

BEA Products

Tools Guide

Version 3.4

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA, Inc. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with that agreement. No part of this guide may be reproduced or retransmitted in any form or by any means electronic, mechanical, or otherwise, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of BEA, Inc.

Copyright © 2001-2007 BEA, Inc. All rights reserved. All BEA Products are trademarks or registered trademarks of BEA, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

TOOLS.....	4
DESIGNER TOOLS.....	4
DATAGEN	5
FLATTEN	6
MAPPER	7
TESTSWIFT	9
EXPORT.....	11
DUMPMESSAGES	11
CODEGEN	12
BUILDING CARTRIDGES WITH ANT	13
CARTRIDGE PUBLISHER	15
SCHEMA2CAR	15
JAVASIMULATOR	16
TEMPLATE	16
CSV EXPORT	17
SQL TOOLS.....	18
SCHEMA GENERATOR	18
SQL GENERATOR	20
EXECUTE SQL	21
SQL CONSOLE	22
QUEUE TOOLS	23
SEND UTILITY	24
RECEIVE UTILITY.....	25
HTTP SEND UTILITY	25
SOAP SEND UTILITY	26

Tools

Designer comes with a set of utilities for the convenience of the user. The available tools are,

- [Designer Tools](#)
- [SQL Tools](#)
- [Queue Tools](#)
- [HTTP Send Utility](#)
- [SOAP Send Utility](#)

Designer Tools

This section explains the various Designer utilities available. These include some utilities, which are specific for testing SWIFT formats only.

The batch files for the tools assume that you are working on Windows NT platform (NT4, 2000, XP or 2003).

The utilities that are supported are

- [DataGen](#)
- [Flatten](#)
- [Mapper](#)
- [TestSwift](#)
- [Export](#)
- [DumpMessages](#)
- [CodeGen](#)
- [Building Cartridges with Ant](#)
- [Cartridge Publisher](#)
- [Schema2car](#)
- [JavaSimulator](#)
- [Template](#)
- [CSV Export](#)
- [SQL Tools](#)

See Also:

[Tools](#)

DataGen

This utility generates test data as .dat files for swift message formats.

Syntax

```
datagen [options] {format} {no of testcases } [random seed]
(or)
datagen [options] {cartridge!format} {no of testcases } [random seed]
```

Where

- **Options**
 - input Generate swift sample with input header
 - output Generate swift sample with output header (default option)
- **format** is the swift message format for which .dat files have to be generated.
- **cartridge!format** is the name of cartridge file(.car) along with the name of the swift format in the cartridge file for which .dat files have to be generated. The format is necessary because a cartridge may contain multiple swift formats. The cartridge name and the format name should be separated by '!' symbol.
- **no of testcases** specifies the number of .dat files to be generated.
- **random seed** is the optional random seed value. Specify the same random seed to generate the same set of test cases.

Note

Please note that if the format for which data has to be generated has network validation rules, the data generated will not always conform to the rules. However the data generated will conform to code validation rules that have been specified in the format.

Output Directories

- If '**format**' option is specified then the .dat files will be generated in the directory from where this utility is run.
- If '**cartridge!format**' option is specified then the .dat files will be generated in the directory where the cartridge file is present.

Examples

```
datagen MT543 100
```

```
datagen MyCar.car!MyFormat 100
datagen -input MT543 100
```

Please refer [Designer Tools](#) for the list of other utilities that are available.

Flatten

This utility generates an XML file to represent the internal message format corresponding to the swift format specified as argument or the swift format in a cartridge in case a cartridge is specified as argument.

This generated file is flattened (i.e.) the inner fields in the format are represented by combining the section names and the field names.

It also generates XML files for input format mapping and output format mapping. These files can be imported to a cartridge containing input and output formats for which these XML files have been generated. This minimizes the task of entering the normalized object fields manually as well as input and output mapping.

Flattening Rules:

Non-repeating sections are removed (flattened) and all its fields are moved to the parent format. This applies to both top-level and nested sections.

If a section to be flattened is optional, all its child fields are made optional (in the flattened structure).

Syntax

```
flatten {format} [-noheadertrailer] [-input|-output]
(or)
flatten {cartridge!format} [-noheadertrailer]
```

Where

- **Format** is the swift message format for which is to be flattened.
- **cartridge!format** is the name of cartridge file(.car) along with the name of the swift format in the cartridge file which is to be flattened. The format is necessary because a cartridge may contain multiple swift formats. The cartridge name and the format name should be separated by '!' symbol.
- If **noheadertrailer** option is specified the flattened XML files will not contain header/trailer fields. By default the flattened XML files will contain header/trailer fields.

