How to Create Language Services

Update 1/14/2014

NICT Language Grid Project

http://langrid.nict.go.jp

Table of Contents

1. Introduction....1
2. What is wrapper?....1
3. Development Environment Settings....2
3.1. JDK setup....2
3.2. Tomcat setup....2
3.3. Eclipse setup....2
3.4. Dowload and setup wrapper libraries....5
4. Wrapper Creation Process....8
4.1. Implementation of a wrapper....8
4.2. Wrapper test....13
4.3. Web service test....17
4.4. Create a deployment package....22
5. Use of Abstract Classes....23
5.1. Outline....23
5.2. List of Service Types....26
6. Using the Database-based Bilingual Dictionary and Parallel Text....28
6.1. Acquiring the DB Wrapper Package....29
6.2. Creating a Database for Stored Language Resources....29
6.3. Set up tables....30
6.4. Deployment of DB Wrapper....33
7. Composite service creation....35
8. Inquiries....35

1. Introduction

The NICT Language Grid Project allows cooperation among Internet-based language resources for the purpose of building a multi-language service base, or Language Grid. To do this, it is necessary to wrap various language resources existing on the Internet (machine translation engines, dictionaries, etc.) and enable Web services. To support these Web services, a NICT service interface is defined in the Language Grid Project, and wrapping libraries are provided for the purpose of developing Web services using the Java language. This manual gives descriptions of how to convert a language resource into Web service, using the wrapping libraries.

First, Section 2 will explain the procedures for configuring the development environment, and Section 3 will describe the wrapper build process. Descriptions of the abstract classes that are used in building a wrapper are provided in Section 4, and finally Section 5 explains methods for handling basic Internet resources. At the last, Section 6 will give the descriptions of how to wrap using the database.

2. What is wrapper?

A wrapper is a program which makes language resources (programs and data) accessible through Web services, and adjusts input and output of language resources to defined input/output using the NICT Language Service Interface. A wrapper is deployed on the language grid service node, and accepts requests from the grid core node. When it receives a request from the core node, it accesses language resource data within the wrapper or accesses language resources running as server programs outside the service node by using HTTP or SOAP, and returns a result to the core node, formatting the necessary data in the NICT Language Service Interface output format.

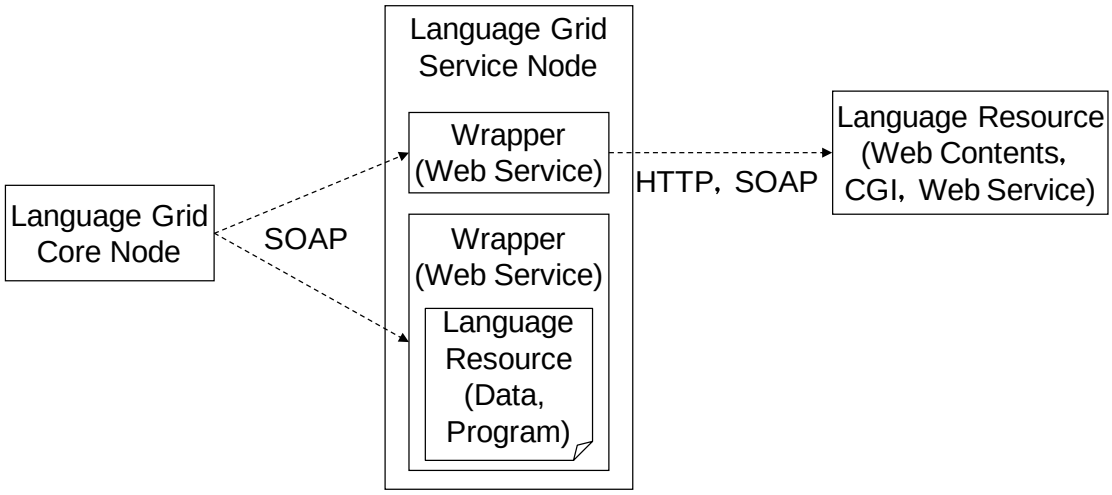


Fig. 1 Configuration diagram of Wrapper

3. Development Environment Settings

This section details the necessary development environment configuration for wrappers. Configure the development environment using the following steps.

3.1. JDK setup

Use JDK 6.0 or later to build a wrapper. Download the kit from the following site and install it on your computer.

- http://www.oracle.com/technetwork/java/index.html

3.2. Tomcat setup

Use Tomcat as web container for wrappers. Install Tomcat 6.0 or later to deploy wappers . Dowload tomcat from the following site and follow the mamual to install it on your computer.

- http://tomcat.apache.org/download-60.cgi

3.3. Eclipse setup

Eclipse3.6 (Helios) is used for wrapper development. First, download the application from the following site and install it on your computer.

- http://www.eclipse.org/downloads/

3.3.1. Setting the Eclipse Java compiler to 1.6

Set the Eclipse compiler compliance level to 1.6 (Change in "Window→Preferences→Java →Compiler→Compiler compliance level.") (Fig. 2) Use 1.6 or later when building all wrappers.

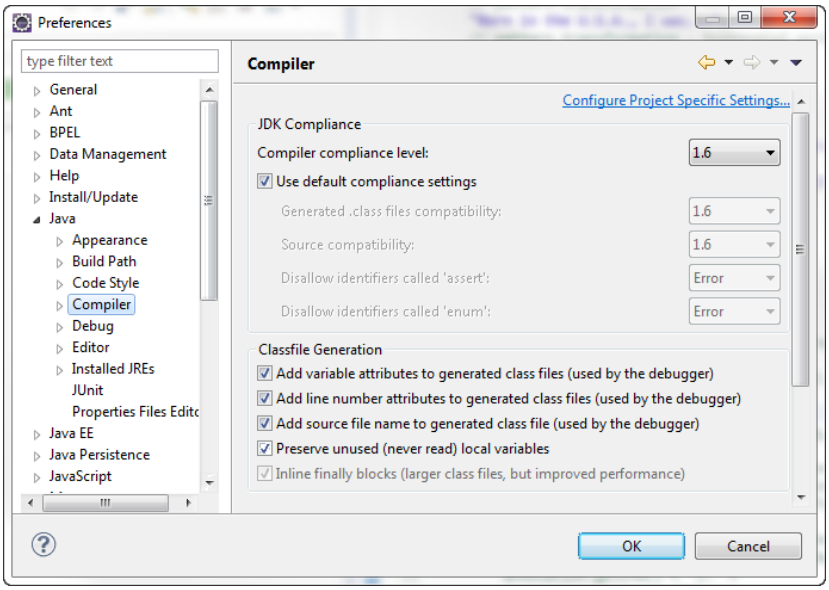


Fig. 2 Eclipse Setup

3.3.2. Setting the Eclipse file encoding to UTF-8.

Set the Eclipse text file encoding to UTF-8 (Change in "Window→Preferences→General →Workspace→Text file encoding.") (Fig. 3)

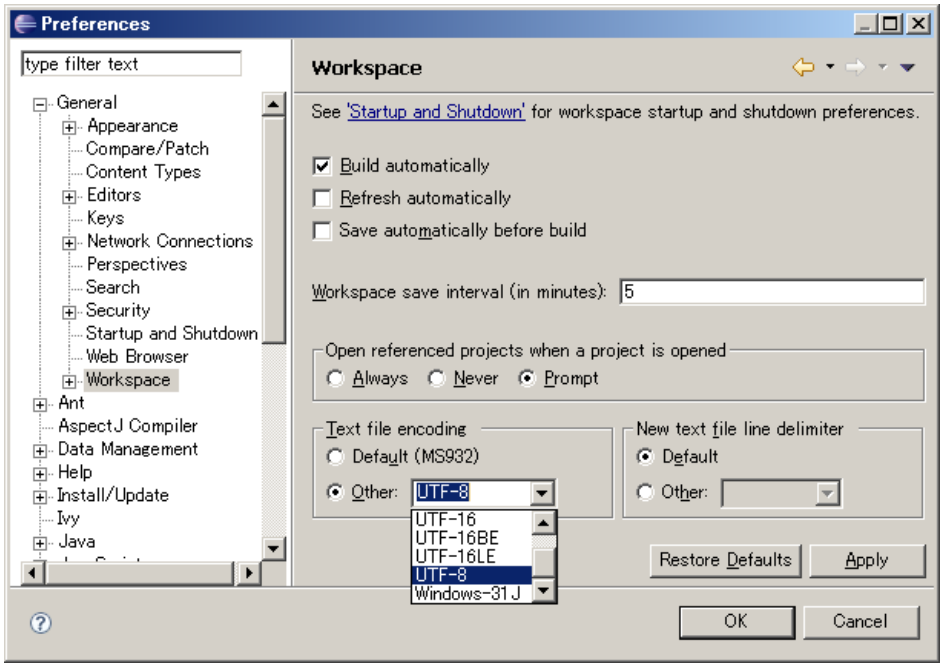


Fig. 3 Eclipse Setup

3.3.3. Server Configuration

Firstly add Tomcat web server in Eclipse. Start Eclipse and press CTRL + N, select Server from wizard list (see Fig. 4).

