

# Lycia Development Suite



**Lycia II Web API Guide**

**Version 2.2 – April 2014**

**Revision number 1.00**



# Querix Lycia II

## Lycia Web API Guide

Part Number: 028-022-001-014

Vitaliy Akimov / Daria Sakhno

Last Update April 2014



## **'Lycia' Release Notes**

Copyright © 2006-2014 Querix Ltd. All rights reserved.

Part Number: 028-022-100-014

### **Published by:**

Querix (UK) Limited. 50 The Avenue, Southampton, Hampshire,  
SO17 1XQ, UK

### **Last Updated:**

April 2014

### **Documentation written by:**

Vitaliy Akimov / Daria Sakhno

### **Notices:**

The information contained within this document is subject to change without notice. It is a stable document and may be used as reference material or cited from another document. If you find any errata or omissions in the documentation, please report errors at [documentation@querix.com](mailto:documentation@querix.com).

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose without the express permission of Querix (UK) Ltd.

Other products or company names used within this document are for identification purposes only, and may be trademarks of their respective owners.



---

## Table of Contents

---

Documentation and Demonstrations .....	6
Document Conventions .....	6
What is WEB API? .....	7
User Guide .....	9
Reference .....	18
Server interface for commands .....	18
Accessing 4GL application with the Web API .....	25
4GL functions To Call Web Services .....	26
Reference designation .....	27
Table of the commands used for Web API objects .....	28
APPENDIX. XSD files for server response .....	35



## Introduction

Welcome to the world of Web Services with the usage of the Querix 'Lycia Studio' Development Suite, the essential toolkit for 4GL and ESQ/C.

This "Web Service: Getting Started" guide will give you an up-close look at the process of creating, developing and deployment of Web Services using Querix 4GL. It covers the basic web interactions of Web Interface and its design patterns that can be put to immediate use.



## Documentation and Demonstrations

### Documentation

#### The Web API Guide


This guide is a non-normative document intended to provide an easily understandable information on the features and specifications of web services.

#### Demonstration Applications

The 'LyciaStudio' installation suite does not include demonstration applications. You can download them from our CVS repository. The instructions on how to download and use the demo applications can be found in the "Importing Demonstration Applications" chapter of the 'Lycia: Getting Started' guide.

#### Document Conventions

The conventions used within this reference guide are described in the table above.

Convention	Description	Usage
Monospace	Used for code fragments or command line functions names.	<pre>DATABASE cms MAIN DISPLAY "Hello World" END MAIN</pre>
	Used in a table to provide additional information, non-trivial but important details, suggestions, prerequisites or indicates any warnings or possible errors.	The keyword "web" simply tells the compiler that this function will be visible/public in the corresponding web service.
<b>Boldface</b>	Used to mark out object categories	<b>web.Response object</b>
<i>Italic type</i>	Used to define datatype, alias and enumeration	<i>Atomic</i> - can be any simple 4GL datatype



# CHAPTER 1

## What is WEB API?

With Lycia Web API any 4GL application may be seamlessly integrated into enterprise application within heterogeneous environment. There are basically two typical parts of the framework, namely server and client. Any 4GL application can act both as server and client. Server part is implemented on top of LyciaWeb application, and client part is a runtime library. Both parts are available immediately after the installation of Lycia toolkit onto any operating system.

With the new Lycia Web API framework, 4GL program interoperates with other programs written in any programming language where similar framework exists (whereas it exists for most of modern programming languages). Any classic 4GL application can be easily turned into a web service accessible under any computing language, e.g. Ruby, JavaScript, Java, etc. This point significantly preserves and leverages the skills of developers as does not require source code rewriting and reading. The program must only be recompiled and then deployed into application server. Thereby, programs developed on 4GL can be used for calling external services and external services can call some business logic written 4GL. This business logic may contain not only function calls and DB access but also any 4GL interaction statement (such as MENU, INPUT, PROMPT, etc). Publishing these interaction statements as service makes 4GL language a very powerful tool for creating large distributed applications, since it provides means for services orchestration. Such 4GL programs still have clear logical reading, where order of actions are obvious, unlike most of other similar frameworks.

By use of Lycia Web API, 4GL application can be a part of large cloud service without time-consuming custom coding, since cloud users consume applications as software interfaces to connect and use resources in different ways.

Lycia Web API is based on HTTP protocol. This makes all available for HTTP services to be available for the interface also. These are: caches, gateways and load balancers.

