Getting Started with Celtix

Table of Contents

Overview

This document shows you how to get started with Celtix. This includes showing you how to set up your development environment and build and run a basic SOAP/HTTP client and server. For information on how to install Celtix, see *The Celtix Installation Guide*, which is included in the product download and available from the Celtix website, `http://forge.objectweb.org/projects/celtix/`.

This document was written for Celtix Milestone 4; as Celtix matures future versions may deviate from the material covered in this document. The Celtix team will endeavor to keep this document as up-to-date as possible.

Setting up Your Environment

Celtix should always be run from an appropriately configured shell. To configure your shell you need to do the following:

1. Set your `CLASSPATH` to pick-up the application files and the correct versions of the Celtix JAR files.
2. Set your `PATH` to ensure you are using the correct Java compiler and the correct version of the Celtix tools.
3. If you are using Ant as a build system, then you will also need to pick up the correct version of the Ant build tools.

Setting the _HOME Variables

Many open-source projects follow the useful convention of having an `_HOME` environment variable to describe the location of the product installation on the file system; when using Celtix you will want to set `JAVA_HOME`, `CELTIX_HOME` and `ANT_HOME` appropriately. The `CELTIX_HOME` variable should be set to the root of your Celtix installation.

Setting the PATH Variable

To ensure that you pick up the correct version of the Java compiler, add the directory `JAVA_HOME/bin` to the `PATH` environment variable. Running `java -version` at the prompt will verify that you are picking up the correct version of Java; for Celtix, you should be using JDK 1.5.0_06 (JDK 5.0, release 6) or higher.

If you want to use command line tools directly to generate Celtix code and compile and run your applications, then add the directory `CELTIX_HOME/bin` to the `PATH` environment variable. This ensures that you will use the Celtix code generation tools like `wsdl2java` and `java2wsdl`.

If you wish to use the Apache Ant build system, add `ANT_HOME/bin` to the `PATH`.

Setting the CLASSPATH Variable

If you want to use the command line tools (that is, `wsdl2java`, `javac`, and `java`) directly, you should add `CELTIX_HOME/lib/celtix.jar` to the `CLASSPATH` so that your environment has access to the Celtix classes.

Using a Script to Set Up the Shell Environment

Rather than setting these variables for every command window, consider using a script. An example script, `setenvs.bat`, for use on Windows with the Celtix binary distribution, is shown below in CodeSnap 1.

```
@echo off
REM Ensure that the values for the following variables are
REM set correctly for your installation.
set CELTIX_HOME=c:\Celtix_bin4\celtix
set JAVA_HOME=c:\jdk1.5.0
set ANT_HOME=c:\Ant\apache-ant-1.6.5

REM You should not have to modify anything below this point.
echo.
echo Take note of the following important variables - are they correct for your
echo system? If not then edit this file and correct them!
echo.
echo CELTIX_HOME = %CELTIX_HOME%
echo JAVA_HOME   = %JAVA_HOME%
echo ANT_HOME    = %ANT_HOME%
echo.
set PATH=%CELTIX_HOME%\bin;%PATH%
set PATH=%JAVA_HOME%\bin;%PATH%
set PATH=%ANT_HOME%\bin;%PATH%

set CELTIX_JAR=%CELTIX_HOME%\lib\celtix.jar
set CLASSPATH=%CELTIX_JAR%;.;.\build\classes;%CLASSPATH%
title Celtix Shell
```

CodeSnap 1: `setenvs.bat`

A corresponding script for use on Windows with the Celtix source distribution is shown in CodeSnap 2. This script is only suitable for compiling and running Celtix applications when using the Apache Ant build system, as described in the following sections of this document.

```
@echo off
REM Ensure that the values for the following variables are
REM set correctly for your installation.
set JAVA_HOME=c:\jdk1.5.0
set ANT_HOME=c:\Ant\apache-ant-1.6.5

REM You should not have to modify anything below this point.
echo.
echo Take note of the following important variables - are they correct for your
echo system? If not then edit this file and correct them!
echo.
echo JAVA_HOME   = %JAVA_HOME%
echo ANT_HOME    = %ANT_HOME%
echo.
set PATH=%JAVA_HOME%\bin;%PATH%
set PATH=%ANT_HOME%\bin;%PATH%

set CLASSPATH=.;.\build\classes;%CLASSPATH%
title Celtix Shell
```

