Cosmos WebServer

Cosmos WebServer (CWS) is an utility that allows you to use Cosmos as a web services provider, allowing you to create REST services.

This utility is available as of version 6.0 of Cosmos, except for the Log file: Cwslog.log, which has been incorporated since version 7.2.

Starting with Cosmos version 7.4, Cosmos WebServer also supports HTTPS connections.

Starting with Cosmos version 7.4.2, you can indicate the version of the Java virtual machine with environment variables, without the need to modify the PATH system variable.

As of version 7.8.3 of Cosmos, you can define the character encoding in which the defined services work.

As of version 8.0 of Cosmos, you can modify the number of «acceptors» and «selectors» of Jetty with an environment variable, install/start/stop/uninstall the service silently and in the log4j file, with the trace in DEBUG mode, the parameters sent in the request can be consulted.

Cosmos WebServer does not work with Cosmos SQL Desktop or Cosmos SQL Workgroup licenses.

BASE100

BASE 100, S.A.
www.base100.com

Index

1. Introduction

Cosmos WebServer (CWS) is an utility that allows you to use Cosmos as a web services provider, allowing you to create REST services

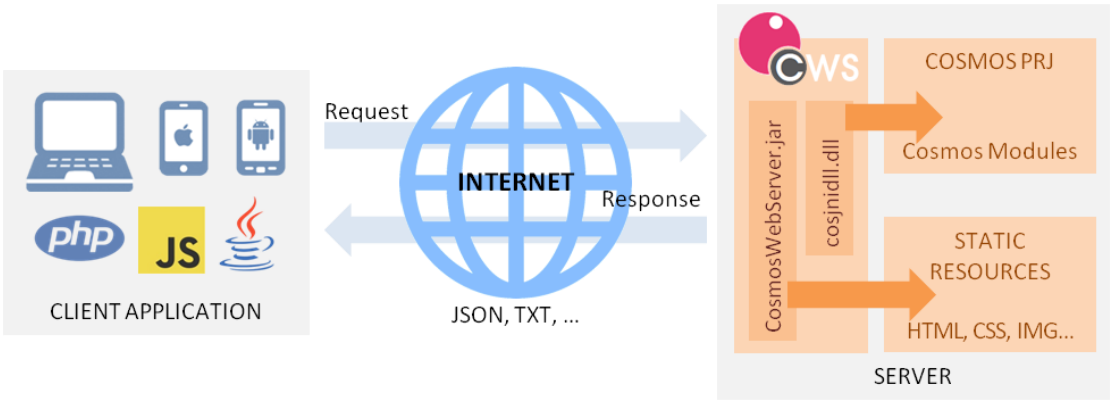
Cosmos WebServer can be started in command line or as a Windows service.

Once started, the server handles the HTTP requests on the defined port (default 8081). A request can become invocation to a method of a configured Cosmos application.

This operation is detailed below.

2. Architecture

The basic scheme of Cosmos WebServer (CWS) is shown in the following figure:



A Cosmos WebServer installation will consist of:

- Web server (coswebserver.exe + cosmoswebserver.jar).
- Cosmos runtime embedded in a DLL (cosjnidll.dll).
- The Cosmos application where you will define the modules that implement the REST services.

Cosmos WebServer uses the "JVM.dll" dll of the Java virtual machine. Therefore, to run CWS it is necessary to have a 32-bit Java virtual machine (version 1.8 or higher) installed on the same machine as Cosmos WebServer.

Starting with Cosmos version 7.4, Cosmos WebServer also supports HTTPS connections.

The Cosmoswebserver.exe application allows you to start the CWS server by command line or as Windows service.

3. Configuration

3.1 CosmosWebServer.ini configuration file

This file includes the following functionalities:

a) One entry for each Windows service to be created.
b) Tells the Cosmos WebServer service which Java virtual machine to use (available as of Cosmos version 7.4.2).

These features are not mutually exclusive.

To install Cosmos WebServer as a Windows service it is necessary to create a file with the name «cosmosweb-server.ini» that must be located in “COSMOSDIR/etc”.

For each service that is created, a section must be defined with the name of the service. Each section will contain a variable, called INIFILE, with the absolute path of the configuration file for the REST server.

