if the online source provides this.

1.1 Installing the Application

The application is provided as a self-installing executable. Just run the installer and follow the instructions to install the application into the desired directory. You can then access it through the start menu or desktop shortcut. The installer also creates a folder in your root user directory. This folder is called `caData` and contains important `config` files and information. You should not delete or move it, or change any of the files in it. You can add new files and use it as your data repository if you wish, but any re-installation will delete this directory and so you should perform a backup of any new files that you add before any reinstallation. Only the folders and files that were originally installed will be deleted though.

2 File Panel

The first panel is an admin panel that allows you to browse for a file and display the contents. As described in the next section, the source code can be loaded in, either from a menu option under the 'File' menu, or by browsing to the file in the appropriate box and then loading in the selected file. This will read and display a text source. If the source is in `Java`, you can parse a jar file to read the classes stored in it. Any of the classes can then be selected and reconstructed for display. The application also now allows you to save a list of file paths, to a database. If you click the `Update Database` button for either the code or jar file sources, it will save the currently loaded list to their respective database file. When you start the application again, this list should be loaded in. You can then search over it by typing text into the search box, where any matching links will be displayed. It should also be possible to enter an online address and load in the resulting text. The database is therefore useful for storing these online addresses in one place. The `Remove Entry` option should then permanently remove the currently selected entry.



Link:

Code file: C:\Users\Kieran\Documents\allTestData\code\java\ServiceCar.java

Search:

Jar file: C:\Users\Kieran\Research\myPrograms\sourceforge\licas\code\licas\dist\licas.jar

Classname: org.licas.autonomic.AM_Module

Search: org.licas.

```
/*
 * ServiceCar.java
 *
 * Created on 12 June 2010
 *
 * Version 1.0
 *
 * Copyright (c) Kieran Greer
 */

package org.car.service;


/**
 * This is the base class for a car service.
 * @author Kieran Greer
 */
```

Figure 1. File Panel with a file loaded.

2.1 File from Code

With this area, you can either `Add` a file link manually, or use the `Browse` button to browse to a local file location. To view, you can then load in the file using the `Load File` button.

The file is read as a text or XML file and displayed in the text area. You therefore have the choice of making sure that an XML file can be parsed correctly, by forcing it to be read as XML first.

2.2 File from Reflection

If the source code is in `Java`, you can parse a jar file from this group area. You use the `Browse` button, to browse to a jar file location instead. The selected jar file is then parsed and all of the classes stored in it are listed in the `Classname` combo box. There is also a search box that can be used to filter the list of classnames further. Enter some text and only the classnames with that text will be displayed. Any of the classes can be selected and then reconstructed using Java Reflection. The class method information is then displayed in the text area after the `Parse File` button is pressed.

With Java reflection, there is no specific name assigned to any method parameter. Each parameter therefore needs to be assigned a unique name, where the following convention applies, with a few conditions:

- The return type is assigned the name of `Return_Type`.
- Each parameter is assigned the name of the method plus the parameter type, plus the parameter position. So for example, the second parameter in the method `makeCall`, which is of type String, would be given the name of `makeCall_java.lang.String_2`. The return type is then stored under the name `makeCall_Return_Type`, for example.
- One other assumption is that overloaded methods with more than one declaration in a class will have the same parameter at the same position, if it is of the same type. This is simply following the rules of Reflection however.

2.3 Menu Options

There is a relatively small selection of menu options, mainly for loading or saving files. The options that are available are as follows:

2.3.1 File Menu

This contains options for loading or saving files. The following options are available:

- **Code File:** this option allows you load a source code file in one go. This is instead of browsing to the file and then clicking the `Load File` button.
- **Jar File:** this options loads in a jar file path, but does not load in any code files. The selected path is then displayed in the `Jar file` text box. A class name from the box then needs to be selected before it can be loaded.

- **Save Script:** this saves the currently created script for the currently selected tab panel.
- **Exit:** this option terminates the application.

2.3.2 Dictionary

This contains options that allow you to configure what dictionaries to use:

- **Select …:** this option opens a new form that is described in section 2.5. The form can be used to change what dictionaries are displayed for selection and what languages are available for each dictionary.

2.3.3 Help

This contains options for displaying help or checking you applications version. The following options are available:

- **Online User Guide:** this option allows you load the online user guide into your browser for viewing.
- **Check for Updates:** this option allows you to check that your application version is the most recent. It compares you application's version number with the one specified on the web site. If there is a difference, then a message informs you of that.
- **About:** this opens an about box with some general information.