- If **input** option is specified then the flattened XML files will contain input header fields.
- If **output** option is specified then the flattened XML files will contain output header fields. This is the default option.

Note:

- If -input|-output options are not given by default the generated XML files will contain output header fields.
- Please note that the -input and -output options need not be specified when you are flattening a format in a cartridge. The flatten utility will generate the XML files based on the type of header selected for the format that is to be flattened.

Output Directories

- If '**format**' option is specified then the XML files for the normalized object and input/output mapping will be generated in the directory from where this utility is run.
- If '**cartridge!format**' option is specified then the xml files for the normalized object and input/output mapping will be generated in the directory where the cartridge file is present.

Examples

```
flatten MT543
flatten MyCar.car!MyFormat
flatten MT543 -input
flatten MT543 -noheadertrailer
```

Please refer [Designer Tools](#) for the list of other utilities that are available.

Mapper

This is similar to 'Flatten' utility, but the generated XML files are not flattened.(i.e) sections in the swift message format are represented in the generated XML files.

These files can be imported to a cartridge containing input and output formats for which these XML files have been generated. This minimizes the task of entering the normalized object fields manually as well as input and output mapping.

Syntax

```
mapper {format} [-noheadertrailer] [-input|-output]
      (or)
mapper {cartridge!format} [-noheadertrailer]
```

Where

- **Format** is the name of the swift message format for which XML files for the normalized object and for input/output mapping have to be generated.
- **Cartridge!format** is the name of cartridge file(.car) along with the name of the swift format in the cartridge file for which XML files have to be generated for the normalized object and input/output mapping. The format is necessary because a cartridge may contain multiple swift formats. The cartridge name and the format name should be separated by '!' symbol.
- If **noheadertrailer** option is specified, the generated XML files will not contain header/trailer fields. By default the generated XML files will contain header/trailer fields.
- If **input** option is specified then the generated XML files will contain input header fields.
- If **output** option is specified then the generated XML files will contain output header fields. This is the default option.

Note:

If -input|-output options are not given by default the generated XML files will contain output header fields.

Please note that the -input and -output options need not be specified when you are generating XML files for format in a cartridge. The mapper utility will generate the XML files based on the type of header selected for the format for which XML files are to be generated.

Output Directories

- If '**format**' option is specified then the XML files for the normalized object and input/output mapping will be generated in the directory from where this utility is run.
- If '**cartridge!format**' option is specified then the XML files for the normalized object and input/output mapping will be generated in the directory where the cartridge file is present.

Examples

```
mapper MT543
mapper MyCar.car!MyFormat
mapper MT543 -noheadertrailer
mapper MT543 -input
```

Please refer [Designer Tools](#) for the list of other utilities that are available.

TestSwift

This utility is an automated test runner used for testing swift runtime.

WORKING:

1. Using the information in the format (xml file) given as argument it generates the parser class for it. The parser parses the input and converts it to internal object structure.
2. It also generates an output writer that converts the internal object back to swift data file.
3. It then generates test cases (again based on the format), loads them using the parser and writes it back with different name using the writer.
4. It then compares the input data file with the output generated and ensures that they are the same.

If it encounters a problem during parsing, writing or during comparison, the error will be logged to the console and processing will continue. Finally the number of testcases completed and the number of failures among them will be displayed in the console.

Syntax

```
testswift {format | cartridge} {#testcases | testcasefile/dir}
[-input | -output] [-nodiff] [-validate] [-ignoreenvr] [-seed random seed]
```

Where

- *format* is the name of the swift message format that is to be tested. It can be one of the known swift formats.

- **cartridge** should be of the form **cartridgeFileName!formatName**. The name of the cartridge file(.car) along with the name of the swift format in the cartridge file that is to be tested should be given. The format is necessary because a cartridge may contain multiple swift formats. The cartridge name and the format name should be separated by '!' symbol.
- **#testcases** specifies the number of test cases to be generated and used.
- **testcasefile** specifies the test case file or a directory containing testcases.
- **[-input / -output]** - The header type to be used while testing the format.By default the header type is Output.
- **[-validate]** - If this option is specified then the validations specified for the format/cartridge will be applied during the testswift process.
- **-nodiff** is an optional argument that specifies whether the input test file and the generated output file should be checked for any difference between them.
- **-validate** is an optional argument. If this is specified then any validations (code as well as network) that have been specified for the format will be applied to the test process. If the data does not conform to validations specified error would be thrown. Otherwise the validations specified will be ignored. In such cases if the data contains some validation errors it will be ignored.
- **-ignorenvr** is an optional argument. If this is specified, network validations that have been specified in the format will be ignored. Only the code validations will be applied to the test process. In such cases even if the data does not conform to network validations no error will be thrown.
- **random seed** is the optional random seed value. This enables you to exactly reproduce a generated test set.

