

## Integrating OsiriX in a HIS/RIS environment

OsiriX is a DICOM Viewer, aka PACS Workstation, but it is not a HIS or a RIS software. If you have such software installed, it can be useful to 'control' OsiriX or 'get' informations from/to another application:

- an application running on the same computer. For example a Java application or a Web based software
- an application running on the same computer, but in a different environment. For example in Windows OS, in Parallels or VMware Fusion software.
- an application running on a different computer on the same network. For example a web server application, like an ASP software.

This application should be able to control OsiriX to facilitate the integration. For example, the other application should be able to:

- Check if a study is available in the OsiriX Database
- Open a specific study, and display the images
- Delete a study
- ...

### XML-RPC Protocol

With OsiriX 3.0, we introduced a very easy and powerful protocol for such control. OsiriX integrates now an HTTP server with an XML-RPC protocol. It means that any software/computer that knows your network address can execute methods in OsiriX. These methods can be built-in in OsiriX or added in plugins. To turn the XML-RPC HTTP server, go to the Listener Preferences window in OsiriX.

### osirix:// URL

With OsiriX 3.9, we introduced a new protocol for such control: osirix:// URL. With this custom URL interface, your web browser can execute orders in OsiriX. The limitation, compared to XML-RPC, is that you cannot get an answer to the order. For example DBWindowFind cannot return the corresponding elements. The other limitation is that this URL is only supported locally: a web browser can only start and control the installed OsiriX, running on the same computer. That means you cannot use the osirix:// URL to send a message from a server to a distant OsiriX. But the server can display an html page on the client computer, and the user can click on the osirix:// URL to execute the action.

### Supported Messages

For now, only few basic messages are supported. If you need more, write a simple XML-RPC plug-in (see below). If you think that your plugin source code should be included in OsiriX, contact me ([rossetantoine@osirix-viewer.com](mailto:rossetantoine@osirix-viewer.com)).

## Integrating OsiriX in a HIS/RIS environment : XML-RPC / osirix:// URL support

Method: DBWindowFind  
Parameters:  
request: SQL request, see 'Predicate Format String Syntax' from Apple doc.  
table: OsiriX Table: Image, Series, Study  
execute: Nothing, Select, Open, Delete  
execute is performed at the study level: you cannot delete a single series of a study  
Example: {request: "name == 'OsiriX'", table: "Study", execute: "Select"}  
Example: {request: "(name LIKE '\*OSIRIX\*')", table: "Study", execute: "Open"}  
Response: {error: "0", elements: array of elements corresponding to the request}

Method: DownloadURL  
Parameters:  
URL: It's the URL where to access the file. Currently only zip files or DICOM files are supported.  
Display: It's a boolean value, indicating if the study has to be selected and opened, after the download is complete. This parameter is optional.  
Example: {URL: "http://www.myserver.com/dicomfile.dcm"}  
Example: {URL: "http://www.myserver.com/dicomfile.dcm", Display: "YES"}  
Response: {error: "0"}

Method: CloseAllWindows  
Parameters: No Parameters  
Response: {error: "0"}

Method: OpenDB  
Parameters:  
path: path of the folder containing the 'OsiriX Data' folder  
if path is valid, but not DB is found, OsiriX will create a new one  
Example: {path: "/Users/antoineRosset/Documents/"}  
Response: {error: "0"}

Method: SelectAlbum  
Parameters:  
name: name of the album  
Example: {name: "Today"}  
Response: {error: "0"}

Method: GetDisplayed2DViewerSeries  
Parameters: No Parameters  
Response: {error: "0", elements: array of series corresponding to displayed windows}

## Integrating OsiriX in a HIS/RIS environment : XML-RPC / osirix:// URL support

```
Method: GetDisplayed2DViewerStudies
Parameters: No Parameters
Response: {error: "0", elements: array of studies corresponding to displayed windows}
```

```
Method: Close2DViewerWithSeriesUID
Parameters:
uid: series instance uid to close
Example: {uid: "1.3.12.2.1107.5.1.4.51988.4.0.1164229612882469"}
Response: {error: "0"}
```

```
Method: Close2DViewerWithStudyUID
Parameters:
uid: study instance uid to close
Example: {uid: "1.2.840.113745.101000.1008000.37915.4331.5559218"}
Response: {error: "0"}
```

```
Method: CMove
Parameters:
accessionNumber: accessionNumber of the study to retrieve
server: server description where the images are located (See OsiriX Locations Preferences)
Example: {accessionNumber: "UA876410", server: "Main-PACS"}
Response: {error: "0"}; -1: server not found, -2 server not responding, -3 no study found
```

You need to send these messages in XML format: for example `Close2DViewerWithStudyUID`, with parameters: `{uid: "1.2.840.113745.101000.1008000.37915.4331.5559218"}` corresponds in XML to:

```
<?xml version="1.0"?>
<methodCall> <methodName>Close2DViewerWithStudyUID</methodName>
<params>
<param>
<value><struct>
<member>
<name>uid</name>
<value><string>1.2.840.113745.101000.1008000.37915.4331.5559218</string></value>
</member>
</struct></value>
</param>
</params>
</methodCall>
```

If you send non-ASCII characters (like é, ç, ä, ...), use UTF-8 encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
```

### **osirix:// URL Supported Messages**

Starting with OsiriX 3.9, you can use all the XML-RPC extensions in the osirix:// URL scheme.

For example, you can do:

```
osirix://?methodName=downloadURL&URL='http://www.myserver.com/dicomfile.dcm'
```

```
osirix://?methodName=retrieve&serverName=Minipacs&filterKey=PatientID&filterValue=296228
```