The server part of the framework is based on the Representation state transfer (REST) architecture style which provides simple and lightweight means for implementation of scalable distributed applications. The main components of REST are resources and URLs. A resource is something that can be stored on a computer, generated and accessed partly or represented as a stream of bits. The data passed to and from the resource are called representations. Not the resources themselves but their representations are being sent or received. These representations can be presented in various formats: XML, HTML, JPEG, GIF, etc. The client's activity on this particular state returns another representation and places the resources into another state while the server application is stateless in the REST approach. Typical 4GL application seems to contradict this constraint since it has internal state in form of variables value and current execution point. Since with Lycia Web API 4GL application plays resource role, not service, no contradictions take place; the resources are typically stateless in any service, for example databases. The service itself is a part of LyciaWeb and is stateless. However, there is an option to run 4GL application in



stateless mode, too. Only a single function can be called in such case and no interaction statements are supported. In cost of dropping state developers get caching, load balancing and other facilities. In stateful mode load balancing is still possible on session level, not on request level. LyciaWeb services are implemented on top of the Spring MVC Framework and run as a servlet on any standard servlet container. Customers can get access to the vast variety of tools available from Spring MVC and its servlet container for application extension. For example, it is possible to change messages formats and URLs wrap messages into SOAP service or use additional security facilities.





# CHAPTER 2

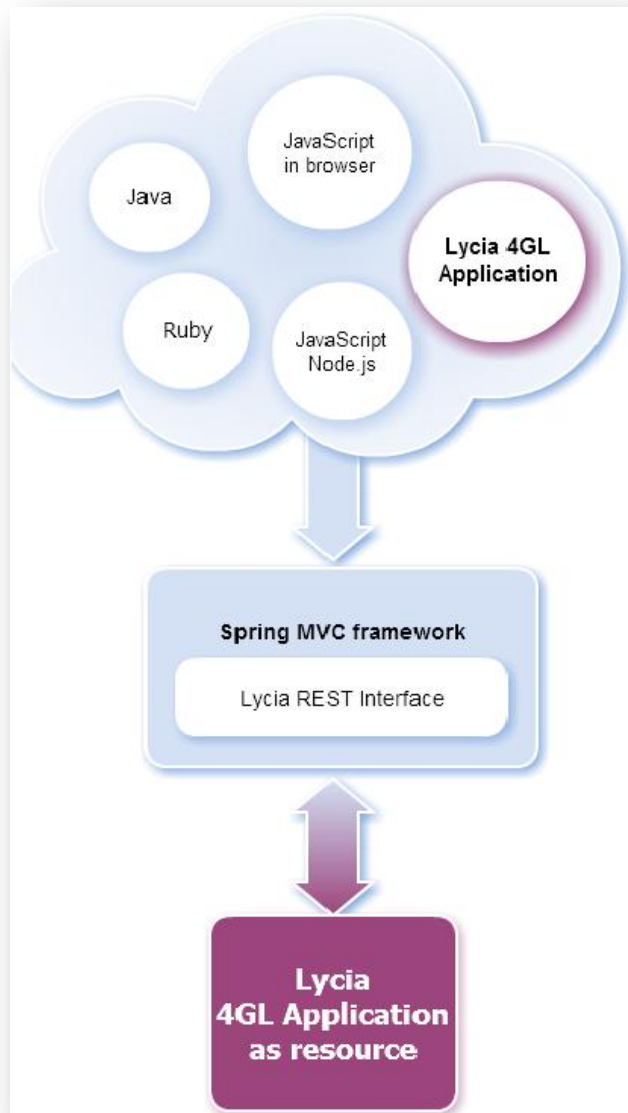
## User Guide

This chapter is dedicated to the overview of the different possibilities in setting up connection between 4GL application (as web client) and Web API (as web service). A few distributed application examples are provided here. Lycia 4GL is used both

for client and server parts in these code samples but any part can be easily replaced with any other modern programming language.

Any 4GL function can be published as web service without any coding or code reading at all. LyciaWeb will translate HTTP requests to 4GL function calls, interaction statement actions and field updates. By default, any deployed application can be called this way. For security reasons only functions starting with the “web\_” prefix can be called. The prefix pattern can be changed during runtime. Full syntax for URLs and messages format is given in Chapter 3 of this Guide.

The API is accessed via /sapi and /api URLs of LyciaWeb, for stateful and stateless usages respectively. For example, in default installation the stateful Web API can be accessed from the same computer with URLs with the <http://localhost:9090/LyciaWeb/sapi> prefix.





The Lycia Web API application is a standard Java servlet and can be run on any standard Java servlet container. Session tracking is done by means of the container. So it has access to all the container options for session handling. By default, session is stored in cookies (the JSESSIONID parameter), but it can be also URL parameter or SSL session.