Output Directories

- If '**format**' option is specified then the .dat files and the swift data file will be generated in the directory from where this utility is run.
- If '**cartridge!format**' option is specified then the .dat files and the swift data file will be generated in the directory where the cartridge file is present.

However in both cases the parser class and the output writer class will always be generated in the directory 'work' which will be created in the directory from where this utility is run.

Examples

```
testswift MT543 100 -nodiff
testswift MT543 MT543Test.dat
testswift MyCar.car!MT543In /usr/swiftdata/MT543 -validate
testswift MyCar.car!MT543In /usr/swiftdata/MT543 -validate -ignoreenvr
```

Please refer [Designer Tools](#) for the list of other utilities that are available.

Export

This is a utility for exporting a message (internal/external) in a cartridge to some external representation (xml).

Syntax

```
export cartridge!format type [filename]
```

Where

- **cartridge!format** is the name of the cartridge file(.car) along with the name of the format in the cartridge file that is to be exported. The format is necessary because a cartridge may contain multiple formats. The cartridge name and the format name should be separated by '!' symbol.
- **type** is the type in which the format is to be exported ('xml').
- **filename** is the name of the file to which the format is to be exported. This is an optional property.

Examples

```
export MyCar.car!MyFormat xml
```

Please refer [Designer Tools](#) for the list of other utilities that are available.

DumpMessages

This is a utility for dumping the structure of a cartridge. The cartridge name, internal message name and names of Input/Output formats will be dumped in a hierarchical structure similar to a tree.

Syntax

```
dumpmessages cartridge
```

Where

- **cartridge** is the name of the cartridge file(.car) whose structure is to be dumped.

Examples

```
dumpmessages MyCar.car
```

Please refer [Designer Tools](#) for the list of other utilities that are available.

CodeGen

This utility generates code for one or more cartridge files in any one of the following platforms: Java, CPP and C#.

Syntax

```
codegen -platform=xxx cartridgefiles ...
```

where

- **xxx** in -platform=xxx can be one of java, CPP or C#
- **cartridgefiles** is the name of the cartridge file(s) for which code has to be generated. If multiple cartridges are specified they must be separated by 'space'.
- The cartridge file name must be a fully qualified file name. Otherwise the cartridge should be present in the same directory from where this utility is being executed.

Examples

- codegen "-platform=java" c:/carts/mycart.car(to generate code for a single cartridge)
- codegen "-platform=CPP" c:/carts/mycart.car c:/other/carttwo.car(to generate code for multiple cartridges)

Please refer [Building Cartridges with Ant](#) for information on the Ant task that can be used for cartridge generation from the Ant build script.

Please refer [Designer Tools](#) for the list of other utilities that are available.

Building Cartridges with Ant

If you are using the Ant tool to build your application, you can include cartridge generation as part of the build process by using the Ant task CGTask bundled with Designer.

Description

The Ant task CGTask can be used to generate a cartridge into platform specific code. The supported platforms are Java, C++ and C#.

Parameters

Attribute	Description	Required
Platform	The platform in which to generate the cartridge. The supported platform values are Java, C++ and C#.	Yes
Cartridge	The path of the cartridge file to be generated.	Yes
Home	The path to the Designer home directory.	Yes

Example

Given below is the content of the build.xml file under the 'installation dir>\docs\Designer\CodeGenerator' directory. This is used to generate PurchaseOrder.car cartridge (under the 'CodeGenerator\Cartridges\PurchaseOrder' directory) in Java platform. The build.xml is explained below.

```
<?xml version="1.0" standalone="yes"?>
<project basedir="." default="compile">
  <property name="designer.home" value="../../.." />
  <property name="build.dir" value="${basedir}" />
  <property name="cart.dir" value="${basedir}/Cartridges" />
  <!--===== -->
  <!-- Define the code generator task -->
  <!-- ===== -->
  <path id="classpath">
    <fileset dir="${designer.home}/lib" includes="*.jar" />
  </path>
  <taskdef name="CGTask"
    classname="com.tplus.transform.design.ui.CGTask">
    <classpath refid="classpath" />
  </taskdef>
</project>
```

```

</taskdef>
<!-- ===== -->
<!-- Builds the cartridge (using the task above) -->
<!-- ===== -->
<target name="compile" description="Compile cartridge">
    <echo message="Building the cartridge..." />
    <CGTask platform="Java"
            cartridge="${cart.dir}/PurchaseOrder/PurchaseOrder.
car"
            home="${designer.home}" />
</target>
</project>