```
osirix://?methodName=findObject&request=(name CONTAINS[cd] 'KNEE')&table=Study&execute=Open
```

To test this URL scheme, launch OsiriX and copy & paste this string in Safari or any web browser: **osirix://?methodName=KillOsiriX**

## How to add your own XML-RPC message protocol?

If you need to add specific functions and behavior in OsiriX for your XML-RPC messages, you can write a simple plug-in. Thanks to the XML support in Cocoa environment (NSXMLDocument), it is very easy to add your own method in OsiriX. An example is available in the Plugins folder of the OsiriX source code: XML-RPC-Plugin. In this plug-in, two new messages are added to OsiriX:

- exportSelectedToPath - {path:"/Users/antoineRosset/Desktop/"}
- openSelectedWithTiling - {rowsTiling:2, columnsTiling:2}

As you see in the source code, a unique plug-in can support multiples messages and parameters decoding is easy:

```

if( [[httpServerMessage valueForKey: @"MethodName"] isEqualToString: @"exportSelectedTo-
Path"])
{
    NSXMLDocument *doc = [httpServerMessage valueForKey:@"NSXMLDocument"];

    NSError *error = 0L;
    NSArray *keys = [doc nodesForXPath:@"methodCall/params//member/name" error:&error];
    NSArray *values = [doc nodesForXPath:@"methodCall/params//member/value" error:&error];
    if (1 == [keys count] || 1 == [values count])
    {
        int i;
        NSMutableDictionary *paramDict = [NSMutableDictionary dictionary];
        for( i = 0; i < [keys count]; i++)
            [paramDict setValue: [[values objectAtIndex: i] objectValue] forKey: [[keys
objectAtIndex: i] objectValue]];

        // Ok, now, we have the parameters -> execute it !

        NSMutableArray *dicomFiles2Export = [NSMutableArray array];
        NSMutableArray *filesToExport;

        filesToExport = [[BrowserController currentBrowser] filesForDatabaseOutlineSele-
ction: dicomFiles2Export onlyImages:YES];
        [[BrowserController currentBrowser] exportDICOMFileInt: [paramDict
valueForKey:@"path"] files: filesToExport objects: dicomFiles2Export];

        // Done, we can send the response to the sender

        NSString *xml = @"<?xml
version=\"1.0\"?><methodResponse><params><param><value><struct><member><name>error</name><val
ue>0</value></member></struct></value></param></params></methodResponse>";
        // Simple answer, no errors

        NSError *error = nil;
        NSXMLDocument *doc = [[[NSXMLDocument alloc] initWithXMLString:xml
options:NSXMLNodeOptionsNone error:&error] autorelease];
        [httpServerMessage setValue: doc forKey: @"NSXMLDocumentResponse"];
        // To tell to other XML-RPC that we processed this order
        [httpServerMessage setValue: [NSNumber numberWithInt: YES] forKey: @"Processed"];
    }
}

```

## How to send an XML-RPC Message?

You are writing an application and you want to communicate with OsiriX. Hence you need to send an XML-RPC message to OsiriX: send the XML message through a HTTP POST procedure.

There is a nice little **XML-RPC Client**: <http://www.ditchnet.org/xmlrpc/>. This software allows you to test your messages with OsiriX:

- Start OsiriX, and activate the HTTP Server in the Listener Preferences
- Start XML-RPC Client, and enter your own address: `http://localhost:8080` (don't forget the 'http://' prefix)
- Enter the method name and the parameters:

Endpoint URI:	<code>http://localhost:8080</code>
Method:	<code>openSelectedWithTiling</code>
Params:	<code>{rowsTiling:2, columnsTiling:2}</code>



- And click Execute button. OsiriX will then try to execute the message and send back an answer in XML format to the sender.

Other software are available to edit XML messages: TextMate is great (<http://macromates.com/>). It allows you to validate your XML message.

Sending an XML message through a HTTP POST is easy in many languages:

In Cocoa framework:

```
NSString *xml = @"<?xml version=\"1.0\"?>";
NSError *error = nil;
NSXMLDocument *doc = [[[NSXMLDocument alloc] initWithXMLString:xml
options:NSXMLNodeOptionsNone error:&error] autorelease];

NSData *data = [doc XMLData];
NSMutableURLRequest *request = [[[NSMutableURLRequest alloc] initWithURL:url
cachePolicy:NSURLRequestReloadIgnoringCacheData timeoutInterval:60.0] autorelease];
[request setHTTPMethod:@"POST"];
[request setHTTPBody:data];
[request setValue:[NSString stringWithFormat:@"%d", [data length]]
forHTTPHeaderField:@"Content-Length"];

NSURLConnection *conn = [[[NSURLConnection alloc] initWithRequest:request delegate:self]
autorelease];

while (![conn isFinished])
{
    [[NSRunLoop currentRunLoop] runMode:NSDefaultRunLoopMode beforeDate:[NSDate distantFuture]];
}

// The answer from the HTTP Server

NSError *error = nil;
NSXMLDocument *reply = [[[NSXMLDocument alloc] initWithData:[conn data]
options:NSXMLNodeOptionsNone error:&error] autorelease];
```

For Apache:

<http://ws.apache.org/xmlrpc/>

In Applescript:

[http://developer.apple.com/documentation/AppleScript/Conceptual/soapXMLRPC/chapter2/chapter\\_2\\_section\\_3.html](http://developer.apple.com/documentation/AppleScript/Conceptual/soapXMLRPC/chapter2/chapter_2_section_3.html)

In other languages (RealBasic, VBasic, C, .NET, #C, ...)

<http://www.xmlrpc.com/directory/1568/implementations>

**To test a XML-RPC message from a file, use the curl tool from the console.app:**

`curl -d @/Users/antoinesosset/Desktop/test.xml "http://localhost:8080"`