The format of URL for access to stateful API should be:

```
http://{host}:{port}/LyciaWeb/sapi/{guiserver_port}/{application}
```

or

```
http://{host}:{port}/LyciaWeb/sapi/{command}/{name}?{arg1}={val1}  
&{arg2}={val2}...
```

where

{host}	host name, e.g. localhost;
{port}	LyciaWebServer port number, which equals 9090 by default;
{guiserver_port}	LyciaWebServer port number, which equals 9090 by default;
{application}	application name from folder 'progs' of Application Server;
{command}	command for execution by current application. E.g. set, get, setfocus, callmethod, closewindow, etc.;
{name}	command parameter, like function name, field name, etc.;
{arg...}	parameter name;
{val...}	parameter value.

This is the example of the link that can be used to start the application:

```
http://localhost:9090/LyciaWeb/sapi/cms/cms?user=username&password=12345
```



**Servlet container is a part of a web server that interacts with the servlets.**  
**Among its known implementations, there are:**

- Apache Tomcat
- Oracle Weblogic
- IBM WebSphere
- GlassFish
- Jetty
- JBoss, etc.



The best way to write a web function is to write a new function inside which an old function is called. This will help you to avoid changing all the names in the calls to the old function inside your 4GL project. Name the function `web_<someone>`, it should have the same name for security reasons.

The following example illustrates how to call 4GL application through another 4GL application. Let's assume that there is some 4GL application called `web_srv_func_sum.exe` which we will use as a test application. This application contains additional function `web_f_sum`:

```
function web_f_sum(a, b)
  define a, b int
  return a + b;
end function
```

To invoke this function in another application, we establish the web session and call the application `web_srv_func_sum.exe`.

```
main
  define ses web.Session
  define r int
  call ses.setLyciaApplication("localhost:9090/LyciaWeb/sapi",
    "default-1889", "web_srv_func_sum.exe")
  call ses.callMethod("web_f_sum", 5,7) returning r
  display "returns : ", r
end main
```

There is an option to include nested arrays in the code. The example below shows it can be realized within the code (CVS source: Server side application `web_srv_func_array2d.exe`):

```
main
  define str string
  define ch CHAR
  define a INTEGER

  let a = 25
  display "web_f_sum(2,3) returns ", web_f_sum(2,3)

  prompt "Input the value:" for str

  if str is null then
    display "string was not entered - exit"
    exit program 0
  end if

  display "WAIT 3 sec"
  sleep 3
end main
```



```
function web_f_sum(a, b)
  define a, b int

  return a + b;
end function

function web_matrix_int4x4_tranform(arr) -- the function is called from
client side

define arr, res array [4,4] of int
define i, j int

for i=1 to 4 step 1
for j=1 to 4 step 1
let res[i,j] = arr [j,i]
end for
end for

return res

end function
```

The program from the client side will look as follows:

```
main
  define ses web.Session
  define rq web.Request
  define rs web.Response
  define r int

  define arr, res array [4,4] of int

  define i, j int

  call ses.setLyciaApplication("localhost:9090/LyciaWeb/sapi", "default-
1889", "web_srv_func_array2d.exe")
  call ses.get() returning rs
  display "get: ", rs.getBody()

  call ses.get() returning rs
  display "get: ", rs.getBody()

  sleep 20

  display "call web_f_sum(102, 304) "
  call ses.callMethod("web_f_sum", 102, 304) returning r

  display "returns : ", r
```



```
for i=1 to 4 step 1
  for j=1 to 4 step 1
    let arr[i,j] = 4*(j-1)+i-1
  end for
end for

for i=1 to 4 step 1
  display arr[i,1], ", ", arr[i,2], ", ", arr[i,3], ", ", arr[i,4]
end for
display ""

call ses.callMethod("web_matrix_int4x4_tranform", arr) returning res
display "result: ", res

for i=1 to 4 step 1
  display res[i,1], ", ", res[i,2], ", ", res[i,3], ", ", res[i,4]
end for

end main
```

The next example illustrates interaction statements usage for server side:

```
main
  define nam cahr(20),
    age int,
    mstat char(1),
    total char(30)

  options input wrap
  options message line last

  open form f1 from "fields_fm"
  display form f1

  display "Press F1 to call the function" at 2,2 ATTRIBUTES(GREEN)
  display "!" TO b_ok
  input nam, age, mstat, total from pers_rec.*

  ON KEY (F1)
    message "KEY EVENT F1"
    let total = trim(nam) || " " || age || " " || trim(mstat)
    display total to formonly.total
  end input
```