```

Explantation

- In the build.xml file, the following define the properties **designer.home**, **build.dir** and **cart.dir**.

```

<property name="designer.home" value="<installation dir>" />
<property name="build.dir" value="${basedir}" />
<property name="cart.dir" value="${basedir}/Cartridges" />

```

Note that **designer.home** refers to the Designer installation directory and changes from machine to machine. Hence, build XML file needs to be updated on each machine. To avoid this, consider using build properties file supported by ANT to externalize designer.home property from build.xml.

- The following defines the classpath for all the JAR files under the '<designer.home>/lib' directory. The classpath will be used by the CGTask.

```

<path id="classpath">
    <fileset dir="${designer.home}/lib" includes="*.jar" />
</path>

```

- The following defines the CGTask with the classpath defined above.

```

<taskdef name="CGTask"
        classname="com.tplus.transform.design.ui.CGTask">
    <classpath refid="classpath" />
</taskdef>

```

- The following defines the **compile** target. It invokes the **CGTask** defined above to generate the '<cart.dir>/PurchaseOrder/PurchaseOrder.car' cartridge in Java platform. Note that the platform code is generated under the Cartridge directory. For example, in case of Java code generation, a directory named 'java' is created under the same directory as that of the cartridge and the generated JAR files and

other files are found under that 'java' directory. From the ANT script you can copy the generated JAR files into the application directory or any other directory as required.

```
<CGTask platform="Java"
        cartridge="${cart.dir}/PurchaseOrder/PurchaseOrder.car"
        home="${designer.home}"/>
```

Please refer [CodeGen](#) for information on utility that generates code for one or more cartridge files.

Please refer [Designer Tools](#) for the list of other utilities that are available.

Cartridge Publisher

This utility publishes a cartridge in HTML format. Note that the cartridge should have been saved in XML format.

Syntax

```
cartridgepublisher cartridgefilename [outputdirectory]
```

where

- **cartridgefile** is the name of the cartridge for which documentation has to be generated.
- **outputdirectory** is the name of the directory where the cartridge is to be published. This property is optional. If this is not specified the cartridge will be published in the directory where the cartridge is present under 'docs' directory.

Examples

- cartridgepublisher C:\MyCarts\MyCart.car
- cartridgepublisher C:\MyCarts\MyCart.car C:\MyCarts\docs

Please refer [Designer Tools](#) for the list of other utilities that are available.

Schema2car

This utility is used to create a cartridge with XML format based on a schema.

Syntax

```
schema2car [options] schemafile
```

where

- **schemafile** is the name of the schema based on which the XML format is to be created.
- **options** should be of the form **-root rootElementName** where 'rootElementName' is the name of the root element in the schema. This property is optional.

Examples

- `schema2car po.xsd`

Please refer [Designer Tools](#) for the list of other utilities that are available.

JavaSimulator

This utility is used to start simulator as a standalone application. You need not start designer to work in this Simulator.

You can deploy directories directly in the java simulator.

Syntax

```
JavaSimulator
```

Please refer [Designer Tools](#) for the list of other utilities that are available.

Template

This utility uses velocity to generate an output file based on an input template file.

Syntax

```
template templateFileName outputFileName [inputparam1 inputparam2 ....]
```

where

- **templateFileName** is the name of the file based on which the output file is to be generated.
- **outputFileName** is the name of the output file to be generated.

- `inputparam1 inputparam2` are parameters that can be applied to the input template file. These parameters are optional and should be specified only if the input template file needs any input values.

Examples

- `template jndi.properties jndi.properties "APPNAME=Transform"`
- `template run.bat execute.bat`

Please refer [Designer Tools](#) for the list of other utilities that are available.

Please refer the section 'New File from Template' in **Designer Guide** documentation for the available templates based on which files can be created.

CSV Export

This utility is used for exporting internal/external message formats and message mappings as CSV. In case of external message formats, CSV export is now supported only for the following plug-ins: ASCII Delimited, ASCII Fixed, Universal and XML.