2.4 Toolbar

The application also comes with a toolbar for quick access to certain formatting options. A summary of these with their related button are shown next.

This button undo's the last operation, up to 5 of the previous operations.

This button redo's the last operation, up to 5 of the previous operations.

This button opens a form that allows you to configure what online dictionaries you use.

2.5 Select Dictionary Form

The application relies on a number of dictionaries for retrieving definitions and keywords about the selected concept terms. An online dictionary might not always be available and so you can configure what ones to display and what ones to hide through this form. There is now also the option to use either a local database or an online source and so at least one dictionary should always be available. The configuration is saved to a file and then loaded in each time the application is started. Figure 2 shows what this form looks like.

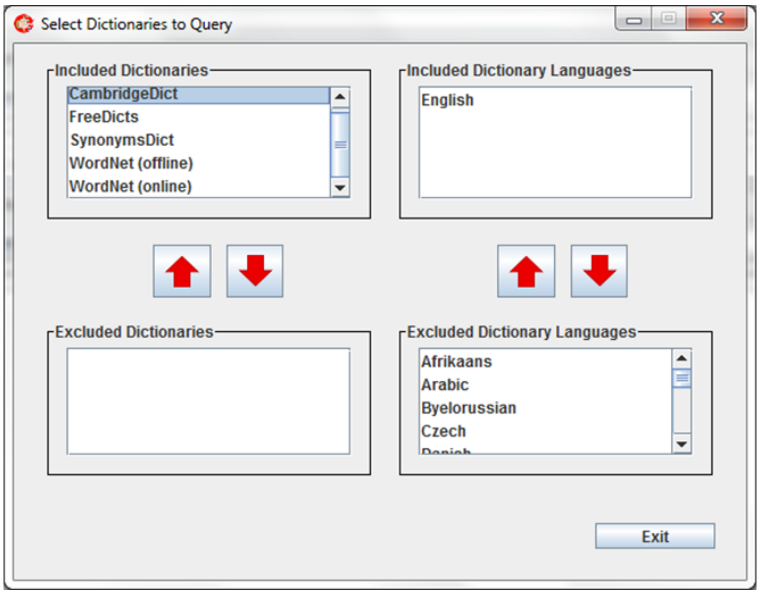


Figure 2. Dictionary selection form.

The Included dictionaries are shown in the `Included Dictionaries` list and excluded ones in the `Excluded Dictionaries` list respectively. If you select an entry, the related upward pointing arrow will move a dictionary from the excluded list to the included list. The related downward pointing arrow will move a dictionary from the included list to the excluded list. Only dictionaries in the included list will then be displayed. Any change is automatically saved and so when you exit, it is made permanent.

If you click on a dictionary in the include list, the languages associated with it are shown in the `Included Dictionary Languages` list. All other remaining languages are shown in the `Excluded Dictionary Languages` list. You can also move the languages in the same way through the related arrows. Then, when you then select a dictionary in the application, only the included languages will be displayed.

2.6 WordNet Dictionary

With the addition of WordNet [1], it is possible to query a local or offline database, as well as the online sources. This requires you to download and install WordNet first. If you then select to query with `WordNet(online)`, it will query the online web page and submit a query there. If you select however the option `WordNet(offline)`, then it will try instead to query using the local database. The path to this database needs to be set and if it is missing you are asked to browse to and select it. If a path is selected, it gets saved in the dictionary config file and so it only needs to be set once. You need to enter the path to the 'dict' folder in the WordNet installation, where after installing WordNet, the path could be:

```
C:\Program Files (x86)\WordNet\2.1\dict
```

If not, then browse to and select the correct folder path.