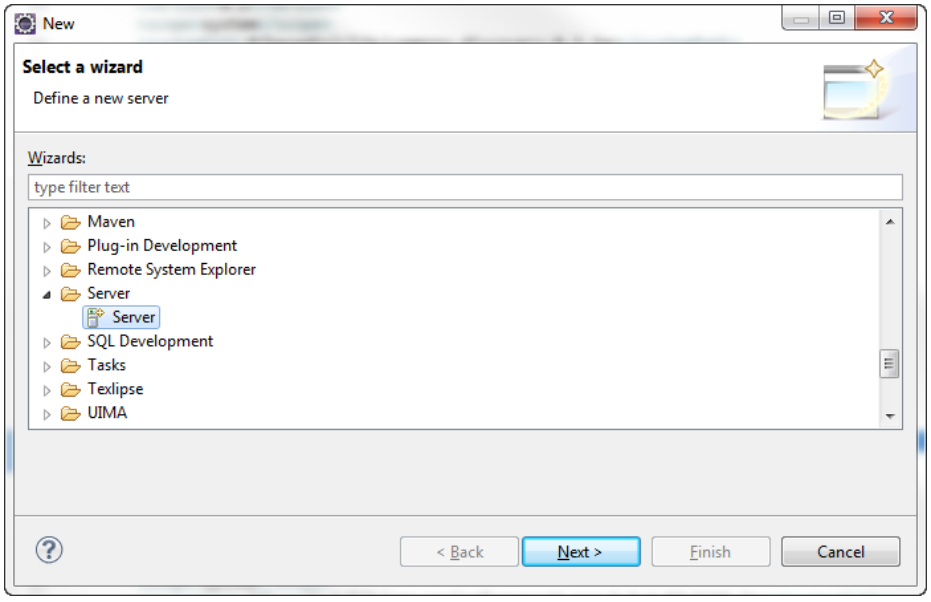


Fig. 4 Server Configuration

Click next and choose Apache Tomcat 6.0 (see Fig. 5)

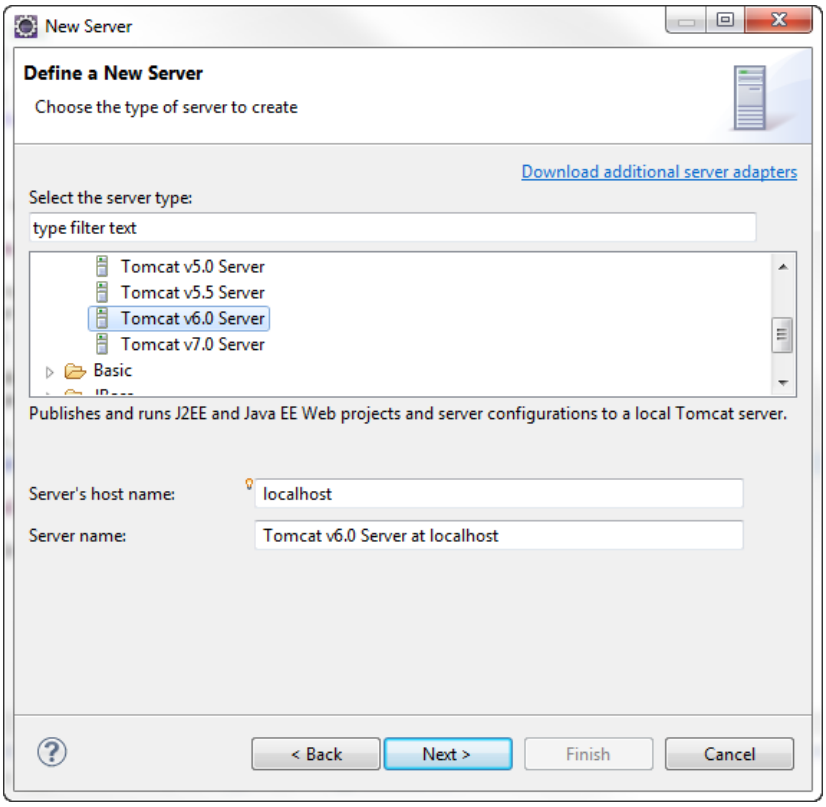


Fig. 5 Select Apache Tomcat Server

Then set Tomcat installation directory to the directory that you installed tomcat in section 3.2 (see Fig. 6)

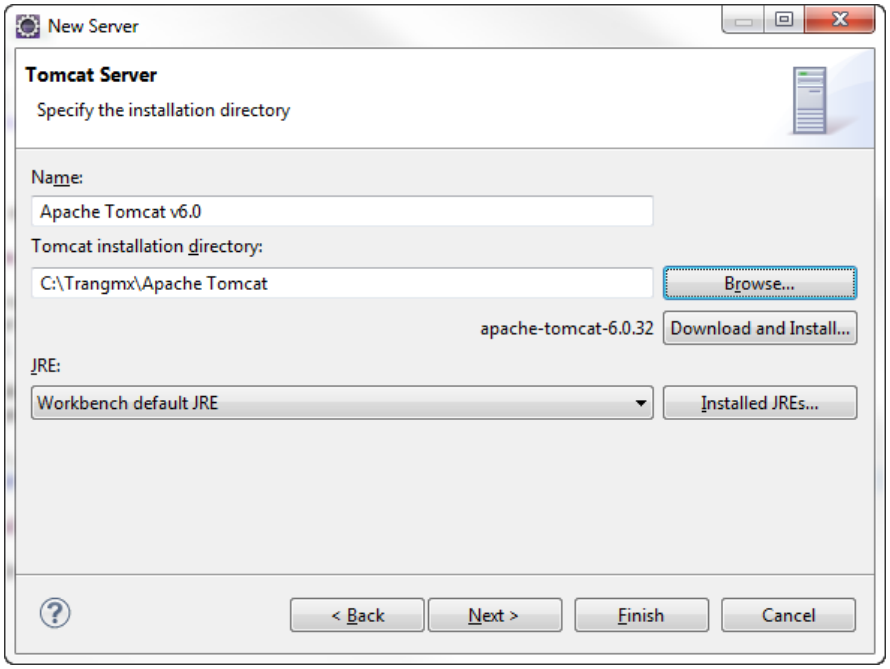


Fig. 6 Select Tomcat installation directory

Click Finish, server configuration for Apache Tomcat in Eclipse will be created (see Fig. 7)

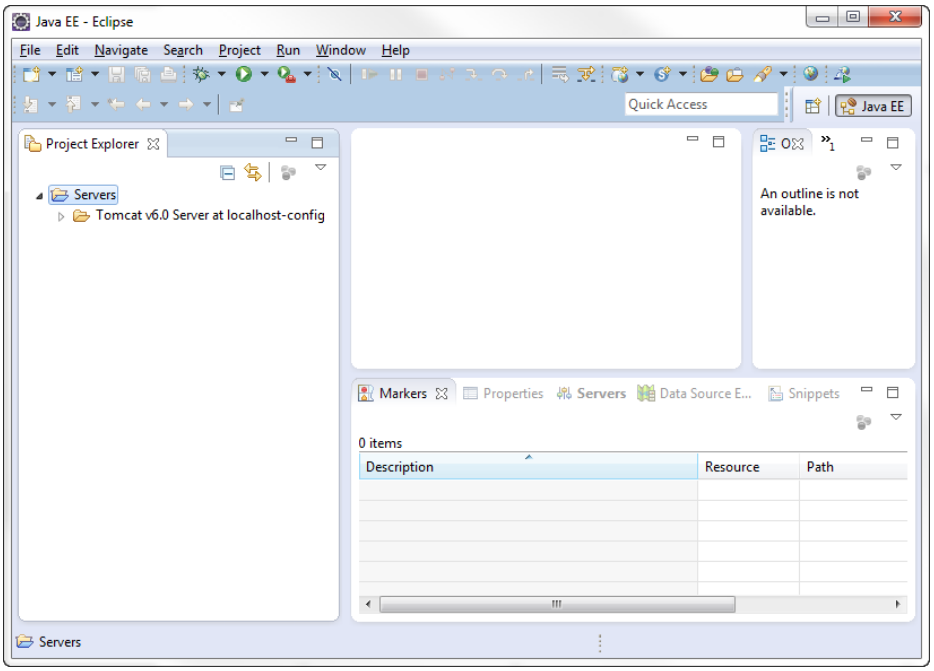


Fig.7 Server Configuration

3.4. Dowload and setup wrapper libraries

This section describes how to download and setup libraries for deploping and testing wrappers.

3.4.1.Wrapper library

Download wrapping library package (langrid-webapps-blank-20130625.zip) from sourcefoge (http://sourceforge.net/projects/servicegrid/files/Language%20Grid/). Unzip the zip file to a folder in your computer.

Import the project into Eclipse. Select File → Import , and choose "Select an import source: General→ Existing Projects into Workspace" and click Next (see Fig. 8)

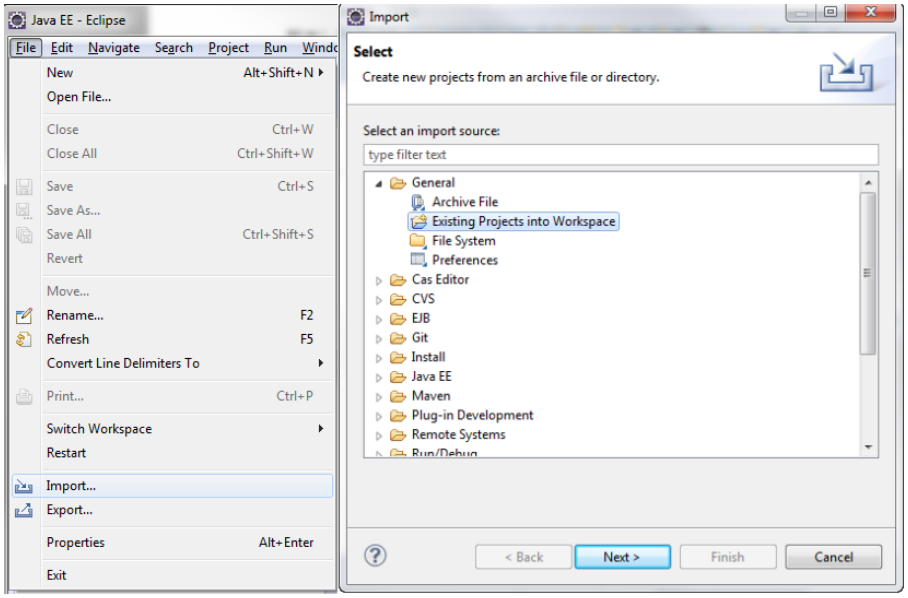


Fig. 8 Import wrapper library as eclipse project 1

Set "Select root directory" to the directory where you unzip the wrapper library. Check "Copy projects into workspace" to copy this project into your Eclipse working space. Click Finish to import the project (see Fig. 9).