Syntax

```
exportcsv cartridge format filename
exportcsv -settings
```

where

- **cartridge** is the name of the cartridge file.
- **format** is the name of the internal/external message whose 'Internal/External Format' design element has to be exported or it is the name of the message mapping whose 'Mapping Rules' design element has to be exported.
- **filename** is the name of the file to which the format has to be exported.
- the second syntax launches the 'Configure CSV Reports' dialog and it allows the user to customize CSV export. In particular, it allows the user to select the design element properties to be included in the generated CSV Report.

Examples

- `exportcsv MyCar.car MyFormat mymessage.csv`

Please refer [Designer Tools](#) for the list of other utilities that are available.

SQL Tools

This section explains the various SQL tools available.

The batch files for the tools assume that you are working on Windows NT platform (NT4, 2000, XP or 2003)

The utilities that are supported are

- [Schema Generator](#)
- [SQL Generator](#)
- [Execute SQL](#)
- [SQL Console](#)
- [Designer Tools](#)
- [Queue Tools](#)

See Also:

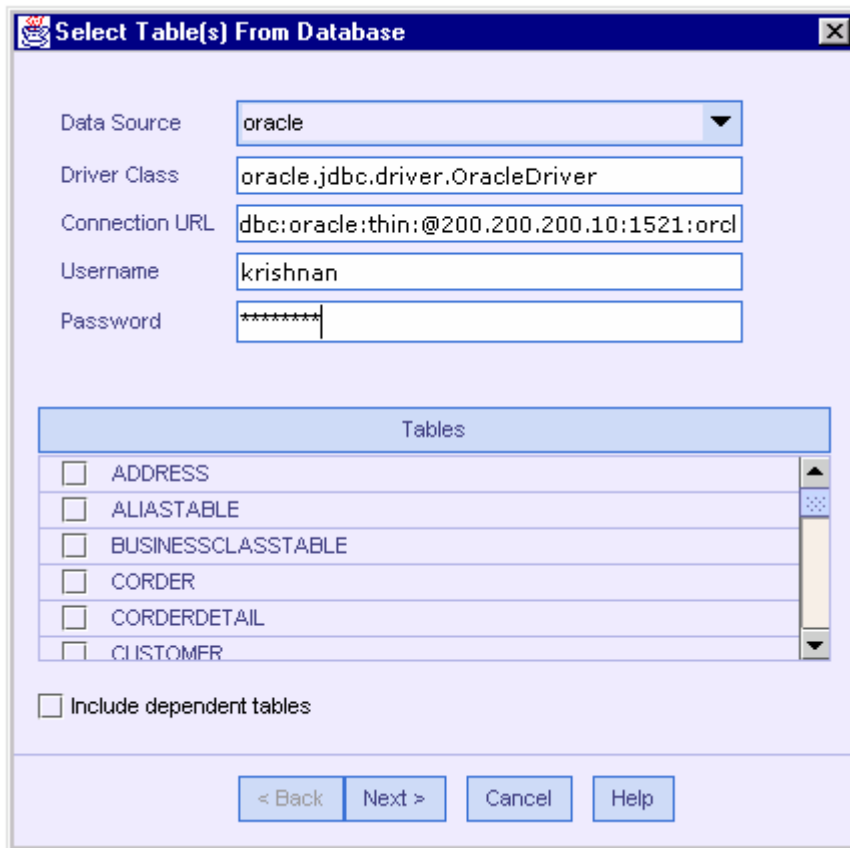
[Tools](#)

Schema Generator

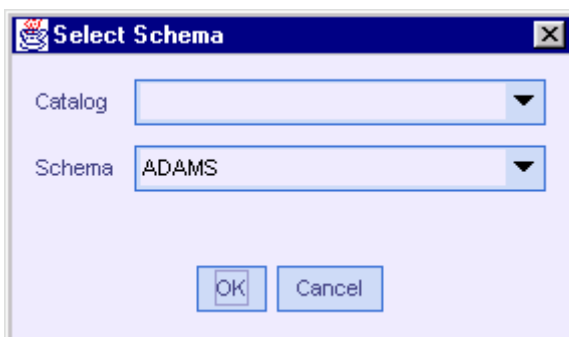
The 'Schema Generator' can be used to generate db independent XML schema files as well as db specific SQL schema files for the tables in a data source.

Steps to Generate a Schema

1. Select the **Schema Generator** menu item from the **Tools** menu.
2. In the "Select Table(s) From Database" dialog that appears, specify the db connection properties and then select the **Tables** button.