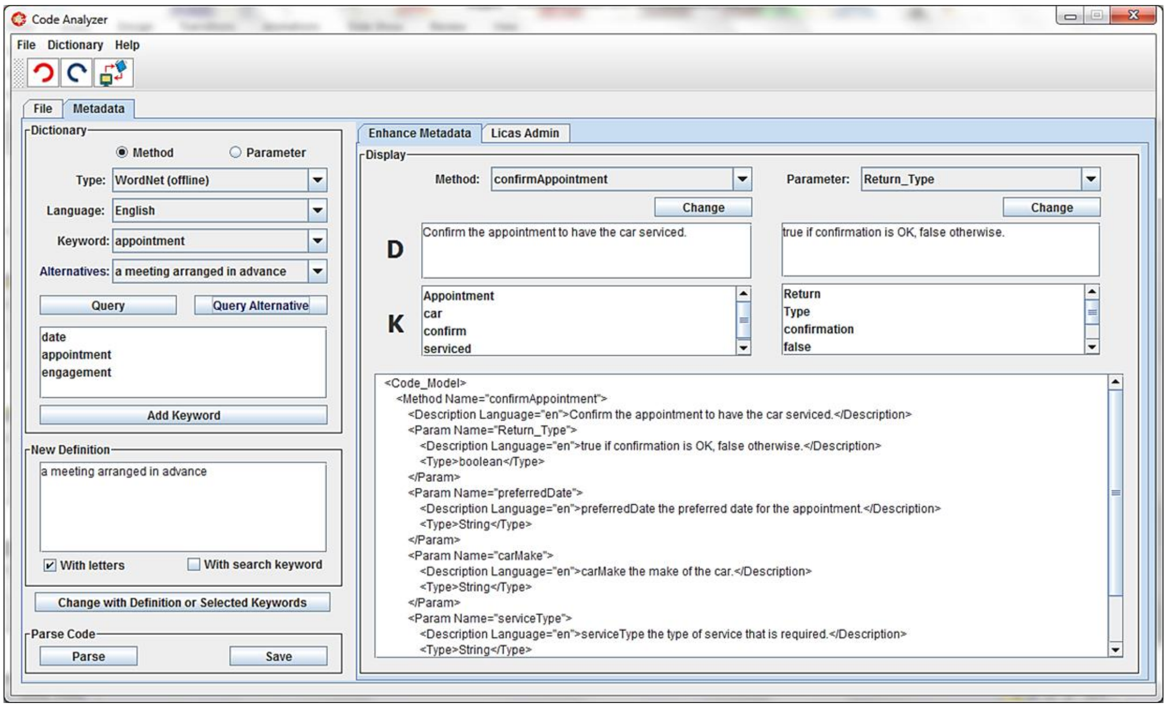
2.7 Dictionary List

The dictionary list may change. The HTML-parsed content will be removed when it becomes out of date and there will be an effort to use free online dictionaries that return in the json format instead. The following dictionaries are therefore available for querying:

- **Glosbe:** is an online source found at 'https://glosbe.com/gapi/'. It can only process queries in the English language and can also return synonyms.
- **SynonymsDict:** is an online resource found at http://www.synonym.com/. It can only process queries in English but does return synonyms.
- **WordNet (online):** is the online page of WordNet found at 'http://wordnetweb.princeton.edu/perl/webwn'. It can only process queries in English but does return synonyms.
- **WordNet (offline):** is the offline version of WordNet that queries the local database. It can only process queries in English but does return synonyms.

3 Metadata and Semantics

With this panel you can add detailed semantic descriptions about your source code files to a script that can be read by any system wanting to access your web service or online web interface. Figure 3 shows what this panel looks like. The application has been converted back to only generate descriptive metadata to enhance a web service description. The second licas tab may get added to a different program.



Dictionary

[✓ Method] Parameter

Type: WordNet (offline)

Language: English

Keyword: appointment

Alternatives: a meeting arranged in advance

date
appointment
engagement

New Definition

a meeting arranged in advance

[✓ With letters] With search keyword

Parse Code

Display

Method: confirmAppointment

Parameter: Return_Type

D

Confirm the appointment to have the car serviced.

true if confirmation is OK, false otherwise.

K

Appointment
car
confirm
serviced

Return
Type
confirmation
false

```
<Code_Model>
  <Method Name="confirmAppointment">
    <Description Language="en">Confirm the appointment to have the car serviced.</Description>
    <Param Name="Return_Type">
      <Description Language="en">true if confirmation is OK, false otherwise.</Description>
      <Type>boolean</Type>
    </Param>
    <Param Name="preferredDate">
      <Description Language="en">preferredDate the preferred date for the appointment.</Description>
      <Type>String</Type>
    </Param>
    <Param Name="carMake">
      <Description Language="en">carMake the make of the car.</Description>
      <Type>String</Type>
    </Param>
    <Param Name="serviceType">
      <Description Language="en">serviceType the type of service that is required.</Description>
      <Type>String</Type>
```

Figure 3: Semantic descriptions panel showing parsed code file and added descriptions.