Example:

```
[ServicioWeb]
INIFILE=c:\cosmos\etc\stockserver.ini
[ServicioWebCompras]
INIFILE=c:\cosmos\etc\comprasserver.ini
```

Section [Enviroment]

Starting with Cosmos version 7.4.2, it is possible to tell Cosmos WebServer which version of the Java virtual machine to use. To do this, it will be necessary to define the environment variables CWSUSEJAVAVERSION or CWSUSELASTJAVAVERSION in the [Environment] section:

- **CWSUSEJAVAVERSION.** It indicates to Cosmos WebServer the version of Java that it will use in its execution and in that of Java methods from Cosmos WebServer itself.
- **CWSUSELASTJAVAVERSION.** Tells Cosmos WebServer to use the latest version of Java installed on the PC.

If Cosmos WebServer is not started as a service, but you want to specify the Java Virtual Machine to use, you will also need to set these variables in *cosmoswebserver.ini*.

3.2 Cosmos WebServer application configuration file

The configuration file will indicate, among other things, the port where the server will listen, the configuration file path of the Cosmos REST services, the parameters of the Java virtual machine (cosmosrestserver.jar file path, memory parameters) and, optionally, the server's HTML resource path.

Configuration file example:

```
[Server]
CONFIG=c:\cosmosRestApp\etc\stockConfiguration.xml
PORT=8080
RESOURCEPATH=c:/cosmosRestApp/resources/webapp
[JAVA]
-Xms256m
```

```
-Xmx1024m
-Djava.class.path=c:\\cosmos\\bin\\cosmoswebserver.jar;
```

3.2.1 Environment variables

CONFIG

This variable indicates the path of the configuration file of the REST services of the server (the configuration of this file is explained in the section "REST's server configuration file").

PORT

Indicates the server port where CWS will listen for HTTP requests.

RESOURCEPATH

This environment variable indicates the directory path where CWS will store resources that will be accessible via the server URL. These resources may be HTML pages, image files, etc.

NUMBEROFTHREADS

It will be necessary to define this variable so that the execution of Cosmos WebServer is multi-threaded, that is, that allows processing more than one REST request at the same time without having to wait for one to finish to process the next one.

It is defined in the [Server] section of the INI file of the service.

The value of this variable will be the number of threads that the server should be able to handle.

To make Cosmos Webserver multi-threaded:

- The definition of the variable is mandatory.
- Its value must be greater than 1.
- Cosmos Webserver must be installed as a service and not as a console application.

When Cosmos Webserver is defined as multi-threaded, the connection to the database can only be done in client-server architecture.

> *Local connections to the database cannot be established. In local connections the following message will be returned:*
> *"Imposible conectar con el servidor de SQL. (Unable to connect to local database from Cosmos WebServer in MULTI-THREAD MODE. The connection must be client-server)".*

Dcom.base100.cosmos.server.threadacceptorselector

This variable modifies the number of «acceptors» and «selectors» for Jetty. The maximum number of the sum of the 2 elements cannot exceed 180. This variable must be defined in the [JAVA] section.

Parameters:

Acceptors.	Threads of the Jetty process that will be in charge of accepting HTTP requests on the port specified in the configuration.
Selectors.	Threads of the Jetty process that will be in charge of managing communication in HTTP connections with the client.

For every HTTP connection, there will be a thread corresponding to a selector.

This variable has been implemented in version 8.0 of Cosmos. In versions prior to 8.0, if many threads were running simultaneously, Cosmos WebServer would crash.

Example:

```
-Dcom.base100.cosmos.server.threadacceptorselector=10,100
acceptors = 10
selectors = 100
```

To configure an HTTPS connection it will be necessary to define the following variables:

SSL_PORT

- Indicates the server port where CWS will listen for HTTPS requests.
- This variable is required if you want to configure an HTTPS connection.
- If the PORT and SSL_PORT variables are not defined, the connection will be HTTP (not secure) over port 8081.
- If PORT is specified and SSL_PORT is not specified, the connection will be HTTP (not secure) over the specified port.
- If PORT is not defined and SSL_PORT is defined, the connection will use HTTPS (secure connection), and it will not be allowed to receive HTTP connections (non-secure).
- If PORT and SSL_PORT are defined, connections will use HTTPS (secure connection) and HTTP (non-secure connection) over the indicated ports. The indicated port numbers must be different.

