Cosmos WebServer

Cosmos WebServer (CWS) is a 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.

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

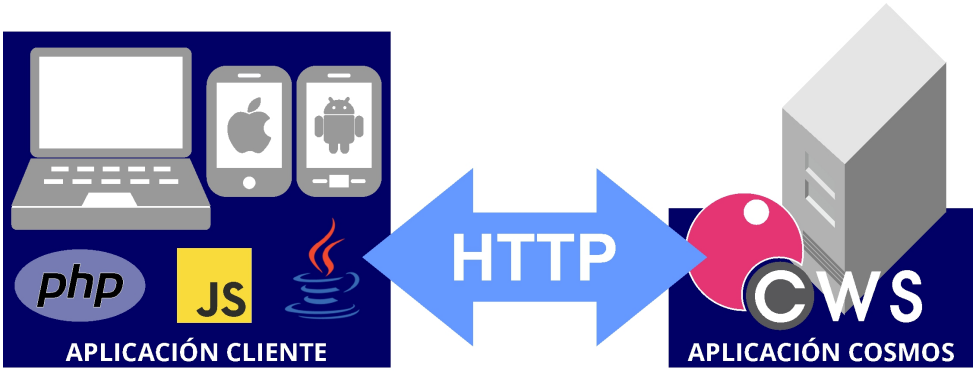
Cosmos WebServer can be started in command line or as a Windows service.

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.

The Cosmoswebserver.exe application allows you to start the CWS server by command line. This application receives as a parameter a configuration file.

Syntax:

```
Cosmoswebserver.exe –ini < configuration file path >
```

Parameter:

-ini	Configuration file path.

3. Configuration

3.1 Cosmos WebServer configuration file

The configuration file received as a parameter by the Cosmoswebserver.exe application 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.

Example:

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

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;
```

3.1.1 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.

3.2 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

3.2.1 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>
```

3.2.2 Configuration file sections

The <project> section

In this section the following properties are defined:

name property	Project identifier on the web server.

dirPath property	Path of the Cosmos project file into the server's file system.
path property	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 property	Name of the Cosmos module.
path property	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 property	Name of the Cosmos method.
path property	Path in the server URL.
http property	The HTTP verb that triggers the method.

If the method receives parameters, the name and, optionally, the default value of each parameter will be indicated, as well as the way in which the parameter arrives at the method. 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 property	The name of the parameter.
defaultValue property	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 property	The name of the parameter.
defaultValue property	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 property	The name of the parameter.
defaultValue property	Default value of the parameter. This is an optional property.

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

3.2.3 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.
puerto	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.

3.3 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. Installation as a Windows service

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

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.

To start Cosmos WebServer as a Windows service, use the following syntax:

```
Cosmoswebserver.exe [–install|start|stop|remove] [service] [–user <user> –
passwd <password>]
```

4.1 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", ie when the server is restarted the service will start automatically.

4.2 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.

4.3 Service stop

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

```
Cosmoswebserver.exe –start ServiceWeb
```

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

4.4 Service uninstall

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

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

5. LOG files

5.1 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).

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

5.2 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
```

5.3 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.

5.3.1 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.