**end main**



The interaction statements usage for client side:

```
main
  define ses web.Session
  define rq web.Request
  define rs web.Request
  define r int

  call ses.setLyciaApplication("localhost:8080/LyciaWeb/sapi", "default-
1889", "web_srv_form1.exe")

  call ses.get() returning rs
  display "get1: ", rs.getBody()

  call ses.set("name")

  call ses.key("nextfield");
  call ses.get() returning rs
  display "get2: ", rs.getBody()

  call ses.set("33")

  call ses.key("nextfield");

  call ses.set("M")
  call ses.get() returning rs
  display "get3: ", rs.getBody()

  call ses.key("F1");
  call ses.get() returning rs
  display "get4: ", rs.getBody()

  call ses.key("nextfield");
  call ses.get() returning rs
  display "get5: ", rs.getBody()

  call ses.key("accept");
end main
```



There is another lightweight mode for automatic program execution similar to Web API which does not require deployment to application server and does not involve any network interaction. It is called headless mode in this document. The application can be run simply from terminal in headless mode if QX\_HEADLESS\_MODE environment variable is set to 1. It will receive requests to its standard input and send responses to standard output. So some timed automatic tasks implemented in 4GL may be run via this mode using some shell script.

See the examples of interaction in Lycia Command Line for headless mode in XML below.

The sample application in cvs: web\_srv\_form1.exe

In the given code, the symbols '<<<' and '>>>' mean server response and client response respectively. Customers can set their own in the end of the code to identify the response direction.

```
<?xml version="1.0" encoding="utf-8"?>
<response><openwindow>SCREEN</openwindow><displayform>f1</displayform><displayat
row="2" col="2" attr="green">Press F1 to call the function</displayat><displayt
o field="b_ok" attr="">!</displayto><input><nam/><age>0</age><mstat/><total/></i
nput><setfocus>formonly.nam</setfocus><interact_dialog/></response>
<<<
```

```
<?xml version="1.0" encoding="utf-8"?>
<fgleftfield name="nextfield"/>
>>>
```

```
<?xml version="1.0" encoding="utf-8"?>
<response><setfocus>formonly.age</setfocus><interact_dialog/></response>
<<<
```

```
<?xml version="1.0" encoding="utf-8"?>
<setvalue>12</setvalue>
>>>
```





```
<?xml version="1.0" encoding="utf-8"?>
<fglevent name="nextfield"/>
>>>
```

```
<?xml version="1.0" encoding="utf-8"?>
<response><setfocus>formonly.mstat</setfocus><interact_dialog/></response>
<<<
```

```
<?xml version="1.0" encoding="utf-8"?>
<setvalue>S</setvalue>
>>>
```

```
<?xml version="1.0" encoding="utf-8"?>
<fglevent name="f1"/>
>>>
```

```
<?xml version="1.0" encoding="utf-8"?>
<response><displayto field="total" attr="">&#32;
</displayto><interact_dialog/></response>
<<<
```

```
<?xml version="1.0" encoding="utf-8"?>
<fglevent name="nextfield"/>
>>>
<?xml version="1.0" encoding="utf-8"?>
<response><setfocus>formonly.total</setfocus><interact_dialog/></response>
<<<
```

```
<?xml version="1.0" encoding="utf-8"?>
<fglevent name="accept"/>
>>>
```



## Reference

### Server interface for commands

Run application:

[http://localhost:9090/LyciaWeb/sapi/default-1889/web\\_srv\\_func\\_sum.exe](http://localhost:9090/LyciaWeb/sapi/default-1889/web_srv_func_sum.exe)

Set value:

<http://localhost:9090/LyciaWeb/sapi/set/{field}>

Value is taken from request body.

Set focus:

<http://localhost:9090/LyciaWeb/sapi/setfocus/{field}>

The 'field' is the name of widget.

Send event:

<http://localhost:9090/LyciaWeb/sapi/key/{name}>

The 'name' is the event value. E.g. "accept".

Call 4GL function:

<http://localhost:9090/LyciaWeb/sapi/call/{name}?p1=v1&p2=v2>

The 'name' is the 4GL function name. The parameters are passed though request body or via parameters (optional). Function name should be started with "web\_".

Get data:

<http://localhost:9090/LyciaWeb/sapi/get>

Read response from server.

Close window:

<http://localhost:9090/LyciaWeb/sapi/closewindow>

Close active 4GL form.

The server sends responses which are messages in XML-format. Codepage is UTF-8.