SSL_KEYSTORE

- This variable defines the absolute path of the certificate store file.
- The character "/" must be used to separate directory names.
- This variable is mandatory to configure an HTTPS connection.

SSL_KEYSTORE_PWD

- This variable indicates the certificate store password.
- This variable is mandatory to configure an HTTPS connection.

SSL_KEYSTORE_ALIAS

- This variable indicates the alias of the certificate that you want to use. If this variable is not defined, it will use the first certificate of the store.
- This variable is optional.
- Although the value indicated in this variable is not valid, Cosmos WebServer will start correctly.

SSL_KEYSTORE_ALIAS_PWD

This variable indicates the alias password defined in SSL_KEYSTORE_ALIAS. If not specified, it will use the password entered in SSL_KEYSTORE_PWD.

This variable is optional.

If the Cosmos WebServer boot process fails, the service will not start and connections of any kind cannot be made. For example, if the certificate store password is incorrect, or if the indicated HTTP or HTTPS listening port is already in use or if the same port is indicated for HTTP and HTTPS connections, the boot process will fail and new connections will not be available.

3.3 REST's server configuration file

This configuration file specifies, in XML format, the name of the Cosmos application, the name of the modules and the methods that will be in charge of implementing the REST services, as well as the path (URL) and HTTP verb (GET , POST, PUT, DELETE, etc.) that must be used to access each of these methods. Each REST service in Cosmos will correspond to a method defined in the <method> sections, which will be accessed through its path and that of the module (<module>) and the project (<Project>) to which they belong (path property of <Project>, < Module> and <method>). Each corresponding Cosmos method with a REST service must return a string with the result of executing the REST service.

3.3.1 Configuration file example

This is an example of a REST server configuration file. It defines the project, the modules and the Cosmos methods that will implement the REST services, as well as the way to access them:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<project name="stock" path="stock/rest" dirPath="c:\cosmosRestApp\cosmosRestApp.PRJ">
        <module name="security" path="security">
            <method name="authenticate" path="authenticate" http="GET">
                <queryParameter name="username" defaultValue = ""/>
                <queryParameter name="password" defaultValue=""/>
                <queryParameter name="domain"/>
            </method>
        </module>
        <module name="states" path="states">
            <method name="resultList" path="resultList" http="POST">
                <queryParameter name="firstResult" defaultValue="0"/>
                <queryParameter name="maxResults" defaultValue="10"/>
                <queryParameter name="orderBy"/>
                <bodyParameter name="filter"/>
            </method>
            <method name="resultCount" path="resultCount" http="POST">
                <bodyParameter name="filter"/>
            </method>
            <method name="findByKey" path="{key}" http="GET">
                <urlParameter name="key"/>
            </method>
            <method name="deleteByKey" path="{key}" http="DELETE">
                <urlParameter name="key"/>
            </method>
            <method name="update" path="{key}" http="PUT">
                <urlParameter name="key"/>
                <bodyParameter name="entity"/>
            </method>
            <method name="create" http="POST">
                <bodyParameter name="entity"/>
            </method>
        </module>
        <module name="items" path="items">
            <method name="resultList" path="resultList" http="POST">
                <queryParameter name="firstResult" defaultValue="0"/>
                <queryParameter name="maxResults" defaultValue="10"/>
                <queryParameter name="orderBy"/>
                <bodyParameter name="filter"/>
            </method>
            <method name="resultCount" path="resultCount" http="POST">
                <bodyParameter name="filter"/>
            </method>
            <method name="findByKey" path="{item}/{supplier}" http="GET">
                <urlParameter name="item"/>
                <urlParameter name="supplier"/>
            </method>
            <method name="deleteByKey" path="{item}/{supplier}" http="DELETE">
                <urlParameter name="item"/>
                <urlParameter name="supplier"/>
            </method>
```

```
            <method name="update" path="{item}/{supplier}" http="PUT">
                <urlParameter name="item"/>
                <urlParameter name="supplier"/>
                <bodyParameter name="entity"/>
            </method>
            <method name="create" http="POST">
                <bodyParameter name="entity"/>
            </method>
        </module>
</project>