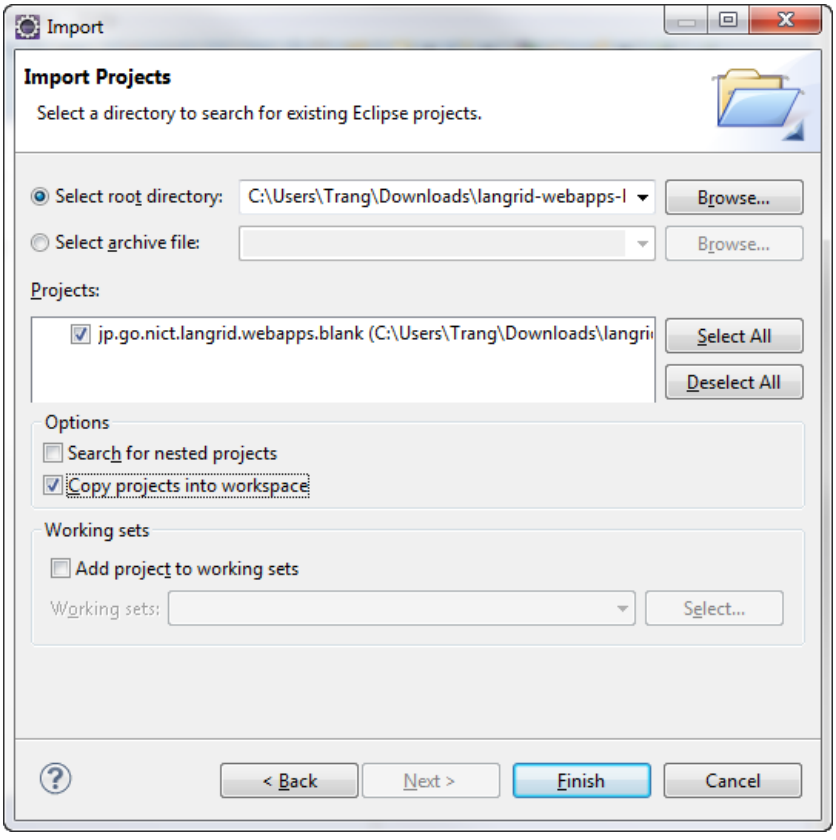


Fig. 9 Import wrapper library as eclipse project 2

The wrapper library is imported as project in Eclipse looks like in Fig. 10.

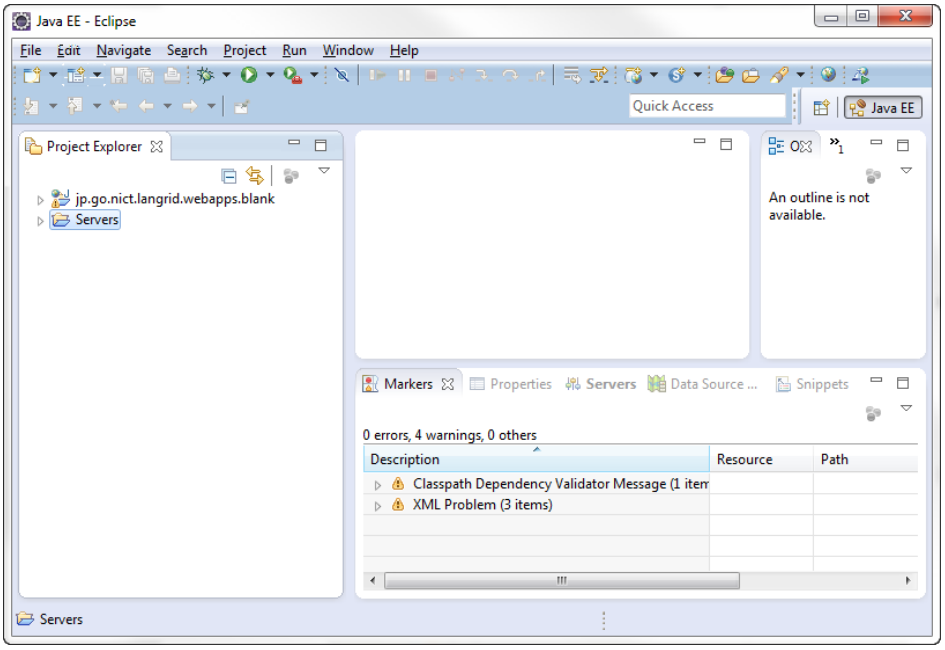


Fig. 10 Import wrapper library

3.4.2. Client library

NICT also provide a client library to test wrappers. Download langrid-clientapps-blank-20130221.zip package from sourceforge (http://sourceforge.net/projects/servicegrid/files/Language%20Grid/). Unzip the package and import as project into Eclipse by the same way as importing wrapper library. After importing projects in eclipse look like in Fig. 11.

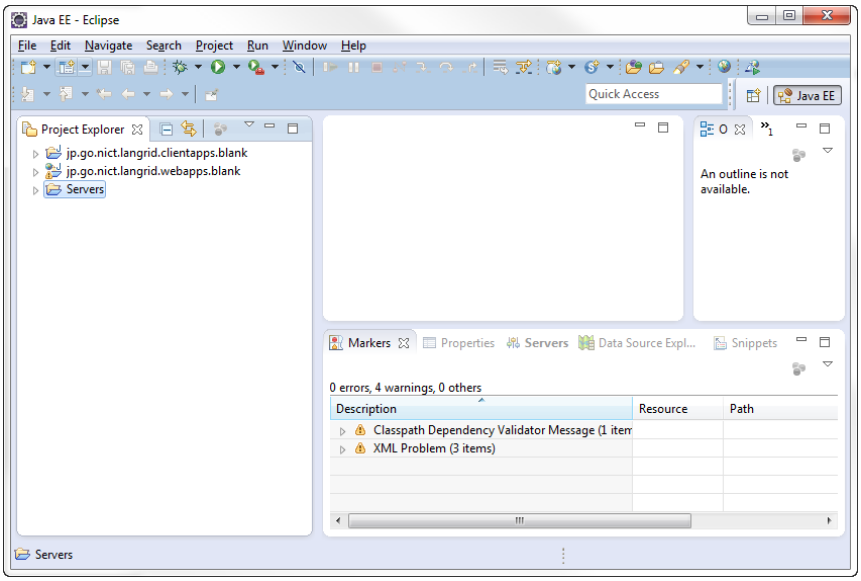


Fig. 11 Import Client library

4. Wrapper Creation Process

This section will explain the rough process for building a wrapper.

4.1. Implementation of a wrapper

As an example, this section will explain how to implement a simple parallel text service , and how to test the service.

4.1.1. Create a class

Create a class that implements the wrapper. In this class, the language resource is being wrapped. This class extends a corresponding abstract class. Let's say we going to wrap a parallel text language resource, here we create a SampleParallelTextService class extending the AbstractParallelTextService. In Eclipse choose "jp.go.nict.langrid.webapps.blank" project press CTRL+N and create a new java class SampeParallelTextService (see Fig. 12)

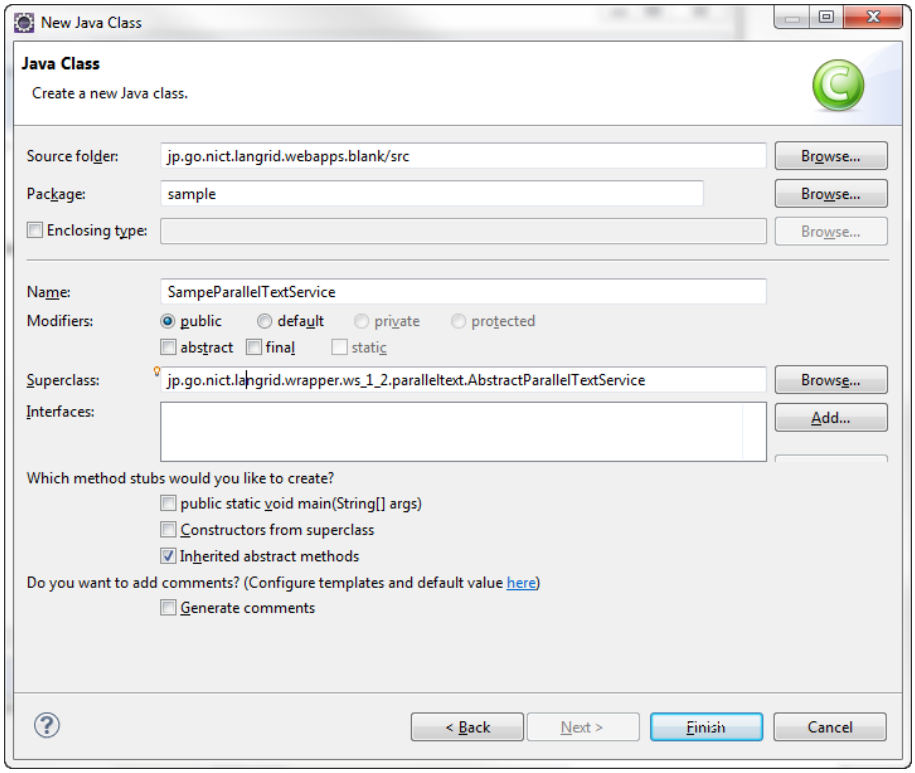


Fig. 12 Create a sample class

The initial source code of the class is automatically generated as in Fig. 13

```java
package sample;

import java.util.Collection;

public class SampeParallelTextService extends AbstractParallelTextService {

    @Override
    protected Collection<ParallelText> doSearch(Language arg0, Language arg1,
            String arg2, MatchingMethod arg3) throws InvalidParameterException,
            ProcessFailedException {
        // TODO Auto-generated method stub
        return null;
    }

}
```

Fig. 13 Generated source code

When developing a wrapper, only the part of the method that actually does any processing should be implemented (in the case of the sample translation wrapper, the doSearch method). The parameter checks and standard methods (getSupportedLanguages, etc.) are implemented by the abstract class. Abstract classes corresponding to the specific details of the implementation provided by the abstract class and to each language interface are explained in Section 5.

As an example of a wrapper implementation, with English to Japanese and Japanese to English as the language pair, a sample translation service from "こんにちは" to "hello" is implemented, employing a matching method corresponding to full matches (COMPLETE), partial matches (PARTIAL), prefix matches (PREFIX), suffix matches (SUFFIX) and regular expressions (REGEX).

