



AGETOR[®]

Accessing Database
Stored Procedures

Table of contents

1	Introduction.....	3
1.1	Access to databases from JAVA	3
1.2	Leveraging your existing application.....	4
1.3	What AGETOR® provides	5
1.3.1	Modification to IDL2JAVA.....	5
1.3.2	Preparation of IDL-objects.....	5
1.3.3	Generation of server.....	5
1.3.4	Possibility to modify auto-generated code.....	5
2	Configuration	6
2.1	Configuring idl2java	6
2.2	Configuring database properties.....	6
2.3	Configuring generated services	6
2.4	Configuring access to Stored Procedures	6
2.4.1	Tag <type>	7
2.4.2	Tag <method>	7
2.5	Configuring arrays	7
3	Technical issues.....	8
3.1	Writing the IDL	8
3.1.1	Order of attributes	8
3.2	Overwriting the server.....	8
3.3	Handling of nulls	8
3.4	Description of used interfaces	8
3.4.1	Interface: java.sql.SQLData	8
3.4.2	Interface: dk.bordong.inside.storage.jdbc.sp.SQLPrepared	9
3.5	Database specific topics	9
3.5.1	Oracle Objects and Records	9
3.5.2	Handling of Arrays and sequences	9

1 Introduction

This guide will show you how to access business logic in Database Stored Procedures using AGETOR® Portal Server. The tools in the toolbox will make this otherwise quite tedious task much easier by auto-generating a substantial part of the code necessary. This will allow you to focus on your application to present the data, rather than the tedious programmatic server-side tasks involved in obtaining access to Database Stored Procedures themselves.

The guide is divided into three parts. The first part is the introductory section you are reading now. It will provide the overall understanding of access to DB Stored Procedures and explain the basic concepts. The second part delves into the configuration needed to setup access to Database Stored Procedures with AGETOR®. Finally, the third part describes the more technical issues, including how to overwrite generated code to suit a specific need.

It is assumed that you have experience with development using AGETOR® Portal Server, Java®, JDBC and other technical areas as these – and other – concepts are not explained herein.

1.1 Access to databases from JAVA

Accessing Database Stored Procedures can be compared to accessing a relational database from JAVA® using JDBC. In that case, you will need to set up a connection to the database using a suitable driver, prepare the statement to the driver, possibly provide some data to the statement, execute the prepared statement and finally you will extract the data returned by the statement from the driver.

Accessing Database Stored Procedures involves the same steps and one additional, namely that you need to prepare the driver for what output to be expected.

Finally, you need to write a client to do what is needed to the data obtained from the Database Stored Procedure.

This is illustrated in Figure 1

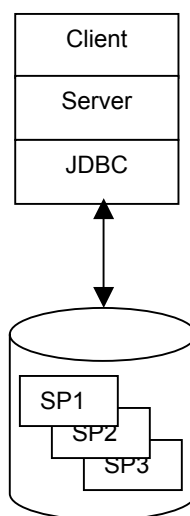


Figure 1: Accessing stored procedures

1.2 Leveraging your existing application

If you have some existing business-logic in a Database Stored Procedure, it is quite easy to access this stored procedure using AGETOR®. All you have to do is write an IDL, use the tool `idl2java`, setup the auto-generated server and you're ready to go.

Figure 2 describes the steps necessary to access Database Stored Procedures using AGETOR® Portal Server.

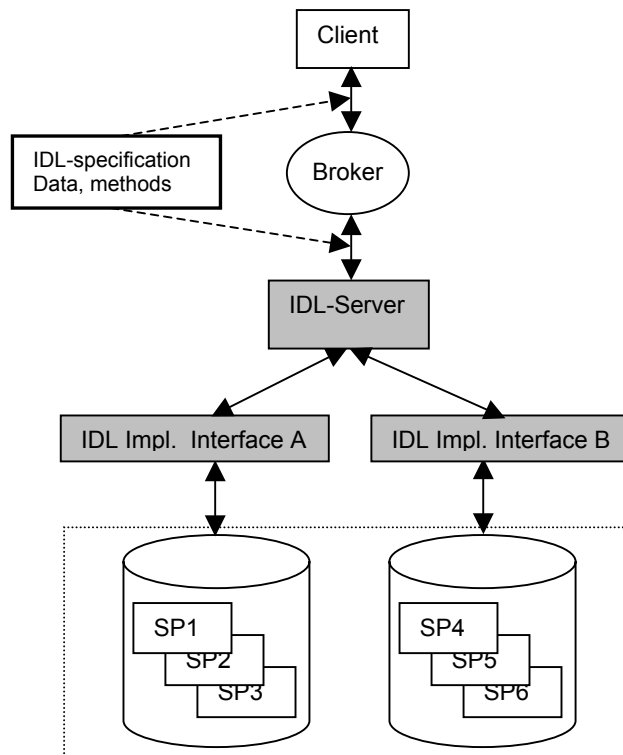


Figure 2: Accessing SP with AGETOR(r) Portal Server

In the figure you'll notice that the IDL-specification is the controlling part defining the mapping between the client and the database containing the Database Stored Procedures (the two databases in the figure is for grouping purposes only. The Database Stored Procedures need to be in one database only – or at least all procedures should be accessed using the same database connection). The gray boxes in the figure are auto-generated by the tool `idl2java`. Notice, that you can extend and overwrite methods in the server if needed. See part III of this guide for an explanation.