The examples of the server responses are provided further.

```
<?xml version="1.0" encoding="utf-8"?>
<setvalue>12345</setvalue>
```



```
<?xml version="1.0" encoding="utf-8"?>
<fglevent name="accept"/>
```

```
<?xml version="1.0" encoding="utf-8"?>
<setfocus>formonly.f001</setfocus>
```

```
<?xml version="1.0" encoding="utf-8"?>
<closewindow/>
```

```
<?xml version="1.0" encoding="utf-8"?>
<response>
<WriteToConsole>
<message>message text </message>
</WriteToConsole>
</response>
```

```
<?xml version="1.0" encoding="utf-8"?>
<response>
<openwindow>SCREEN</openwindow>
<displayat row="1" col="0" attr=""/>
<setfocus>Prompt Field</setfocus>
<interact_prompt/>
</response>
```



When you access application as a resource, there is a set of the command customers can use. Below you can find the table with the command and their descriptions.

Function name	Result
FglEvent	Calls a specified action to be taken by the program on the server. Execution point of the application should be in the corresponding interaction statement.
CallMethod	Calls the specified Method by name. The parameter NAME must be indicated. The execution point should be in any interaction statement (PROMPT, MENU, INPUT, etc.). The call itself does not change its execution state, it is left in the interaction statement You can set the option of which method can be called and which one cannot. The name of method should start with web_. This pattern can be set.
SetFocus	Sets the focus to the required field. The parameter NAME must be indicated. If you work with an array, specify the parameter ROW. Execution should be in the corresponding interaction statement; all the handlers, which are required to be called from moving focus from one field to another, will be executed.
SetValue	Sets the value for a specified field. Execution point of the application should be in the corresponding interaction statement.
CallMenuItem	Calls a specified menu item. The parameter NAME must be indicated. Execution point of the application should be in the corresponding interaction statement.
CallTopMenuItem	Calls a specified item from the top menu. Execution point of the application should be in the corresponding interaction statement.
CloseWindow	Closes the currently active window. Execution point of the application should be in the corresponding interaction statement. This simulates pressing close button on window title.



Developers can take field names directly from Lycia Theme Designer; it will save their time when wrapping 4GL application into web service as there is no need to read 4GL sources or forms.



Once server gets the request from client, it sends back a message informing about the corresponding action taken on server. Web client can either process data or ignore, if it does not assume handling of such messages. The list of the server responses that do not require any client's reply is as follows:

- fgl\_bell()
  - fgl\_upload() and fgl\_download()
  - fgl\_dialog\_setcursor()
  - fgl\_dialog\_setselection()
  - fgl\_grid\_export()
  - fgl\_grid\_viewexport()
  - exec\_program()
  - fgl\_winmessage()
  - fgl\_setproperty()
  - WriteToConsole
  - WriteToTextViewer
  - ShowSystemMessageBox
  - open\_url
  - start\_program
- 
- RUN
  - DISPLAY ARRAY
  - INPUT
  - INPUT ARRAY
  - EXECUTE FUNCTION

Since 4GL supports a set of statements that allows users to interact with the program and to maintain its execution, server relates with the client by means of the number of interactions which the client must give response to.

- interact\_prompt()
- interact\_dialog()
- interact\_menu()
- interact\_fgl\_getkey()

An XML schema defines the rules about how your request should be composed. Generally, you define this schema for each representation you wish to send to the web server.

XSD (XML Schema Definitions ) formally describes the structure of XML document. XSD is used to represent interrelations between the elements and attributes of an XML object. Here is the list of



functions which represents functions. You can find the corresponding XSD-schemes in the **APPENDIX. XSD files for server response.**

The functions updates a specified row in the array. The array and row number must be indicated as the parameters inside the function:

```
<updaterow name="arr_name" row="6">...</updaterow>
```

The function inserts a new row in the array. The array and row number must be indicated as the parameters inside the function:

```
<insertrow name="arr_name" row="6">...</insertrow>
```

The function removes a row from the array. The array and row number must be indicated as the parameters inside the function:

```
<deleterow name="arr_name" row="6">...</deleterow >
```

The function displayed a specified form in the array. The array and row number must be indicated as the parameters inside the functions:

```
<displayform>form_name</displayform>
```

The function displays a specified value in the field. The name of the field must be indicated as the parameters inside the function; attributes and row number are optional:

```
<displayto field="field_name" attr="attributes" row="row_num_optional">string</displayto>
```

The function opens a window. Window name must be indicated:

```
<openwindow>window_name</openwindow>
```

The function displays the specified value at the row. The row and column must be indicated in the parameters:

```
<displayat row="2" col="4" attr="attributes">string</displayat>
```

The function sets focus in the requested field. The row must be indicated in the parameters inside the function:

```
<setfocus row="row_num_optional">field_full_name</setfocus>
```

The function displays the result of the execution if the statement EXECUTE FUNCTIONS.

```
<results>...</results>
```

Besides the secondary list of responses from 4GL application, there is the list of functions that require the client`s response. Until the server receives the response from the client, the server



will be in the waiting mode. These functions should be responded by client. Client does not have to perform them or to do some other action.