</configuration>
```

3.3.2 Configuration file sections

The <project> section

In this section the following properties are defined:

name	Project identifier on the web server.
dirPath	Path of the Cosmos project file into the server's file system.
path	Path from which the project methods are accessed in the server URL.

The <module> section

For each module of the Cosmos project that defines a REST service, a <module> section must be defined. The <module> sections should be defined as child sections of the <project> section, one for each module. In this section the following properties are defined:

name	Name of the Cosmos module.
path	Path in the server URL.

The <method> section

For each of the REST server methods defined in a module, a section called <method> is defined as the child section of the corresponding <module> section. A <method> section must be defined for each module method that implements a web service.

name	Name of the Cosmos method.
path	Path in the server URL.
http	The HTTP verb that triggers the method.
parametersCharset	Character encoding of the parameter received by the Cosmos function that implements the request. If not defined, it defaults to ANSI with the local code page. Possible values are: UTF-8 or a number that will indicate the ANSI code page.
returnCharset	Character encoding of the return value of the Cosmos function that implements the request. If not defined, the default value is ANSI with local code page. Possible values are: UTF-8 or a number that will indicate the ANSI code page.

If the method receives parameters, the name and, optionally, the default value of each parameter and the way in which the parameter arrives at the method will be indicated. A parameter can arrive through the URL or through the BODY section of the HTTP request.

Depending on the type of parameter that the method receives, the following sections must be defined within the <method> section:

<queryParameter> section

If a parameter is defined in the URL with this format:

http:\\<server>:<port>\prjpath\modulepath\methodname?param1=value1

It corresponds to a query parameter, and will be defined within this section.

Within this section the following properties are defined:

name	The name of the parameter.
defaultValue	Default value of the parameter. This is an optional property.

<urlParameter> section

If a parameter is defined in the URL, without method name, with this format:

"http:\\<server>:<port>\prjpath\modulepath\value1\value2"

It corresponds to a url parameter, and will be defined within this section.

Within this section the following properties are defined:

name	The name of the parameter.
defaultValue	Default value of the parameter. This is an optional property.

<bodyParameter> section

If a parameter is defined in the body section of the HTTP request, it corresponds to a request body parameter, and must be defined in this section.

Within this section the following properties are defined:

name	The name of the parameter.
defaultValue	Default value of the parameter. This is an optional property.

<headerParameter> section

This parameter allows to query all the information in the HTTP header.

Within this section the following properties are defined:

name	The name of the parameter.

The name of the parameter in the configuration file, either of type <queryParameter>, <urlParameter>, <bodyParameter> or <headerParameter> , must match the name of the parameter specified in the Cosmos source code of the method.

3.3.3 Example of a Cosmos REST service request

According to the configuration XML file defined above, to execute the service resultCount of the states.smd module, a POST type operation must be executed from the client side at the URL: http://<server>:<port>/stock/rest/states/resultCount

host	Name or IP address of the server where Cosmoswebserver.exe is running.
port	Port number specified in the PORT environment variable of the configuration file passed as a parameter on the Cosmoswebserver.exe command line.
stock/rest	Path to the Cosmos project defined in the "path" property of the <project> section in the configuration XML file.
states	Path to the "states" module, defined in the "path" property of the <module> section.
resultCount	Path to the "resultCount" method, defined in the "path" property of the <method> section.

That is, the "name" property of <project> stores the project identifier, and the "name" property of <module> and <method> stores the name of the Cosmos module and the name of the Cosmos method respectively. The "path" property stores the term to be used in the URL to access the project, module and method.

3.4 Creating a REST service in Cosmos

A REST service in Cosmos is defined as a function or method defined in a module of a project in Cosmos, which receives some parameters from the URL or from the body of an HTTP request, and returns a string with the result of its execution, and a status code indicating whether the execution has been successful, failed, etc.

This code must be a standard HTTP status code (200 - ok, 201 - created, 405 - method not allowed, etc.).

This status code will be set by the Cosmos method or function corresponding to the REST service by executing the SetExecStatus method of the Module class at the end of the function body.

Example of the REST service findByKey implemented in the findByKey function of the States.smd module. It connects to the database and indicates that it does not display errors on the screen. The processQuerySingleResult function performs the search, collects the records in a JSON object, returns it as a string, indicates the state of execution with the SetExecStatus method, and disconnects from the database.