4.1.2. Constructor implementation

A constructor (followed by the declared init method) is displayed in Fig. . For all source codes including import sentence, see 4.1.4.

```java
/**
* Constructor.
*/
public SampleParallelTextService(){
 // Supports Japanese->English and English->Japanese
 // MatchingMethod supports COMPLETE, PARTIAL, PREFIX, SUFFIX, REGEX.
 setSupportedLanguagePairs(Arrays.asList(
  new LanguagePair(en, ja)
  , new LanguagePair(ja, en)
  ));
 setSupportedMatchingMethods(new HashSet<MatchingMethod>(
  Arrays.asList(COMPLETE, PARTIAL, PREFIX, SUFFIX, REGEX)));
}
```

Fig. 14 Contents of the Constructor

In the constructor, the setSupportedLanguagePairs is declared, and the corresponding language pairs are specified. This method takes a collection (java.util.Collection) of a LanguagePair (jp.go.nict.language.LanguagePair) into the argument. Using the asList method from the Arrays (java.util.Arrays) class, it generates a LanguagePair collection and passes it to setSupportedLanguagePairs. setSupportedMatchingMethods is capable of designating a utilizable matching method, and takes java.util.Set<jp.go.nict.langrid.service_1_2.typed.MatchingMethod> as an argument. Like setSupportedLanguagePairs, it uses the asList method from the Arrays (java.util.Arrays) class to generate a MatchingMethod collection, and passes it to setSupportedMatchingMethods. In this example, it is being set to support COMPLETE (complete matches), PARTIAL (partial matches), PREFIX (prefix matches), SUFFIX (suffix matches) and REGEX (regular expressions).

4.1.3. Implementing abstract methods

Next, the actual method that does the processing necessary for the language resource to be utilized is implemented (doSearch, in the case of the sample translation wrapper). In this example, only the contents of the sample translation, "犬" and "hello", are handled. Fig. 4 shows the code.

```
@Override
protected Collection<ParallelText> doSearch(Language sourceLang,
 Language targetLang, String source, MatchingMethod matchingMethod)
 throws InvalidParameterException, ProcessFailedException {
List<ParallelText> result = new ArrayList<ParallelText>();
for(ParallelTextResource r : resources){
 if(matches(r, sourceLang, source, matchingMethod)){
  result.add(new ParallelText(
   r.get(sourceLang)
   , r.get(targetLang)));
 }
}
return result;
}

private static class ParallelTextResource{
 public ParallelTextResource(String enText, String jaText){
  this.enText = enText;
        this.jaText = jaText;
 }

 public String get(Language language){
  if(language.equals(en)){
   return enText;
  } else{
   return jaText;
  }
```

```
 }
 private String enText;
 private String jaText;
}

private boolean matches(ParallelTextResource resource, Language language
 , String source, MatchingMethod matchingMethod){
 String text = resource.get(language);
 switch(matchingMethod){
  default:
  case COMPLETE:
   return text.equals(source);
  case PREFIX:
   return text.startsWith(source);
  case SUFFIX:
   return text.endsWith(source);
  case PARTIAL:
   return text.indexOf(source) != -1;
  case REGEX:
   return text.matches(source);
 }
}

private ParallelTextResource[] resources = {
 new ParallelTextResource("Hello.", "
 };
```

Fig. 4 Search Method Contents

ThisdoSearchmethodreceivesen/jaandja/enfortheargumentssourceLangandtargetLang, and COMPLETE,PARTIAL,PREFIX,SUFFIX,and REGEX for the matchingMethod. When other parameters besides these are designated, they are detected by the abstract class parameter check, and an exception is returned.

In the doSearch method, the ParallelTextResource-type sample translation data are read one at a time, and checked with a matching method to determine whether there are samples that match the query using the designated matching method. If there is a matching sample, it generates an instance of ParallelText and adds it to an array for output. The instance of ParallelText consists of an English to Japanese pair when sourceLang is en, and a Japanese to English pair when it is ja.

4.1.4. Complete wrapper source code

The complete code for the SampleParallelTextService class is shown in Fig. 5.

```
package sample;

import static jp.go.nict.langrid.language.ISO639_1LanguageTags.en;
import static jp.go.nict.langrid.language.ISO639_1LanguageTags.ja;
```

```
import static jp.go.nict.langrid.service_1_2.typed.MatchingMethod.COMPLETE;
import static jp.go.nict.langrid.service_1_2.typed.MatchingMethod.PARTIAL;
import static jp.go.nict.langrid.service_1_2.typed.MatchingMethod.PREFIX;
import static jp.go.nict.langrid.service_1_2.typed.MatchingMethod.SUFFIX;
import static jp.go.nict.langrid.service_1_2.typed.MatchingMethod.REGEX;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.List;
import java.util.HashSet;

import jp.go.nict.langrid.commons.ws.ServiceContext;
import jp.go.nict.langrid.language.Language;
import jp.go.nict.langrid.language.LanguagePair;
import jp.go.nict.langrid.service_1_2.InvalidParameterException;
import jp.go.nict.langrid.service_1_2.ProcessFailedException;
import jp.go.nict.langrid.service_1_2.paralleltext.ParallelText;
import jp.go.nict.langrid.service_1_2.typed.MatchingMethod;
import
jp.go.nict.langrid.wrapper.ws_1_2.paralleltext.AbstractParallelTextService;

/**
* Sample of Parallel Text Srvice.
*/
public class SampleParallelTextService extends AbstractParallelTextService {
 /**
 * Constructor.
 */
 public SampleParallelTextService (){
 // Supports Japanese->English and English->Japanese.
 // MatchingMethod supports COMPLETE, PARTIAL, PREFIX, SUFFIX, REGEX.
 setSupportedLanguagePairs(Arrays.asList(
  new LanguagePair(en, ja)
  , new LanguagePair(ja, en)
  ));
 setSupportedMatchingMethods(new HashSet<MatchingMethod>(
  Arrays.asList(COMPLETE, PARTIAL, PREFIX, SUFFIX, REGEX)));
 }

 @Override
 protected Collection<ParallelText> doSearch(Language sourceLang,
  Language targetLang, String source, MatchingMethod matchingMethod)
  throws InvalidParameterException, ProcessFailedException {
 List<ParallelText> result = new ArrayList<ParallelText>();
 for(ParallelTextResource r : resources){
  if(matches(r, sourceLang, source, matchingMethod)){
   result.add(new ParallelText(
    r.get(sourceLang)
    , r.get(targetLang)));
  }
 }
 return result;
 }
```

```
private static class ParallelTextResource{
 public ParallelTextResource(String enText, String jaText){
  this.enText = enText;
  this.jaText = jaText;
 }
 public String get(Language language){
  if(language.equals(en)){
   return enText;
  } else{
   return jaText;
  }
 }

 private String enText;
 private String jaText;
}

private boolean matches(ParallelTextResource resource, Language language
  , String source, MatchingMethod matchingMethod){
 String text = resource.get(language);
 switch(matchingMethod){
  default:
  case COMPLETE:
   return text.equals(source);
  case PREFIX:
   return text.startsWith(source);
  case SUFFIX:
   return text.endsWith(source);
  case PARTIAL:
   return text.indexOf(source) != -1;
  case REGEX:
   return text.matches(source);
 }
}

private ParallelTextResource[] resources = {
  new ParallelTextResource("Hello.", "
  };
}
```

Fig. 5 Complete Wrapper Source Code

4.2. Wrapper test

Once wrapper implementation is complete, define the test code. Actually, we recommend that you prepare the wrapper method declaration and define the test code at the compilation path phase (where logic has not yet been implemented), and implement logic while implementing the test. Using this method will allow you to see how implementation is proceeding by looking at the number of successful tests.

4.2.1. Creating the test class

Create a test class for the service. Select the SampleParallelTextService class in Project Ẽplorer and press **CTRL+N** (see Fig. 17). Choose Java→JUnit→JUnit Test Case.

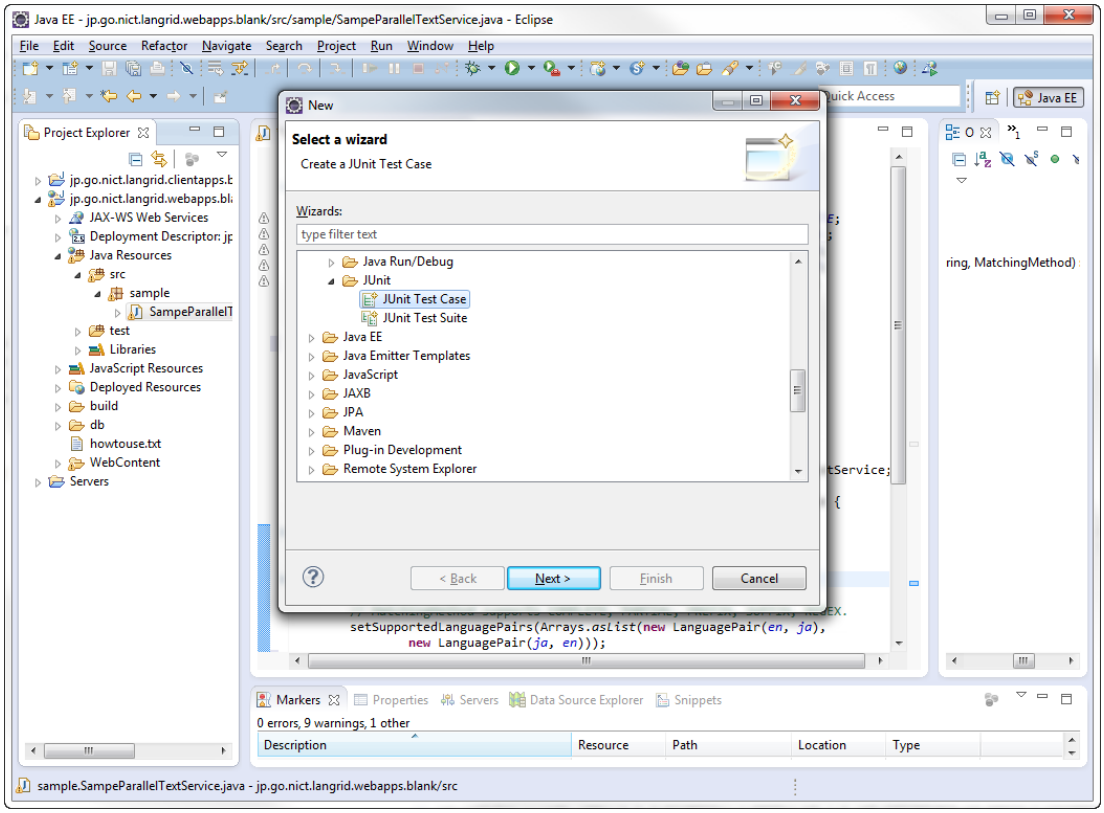


Fig. 17 Wrapper test 1

Click next to enter information of the test class. Choose “New JUnit 4 test”, Source folder is“test”,Packageis“sample”.EnterSampleParallelTextServiceTestasnameofthetestclass, and fill “sample.SampeParallelTextService” in the Class under test. (see Fig. 18)