3.1 Dictionary Panel

This panel allows you to retrieve dictionary definitions of keywords and add them to the description of your method. For the web service script, each description is added to the method or parameter description itself. In Figure 3 you can see some keywords that have been added at the top of the script. These relate to the method name itself. The dictionary panel contains the following boxes:

- The combo box at the top lists a number of online dictionaries that can be selected from. The online dictionaries can be queried either for English descriptions or other languages.
- The list of possible languages for each dictionary is shown in the `Language` combo box. That is where you select the language that you want the answer to be returned in.

- The word to be queried is shown in the `Keyword` combo box. This is a list of alternative words and you select the one that you want to query there. All of the words transferred to this list are converted into lower case, as this can affect how the online dictionary interprets them. Note that different languages might return different types of information, such as simply a language translation of a keyword, rather than a full dictionary definition. Some effort will be made to try and maintain currently allowed links. If these change then not all of the dictionaries will be available.
- After selecting the keyword, if you then click the `Query` button the selected dictionary is queried for a definition of the word. It could be the case that the keyword has more than one meaning. When that happens the dictionary will return several options and these will be listed in the `Alternatives` combo box. You then select the alternative that most closely matches your intended meaning and click the `Query Alternative` button. The dictionary will then return a definition of the keyword and also an optional list of synonyms. If there are no alternative meanings, then this information is returned after the initial query.

There are also some selection buttons at the top of this panel:

- If the `Method` button is selected, then the currently selected method description can be changed.
- If the `Parameter` button is selected, then the currently selected parameter definition can be changed.

From the keywords list, you can select any number of these to add as a definition to your script. They are added to the script using the large `Change with Definition or Selected Keywords` button. Note that some need to be selected or highlighted first. You can add keyword lists for definitions to each parameter or method. Each new description that is added is additional to the existing metadata. The new metadata is then displayed in the form of a keyword with related source and definition as follows:

- The keyword added is the one in the `Keyword` combo box, which is the one that was originally queried.
- The source is the URL of the online dictionary.
- The language is the code of the language in the `Language` combo box.
- The definition is either the definition returned by the query, in the `New Definition` text area; or a comma separated list of keywords, if they are chosen instead.

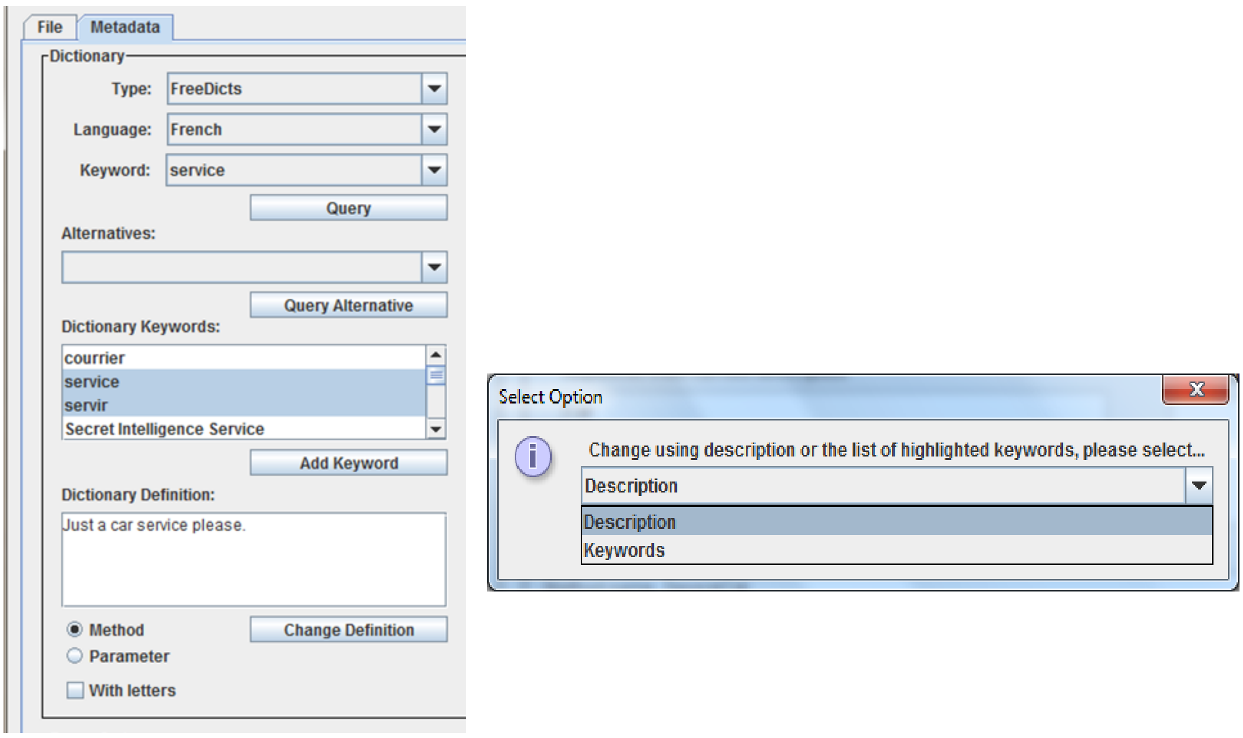
There are two further options as determined by the following check boxes:

- If you select the `With letters` check box, this makes sure that any concept or word that is included for selection as a keyword must have letters in it. In that case, words without any letters (only numbers or symbols) would not be allowed.

In Figure 3 the method's metadata is being looked at and the selected keyword is '`appointment`'. If you click the `Add Keyword` button however, the selected keywords are added back to the `Keyword` combo box list. These would typically be synonyms, or alternative words, for the intended one. These can then themselves be selected for definition retrieval. When you have added sufficient metadata descriptions to all of your methods and parameters, you can save the description as an XML script by clicking the `Save` button. This script or metadata description can then be read by another program and will describe your Web Service interface in great detail. Appendix B gives one example of what this script might look like, while section 4 describes how to use this script as part of your service interface.

3.1.1 Different Languages

It is possible to query the dictionary with different languages. The available languages are shown in the `Language` combo box. You select the language to query with from there. The dictionary however might return different information depending on what language is selected. The `FreeDicts` dictionary is one that is provided with different languages. If you query in English, for example, you will be returned a definition and possible list of synonyms. If you query in another language however, you are returned a list of keywords in the selected language. There may be no full description. To use these keywords, you will then possibly want to add them to your metadata description as alternative meanings and this can simply be done in the standard way. If there is a definition in the `New Definition` text area and also a keyword list, you are asked to select which list of words you want to use to update the metadata description. Figure 4 illustrates this sort of scenario. If only one of the areas has a list of selected words, then only that area will be used. This means however that you can give alternative meanings for English words, for example, as well as full definitions. It is also possible to select the keywords and add them to the `Keyword` list, to be queried directly, if a suitable source is available.


File | Metadata

Dictionary

Type: FreeDicts

Language: French

Keyword: service

Query

Alternatives:

Query Alternative

Dictionary Keywords:

- courrier
- service
- servir
- Secret Intelligence Service

Add Keyword

Dictionary Definition:

Just a car service please.

(●) Method ( ) Parameter

Change Definition

[ ] With letters

Select Option

Change using description or the list of highlighted keywords, please select...

Description

- Description
- Keywords

Figure 4: Keyword retrieval in a different language, with the option to select which definition to add to the metadata description.

3.2 Metadata for Web Services

The two different panels work in the same way with respect to adding keyword definitions. You need to select a panel first – `Enhance Metadata` in Figure 3. If you then click the `Parse` button, the source code file that you have chosen is parsed and added to the different boxes in that panel for selection purposes. The following sections describe how you can then generate the additional metadata through the functionality provided. Note that some of the text areas also allow for direct copy and paste, which is another way to transfer text from one place to another.

3.3 Parse the Source File

To parse the file you need to click the `Parse` button in the `Parse Code` box. This opens another box to allow you to select the 'file link' or the 'jar file class'. This is because the admin panel with the source file path can have both entries. Select which one is relevant to your source file. Note that only Java classes can be reconstructed through the jar files, but the source code option can read other types of source as well. The file type is defined by its extension, where the following types are currently supported:

- A WSDL file is defined with a `.wsdl` or `.xml` extension. The encoding for the WSDL file should be set to 'UTF-8'.
- A Java file is defined with a `.java` extension.
- A .Net C# file is defined with a `.cs` extension.
- A .Net Visual Basic file is defined with a `.vb` extension.