Querix 4GL supports Dynamic Data Exchange (DDE), a Microsoft communications protocol for exchanging data between applications. To invoke DDE, users can operate a whole number of functions:

- DDEConnect()
- DDEExecute()
- DDEFinish()
- DDEFinishAll()
- DDEGetError()
- DDEPeek()
- DDEPoke()The FrontCall method executes function calls to the current front end:
- FrontCallExecute
- FrontCallShellExec
- FrontCallLaunchUrl
- FrontCallGetWindowId
- FrontCallOpenDir
- FrontCallOpenFile
- FrontCallSaveFile
- FrontCallCbClear
- FrontCallCbSet
- FrontCallCbGet
- FrontCallCbAdd
- FrontCallCbPast
- FrontCallHardCopy

Other functions:

- fgl\_dialog\_getcursor()
- fgl\_dialog\_getselectionend()
- fgl\_message\_box
- fgl\_winquestion
- fgl\_winbutton
- fgl\_winprompt
- fgl\_file\_dialog
- fgl\_getproperty
- execute\_menu()
- InterfaceGetContainer



- InterfaceGetChildCount
- WinExec

You can record and reply all the Web API messages that run the application in the GUI mode. To set up the option, add the corresponding environment variable to your inet.env file.



**Inet.env is a configuration file that contains environment variables required by GUI server. Double clicking the environment tab allows you to customize the settings in the form of environment variables.**

In LyciaStudio, go to **Window ->Preferences ->4GL ->Run/Debug ->GUI Servers**. Choose the current GUI server instance and press the radio button **Edit Environment**.

Now, when the file is opened in the editor area of the Studio, find QXDEBUG in the list of current environment variables. Make sure that you have the underlined chars in this string:

QXDEBUG=! **zA**

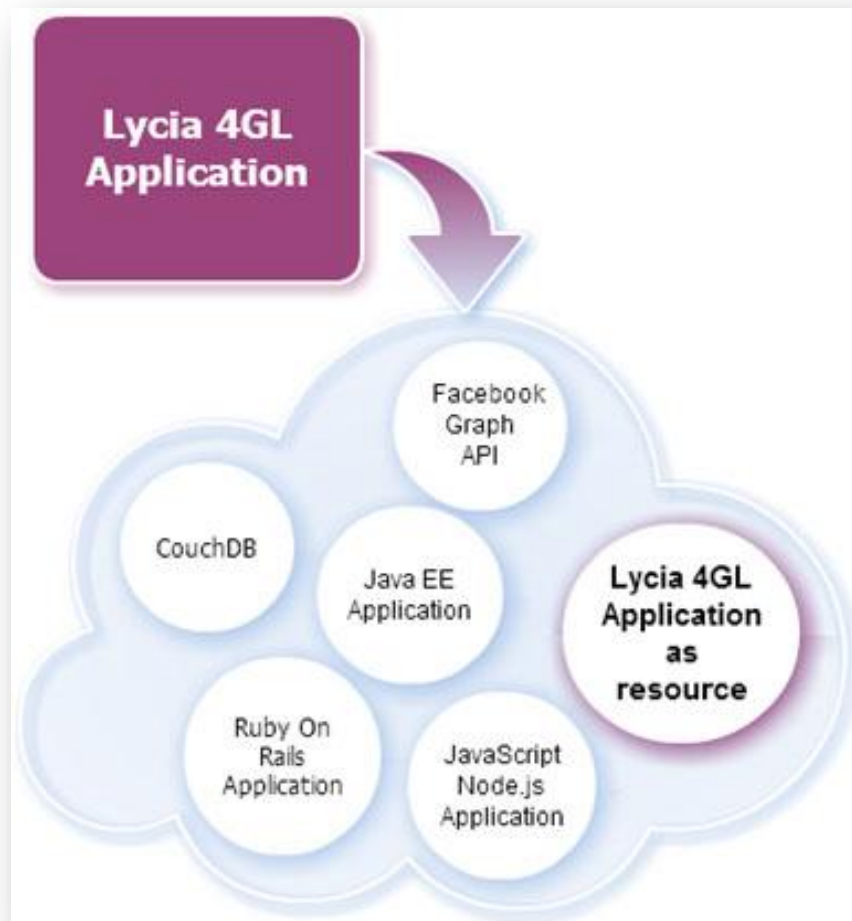
Once you make the modifications, you will be able to check the recorded logs in the corresponding folder "logs". The default path to the folder is *C:/ProgramData/Querix/Lycia6/logs*. Every application is recorded to a single log file which looks as a plain text file. As output, you will receive files in the XML format. In the current version, this format is prevalent though, we are planning to extend the package of format (XML/HTML/JPEG/GIF/HEADLESS mode).