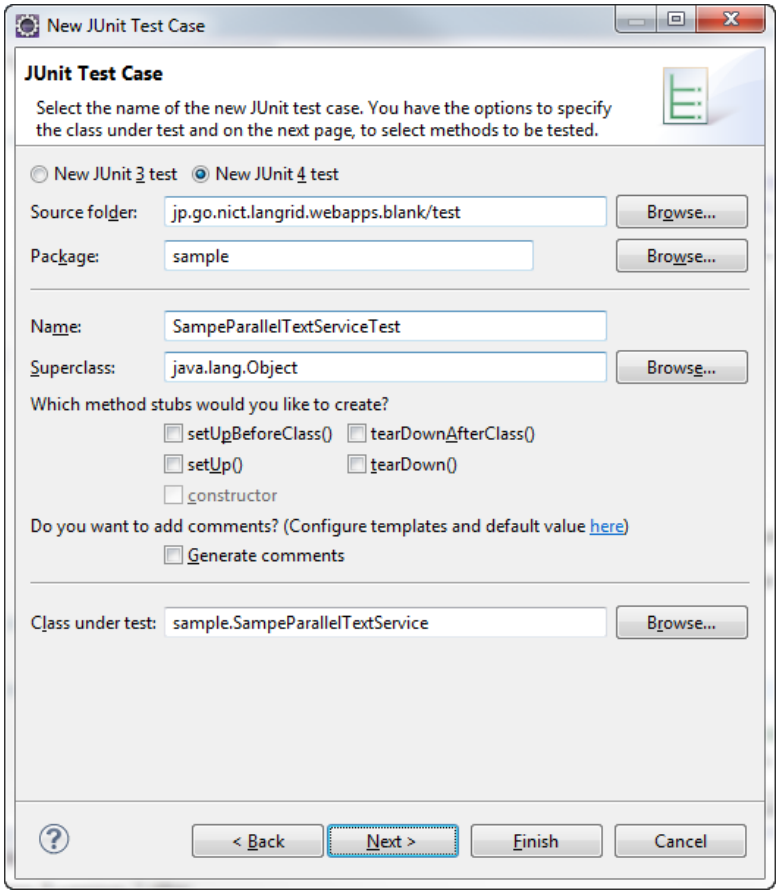
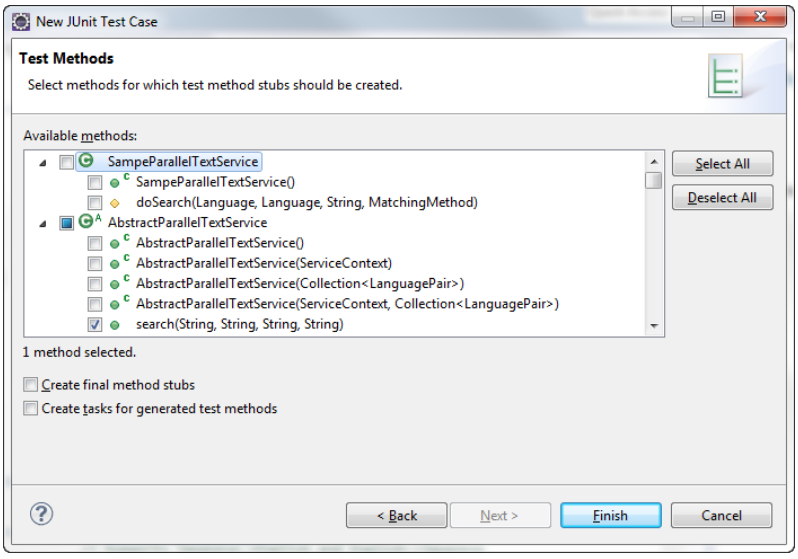


Fig. 18 Wrapper test 2

Click Next, the screen to select methods for generating test code will be diplay. Check the method to create the test (search) from the scroll down list of methods in "Available methods:" and press "Finish".



4.2.2.Implemeting the test code

If there is no error from the test code class, implement the test code and perform the following test. For more information, please refer to the many websites available detailing the JUnit assertion method necessary for implementing the test code.

- Function Test

  Check whether the expected results when combining all values that can be extracted from the parameters (all supported languages and translations, all supported matching methods, etc.) are received.

- Error Test

  Check whether the proper error is returned if unauthorized data (null parameters, empty character strings, language codes for unsupported languages, etc.) are inputted.

The test class generated is shown in Fig. 6.

```
public class SampleParallelTextServiceTest extends TestCase {
public void testSearch() {
 fail("Not yet implemented");
}
}
```

Fig. 6　Generated Test Class

Define the function test in the testSearch method. There are multiple tests defined in a normal function test. Here, you should basically divide methods so that there is one method to one test. Also, test various possible combinations of all inputted supported language resource languages/language pairs, all matching methods, and so on in the test code.
Fig. shows a function test code. The testSearch method is deleted, and the Japanese-English bilingual retrieval and English-Japanese bilingual retrieval are defined. It uses multiple services as the service entity and generates a SampleParallelTextService instance.

```
public void testSearch_en_ja_COMPLETE() throws Exception{
 assertEquals("
 , service.search("en", "ja", "Hello.", "COMPLETE")[0].getTarget());
}

public void testSearch_ja_en_COMPLETE() throws Exception{
 assertEquals("Hello.", service.search("ja", "en", "
 , "COMPLETE")[0].getTarget());
}

private ParallelTextService service = new SampleParallelTextService();
```

Fig. 20 Function Test

As an example of an error test, Fig. 7 shows the code to test actions when a null is inputted into the sourceLang.

```
public void testSearch_null_for_sourceLang() throws Exception{
 try{
  service.search(null, "ja", "Hello.", "COMPLETE");
  fail();
 } catch(InvalidParameterException e){
  assertEquals("sourceLang", e.getParameterName());
 }
}
```

Fig. 7 Error Test

In the error test it will fail to properly close execution, so it will define fail (a method to forcibly fail a test) in the higher order declared by the search or other implemented method. Define a code to verify whether the proper exception is returned when it is caught. In the example, the test will pass only if an error occurs for InvalidParameterException in the sourceLang parameter.

Functions such as normal null checks and supported language/search method checks are performed after the abstract class test, so there is no need to define them in the error test. Define error tests when unspecified actions occur at the abstract class level.

4.2.3. Local individual tests

Once the test code has been defined, select and implement "Run→Run As→JUnit Test" from the Eclipse menu while there is focus in the test code.

4.3. Web service test

A wrapper will be located on the Tomcat server, and is called as web service. Therefore, it is necessary to test the wrapper as web service. This section shows how to test wrapper as web service under Tomcat on Eclipse.

4.3.1. Create web service definition file

Create a service definition file and place it in WebContent/WEB-INF/serviceimpl/. The file name is the name of service + ".xml". Figure 22 shows the content of the service definition file "SampleParallelTextService.xml".

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
```

```
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="target"
    class="jp.go.nict.langrid.servicecontainer.handler.TargetServiceFactory">
    <property name="service">
       <bean class="sample.SampleParallelTextService" />
    </property>
  </bean>
</beans>
```

Fig. 22　Service definition file

4.3.2. Test service with ServiceLoader

Create a test code with SeriviceLoader on test folder. Figure 23 show an example of sample.SampleParallelTextServiceLoaderTest.