```
public function findByKey(key as char) return char
objects begin
    retStr stmtStr as char default NULL
end
begin
    Conecta();
    ErrorLevel(0);
    stmtStr = "select * from states ";
    if key is not null then begin
        stmtStr = stmtStr + " where state = '" + key + "'";
```

```
        retStr = processQuerySingleResult(stmtStr);
    end else begin
            Module.SetExecStatus(BAD_REQUEST);
    end
    Desconecta();
    return retStr;
end

private function processQuerySingleResult(stmtStr as char) return char
objects begin
    retStr as char default ""
    miCursor as SqlCursor
    jsonArray as json
    jsonRecord as json
    stateStr nameStateStr as char
    retStatus as integer
    found as boolean default FALSE
end
begin

    miCursor.Prepare(stmtStr);
    if Sql.Error() == 0 then begin
        miCursor.Open();
        if miCursor.Fetch(stateStr, nameStateStr).Found() then begin
            jsonRecord.Clear();
            jsonRecord.Set("state", stateStr);
            jsonRecord.Set("sname", nameStateStr);
            found = TRUE;
        end
        miCursor.Close();
        if not found then retStatus = getNoRecorsStatusCode();
    end else begin
        retStatus = BAD_REQUEST;
    end
    miCursor.Free();

    Module.SetExecStatus(retStatus);
    retStr = jsonRecord;

    return retStr;
end
```

4. Character encoding

Character encoding in versions of Cosmos earlier than 7.8

In versions of Cosmos prior to 7.8, Cosmos converted the character string sent in the request parameter to ANSI with the local code page. This caused that, if there was no correspondence between the characters of the request and those of the local ANSI encoding of the computer where Cosmos WebServer is running, the conversion and graphic representation of these would not be correct.

Cómo funcionará a partir de la versión 7.8

By default it will work as in previous versions.

If it is necessary to work with a character set other than the ANSI of the local machine, it will be necessary to define the parametersCharset and returnCharset properties in the Cosmos Web service configuration xml file. By defining this property, Cosmos will convert to the charset indicated in parametersCharset, and the character strings sent in the response will use the character set indicated in the returnCharset property, being ANSI by default with the local code page.

parametersCharset and returnCharset properties.

Possible values are: UTF-8 or a number indicating the ANSI code page.

They are defined in the service configuration xml file at the method level.

parametersCharset. Character encoding of the parameter received by the Cosmos function that implements the request. If not defined, defaults to ANSI with ANSI local code page.

returnCharset. Character encoding of the return value of the Cosmos function that implements the request. If not defined, defaults to ANSI with ANSI local code page.

Example.

If a JSON with UTF-8 encoding with Cyrillic text is sent in the body of the request and the Cosmos service wants to work in ANSI, the value that must be indicated in the parametersCharset property must be "1251", which is the corresponding to ANSI Cyrillic encoding. If you want to work in UTF-8, the value to be indicated in the parametersCharset property must be "UTF-8".

If you want to send the response in UTF-8, the value of the returnCharset property must be "UTF-8". The Cosmos function that implements the request must be responsible for converting to the character set indicated in returnCharset.

5. Install and execution of Cosmos WebServer

The Cosmos WebServer utility can be started from the command line or as a Windows service.

Before starting the installation of the application, the Java virtual machine must be installed and make sure that it includes the DLL JVM.dll.

In versions of Cosmos prior to 7.4.2, for Cosmos WebServer to locate the DLL, the path where the DLL is located must be added to the system's PATH variable.

As of version 7.4.2 of Cosmos it is not necessary to indicate the path of the JVM.dll in the PATH environment variable of the operating system. For this, the environment variables CWSUSEJAVAVERSION and CWSUSELASTJAVAVERSION have been implemented. These two environment variables must be defined in the **[Environment]** section of the *cosmoswebserver.ini* configuration file.

This implies that, as of version 7.4.2 of Cosmos, the cosmoswebserver.ini file must necessarily exist in case of using the mentioned environment variables, whether the application is started from the command line or from a Windows service.

The *cosmoswebserver.ini* file should be located in “COSMOSDIR\etc”.

5.1 Execution from command line

To start it from the command line we will have to use the following syntax:

```
Cosmoswebserver.exe –ini c:\cosmos\etc\stockserver.ini
```

The problem of starting Cosmos WebServer from the command line is that, when the server where it is installed is restarted, it is necessary to manually restart Cosmos WebServer.

This problem is solved by installing Cosmos WebServer as a Windows service.

5.2 Execution as service

5.2.1 Service installation

To install Cosmos WebServer as a service it will be necessary to run Cosmos WebServer with the parameter "-install <service_name>":

```
Cosmoswebserver.exe –install ServiceWeb –user .\user –passwd passwd
```

The parameter "service_name" will indicate a non-existent Windows service name, which will be the one that will identify the Cosmos WebServer service.

The parameters "-user" and "-passwd" indicate respectively the user who starts the service and its password. These parameters are optional. If not indicated, the service will boot with a local system account.

The service will be installed by default as "AUTOMATIC", so when the server is restarted the service will start automatically.

The service can be installed silently using the «-silent» parameter.

5.2.2 Service start

To start the Cosmos WebServer service installed with "-install" you must run CosmoswebServer with the parameter "-start <service_name>":

```
Cosmoswebserver.exe –start ServicieWeb
```

As we saw in section 3.1, CosmosWebServer needs the name of a config file. This name will be obtained by CosmosWebServer from the cosmoswebserver.ini file, which should be located in "COSMOSDIR\etc". For each service that we want to install, we must define a section with the name of the service and a variable, called INIFILE, with the absolute path of the configuration file for that service.

Example of cosmoswebserver.ini with two CosmosWebServer services configured:

```
[ServiceWeb]
INIFILE=c:\cosmos\etc\stockserver.ini
[ServiceWebPurchases]
INIFILE=c:\cosmos\etc\purchasesserver.ini
```

From then on, the service will be available to accept new connections.

5.2.3 Service stop

To stop the Cosmos WebServer service installed with "-install" you must run CosmosWebServer with the parameter "-stop <service_name>":

```
Cosmoswebserver.exe –stop ServiceWeb
```

From that moment the service will be installed, but not available to accept new connections, until it is not rebooted with "-start".

5.2.4 Service uninstall

To uninstall the CosmosWebServer service installed with "-install", you must run CosmosWebServer with the parameter "-remove <service_name>":

```
Cosmoswebserver.exe –remove ServiceWeb
```

6. LOG files

6.1 Log4j file

The Cosmos WebServer server allows the generation of a log file in which information about the operations of Cosmos WebServer will be stored.

Cosmos WebServer uses the Log4j2 tool to generate the log file.

By default, CosmosWeb Server does not generate any log files. If you want the application to store log information, it should be indicated in the configuration file of Cosmos WebServer. (see the section 3.1).

The name of the log configuration file will be indicated in the [JAVA] section by defining the following variable:

```
-Dlog4j.configurationFile=c:/cosmos/cosmoswebserver/log4j2.xml
```

In this case, the log configuration file is called log4j2.xml. This is a configuration file in XML format where you can indicate the path and the name of the log file, the output format and the level of log you want (ALL, DEBUG, ERROR, FATAL, INFO, OFF, TRACE and WARN).

As of version 8.0 of Cosmos, if the «DEBUG» level is indicated in the log4j file, the parameters sent in the request can be consulted.

Example of log4j2.xml configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
LogLevel=ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF
See: http://logging.apache.org/log4j/2.x/manual/index.html
 -->
<Configuration status="INFO">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
    <RollingFile name="RollingFile" fileName="c:/cosmos/cosmosdata/logs/app.log"
                 filePattern="logs/$${date:yyyy-MM}/app-%d{MM-dd-yyyy}-%i.log.gz"       ap-
pend="true">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
      <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="250 MB"/>
      </Policies>
    </RollingFile>
  </Appenders>
  <Loggers>
    <Root level="All">
      <AppenderRef ref="Console"/>
      <AppenderRef ref="RollingFile" level="INFO"/>
    </Root>
  </Loggers>
</Configuration>    retStr stmtStr as char default NULL
```

This configuration file indicates the full path of the log file (fileName property of the RollingFile section), and the log level (property level of section AppenderRef).

For further information, see: http://logging.apache.org/log4j/2.x/manual/index.html

6.2 CosmosWebServer.log file

This file is created in the c:\tmp directory, and in it Cosmos WebServer will write the information generated in the installation, start-up, stop and removal of the Windows service or in the startup process of Cosmos Web-Server as an application.