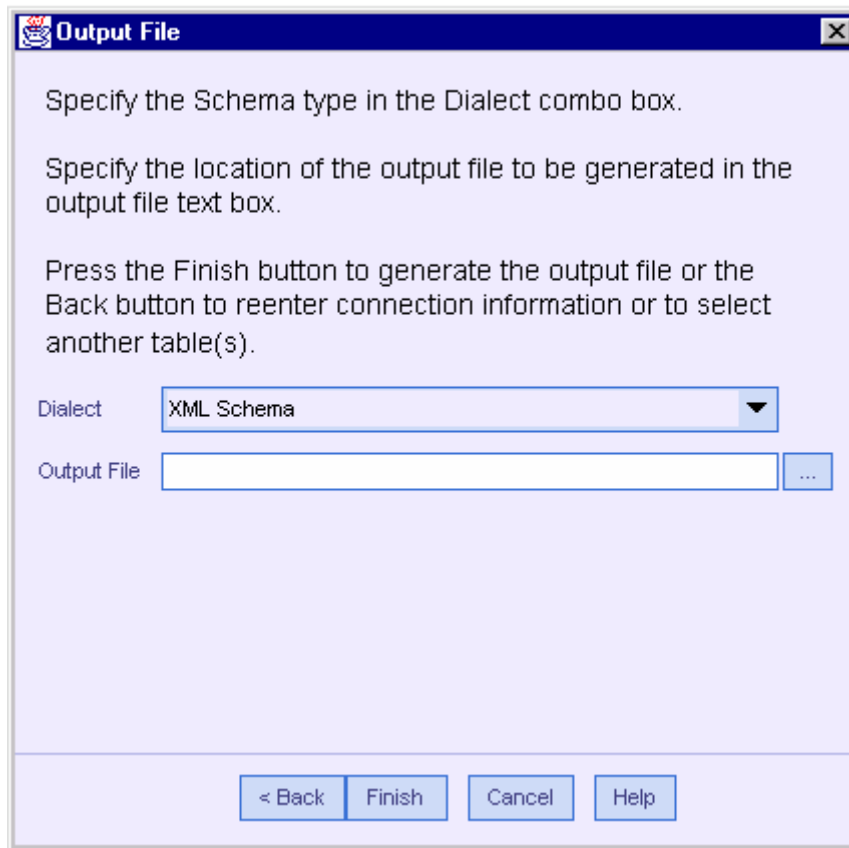


3. In case of Oracle this step brings the "Select Schema" dialog. Once the required catalog and schema are selected, click the OK button to populate tables from the specified data source.



4. Specify the tables whose schema needs to be inserted by selecting the check box besides the corresponding table.
5. Select the **Include Dependent Tables** check box, if you want the dependent tables of the selected tables should also be inserted and select the Next button. (In case of a table having its dependent tables, the entire tree of tables is inserted with their parent and child relationships intact.)

6. In the **Output File** dialog box, specify the Schema type in the Dialect combo box and the location of the output file to be generated in the Output File text box and select Finish.



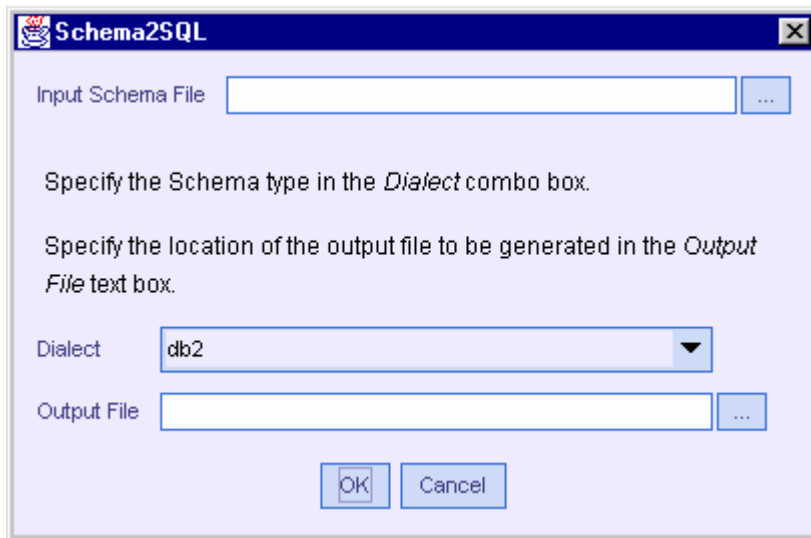
Please refer [SQL tools](#) for list of other available SQL tools.