```
public void testSearch_en_ja_COMPLETE() throws Exception {
ParallelTextService service = new ServiceLoader(
new EclipseServiceContext()).load("SampleParallelTextService",
           ParallelTextService.class);
assertEquals("こんにちは。",
        service.search("en", "ja", "Hello.", "COMPLETE")[0].getTarget());
}
```

Fig. 23　Load service with ServiceLoader

EclipseServiceContext is a implementation of ServiceContext to execute a service on Eclipse. When loading the service, the root directory for service definition is /WebContent. The service definition file is WEB-INF/serviceimpl/service ID.xml. When you run the test the service is loaded base on the definition file. If the test fails, please check file name and location of the definition file.

4.3.3. Start service in Eclipse

Start the wrapper that you have created as web service. On the tab “Server” on Eclipse, right-click to the Server that you added in Section 3.3, and click “Add and Remove”. See Figure 24.

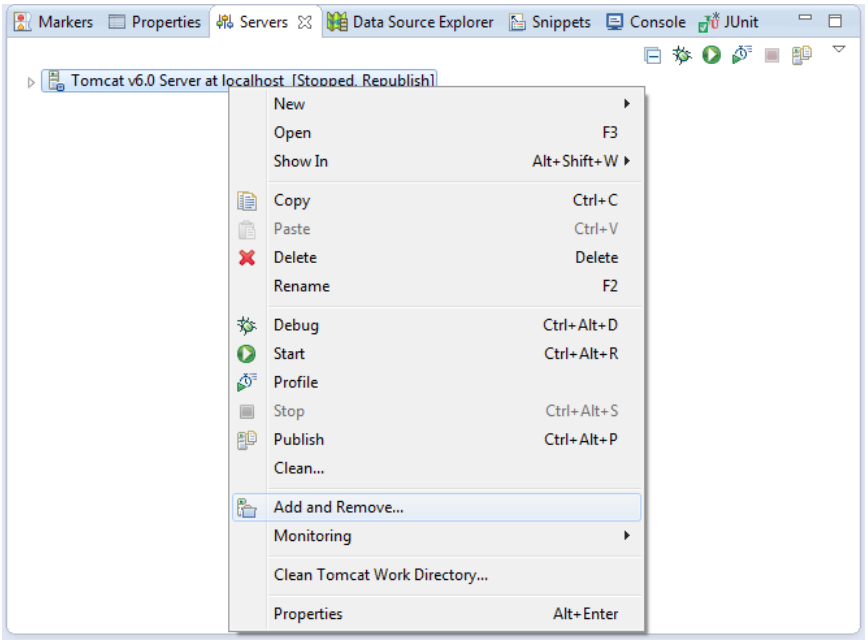


Fig. 24　Add server configuration for the Wrapper 1

Chose project “jp.go.nict.langrid.webapps.blank” click add and click “Finish”. (See Figure 25)

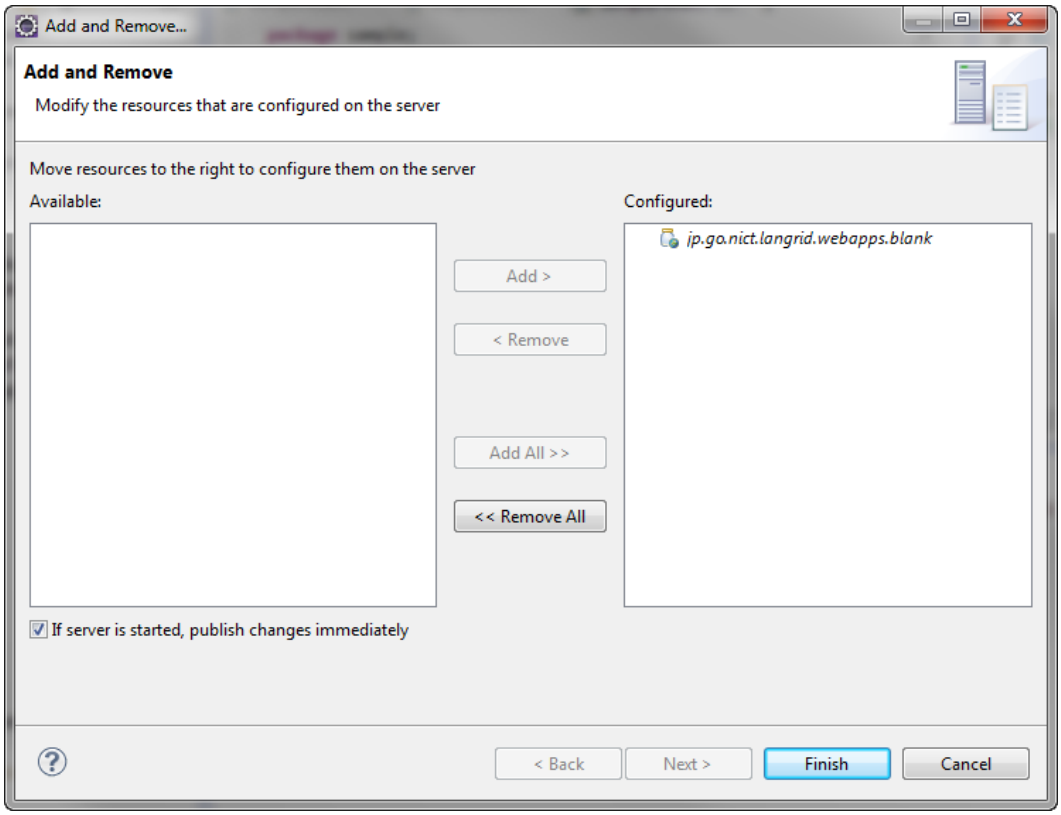


Fig. 25　Add server configuration for the Wrapper 2

Then click start on the “Server” view to start the Tomcat. (See Figure 26)

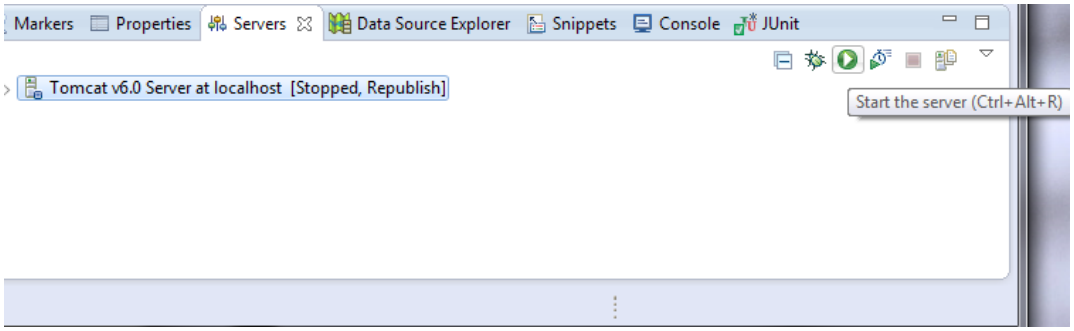


Fig. 26 Start Tomcat

After Tomcat started, open this URL: http://localhost:8080/jp.go.nict.langrid.webapps.blank/services in a browser. If the wrapper runs successfully, the list of services will be displayed. (See Figure 27)

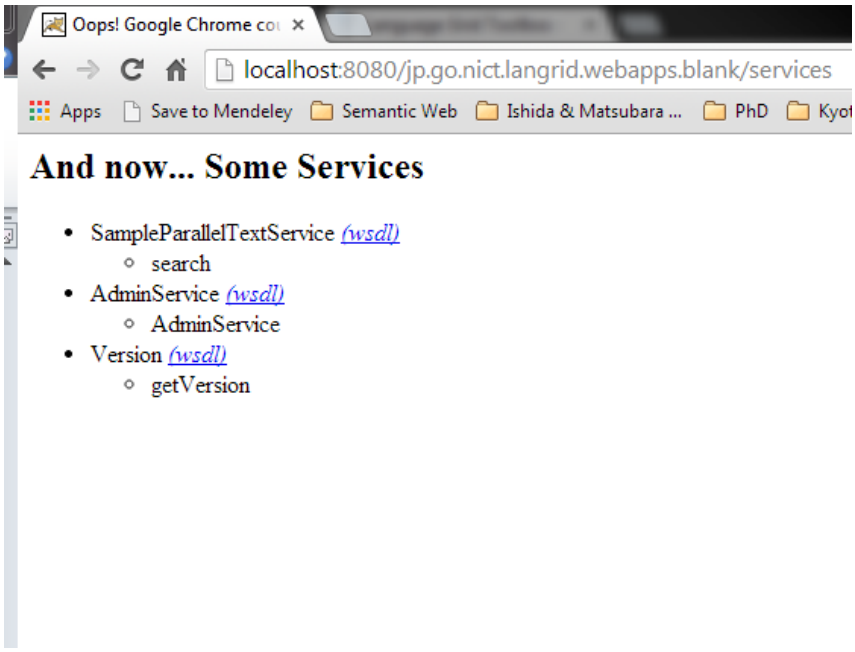


Fig. 27 Wrapper started

AdminService and Version are Axis services. You can click on the “wsdl” link to get WSDL file of the services.

4.3.4. Create a test class

Use the client project that you added in section 3.4.2 (jp.go.nict.langrid.clientapps.blank) to create test class below:

Project	jp.go.nict.langrid.clientapps.blank
Source folder	test
Paclage Name	sample
Class Name	SampleParallelTextSoapTest

4.3.5.Implementation of test code

Add the testSearch_en_ja_COMPLETE method to the test class. The code is shown in Figure 28.

```
package jp.go.nict.langrid.clientapps.blank;

import static jp.go.nict.langrid.language.ISO639_1LanguageTags.en;
import static jp.go.nict.langrid.language.ISO639_1LanguageTags.ja;
import static jp.go.nict.langrid.service_1_2.typed.MatchingMethod.COMPLETE;
import static org.junit.Assert.assertEquals;
import java.net.URL;
import jp.go.nict.langrid.client.ws_1_2.ClientFactory;
import jp.go.nict.langrid.client.ws_1_2.ParallelTextClient;
import org.junit.Test;