3.4 Display Panel

This panel displays the created metadata script, but for one method only. The text area at the bottom shows the current metadata description of the code in script format. That is, the XML-based description of the method that is currently selected is shown with any additions of keyword definitions, etc., as it would be saved in the script. The left-hand side of the `Display` panel relates to the method and the right-hand side to its parameter list. For either, there is a general description (in the `D` box) and a list of keywords (in the `K` box). These are selected initially from the metadata tags in the source code file, but can then be changed or updated. For example, you can manually change the method or parameter descriptions and then click the related `Change` button. After confirmation you will see the change as part of the new string-based description in the text area.

3.4.1 Assumed Coding Rules

The method or parameter name itself is also used as part of the keyword list that is queried. It is assumed that coding principles are being used and that separate words in either of these cases can be identified as follows:

- The next word would start with a new capital letter.
- The words are separated by a symbol that is not a letter or a number.

The method or parameter name is parsed based on this separating criterion and the parsed words added to the related keywords list. Some of the more commonly occurring words are automatically removed and not included as keywords. These can be found in the `wordsToRemove.txt` file in the `files` folder in the `caData` data folder. The included keywords are also transferred over to the dictionary panel so that they can be queried for further descriptions. This process is described next.

3.5 Save the Metadata Script

When the metadata script has been constructed, you can click the `Save` button to save it as an XML script. This script can then be used as part of your web interface.

4 Using the Script as part of a Web Interface

Section 3 described how you can create a semantic or metadata script that can be used to describe your web service interface in a standard and more detailed way. This script can be saved to a file, when you will then want to use it as part of your service deployment. To help you with this, the package comes with an additional library package called `ca_xmlscript.jar`, also for the Java environment. This can now be downloaded from the web site at http://distributedcomputingsystems.co.uk/codeanalyzer.aspx#ca_xmlscript. The package is self-contained and so you just need to add it to your web service library path. You then need to update your web service interface as described in the following sections. There is then a similar package for the .Net environment called `ca_xmlscript.dll` that is also downloaded in the same zip file. Alternatively, the script contents are clear, so it would not be too difficult to write your own parser or integrate the script into your interface in some other way. Following is one suggestion on how to use it.

4.1 Web Service Interface

This example uses the Java programming language, but it applies equally to a .Net web service, where the exact same procedures should be used. Your web service interface will typically be a code file that describes the methods that can be invoked on our service. There is an additional package provided with the application called `ca_xmlscript`. This package contains a class called `ScriptParser` that provides two methods – one to return a method description and one to return a parameter description. These methods, plus a third one to load in the script, look like:

```
/**
 * Parse the script to retrieve the relevant method information.
 * @param methodName the name of th emethod to retrieve information about.
 * @param language the language the description should be written in.
 * @return the method description.
 * @throws java.lang.Exception any error.
 */
public static String describeMethod(String methodName, String language)
throws Exception


/**
 * Parse the script to retrieve the relevant method information.
 * @param methodName the name of the method the parameter belongs to.
 * @param parameterName the name of the parameter to retrieve information
about.
 * @param language the language the description should be written in.
 * @return the parameter description.
 * @throws java.lang.Exception any error.
 */
public static String describeParameter(String methodName, String
parameterName, String language) throws Exception


/**
 * Read the script and create the code model from it.
 * @param filePath the full path to the file to load.
 * @throws java.lang.Exception any error.
 */
public static void loadScript(String filePath) throws Exception
```

You can create your web service interface independently of these methods, but you should provide at least two additional public methods in the interface - one to allow the user to retrieve a method description and one to allow a user to retrieve a parameter description. These should be based on the `describeMethod` and `describeParameter` examples just shown. For an EJB web service, the implementation of the `describeMethod` method might then look like:

```
/**
 * Web service operation
 */
@WebMethod(operationName = "whatIsThisMethod")
public String whatIsThisMethod(@WebParam(name = "methodName")
    final String methodName, @WebParam(name = "language")
    final String language)
{
    try
    {
      ScriptParser.loadScript("./scripts/ServiceCar.xml");
      String reply = ScriptParser.describeMethod(methodName, language);
      return reply;
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }

    return null;
}
```

So you declare the method interface when you create your web service and then you can add code to the method body similar to what has just been shown afterwards. This implementation simply tries to load in a script and then parse it to read the part relating to the method with the specified name and in the specified language. You can specify the file path and name of the script as part of this code, or some other way. In this example it is placed in a scripts file in the root execution directory and is called `ServiceCar.xml`.