SQL Generator

The 'SQL Generator' can be used to generate db specific SQL commands for the given XML schema file, which is a db independent way of specifying SQL commands.

Steps to Convert an XML Schema to an SQL Schema

1. Select the **SQL Generator** menu item from the **Tools** menu.
2. In the **Schema2SQL** dialog box that appears specify the following details and select OK.



3. In the **Input Schema File** text box, specify the XML scheme file that needs to be converted to db specific SQL schema.
4. In the **Dialect** combo box, select the db for which you want to generate the SQL schema.
5. In the **Output File** text box, specify the location of the SQL Schema file to be generated.

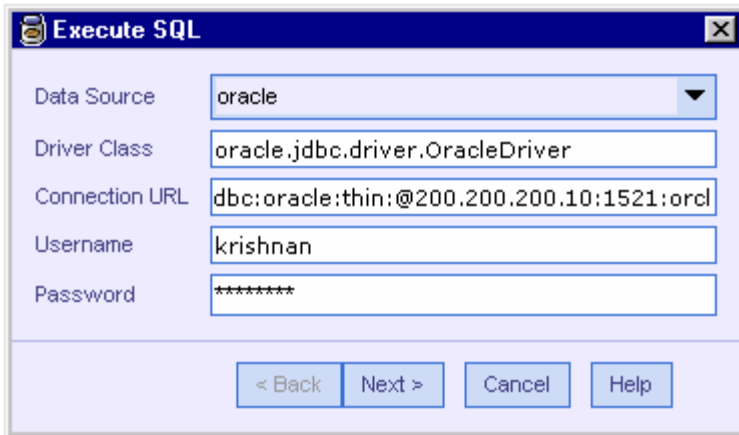
Please refer [SQL tools](#) for list of other available SQL tools.

Execute SQL

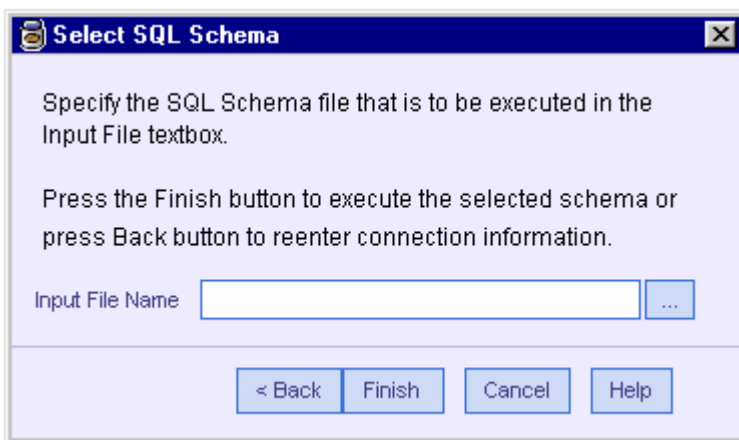
This tool executes a set of SQL commands against the specified data source, based on the given schema file. This tool supports both XML schema files and SQL schema files.

Steps in Executing a Schema

1. Select the **Execute SQL** menu item from the **Tools** menu.
2. In the **Connection Information** dialog box that appears specify the db connection details and select the Next button.



3. In the **Select SQL Schema** dialog that appears, specify the SQL schema file to be executed in the **Input File Name** text box.



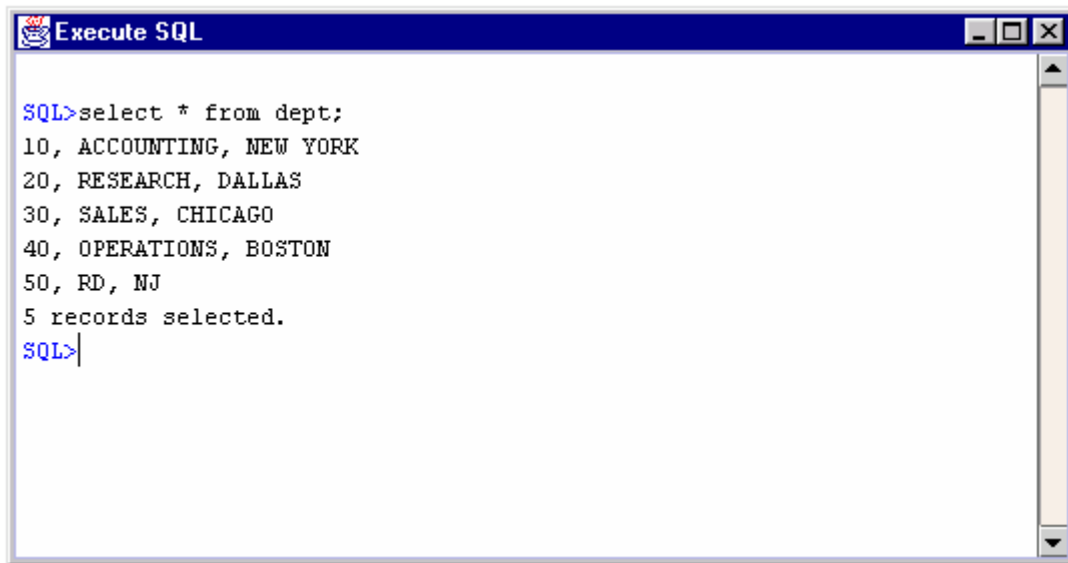
Please refer [SQL tools](#) for list of other available SQL tools.