public class SampleParallelTextSoapTest {
        @Test
        public void testSearch_en_ja_COMPLETE() throws Exception {
            ParallelTextClient service = ClientFactory
                    .createParallelTextClient(new URL(
                        "http://localhost:8080/jp.go.nict.langrid.webapps.blank"
                          + "/services/SampleParallelTextService"));
            assertEquals("こんにちは。",
                service.search(en, ja, "Hello.",COMPLETE)[0].getTarget());
        }
}
```

Fig. 28　Web service test code

ClientFactory crate client to call web service via SOAP. Create method for each service type is defined, the method take the URL to indicate the endpoint of the service.

4.3.6.Run the test

Run the test similar with in section 4.2.3. If the service is started and work as expected, the test will succeed. If any error occurs, an exception is returned. The error codes and exeptions are shown in the table below.

Table 1 Error Codes and Exceptions

E052	Exception Class	InvalidParameterException
	Description	Improper parameters are passed.

E053	Exception Class	LanguageNotUniquelyDecidedException
	Description	The actual language that should be processed from the language passed to the service is not uniquely decided. For example, this will occur if the service supports zh-CN and zh-TW, but zh parameters are passed.
E054	Exception Class	UnsupportedLanguageException
	Description	The language passed to the service is not supported.
E055	Exception Class	LanguagePairNotUniquelyDecidedException
	Description	The actual language pair that should be processed from the language pair passed to the service is not uniquely decided. For example, this will occur if the service supports ja, zh-CN and ja, zh-TW, but the ja, zh parameters are passed.
E056	Exception Class	UnsupportedLanguagePairException
	Description	The language pair passed to the service is not supported.
E062	Exception Class	UnsupportedMatchingMethodException
	Description	The matching method passed to the service is not supported.

4.4. Create a deployment package

Use the export feature of Eclipse to create deployment package to a server environment. Right-click on the jp.go.nict.langrid.webapps.blank project and chose export WAR file. (See Figure 29)

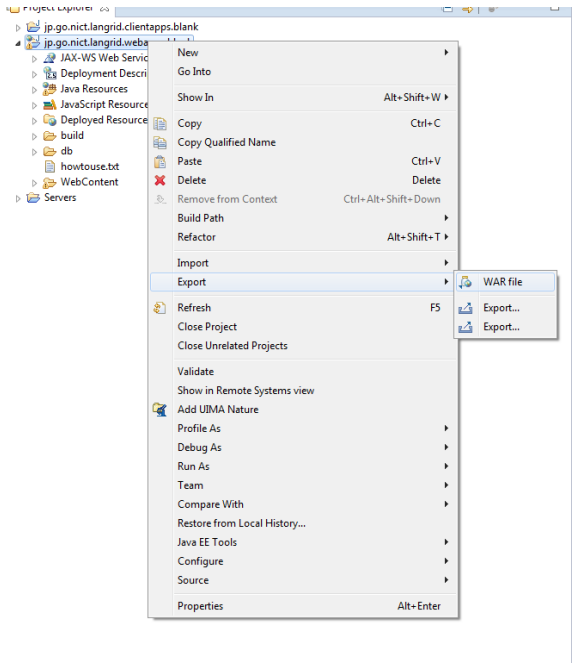


Fig. 29 War file creation 1

Next, select output directory and click Finish to export the war file. (See Figure 30)

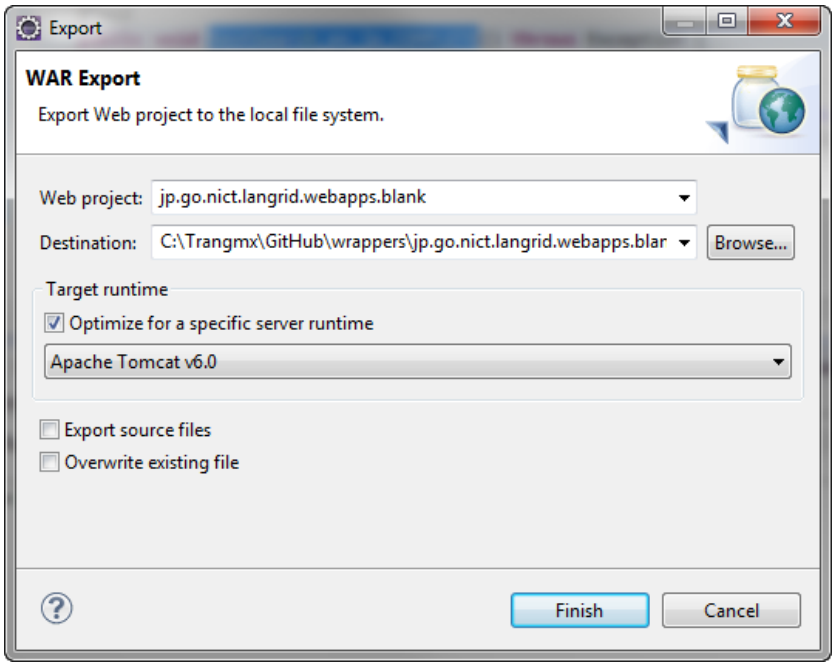


Fig. 30　War file creation 2

Place the war file uder webapps folder in Tomcat, the deployment on Tomcat is don automatically.

5. Use of Abstract Classes

5.1. Outline

In order to reduce the cost of implementation of each language service, we have prepared an abstract class for each type of language service. Those classes will be explained here. For each abstract class, the interface that supports the language resource is implemented, and after a parameter check (5.1.5), parameter conversion (5.1.6), and thread limit check (5.1.7) are performed, each abstract method is declared.

When developing the wrapper using the abstract class, inherit the abstract class, configure the interaction language within the construct (setSupportedLanguages/setSupportedLanguagePairs) or the matching method (setSupportedMatchingMethods), and override the abstract method that implements each service's logic.

5.1.1. Initialization of Parameter Acquisition

The initialization parameter of the service can be described in a definition file of the service. For example, if SampleParallelText has parameters p1 you can set "hello" to p1 in the service definition file. (See Figure 31)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
<bean id="target"
class="jp.go.nict.langrid.servicecontainer.handler.TargetServiceFactory">
   <property name="service">
      <bean class="sample.SampleParallelTextService">
         <property name="p1" name="hello">
      </bean>
   </property>
 </bean>
</beans>
```

Fig. 31 Initialization parameter definition

In your service code, write setP1 method to set "hello" to your p1 parameter when the service is loaded.

5.1.2. Configuring the supported languages

Configure the supported languages within the general construct. The configuration method is different for the first language service and the language pair service. Configure the first language service as below. (Below, LanguagePair is jp.go.nict.langrid.language.LanguagePair, Arrays are java.util.Arays, ja, en, ko, zh are supported by jp.go.nict.langrid.language.ISO639_1LanguageTags. Ja, en, ko and zh are static imports.)

```
public MorphologicalAnalysis(){
 setSupportedLanguages(Arrays.asList(ja, en, ko, zh));
}
```

In this example, English, Japanese, Korean and Chinese are supported.

To handle the language pair, configure as below.

```
public BilingualDictionary(){
 setSupportedLanguagePairs(Arrays.asList(
 new LanguagePair(ja, en), new LanguagePair(en, ja)
 ));
}
```

In this example, Japanese-English and English-Japanese are supported.

5.1.3. Work directory

The work directory can be acquired by declaring the getWorkDirectory method implemented by the abstract class. For a web service deployed by Axis, the WEB-INF/wrapperwork directory path is returned; if it is not deployed by Axis, the current directory's wrapperwork directory is returned. Use a temporary file storage location in a working directory to process the wrapper.

5.1.4. Exceptions

The following lists possible exceptions you can throw in each abstract method (do~).

InvalidParameterException

Throw this when the unique language resource parameter restriction is violated, such as when argument contents are larger than the size supported by the language resource.

ProcessFailedException

Throw this when the process fails because another exception does not apply. If a serious exception occurs, do not use only ProcessFailedException when the exception is caught to determine the cause; enter the code below to save the server log.

```
try{
..
} catch(SomeSevereExceptin exception){
logger.log(Level.SEVERE, "例外", exception);
throw new ProcessFailedException(exception);
}
```

5.1.5. Parameter Check

The Parameter Check checks whether the value that is normally supported by the given language parameter (language, sourceLang, targetLang, headLang etc.) and the matching method (matchingMethod) is given, and whether the language resource is supported (whether supportedPairs, supportedLanguages, supportedMatchingMethods, etc. exist). If the check fails, the following exceptions are declared and the parameter is returned to its original state.

- InvalidParameterException (an invalid parameter or null is returned)
- LanguagePairNotUniquelyDecidedException (the language code is incomplete, and the interaction language pair is not uniquely determined)
- LanguageNotUniquelyDecidedException (the language code is incomplete, and the interaction language is not uniquely determined)

- UnsupportedLanguagePairException (the language pair is not supported)
- UnsupportedLanguageException (the language is not supported)
- UnsupportedMatchingMethodException (the matching method is not supported)

Configure supported language parameters such as supportedPairs and supportedLanguages using setSupportedLanguages within the general construct. Configure supportedMatchingMethod using the setSupportedMatchingMethod method.

5.1.6. Parameter conversion

Parameter conversion converts the language sent as a character string or the matching method into an appropriate type (Language, MatchingMethod, etc.). When parameter conversion occurs, it may behave as a language code matching (Language.matches) or enumerated value, so there are fewer descriptor codes than for the handling of character strings, and it is more difficult for bugs to occur. If the appropriate conversion does not take place, InvalidParameterException is returned to the access origin.

The rules below are used for language parameter matching for the conversion of language parameters.

Perform a leading section match and convert the parameter to the matching language. (Ex: The parameter “ja” matches “ja-JP” and “ja-JP-osaka” supported by the service.)

However, if there are several candidates, a match is not made, and a Language[Pair]NotUniquelyDecided exception occurs. (Ex: If the service supports both “zh-Hans” and “zh-Hant”, the parameter “zh” will not match with anything.)

5.1.7. Thread Limit Check

The thread limit check functions to restrict the number of simultaneous threads in VM(ClassLoader) to the value established for the langrid.maxThreads parameter in the wsdd file (descriptor deployment file). If the number of threads has already reached the upper limit, it will wait until currently existing threads close.

5.2. List of Service Types

We offer the following abstract wrapping classes for the language services below.

For the descriptions of how to use each abstract class, see JavaDoc(http://langrid.nict.go.jp/developer/en/apidocs/).

Language Service (Type)	Abstract Class Name
Translation	AbstractTranslationService
Description	

General translation. Translates text from one specified language into another.

Language Service (Type)	Abstract Class Name
Bilingual Dictionary	AbstractBilingualDictionaryService
Description	
Displays a bilingual dictionary. Can support several languages as a dictionary, but can only retrieve definitions in one language.	

Language Service (Type)	Abstract Class Name
Bilingual Dictionary Longest Match Search	AbstractBilingualDictionaryWith LongestMatchSearchService
Description	
Bilingual dictionary that supports a longest match search. This service takes parameters for morpheme arrays and uses this information to search for longest language array matches.	

Language Service (Type)	Abstract Class Name
Pictogram Dictionary	AbstractPictogramDictionaryService
Description	
This service searches for pictograms signifying words in a language.	

Language Service (Type)	Abstract Class Name
Concept Dictionary	AbstractConceptDictionaryService
Description	
This service searches for the concepts a word possesses.	

Language Service (Type)	Abstract Class Name
Parallel Text	AbstractParallelTextService
Description	
This service searches for parallel text.	

Language Service (Type)	Abstract Class Name
Adjacency Pair	AbstractAdjacencyPairService
Description	
This service searches for pair responses.	

Language Service (Type)	Abstract Class Name
Morphological Analysis	AbstractMorphologicalAnalysisService
Description	
General morphological analysis service. Analyzes and returns morpheme arrays for an inputted text.	

Language Service (Type)	Abstract Class Name
Paraphrase	AbstractParaphraseService
Description	
This service searches for and returns a paraphrased expression in the same language. Well suited for converting simple, easy to handle text in translation.	

Language Service (Type)	Abstract Class Name
Similarity Calculation	AbstractSimilarityCalculationService
Description	
Calculates the level of similarity of two texts in the same language.	

Language Service (Type)	Abstract Class Name
Dependency Parser	AbstractDependencyParserService
Description	
Parses the dependency in a text.	

Language Service (Type)	Abstract Class Name
Speech Recognition	AbstractSpeechRecognitionService
Description	
Recognizes an inputted voice audio file and returns the recognized text as String.	

Language Service (Type)	Abstract Class Name
Text to Speech	AbstractTextToSpeechService
Description	
Synthesizes and returns a voice audio file for an inputted text.	