1.3 What AGETOR® provides

1.3.1 Modification to IDL2JAVA

To enable access to Database Stored Procedures of IDL datatypes, AGETOR® provides an extended *idl2java* tool, which will generate the necessary code to enable IDL datatypes to be sent to Database Stored Procedures. You will write an interface, which contains the stored procedures you wish to access. You will then be able to invoke a Database Stored Procedure from your client like you would do with any other backend call in AGETOR® Portal Server.

1.3.2 Preparation of IDL-objects

To get an IDL-datatype (a structure if you will) the corresponding Java Object must implement a specific Interface (`java.sql.SQLData`). This interface is needed by the database-driver to instruct the driver how to map attributes in the Object.

1.3.3 Generation of server

AGETOR® Portal Server will generate an implementation of every IDL-specified Interface and default server that uses this implementation. This allows you to invoke Database Stored Procedures instantly without writing a single line of code (apart from your client that is).

Basically, all you need to do is to configure the generated server within AGETOR® Portal Server and start using it like any other server in the framework. There are other configurations necessary as explained in more detail in part II of this guide.

1.3.4 Possibility to modify auto-generated code

Why this separation of server and implementation? If the auto-generated server does not meet your specific need, you can easily overwrite the generated server to do exactly what you want. This situation could arise from a number of different reasons. For instance, it could happen that not all input data is provided from the client directly in the method invocation. In that case, you will need to provide this data before the actual invocation of the Database Stored Procedure. You obtain this by simply extending the auto-generated server and overwriting the methods necessary. This topic is covered in detail in part III of the guide.

2 Configuration

This part of the guide explains the configurations necessary to make access to Database Stored Procedures.

Below, we refer to the generic `inside.properties`, `services.cfg` etc. files. You can choose any file as long as these are set up in your `conf.cfg`. For information on configuration scopes, see the AGETOR® Runtime Guide.

2.1 Configuring idl2java

In order to get `idl2java` to generate the necessary code you will need to set a property. You set this in the `inside.properties` file as show below:

```
inside.tools.idl2java.structfactory.sql.sp=true
```

2.2 Configuring database properties

You will also need to specify which JDBC driver to use a long with a valid combination of username and password for the connection that will be established.

This is done in `inside.properties` as follows:

```
jdbc.sp.dbdriver=oracle.jdbc.OracleDriver
jdbc.sp.dburl=jdbc:oracle:thin:@nixon:1521:ase
jdbc.sp.dbuser=pjo
jdbc.sp.dbpassword=pjo
```

In the above example the Oracle thin-driver is used.

2.3 Configuring generated services

In order to use the auto-generated server, it must be set up in your `services.cfg` file as shown below:

```
<SERVICES LOGBASE="${INSIDE_HOME}/logs">
  <PROGRAM NAME="dk.bording.idl.mapidl2sp.SPServer" TYPE="Java-method"
    DESCRIPTION="SP Server" LOGFILE="spserver.log">
    <PARAM NAME="-p" VALUE="30012"/>
  </PROGRAM>
</SERVICES>
```

You will also need to configure your `broker.cfg` with Question Number and port number as for any other server. See the AGETOR® Runtime guide for more on this subject.

2.4 Configuring access to Stored Procedures

In order to let AGETOR® Portal Server and the tools know what how to map your existing Database Stored Procedures you need to configure a new configuration file called `sql.cfg`. It must be located in your `$INSIDE_HOME/conf` directory.

An example `sql.cfg` file is shown below and it is explained in detail in the following two subsections:

```
<sqlmap>
  <type sql='PJO.NESTEDTYPE' sequence='PJO.NESTEDTYPES'
      class='dk.bording.idl.mapidl2sp.NestedType' />
  <type sql='LILLE_PERIODE_OBJ' sequence='PJO.PERIODES'
      class='dk.bording.idl.mapidl2sp.Periode' />
  <type sql='PJO.SEQSTRUCT' sequence='PJO.SEQSTRUCTS'
      class='dk.bording.idl.mapidl2sp.SeqStruct' />
  <type sql='PJO.TYPESPSTRUCT' sequence='PJO.TYPESPSTRUCTS'
      class='dk.bording.idl.mapidl2sp.TypeStruct' />

  <method interface='dk.bording.idl.mapidl2sp.MapIdl2SP'
      name='callSPNestedType' procedure='spnestedtype' />
  <method interface='dk.bording.idl.mapidl2sp.MapIdl2SP'
      name='callSPPeriode'
      procedure='demo_sp.hent_periode_objekter' />
  <method interface='dk.bording.idl.mapidl2sp.MapIdl2SP'
      name='callSPSimple' procedure='spsimple' />
  <method interface='dk.bording.idl.mapidl2sp.MapIdl2SP'
      name='callSPStruct' procedure='spstruct' />
  <method interface='dk.bording.idl.mapidl2sp.MapIdl2SP'
      name='callSPSequence' procedure='spsequence' />
</sqlmap>
```