CodeSnap 2: `setenvs.bat`

Celtix Development Environment

Developing applications with Celtix code is no different from developing with any other Java library or API. You just need to set the `CLASSPATH` appropriately and begin coding. You can develop with your favorite text editor, Integrated Development Environment (IDE) or build system. In this section, we recommend two open-source tools used extensively by developers of Celtix:

- Eclipse
- Ant

Using Celtix with Eclipse

Eclipse (available from `http://www.eclipse.org`) provides an excellent Java IDE for Celtix development. You must use Eclipse 3.1.1 or higher, as Celtix requires support for Java 1.5 language constructs, which was not available in earlier versions of Eclipse.

Eclipse provides a way to store “User Libraries”: collections of JARs and classes that can be reused across projects. Create a user library for Celtix by navigating to the “User Libraries” dialog box in Eclipse.

Window → Preferences → Java → Build Path → User Libraries

Add the file `celtix.jar` to the user library. In the binary distribution of Celtix, this can be found in `CELTIX_HOME/lib/celtix.jar`.

At the time of writing, Eclipse is unable to pickup the manifest classpath present in `celtix.jar`. As a result you will also have to explicitly add all the JAR files for JAX-WS to your user library. For a binary distribution, these files will reside in the `CELTIX_HOME/lib` directory.

After you have created a user library for Celtix, you can add it to the Java project build path and Eclipse will auto-compile your code.

Future documentation will cover this topic in greater detail.

Using Celtix with Ant

Many Java developers will be familiar with the Ant build system, downloaded from `http://ant.apache.org`. CodeSnap 3 shows an Ant build file typical of those provided with the Celtix samples. The build file imports the `common_build.xml` file, which includes most of the commands that compile and run the sample applications. This build file offers a number of features:

- The variable `codegen.notrequired` is true if no XSD or WSDL files in the project have changed since the last run of `wsdl2java`. If you do not declare `wsdl.dir` as a property that identifies the location of XSD and/or WSDL files, then the default value of `./wsdl` is used.
- The `wsdl2java` task can be used to generate Java code.
- The `celtixrun` task can be used to run a Java class with appropriate `CLASSPATH` and JVM argument settings for use with Celtix.

```
<project default="build">
    <!-- Import generic celtix build.xml file -->
    <property environment="env"/>
    <import file="${env.CELTIX_HOME}/samples/common_build.xml"/>

    <target name="generate.code" unless="codegen.notrequired">
      <echo message="Generating code using wsdl2java..."/>
       <wsdl2java file="HelloWorld.wsdl"/>
      <touch file="${codegen.timestamp.file}"/>
    </target>

    <!-- Targets to run the client and server -->
    <target name="server" depends="build">
        <celtixrun classname="helloworld.Server"/>
    </target>

    <target name="client" depends="build">
        <celtixrun classname="helloworld.Client"/>
    </target>
</project>
```

CodeSnap 3: Sample `build.xml` file for use with Celtix sample applications.

The `common_build.xml` and `build.xml` files that are supplied with the Celtix sample applications are useful for applications built along a similar approach; that is, you use a directory hierarchy modeled on the Celtix samples and you do not use the `wsdl2java` utility to generate starting point code for your client and server mainlines and implementation object.

Later in this document, you will learn how the Celtix `wsdl2java` utility will generate an Ant build file for the applications that you build. Ant build files generated by the `wsdl2java` utility are complete and do not import the `common_build.xml` file. Additionally, they employ a different directory hierarchy than the one used by the Celtix samples, as will be discussed in the following section.

Directory Structure for Celtix Projects

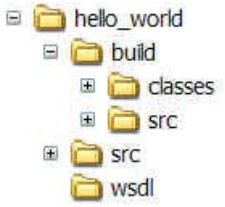
How you arrange a project's directory structure depends on whether you are modeling your hierarchy after the product samples or using the `wsdl2java` utility to generate the full complement of starting point code files.