## CHAPTER 3

### Accessing 4GL application with the Web API



Web API is a framework, that turns creation of HTTP services into an easy and flexible process. It embraces a wide range of clients, including browsers and mobile devices, allowing the customers to easily show off their service data to any browser or device.

Among Web API distinctive features is that it allows the business logic 4GL projects integration into the cloud. Moreover, there is a possibility not just to call services from within a 4GL application, but to call 4GL service by a third party service as well. Because of the structure flexibility and freedom, even programs with different interaction statements can be operated on with no restrictions.



## 4GL functions To Call Web Services

### 4GL API Reference

#### URL Structure

##### Any URL



For the better understanding of URL structural components meaning, we suggest you to consider the following usage example, that is a typical URL sample aimed at a web application creation:

```
http://user:password@localhost:9090/www.querix.com/directory/page.html?par1=1&param2=2#123"
```

Part	Meaning	Sample
protocol	used to communicate with a resource	http
login and password	allows <i>username</i> and <i>password</i> indication inside an URL	user:password@localhost:9090/
Host	determines the server where the resource is located	www.querix.com
Path	determines the definite resource on the server	/directory/page.html
Query	instructs the server to perform additional operation to identify the exact chunk of a resource that is desired.	par1=1 param2=2#123



## Reference designation

*Atomic* - can be any simple 4GL datatype

*UrlString* - any simple alias of a string

*Method* - enumeration that is simple 4GL string with a predefined case insensitive value



## Table of the commands used for Web API objects

### web.Response object

Method Name	Argument	Returned Value	Description
getHeader		Atomic	Returns the value of the specified request header as a <code>String</code> .
getParameter		Atomic	Returns any requested parameter.
getSession		Session	Returns the current session associated with this request, or if the request does not have a session, creates one.
getBody		(Xml   Atomic)*	Returns HTTP-body of the requested URL.
getRequest		Request	Returns the value of a request parameter as a <code>String</code> , or <code>null</code> if the parameter does not exist.
getUrl		URL	Download one or more URLs.
getHost		String	Returns the requested host.
getPort		String	Enumerates the ports available on a service.
getCookie	String	(Atomic)*	Retrieves a cookie for the location specified by a URL. Cookies parameters are specified by position.
setHeader	Atomic	Void	Sets a single header value for implicit headers. If this header already exists in the to-be-sent headers, it's value will be replaced.
setParameter	Atomic	Void	Sets the value of one or more parameters to be used in subsequent transformations. If the parameter doesn't exist on the receiving side, it will be ignored.
setSession	Session	Void	Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.
setHost	String	Void	Sets the connection to the host.
setPort	String	Void	Sets the class port configuration parameters.



setBody	(Xml Atomic)*	Void	Sets the body text of the mail message. This method replaces any existing body text with the string you specify.
setCookie	String (Atomic)*	Atomic	Defines a cookie to be sent along with the rest of the HTTP headers. Cookies must be sent <i>before</i> any output from your script (this is a protocol restriction).

#### web.Request object

Method Name	Argument	Returned Value	Description
getHeader		Atomic	Returns the value of the specified request header as a <code>String</code> .
getParameter		Atomic	Returns any requested parameter.
getSession		Session	Returns the current session associated with this request, or if the request does not have a session, creates one.
getBody		(Xml   Atomic)*	Returns HTTP-body of the requested URL.
getUrl		URL	Download one or more URLs.
getCookie	String	(Atomic)*	Retrieves a cookie for the location specified by a URL. Cookies parameters are specified by position.
getMethod		Method	Retrieves whatever information (in the form of an entity) is identified by the Request-URL
perform	(UrlString  (Method UrlString?))? (Request Url  Atomic  Session Xml)*	(Response   Atomic  Session Xml)*	Performs HTTP-request to the server's side
setHeader	Atomic		Sets a single header value for implicit headers. If this header already exists in the to-be-sent headers, it's value will be replaced.
setParameter	Atomic		Sets the value of one or more parameters to be used in subsequent transformations. If the parameter doesn't exist on the receiving side, it will be ignored.



setSession	Session		Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.
setBody	(Xml Atomic)*		Sets the body text of the mail message. This method replaces any existing body text with the string you specify.
setCookie	String (Atomic)*		Defines a cookie to be sent along with the rest of the HTTP-headers. Cookies must be sent <i>before</i> any output from your script (this is a protocol restriction).
setMethod	Method		Creates and saves a formal method for a given function and list of classes.