SQL Console

This tool can be used to execute SQL commands against the specified data source just like Oracle SQL*Plus.

Steps to launch the SQL Console

1. Select the **SQL Console** menu item from the **Tools** menu.
2. In the Connection Information dialog box specify the db connection details and select the OK button to bring the console on screen.



You can use the **cls** command to clear the console and use the up arrow to select a command from the command history.

Please refer [SQL tools](#) for list of other available SQL tools.

Queue Tools

This section explains the various Designer queue utilities available. These utilities can be used to send/receive data to/from a queue defined in an Application server. The Application servers for which these utilities are supported are

- WebSphere
- WebLogic
- JBoss
- Orion

The batch files for the tools assume that you are working on Windows platform (NT4, 2000, XP or 2003)

The queue utilities that are available are

- [Send Utility](#)
- [Receive Utility](#)
- [SQL Tools](#)
- [SOAP Send Utility](#)
- [HTTP Send Utility](#)

See Also:

[Tools](#)

Send Utility

This utility is used to send data to a queue present in an Application server/ IBM MQ.

- In setpath.bat set the correct path for the Application server where the queue to which data is to be sent is present.
- In send.properties specify the appropriate value properties such as Queue Name, Queue Connection factory etc.

Syntax

```
send [options] {datafile}
```

Where

- **Options**

-bytes

Writes the data to queue in bytes format.

-properties propertyfile

Writes the properties specified in the property file to the message before sending it to the queue.

-count countvalue

Sends the data file specified, to the queue as many times as specified by the count value. If the count value specified is 10, the data file will be sent to the queue 10 times.

- **datafile** is the name of the file whose content is to be sent to the queue.

Note:

By default the data is written to the queue in Text format.

Examples

```
send data1.txt
```

```
send -bytes data1.txt
```

```
send -properties message.properties data1.txt
```

```
send -count 10 data1.txt
```


See Also:

[Receive Utility](#)

Receive Utility

This utility is used to retrieve data that is present in a queue in an Application Server/ IBM MQ.

- In setpath.bat, set the correct path for the Application server where the queue from which data is to be retrieved is present.
- In receive.properties specify the appropriate value properties such as Queue Name, Queue Connection factory etc.

If the queue from which data is to be retrieved does not have any data, the utility will be waiting until data is put into the queue. Once data is put into the queue it will be retrieved by the utility.

Syntax

```
receive
```

The message present in the queue along with its properties will be dumped in the console from where the receive utility is being executed.

Examples

```
receive
```

See Also:

[Send Utility](#)

HTTP Send Utility

This utility can be used to send HTTP requests to the specified host and port.

Syntax

```
send url method [file]
```

Where

- `url` is the request URL. URL of the resource to be requested.

- **method** is the HTTP method to be invoked. The methods are listed below.

GET
POST
HEAD
OPTIONS
PUT
DELETE
TRACE

- **file** is the file that contains the content to be included in the HTTP request body.

Examples

send "http://localhost:8086/test?param=value" GET

See Also:

[SOAP Send Utility Tools](#)

SOAP Send Utility

This utility can be used to send SOAP requests.

Syntax

```
send wsdlLocation operationName[<portname>][argument1...]
```

Where,

- **wsdlLocation** is the URL for the WSDL file.
- **operationName** is the name of the WebService to be invoked.
- **argument1...** is the optional list of arguments to be passed to the WebService. You can specify a file name or the actual value.

Examples

```
send http://localhost:8081/axis/services/pass?wsdl PassThroughFlow  
data1.txt
```

See Also:

[HTTP Send Utility
Tools](#)