2.4.1 Tag <type>

For every struct defined in your IDL-specification, you should make an entry in the sql.cfg file. The parameter sql denotes the corresponding object type in the database, whereas class-identifier is the IDL-generated class. If there is a sql sequence type based on this struct (a typedef), you will specify this in the sequence field.

2.4.2 Tag <method>

For every method in an interface, you add a method entry to the sql.cfg file. The interface parameter specifies the java interface the method belongs to and the name is the name of the method. Finally, the procedure parameter denotes the corresponding Database Stored Procedure.

2.5 Configuring arrays

If you need to send or receive arrays or sequence, you need to tell the driver how the mapping between IDL-sequences and the database driver implementation of JDBC-arrays is handled. The following property instructs AGETOR® Portal Server which class handles this mapping:

```
agetor.storedprocedure.array.factory=dk.bording.inside.storage.jdbc.sp.oracle.SQLArray
```

See part III of this guide for a detailed explanation of the concepts behind this property.

3 Technical issues

This part of the guide contains information on the technical issues regarding Database Stored Procedures with AGETOR® Portal Server. If you have a question that you can not find an answer to here, please contact agetorsupport@bording.dk.

3.1 Writing the IDL

You will write your IDL-specification the same way as you would any other IDL-specification. However, a few guidelines is provided here.

3.1.1 Order of attributes

When you specify your IDL-structs to match your existing database objects (e.g. Oracle Objects), the order of the attributes should be the same as the order of the attributes in your existing objects. This is necessary, since the database driver will extract (or submit) the values in that order. Therefore, you need an IDL-object with the same signature.

3.2 Overwriting the server

The tools `idl2java` will based on your IDL-specification generate one *implementation* for each Interface in the IDL. Also, the tool generates a default server (called `SPServer.java`), which utilizes each interface implementation. You can go ahead and use this server like any other server or you can extend the generated server and overwrite any method that you choose, thus obtaining the functionality needed.

Why is this important? The generated server does nothing but call upon the corresponding method in the interface. If you wish to do other things (such as updating other databases or adding data not provided by the client to the input), you can!

3.3 Handling of nulls

The AGETOR® broker does not support relaying of nulls. If your data in your database contains null values, you might experience error when fetching such data. Also, you will not be able to stored null-values in the database using the auto-generated code.

There is one exception though: If you have null-values in strings (varchar), the framework will translate these to an empty string (""). Please note, that if your client saves data back to the database to that field, the null value will be lost and substituted with the value from the client. This value might be the empty string ("") instead of null-value even if your client did not modify the value per se.

3.4 Description of used interfaces

3.4.1 Interface: `java.sql.SQLData`

The tool `idl2java` uses an interface in order to prepare your IDL-structures (or rather IDL Objects) to enable your objects to be sent to and from the database using JDBC. This interface contains methods invoked by the driver upon reading and writing the object to the data-stream.

3.4.2 Interface: `dk.bording.inside.storage.jdbc.sp.SQLPrepared`

If your data structures contains attributes of complex type (i.e. nested types), the driver need to have prepared appropriate memory space for these structures. This is made sure by implementing the only method in this interface: `prepareSQL`.

3.5 Database specific topics

Currently, the only supported databases are Oracle Databases. This section discusses topics related to this database. If you wish use another database, please contact agetorsupport@bording.dk.

3.5.1 Oracle Objects and Records

If you are using a Oracle Thin Database driver, please note, that this driver does not support fetching/updating of Oracle Record types.

In order to manipulate Oracle Records, you should write a translator-method in your existing Oracle application that will take a record as input and return an Oracle Object with the same attributes. Also, define a database wrapper procedure with the same signature as you existing procedure, but using Database Objects instead of Database Records. You will the call the wrapper method from AGETOR® Portal Server.

3.5.2 Handling of Arrays and sequences

In case you need to fetch or store arrays of data you will need a java class that implements `java.sql.Array` to hold this array. In some cases, the driver might have a suitable class (in fact the oracle Thin Driver does); otherwise you will have to write one yourself.

In this version of accessing Database Stored Procedures, the provided Oracle class is used.

Which class is used is handled by the property `agetor.storedprocedure.array.factory`. Please, refer to section 2.5 for more information.