The "Celtix Sample Application" Approach

With the Celtix sample applications, the `wsdl2java` utility is used to generate files that represent the types, service proxy, and service interface for an application. The `wsdl2java` utility is not used to generate starting point code for the client and server mainlines or the implementation object. These files are provided as completed implementations so that the sample applications will run without requiring the user to add processing logic.

The Celtix sample applications contain the following directories:

- `build/classes` contains compiled Java classes, including those generated by `wsdl2java` utility.
- `build/src` contains Java source code generated by `wsdl2java` utility.
- `src` contains the supplied Java source code. These files may be in a different Java package than the Java source code files generated by the `wsdl2java` utility.
- `wsdl` contains WSDL and XSD files.



The top level project directory contains:

- The Ant build file (`build.xml`).

As shown in CodeSnap 3 above, the `build.xml` file imports the `common_build.xml` file, which is located in the samples directory. These files are written to use the directory hierarchy employed by the sample applications.

The "Developer Driven" Approach

When you use the `wsdl2java` utility to generate the complete complement of source code files, starting point code for the client mainline, server mainline, and implementation object will be generated. These files are included in the same Java package as the files representing the types, service proxy, and service interface.

Developer written Celtix applications contain the following directories:

- A project directory that includes:
    - One, or more, top_level_application_directories that contain:
        - The source code files for the application. The source code files include both the files generated by the `wsdl2java` utility as well as any other files you want to include in the application.

- A `build/classes` subdirectory that contains the compiled application files.
- The Ant build file (`build.xml`) generated by the `wsdl2java` utility.

- A `wsdl` directory that contains WSDL and XSD files.

The following figure illustrates this directory hierarchy.

- The project directory is `hello_world`.
  - The top_level_application_directory is `client`.
    - The source code generated by the `wsdl2java` utility is in the package hierarchies `org/objectweb/hello_world_soap_http` and `org/objectweb/hello_world_soap_http/types` under the `hello_world/client` directory. The package name was derived from the target namespace defined in the WSDL file.
    - The `hello_world/client/build/classes` directory includes the compiled code in the package hierarchies `org/objectweb/hello_world_soap_http` and `org/objectweb/hello_world_soap_http/types`.
  - The wsdl directory contains the WSDL file used to generate the application code.

Writing a SOAP/HTTP Client and Server with Celtix

The "Hello, World" interface used here is defined in the WSDL file `HelloWorld.wsdl`. While there is a "Hello, World" demo in the Celtix distribution, it uses a slightly different WSDL contract than that used here. The version used for this demo corresponds to the Java interface shown in CodeSnap 4.

```
public interface HelloWorld {
    public String sayHello(String message);
}
```

CodeSnap 4: `HelloWorld` *interface.*

The full WSDL contract follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--WSDL file template-->
<!--(c) 2005, IONA Technologies, Inc.-->
<definitions name="HelloWorld.wsdl"
      targetNamespace="http://www.celtix.org/courseware/HelloWorld"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:tns="http://www.celtix.org/courseware/HelloWorld"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <types>
    <schema targetNamespace="http://www.celtix.org/courseware/HelloWorld"
            xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
      <element name="sayHello">
        <complexType>
          <sequence>
            <element maxOccurs="1" minOccurs="1" name="message"
                     nillable="true" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
      <element name="sayHelloResponse">
        <complexType>
          <sequence>
            <element maxOccurs="1" minOccurs="1" name="return"
                     nillable="true" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
    </schema>
  </types>
  <message name="sayHello">
    <part element="tns:sayHello" name="parameters"/>
  </message>
  <message name="sayHelloResponse">
    <part element="tns:sayHelloResponse" name="parameters"/>
  </message>
  <portType name="HelloWorld">
    <operation name="sayHello">
      <input message="tns:sayHello" name="sayHello"/>
      <output message="tns:sayHelloResponse" name="sayHelloResponse"/>
    </operation>
  </portType>
  <binding name="HelloWorld_DocLiteral_SOAPBinding" type="tns:HelloWorld">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sayHello">
      <soap:operation soapAction="" style="document"/>
      <input name="sayHello">
        <soap:body use="literal"/>
      </input>
      <output name="sayHelloResponse">
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="HelloWorldService">
    <port binding="tns:HelloWorld_DocLiteral_SOAPBinding" name="SOAPOverHTTP">
```