Now when anybody invokes this method on your web service, a detailed xml description will be returned in String format. This is the description that you created with the Code Analyzer application. This can then be read by the calling program or user, as it is in XML format. You can of course change this code in any way that you wish – you only need to make the `ScriptParser` method calls the same. To retrieve a parameter description, the code might look like:

```
/**
 * Web service operation
 */
/**
 * Web service operation
 */
@WebMethod(operationName = "whatIsThisParameter")
public String whatIsThisParameter(@WebParam(name = "methodName")
    final String methodName, @WebParam(name = "parameterName")
    final String parameterName, @WebParam(name = "language")
    final String language)
{
    try
    {
        ...

        ScriptParser.loadScript("./scripts/ServiceCar.xml");
        String     reply     =     ScriptParser.describeParameter(methodName,
parameterName, language);

        ...
    }
}
```

The `loadScript` method call only needs to be made once and stores a permanent description of the web service interface (the specified script) in a code model. Calling it multiple times however does no harm. So to summarise:

1. You will probably need to generate the Web Service interface first to produce a WSDL script or source code file, for example.
2. You can then parse that in the Code Analyzer application to generate the enhanced metadata description.
3. You then use this enhanced script with the stand-alone library package and add the method body implementation details to your service to allow the script details to be retrieved.

This will enhance the descriptive information about your web service to a level that might even make it possible for other computers to understand it. One other additional method that you might want to add would return the different languages that information can be retrieved in. The user would enter a country or language and receive the related language code. This code value would then be used to query the metadata. The method could look like:

```
public String getLanguages()
```

This could return an xml-based description that looks something like:

```
<Languages>
  <Language Name="English">en</Language>
  <Language Name="French">fr</Language>
  <Language Name="German">de</Language>
  <Language Name="Spanish">es</Language>
</Languages>
```

This however is very application dependent and so you will need to add this method yourself. Examples of language codes can be found at http://www.loc.gov/standards/iso639-2/php/English_list.php.

References

[1]WordNet, (2013), http://wordnet.princeton.edu/wordnet/.

Appendix A – Example Metadata Script

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<Code_Model>
	<Method Name="confirmAppointment">
		<Description Language="en">Confirm the appointment to have the car serviced.</Description>
		<Keywords>
			<Keyword Name="Appointment" Language="en">
				<Source>http://www.memidex.com/</Source>
				<Description>a meeting arranged in advance</Description>
			</Keyword>
			<Keyword Name="confirm" Language="en">
				<Source>http://www.memidex.com/</Source>
				<Description>to strengthen or make more firm</Description>
			</Keyword>
		</Keywords>
		<Parameter Name="Return_Type">
			<Description Language="en">true if confirmation is OK, false otherwise.</Description>
			<Type>boolean</Type>
			<Keywords>
				<Keyword Name="confirmation" Language="en">
					<Source>http://www.memidex.com/</Source>
					<Description>information that confirms or verifies</Description>
				</Keyword>
			</Keywords>
		</Parameter>
		<Parameter Name="preferredDate">
			<Description Language="en">preferredDate the preferred date for the appointment.</Description>
			<Type>String</Type>
			<Keywords>
				<Keyword Name="Date" Language="en">
					<Source>http://www.memidex.com/</Source>
					<Description>the specified day of the month</Description>
				</Keyword>
				<Keyword Name="preferred" Language="en">
					<Source>http://www.memidex.com/</Source>
					<Description>more desirable than another</Description>
				</Keyword>
			</Keywords>
		</Parameter>
		<Parameter Name="carMake">
			<Description Language="en">carMake the make of the car.</Description>
			<Type>String</Type>
			<Keywords>
				<Keyword Name="Make" Language="en">
					<Source>http://www.memidex.com/</Source>
					<Description>a recognizable kind</Description>
				</Keyword>
				<Keyword Name="car" Language="en">
					<Source>http://www.memidex.com/</Source>
					<Description>a motor vehicle with 4 wheels usually propelled by an internal combustion engine</Description>
				</Keyword>
			</Keywords>
		</Parameter>
		<Parameter Name="serviceType">
			<Description Language="en">serviceType the type of service that is required.</Description>
			<Type>String</Type>
			<Keywords>
				<Keyword Name="Type" Language="en">
					<Source>http://www.memidex.com/</Source>
					<Description>a subdivision of a particular kind of thing</Description>
				</Keyword>
				<Keyword Name="service" Language="en">
					<Source>http://www.memidex.com/</Source>
					<Description>to make fit for use</Description>
				</Keyword>
			</Keywords>
		</Parameter>
	</Method>
</Code_Model>
```