6. Using the Database-based Bilingual Dictionary and Parallel Text

The NICT Language Grid Project provides users with a database (DB) wrapper package to wrap the stored language resources in the relational database for implementing the

bilingual dictionary (BilingualDictionary) and parallel text (ParallelText) as single wrapper. Using the DB wrapper package allows users to deploy the wrapper instead of defining the Java source code.

6.1. Acquiring the DB Wrapper Package

Confirm that the following db directory of the wrapping library that you obtained in section 3.4.1. This directory stores related files of DB wrapper.

- db
  - data
    - createTable-DBWraper.sql (SQL for creating database for Stored language resources)
    - Dictionary.xla (Microsoft Excel add-in for outputting CSV files in UTF8)
  - Lib (Jar files coppy from lib directory of Tomcat)

6.2. Creating a Database for Stored Language Resources

Use PostgreSQL to create a database to store language resources. The following discusses installation of PostgreSQL.

6.2.1. Installing PostgreSQL (Fedora)

The following commands can be used to install PostgreSQL when using Fedora 9.

```
yum install postgresql
yum install postgresql-server
```

6.2.2. Installing PostgreSQL (Windows)

The PostgreSQL download (http://www.postgresql.jp/PostgreSQL/) can be downloaded from Japan Postgre SQL Users Group using a Windows installer. Refer to PgFoundry's PostgreSQL Installer (http://pginstaller.projects.postgresql.org/) for installation steps. Set the Superuser name on the Initialise database cluster setting screen to the same name as your Windows login name. This user name will thereafter be displayed as `${USERNAME}`.

You can navigate the PostgreSQL through commands after installation is complete by running the command prompts in the following order. You will need your Superuser password set during installation when executing the commands.

[Start] menu → All programs → PostgreSQL 8.3 → Command Prompt

6.2.3. Creating a Database

Create a database with a random name to store the language resources. The database name will thereafter be displayed as `${DBNAME}`. You will be able to create a database by executing the following command using your postgres user authority.

```
createdb ${DBNAME}
```

When executing on Linux, work as a postgres user. When executing on Windows, you can attach a username option to make a postgres user, or work using a PgAdmin or other GUI tool. These will both create the same user below.

6.2.4. Creating a Connecting User

Create a user to connect the database to the wrapper. This new user will thereafter be displayed as ${DBUSER}, and the user password as ${DBPASS}. The command is as follows.

```
createuser -S -D –R -P ${DBUSER}
```

Option meanings are given below.

-S: New users do not become Superusers

-D: Databases will not be created for new users.

-R: New rolls will not be created for new users.

-P: New user password prompts will be displayed.

6.3. Set up tables

This section describes how to create a table to in DB wrapper, to import the data.

6.3.1. Creating Tables

Create a table to store bilingual data in the ${DBNAME} database. Change the content of createTable-DBWrapper.sql included in the DB wrapping package you downloaded to match each environment. The absolute path of the createTable-DBWrapper.sql will thereafter be displayed as ${SQLPATH}.

```
CREATE TABLE ${TABLENAME}
(
id serial PRIMARY KEY,
ja text,
en text,
"zh-CN" text,
ko text,
date timestamp DEFAULT now()
);
ALTER TABLE ${TABLENAME} OWNER TO ${DBUSER};
```

createTable-DBWrapper.sql

- Change ${TABLENAME} to the name of the table storing the language resource as appropriate.

- Change the ${DBUSER} to the user name when the wrapper connects to the database as appropriate.
- Table attributes are given in examples for Japanese, English, Chinese, and Korean. Change the attributes to match the corresponding wrapper. Use RFC3066 for the language code format. When using a language code with a hyphen, enclose in double quotes, such as “zh-CN”.

Execute createTable-DBWrapper.sql and create a table to store the bilingual data. The command is as follows.

```
psql ${DBNAME} < "${SQLPATH}"
```

6.3.2. Creating Bilingual Data

Create bilingual data using Microsoft Excel. The following examples show bilingual data for Japanese, English, Chinese, and Korean. Bilingual data in the following are stored in order of Japanese (ja), English (en), Chinese (zh-CN), and Korean (ko).

	A	B	C	D
1	日本語	Japanese	日语	일본어
2	英語	English	英语	영어
3	中国語	Chinese	中文	중국어
4	韓国語	Korean	韩语	한국어

Using the Dictionary.xla Excel add-in included in the DB wrapping package will allow you to use UTF8 character code to convert the Microsoft Excel sheet you created into a CSV file.

<When Using Windows XP and Excel 2003>

With the Excel file where the bilingual data is stored closed, copy Dictionary.xla into the following folder. If you are unable to view the application data in Explorer, enter the following path directly into Explorer's address bar and move the folder.

```
C:¥Documents and Settings¥${USERNAME}¥Application Data¥Microsoft¥AddIns
```

Open the Excel file where the bilingual data is stored and select Tool → Add-in from the menu. Check Dictionary from the list of valid add-ins and click OK. Add Create CSV to the menu. Click on Create CSV and output the CSV file into the folder containing the Excel file.

<When Using Windows Vista and Excel 2007>

With the Excel file where the bilingual data is stored closed, copy Dictionary.xla into the following folder.

```
C:¥Users¥${USERNAME}¥AppData¥Roaming¥Microsoft¥AddIns
```

Open the Excel file where the bilingual data is stored and click Excel Options →Add-ins→ Settings (G) in that order from the upper left Office button. Check Dictionary from the list of valid add-ins and click OK. Add-ins will have been added to the menu. Click Add-ins →Create CSV to output the CSV file into the folder where Dictionary.xla is copied.

6.3.3. Importing Data

This section describes the steps to import the bilingual data CSV file into the database. The CSV file's absolute path (/break) will thereafter be displayed as `${CSVPATH}`. The following example shows `${CSVPATH}` when using Windows XP.

```
C:/Documents and Settings/${USERNAME}/My Documents/Sheet1.csv
```

First, log into the database.

```
psql -d ${DBNAME}
```

When using Windows XP, change the client encoding to UTF8. The command is as follows.

```
¥encoding UTF8
```

Next, import the bilingual data CSV file (`${SCVPATH}`) into the database `${DBNAME}` table `${TABLENAME}`.

```
COPY ${TABLENAME}(${LANGUAGES}) FROM '${CSVPATH}' CSV;
```

Define the absolute path in `${CSVPATH}`. `${LANGUAGES}` is written in the CSV file where the bilingual dictionary is stored. The following is an example of `${LANGUAGES}`.

```
Ja, en, "zh-CN", ko
```

6.3.4. Setting the Database Connection Authentication Method

Change pg_hba.conf as below. (Add the parts written in red.) When using Fedora, this will be in /var/lib/pgsql/data/pg_hba.conf. When using Windows, you can edit by [Start] Menu → All Programs → PostgreSQL 8.3 → Configuration Files → Edit pg_hba.conf.

```
# IPv4 local connections:
#host all  all  127.0.0.1/32  md5
host all  all  127.0.0.1/32  password
```

Setting of pg_hba.conf

6.4. Deployment of DB Wrapper

This chapter explains how to create database-based bilingual dictionary and parallel text wrappers.

6.4.1.DB connection setting

Connection setting to connect DB wrapper with database is defined in context.xml file (WebContent/META-INF/context.xml). This file declares authentication information and connection URL of the database. The contentof the context.xml file is shown in Figure 32.

```
<?xml version='1.0' encoding='utf-8'?>
<Context reloadable="true" displayName="Langrid DB Services">
   <Resource name="jdbc/langrid-service-db" auth="Container"
       type="javax.sql.DataSource" maxActive="100" maxIdle="50" maxWait="10000"
       username="${DBUSER}" password="${DBPASS}"
       driverClassName="org.postgresql.Driver"
       url="${DBURL}" />
</Context>
```

Fig. 32　DB connection settings

Replace parameters with your environment variables below:

- ${DBURL} - database connection URL (e.g jdbc:postgresql://localhost:5432/${DBNAME}, ${DBNAME} is database that you careted in section 6.2.3)
- ${DBUSER} - Database user
- ${DBPASS} - Password of the database user

“jdbc/langrid-service-db” is the name of the resource.

6.4.2.Service definition of DB Wrapper

The service definition of DB wrapper is defined corresponding with DB connection information insection 6.4.1. Figure 33 shows an exampleof bilingual dictionary service definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <bean id="target"
          class="jp.go.nict.langrid.servicecontainer.handler.TargetServiceFactory">
          <property name="service">
            <bean
               class="jp.go.nict.langrid.wrapper.common.db.bilingualdictionary.BilingualDictionaryService">
               <property name="tableName" value="${TABLENAME}" />
               <property name="languageColumnNames" value="ja en zh-CN ko" />
               <property name="connectionParameters">
                  <bean class="jp.go.nict.langrid.wrapper.common.db.ConnectionParameters">
                    <property name="dataSourceName" value="jdbc/langrid-service-db" />
                    <property name="dbDictionary" value="POSTGRESQL" />
                  </bean>
               </property>
               <property name="maxResults" value="100" />
            </bean>
          </property>
    </bean>
</beans>
```

Fig. 32 Service definition of DB wrapper

This example is information for deploying jp.go.nict.langrid.wrapper.common.db.bilingualdictionary.BilingualDictionaryService service and jp.go.nict.langrid.wrapper.common.db.paralleltext.ParallelTextService service. Following is the description of common parameter for parallel texts and bilingual dictionary.

- tableName
  - data is stored in the table created in section 6.3.1
- languageCollumnNames
  - list of languages separeated by space
- connectionParameter
  - Set the DB connection information. The connection information is defined in jp.go.nict.langrid.wrapper.common.db.ConnectionParameters. the dataSourceName refers to DB connection name (defined in section 6.4.1 “jdbc/langrid-service-db”)
- maxResults
  - Number of maximum return of search results

6.4.3.Deployment of the DB Wrapper

First coppy following jar file in the db/lib folder to the lib directory of the Tomcat.

commons-dbcp-1.2.2.jar
commons-pool-1.3.jar
JDBC driver (postgresql-8.3-604-jdbc3.jar, etc)

Use export feature in eclipse to create war file for the wrapper, and place the war file to the webapps folder of Tomcat.

7. Composite service creation

To be updated

8. Inquiries

Please send any inquiries to the following address:
langrid@khn.nict.go.jp