```
      <soap:address location="http://localhost:9090/helloworld"/>
    </port>
  </service>
</definitions>
```

Using the "Celtix Sample Application" Approach

In the Celtix source distribution, the collection of JAR files that comprise the Celtix product are distributed throughout the product directories. Consequently, it is somewhat involved to list all of these JAR files on the `CLASSPATH`. In contrast, the Celtix binary distribution includes all these JAR files in a single directory and it is less complicated to add all of these files to the `CLASSPATH`. When you use the Celtix sample application approach, the `common_build.xml` file correctly sets the `CLASSPATH` for either the source or binary product distributions. It is, therefore, recommended that you use this approach to building applications when working with the Celtix source distribution.

The easiest way to start is to replicate the directory hierarchy used in the sample applications and then place your WSDL file into the `wsdl` directory. You can also copy a `build.xml` file from one of the sample applications and modify as required. You then write the client mainline, server mainline, and implementation object source code files and place add them to the `src` directory. Finally, you use Ant to generate code for the type, service proxy, and service interface files, and then to compile and run the applications.

As an example, the `HelloWorld` directory contains the `build.xml` file, the `src` directory contains the source code files (which you provide) for the client mainline, server mainline, and implementation object, and the `wsdl` directory contains the `HelloWorld.wsdl` file.

For this example, CodeSnap 5 shows the content of the `build.xml` file.

```
<?xml version="1.0"?>
<project name="HelloWorld Application" default="build" basedir=".">

    <import file="../common_build.xml"/>

    <target name="client" description="run client">
        <property name="param" value=""/>
        <celtixrun classname="<full package name of client>"
                   param1="${basedir}/wsdl/HelloWorld.wsdl"
                   param2="${op}" param3="${param}"/>
    </target>

    <target name="server" description="run server">
        <celtixrun classname="<full package name of server>"
                   param1="${basedir}/wsdl/HelloWorld.wsdl"/>
    </target>

    <target name="generate.code">
        <echo level="info" message="Generating code using wsdl2java..."/>
        <wsdl2java file="HelloWorld.wsdl"/>
    </target>

</project>
```

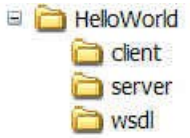
CodeSnap 5 `build.xml` *file.*

Now issue the command `ant build`, which creates the `build/classes` directory hierarchy and generates the type, service proxy, and service interface files and compiles the applications. Finally, use the commands `ant server` and `ant client` to run the applications.

Using the “Developer Driven” Approach

With this approach, the `wsdl2java` utility will generate an Ant `build.xml` file that accurately lists all Celtix JARs on the `CLASSPATH`. This approach is suitable when you are working with the Celtix binary distribution.

To start, you need to create the project, top_level_application, and `wsdl` directories, as shown in the following figure.



Then place the WSDL file into the `wsdl` directory and use the `wsdl2java` utility to generate the starting point code directly into a top_level_application directory.

For example, from the `HelloWorld` directory, use

```
wsdl2java -client -d client -ant .\wsdl\HelloWorld.wsdl
```

to generate starting point code, including the client mainline, for the client application, and

```
wsdl2java -server -impl -d server -ant .\wsdl\HelloWorld.wsdl
```

to generate starting point code, including the server mainline and implementation object, for the server application.

In each of the top_level_application directories – `client` and `server` – the `wsdl2java` utility creates an Ant `build.xml` file. After you complete coding in the client mainline and implementation object, issue the command `ant build` in each of these directories to compile your applications.

Running the Applications using Ant

You can use Ant to run each of the applications. The actual syntax of the command depends on the name of the port type defined in the WSDL file. For example, in the `HelloWorld.wsdl` file, the port type is named `HelloWorld`, and the `build.xml` file targets used to run the client and server applications will be `HelloWorld.Client` and `HelloWorld.Server`.

Running the Applications using java

The Celtix runtime uses the `java.util.logging` framework. You can configure Celtix logging levels by pointing the JVM to a `logging.properties` file by defining the JVM system variable `java.util.logging.config.file`. Celtix provides a default `logging.properties` file in the `etc` directory, so you can use:

```
-Djava.util.logging.config.file=%CELTIX_HOME%/etc/logging.properties
```

To run the server application using `java`, move to the `server/build/classes` directory and issue the command:

```
java -Djava.util.logging.config.file=%CELTIX_HOME%/etc/logging.properties
     <full package name of server>
```

And to run the client application, move to the `client/build/classes` directory and issue the command:

```
java -Djava.util.logging.config.file=%CELTIX_HOME%/etc/logging.properties
     <full package name of client> <path to WSDL file>
```

Note that you must supply the relative, or absolute, path to the WSDL file when running the client.

The Celtix wsdl2java Utility

Using the `wsdl2java` utility gives you greater control over the code generation process. The syntax of the command is summarized in CodeSnap 6.

```
wsdltojava -p <[wsdl namespace =]Package Name>* -b <binding-name>*
           -d <output-directory> -compile -classdir <compile-classes-directory>
           -client -server -impl -all -ant
           -nexclude <schema namespace [= java packagename]>*
           -exsh <enable extended soap header message binding (true, false)>
           -v -verbose -quiet <path to WSDL file>
```

CodeSnap 6 `wsdl2java` syntax.

The various command line arguments that you use to manage code generation are reviewed in the following table. Refer to the online documentation for the most up-to-date description.

Command Line Argument	*Interpretation*
-? -help -h	Display the online help for this utility
-p <[wsdl namespace=]Package name>*	Zero, or more, package names to use for the

Command Line Argument	*Interpretation*
	generated code. Optionally, specify the WSDL namespace to package name mapping.
-b <binding file name>*	Zero, or more, JAXWS or JAXB binding files. Use the space character to separate multiple entries.
-d <output directory>	The directory into which to place the generated code.
-compile	Compile generated java file
-classdir	Specify the directory to place the compiled classes
-client	Generate starting point code for the client mainline.
-server	Generate starting point code for the server mainline.
-impl	Generate starting point code for the implementation object.
-all	Generate all starting point code: types, service proxy, service interface, server mainline, client mainline, implementation object, Ant `build.xml` file.
-ant	Generate the Ant `build.xml` file.
-nexclude <schema namespace [= java packagename]>	The wsdl namespace to exclude for generating code. This option can be specified multiple times. Also, Optionally specify the java package name to use for this wsdl namespace.
-exsh <enable extended soap header message binding (true, false)>	To enable processing of extended soap header message binding
-v	Display the version number for the tool.
-verbose	Display comments during the code generation process.
-quiet	Suppress comments during the code generation process.

All command line arguments are optional and, if used, may be listed in any order. You must, however, include the absolute or relative path to the WSDL file as the last argument.

The Celtix java2wsdl Utility

The java2wsdl tool generates the wsdl used in celtix development ,deployment . This tool ues web service endpoint class and types classes to genearte wsdl. The syntax of the command is summarized in CodeSnap 7

```
javatowsdl -o <output-file>
           -t <target-namespace>
           -servicename <service-name>
           -h
           -v
           -verbose
           -quiet <classname>
```

CodeSnap 7 `java2wsdl` *syntax.*

Command Line Argument	*Interpretation*
-? -help -h	Display the online help for this utility
-o <output-file>	Zero or more. Specify the generated wsdl file name . Default we use the “” to name the generated wsdl file.
-t <target-namespace>	Zero, or more, specify the target namespace to use for the generated wsdl.
-servicename <service-name>	Zero, or more , specify the `wsdl:service` name for the generated WSDL .
<classname>*	Name of the service endpoint class
-v	Display the version number for the tool.
-verbose	Display comments during the code generation process.
-quiet	Suppress comments during the code generation process.

The Celtix wsdl2soap Utility

This tool will generate a SOAP binding for a given wsdl portType from an exsiting wsdl .