Cosmos WebServer as service

```
2018-07-19 16:20:49  Checking JVM:JVM found
2018-07-19 16:20:49  Installing the service:cosmoswebserver
2018-07-19 16:20:49  cosmoswebserver instalado
2018-07-19 16:20:50  La configuración del servicio cosmoswebserver ha sido cambiada.
2018-07-19 16:21:07  Checking JVM:JVM found
2018-07-19 16:21:07  Starting the service:cosmoswebserver
2018-07-19 16:21:07  La configuración del servicio cosmoswebserver ha sido cambiada.
2018-07-19 16:21:08  Checking JVM:JVM found
2018-07-19 16:21:08  Running as service
2018-07-19 16:21:08  cosmoswebserver puesto en marcha.
2018-07-19 16:21:08  Starting Cosmos Web Server
2018-07-19 16:21:08  Ini file loaded: c:\cosmos\etc\cwsdemo.ini
2018-07-19 16:21:48  Checking JVM:JVM found
2018-07-19 16:21:48  Stopping the service:cosmoswebserver
2018-07-19 16:21:48  cosmoswebserver parado.
2018-07-19 16:21:52  Checking JVM:JVM found
2018-07-19 16:21:52  Removing the service:cosmoswebserver
2018-07-19 16:21:52  cosmoswebserver eliminado.
```

Cosmos WebServer as application

```
2018-07-19 16:22:16  Checking JVM:JVM found
2018-07-19 16:22:16  Starting Cosmos Web Server
2018-07-19 16:22:16  Ini file loaded: c:\cosmos\etc\cwsdemo.ini
```

6.3 Cwslog.log File

The output of the errors generated by the Cosmos WebServer runtime during the processes can be to a file if a file with the name **CWSLog.log** is created in the temporary directory.

Cosmos WebServer as service

If an error occurs during the execution of a Cosmos service and the ErrorLevel is greater than 0, information about the error will be stored in the log file, as well as the method and module where the error occurred, and the call stack from the application.

If an error occurs during execution and the ErrorLevel equals 0, the error information will not be written to the log file.

If the Trace method is executed in the source code of the Cosmos service, the text associated with the Trace method will be written to the log file.

Cosmos WebServer as application

If an error occurs during the execution of a Cosmos service and the ErrorLevel is greater than 0, information about the error will be stored in the log file, as well as the method and module where the error occurred, and the call stack of the application. This same information will be displayed on the screen through a MessageBox.

6.3.1 Example

The following modifications have been made in the example project:

- In the findByKey function of the states module, the select statement is changed to cause an error.
- To check the value of some variables, the Trace method is called twice.

Function findByKey:

```
public function findByKey(key as char) return char
objects begin
    retStr stmtStr as char default NULL
end
begin
    Conecta();
    key.Trace();
 //   ErrorLevel(0);
    stmtStr = "select * from1 states";
    stmtStr.Trace();
    if key is not null then begin
        stmtStr = stmtStr + " where state = '" + key + "'";
        retStr = processQuerySingleResult(stmtStr);
    end
    Desconecta();
    return retStr;
end
```

Cosmos WebServer as service

The following information will be displayed in the log file:

```
[2018-07-19 10:35:40] CWS Service Mode
[2018-07-19 10:35:40] Char::Trace
MA
[2018-07-19 10:35:40] CWS Service Mode
[2018-07-19 10:35:40] Char::Trace
select * from1 states
[2018-07-19 10:35:40] CWS Service Mode
[2018-07-19 10:35:40] Warning
Warning (M50700000)
File STATES
Method STATES::processQuerySingleResult
Falta la clausula FROM en la sentencia.
==========
CALL STACK
==========
-->SqlCursor::Prepare
STATES::processQuerySingleResult
STATES::findByKey
```

Cosmos WebServer as application

The same information will be displayed on the screen:

The following information will be displayed in the log file:

```
[2018-07-19 10:44:29] CWS Application Mode
[2018-07-19 10:44:29] Warning
Warning (M50700000)
File STATES
Method STATES::processQuerySingleResult
Falta la clausula FROM en la sentencia.
==========
CALL STACK
==========
-->SqlCursor::Prepare
STATES::processQuerySingleResult
STATES::findByKey
```

NOTE: In this case, since the SetExecStatus method was not used in the header of the response, the Web server will return: 500 Server Error.