## **web.Url object**

Method Name	Argument	Returned Value	Description
getProtocol		String	Returns the protocol name
setProtocol	String		Sets the protocol (HTTP, HTTPS)
getUser		String	Returns the user's login
setUser	String		Sets the user name
getPassword		String	Returns the password
setPassword	String		Sets the password
getHost		String	Returns the requested host
setHost	String		Sets the host
getPort		String	Returns TCP port number.
setPort	String		Sets the port component.
getQuery		String	Returns parameters
setQuery	String		Sets parameters
getFragmentId		String	Returns fragment id
setFragmentId	String		Sets fragment id
getUrl		String	Convert object to String
setUrl	String		Sets all field from String representation



## web.Session object

Method Name	Argument	Returned Value	Description
getCookie	String	Atomic*	Retrieves a cookie for the location specified by a URL
get		(Response  Atomic  Xml)*	Returns the value of a request parameter as a String
setCookie	String (Atomic)*	Atomic	Defines a cookie to be sent along with the rest of the HTTP-headers. Cookies must be sent <i>before</i> any output from your script (this is a protocol restriction)
setLyciaApplication	String String String String?	Void	Run a specified Lycia application and sets connection to it. All required parameters should be placed here in the order: Prefix, instance, application, parameters (optional).
setFocus	String	(Response  Atomic  Session  Xml)*	Sets the keyboard focus to the specified window.
set	String	(Response  Atomic  Session  Xml)*	Sets the specified value to the current field.
callMethod	String	(Response  Atomic  Session   Xml)*	Is used to call the method during the current session.
create	()	Session	static method which creates the session.
action	String	(Response  Atomic  Session  Xml)*	Calls the needed command from the server's side
key	String	(Response  Atomic  Session X ml)*	Simulates a key pressing.
perform	(UrlString  (Method UrlString?))? (Request  Url  Atomic	(Response  Atomic  Session  Xml)*	Performs http-request to the server's side.





	Session   Xml)*		
Request. perform	(UrlString  (Method UrlString?))? (Request  Url  Atomic  Session   Xml)*	(Response  Atomic  Session  Xml)*	Performs http-request to the server's side.
focus	String Integer		<p>Specifies the required field bringing it into focus.</p> <p>ROW is used within DISPLAY ARRAY and INPUT ARRAY statements; it is taken into the square brackets as it is optional.</p> <p>URL invokes 4GL BEFORE/AFTER FIELD.</p>



### Other static methods

Method Name	Argument	Returned Value	Description
Request.create	(Method UrlString?)? (Url   Atomic  Session Xml)*	Request	Requests either creation of or access to a file.
Response.create	(Atomic Session  Xml Request)*	Response (Request is required but order of arguments is not important - may be simplified with predefined order though)	Request creation of the response from the server side.

### Implicit Conversations

Object *web.Response* can be implicitly converted to *String* value which takes a value of *Response* body.

Object *web.URL* can be implicitly converted to *String* which will be the same like result of its *getUrl()* method.



## APPENDIX. XSD files for server response.

### Rows updating

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="updaterow">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:string" name="name"/>
          <xs:attribute type="xs:byte" name="row"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

### Row inserting

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="insertrow">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:string" name="name"/>
          <xs:attribute type="xs:byte" name="row"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



## Rows deleting

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="deleterow">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:string" name="name"/>
          <xs:attribute type="xs:byte" name="row"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

## Forms displaying

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="displayform" type="xs:string"/>
</xs:schema>
```

## Displaying to fields

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="displayto">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:string" name="field"/>
          <xs:attribute type="xs:string" name="attr"/>
          <xs:attribute type="xs:string" name="row"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



## Opening Windows

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="openwindow" type="xs:string"/>
</xs:schema>
```

## Displaying At Rows

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="displayat">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:byte" name="row"/>
          <xs:attribute type="xs:byte" name="col"/>
          <xs:attribute type="xs:string" name="attr"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

## Setting Focus

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="setfocus">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:string" name="row"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

## Set

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="setvalue" type="xs:string"/>
</xs:schema>
```