```
wsdltosoap -i <port-type-name> -b <binding-name>
           -d <output-directory> -o <output-file>
           -n <soap-body-namespace(only for rpc style)>
           -style <soap binding/operation style(document/rpc)>
           -use <soap body use(literal/encoded)>
           -h -v -verbose -quiet
           <wsdlurl>
```

CodeSnap 8 `wsdl2soap` *syntax.*

Command Line Argument	***Interpretation***
-? -help -h	Display the online help for this utility
-i <port-type-name>*	Specify the portType to use.
-b <binding-name>	Specify the generated SOAP binding name
-d <output-directory>	Specify the directory to place generated wsdl file
-o <output-file>	Specify the generated wsdl file name
-n <soap-body-namespace(only for rpc style)>	Specify the soap body namespace when style is RPC
-style <soap binding/operation style(document/rpc)>	Specifies the encoding style(Document/RPC) to use in the SOAP binding(default : Document)
-use <soap body use(literal/encoded)>	Specify the binding use (encoded/literal) to use in the soap binding.(default : Literal)
<wsdlurl>*	The existing wsdl location
-v	Display the version number for the tool.
-verbose	Display comments during the code generation process.
-quiet	Suppress comments during the code generation process.

The Celtix wsdl2service Utility

```
wsdltoservice -transport <soap/http> -e <service-name>
              -p <port-name> -n <attribute-binding-name>
              [ -a <address> ] -o <output-file>
              -d <output-directory> -h -v -verbose -quiet
              <wsdlurl>
```

CodeSnap `wsdl2service` *syntax.*

Command Line Argument	*Interpretation*
-? -help -h	Display the online help for this utility
-transport <soap/http>	If the payload being sent over the wire is SOAP, use -transport soap. For all other payloads use -transport http.
-e <service-name>	Specifies the name of the generated service
-p <port-name>	Specifies the value of the name attribute of the generated port element, sperate by space
-a <address>	Specifies the value used in the address element of the port, sperate by space
-o <output-file>	The output wsdl file name
-d <output-directory>	The directory in which the generated wsdl is placed
-v	Display the version number for the tool.
-verbose	Display comments during the code generation process.
-quiet	Suppress comments during the code generation process.

The Celtix wsdl2xml Utiltily

```
wsdltoxml -i <port-type-name> -b <binding-name> -e <service-name>
          -p <port-name> -a <address> -d <output-directory>
          -o <output-file> -n <soap-body-namespace(only for rpc style)>
          -style <soap binding/operation style(document/rpc)>
          -use <soap body use(literal/encoded)> -h -v -verbose -quiet
          <wsdlurl>
```

CodeSnap `wsdl2xml` *syntax.*

Command Line Argument	*Interpretation*
-? -help -h	Display the online help for this utility
-i <port-type-name>*	Specify the portType to use.
-b <binding-name>	Specify the generated SOAP binding name
-e <service-name>	Specifies the name of the generated service
-p <port-name>	Specifies the value of the name attribute of the generated port element, sperate by space
-a <address>	Specifies the value used in the address element of the port, sperate by space
-d <output-directory>	Specify the directory to place generated wsdl file
-o <output-file>	Specify the generated wsdl file name
-n <soap-body-namespace(only for rpc style)>	Specify the soap body namespace
-style <soap binding/operation style(document/rpc)>	Specifies the encoding style(Document/RPC) to use in the SOAP binding(default : Document)
-use <soap body use(literal/encoded)>	Specify the binding use (encoded/literal) to use in the soap binding.(default : Literal)
-v	Display the version number for the tool.
-verbose	Display comments during the code generation process.
-quiet	Suppress comments during the code generation process.

The Celtix xsd2wsdl Utiltily

```
xsdtowsdl -t <target-name-space> -n <wsdl-name>
          -d <output-directory> -o <output-file>
          -h -v -verbose -quiet
          <xsdurl>
```

CodeSnap `xsd2wsdl` *syntax.*

Command Line Argument	*Interpretation*
-? -help -h	Display the online help for this utility
-t <target-name-space>	Specify the target name space for the generated wsdl
-n <wsdl-name>	Specify the wsdl:definition name for the generated wsdl
-d <output-directory>	The directory in which the generated wsdl is placed
-o <output-file>	The output wsdl file name.
-v	Display the version number for the tool.
-verbose	Display comments during the code generation process.
-quiet	Suppress comments during the code generation process.