ADOBE® INDESIGN® CS5 SERVER

WORKING WITH ADOBE INDESIGN CS5 SERVER SOAP

Adobe

Working with Adobe® InDesign® CS5 Server SOAP

Working with Adobe InDesign CS5 Server SOAP

Contents

Overview

Adobe® InDesign® Server opens communication channels via SOAP and CORBA. This document describes the SOAP implementation for InDesign Server, namely, the `RunScript` method. (For information on InDesign Server and CORBA, see *Working with Adobe InDesign CS5 Server Java*.)

Terminology

SOAP (Simple Object Access Protocol) is an XML-based syntax and protocol specification for exchanging data between applications in a networked environment. SOAP provides the framework by which application-specific data may be conveyed in an extensible manner, as well as a full description of the required actions taken by a *SOAP node* on receiving a *SOAP message*.

A *SOAP message* is defined as "The basic unit of communication between SOAP nodes." A *SOAP node* is defined as "The embodiment of the processing logic necessary to transmit, receive, process and/or relay a SOAP message, according to the set of conventions defined by this recommendation. A SOAP node is responsible for enforcing the rules that govern the exchange of SOAP messages (see below). It accesses the services provided by the underlying protocols through one or more SOAP bindings." (Definitions are from http://www.w3.org/TR/soap12-part1/#terminology.)

InDesign Server WSDL (Web Services Description Language)

What is WSDL?

WSDL is an XML-formatted language used to describe a Web service's capabilities as collections of communication endpoints capable of exchanging messages. In this context, a WSDL file is an XML file that defines the types, method(s), parameters, and result structures of InDesign Server's SOAP implementation.

There are many programming languages that can be used to implement a SOAP solution, and there are a variety of toolkits and APIs that can be used depending on the language you choose for your solution. In general, when using a statically typed language (like Java, C++, and C#), the toolkit provides a tool that generates source code based on the WSDL. For example, Apache Axis provides the `wsdl4j` tool, which generates Java code based on a WSDL. You then include the generated source code in your solution. For dynamically typed languages (like PHP and ASP.NET), the WSDL is used by the API dynamically at runtime.

Location of the InDesign Server WSDL

There are two ways to access the InDesign Server WSDL:

- Use the `IDSP.wsdl` file located in the `docs/references` folder of the InDesign Server SDK. The InDesign Server WSDL contains a `<SOAP:address location>` element that defines the default instance of InDesign Server with which a client using the WSDL will interact. The default location is `http://localhost:80`. You can modify this default by editing your version of `IDSP.wsdl` to use the desired instance of InDesign Server. The location element is located near the end of the file.
- Use HTTP to request the WSDL from a running instance of InDesign Server. For example: `http://localhost:12345/service?wsdl`. When accessing the WSDL through HTTP, the location element gets defined as the instance of InDesign Server used to generate the WSDL. In this example, InDesign Server was started with the -port option set to 12345; therefore, the instance is `http://localhost:12345`.

What is in the InDesign Server WSDL?

The primary types defined in the InDesign Server WSDL are described here:

- **`IDSP-ScriptArg`** — A sequence containing strings for name and value. The `IDSP-ScriptArg` type is used within the RunScriptParameters type.

```
<complexType name="IDSP-ScriptArg">
    <sequence>
        <element name="name" type="xsd:string" minOccurs="1" maxOccurs="1" />
        <element name="value" type="xsd:string" minOccurs="1" maxOccurs="1" />
    </sequence>
</complexType>
```

- **`RunScriptParameters`** — A sequence containing `scriptText`, `scriptLanguage`, `scriptFile`, and `scriptArgs` elements. The `RunScriptParameters` type is used as a parameter to the RunScript method.

```
<complexType name="RunScriptParameters">
    <sequence>
        <element name="scriptText" type="xsd:string"
            minOccurs="0" maxOccurs="1" nillable="true" />
        <element name="scriptLanguage" type="xsd:string"
            minOccurs="0" maxOccurs="1" nillable="true" />
        <element name="scriptFile" type="xsd:string"
            minOccurs="0" maxOccurs="1" nillable="true" />
        <element name="scriptArgs" type="IDSP:IDSP-ScriptArg"
            minOccurs="0" maxOccurs="unbounded" />
    </sequence>
</complexType>
```

- **`RunScript`** — The single method made available by InDesign Server.

```
<element name="RunScript">
    <complexType>
        <sequence>
        <element name="runScriptParameters" type="IDSP:RunScriptParameters"
            minOccurs="0" maxOccurs="1" nillable="true" />
        </sequence>
    </complexType>
</element>
```

- **`RunScriptResponse`** — The return type of the `RunScript` method.

```
<element name="RunScriptResponse">
    <complexType>
        <sequence>
            <element name="errorNumber" type="xsd:int" minOccurs="1" maxOccurs="1"/>
            <element name="errorString" type="xsd:string"
                minOccurs="0" maxOccurs="1" nillable="true"/>
            <element name="scriptResult" type="IDSP:Data"
                minOccurs="0" maxOccurs="1" nillable="true" />
        </sequence>
    </complexType>
</element>
```

The RunScript method

The InDesign Server SOAP implementation provides one method, `RunScript`. This method, its parameter list, and its result structure are defined by the InDesign Server WSDL. The `RunScript` method passes an InDesign Server-compatible script to InDesign Server. InDesign Server executes the script and returns the result to the caller.

The script can be written in JavaScript, AppleScript, or VBScript. It must be based on the InDesign Server scripting DOM (Document Object Model). Information about the InDesign Server scripting DOM is in *Adobe InDesign CS5 Server Scripting Guide*, located in the `scripting` folder within the InDesign Server SDK.

How you call `RunScript` depends on the language with which you are building your solution and what toolkit or API you are using. Generally, you use a SOAP API to create a `client` object that can call the methods defined in the WSDL. In interpreted languages (for example, PHP and ASP), you use a reference to the InDesign Server WSDL to create the client at runtime. In compiled languages (for example, C++, C#, and Java), you must first generate client code based on the WSDL, include that code in your project, and then instantiate a client object using the generated code.

For specific examples of how to create a client object and call `RunScript`, look at the sample client projects included with the InDesign Server SDK. There are samples for Java, Flex, C#, PHP, and ASP.NET, and in the InDesign Products SDK, there is a C++ example, SampleClient.

The InDesign Server SOAP message

SOAP works by passing XML messages between applications. The XML message that is sent from a client to InDesign Server is called a *SOAP request*, and the XML message that InDesign Server returns to the client is called a *SOAP response*. A SOAP XML message contains a root element of type `SOAP-ENV:Envelope`. The envelope element contains one element, of type `SOAP-ENV:Body`.

If the message is a request being sent to InDesign Server, the first element within the body element is a `RunScript` element. If the message is a response being returned from InDesign Server's RunScript method, the first element within the body element is a `RunScriptResponse` element.

The InDesign Server SOAP request envelope

The body of the request envelope contains one element, `RunScript`, which contains an element named `runScriptParameters`. The `runScriptParameters` element contains the following four elements that tell InDesign Server all it needs to know to run your script:

- *`scriptText`* — A string passed to InDesign Server, representing the entire script to be executed. This parameter is used only if the `scriptFile` parameter is empty.
- *`scriptLanguage`* — One of `javascript`, `applescript`, or `visual basic`.
- *`scriptFile`* — The path to the script to be executed by InDesign Server. The path can take one of two forms:
  - An absolute path to the script based on the file system of the targeted InDesign Server instance. For example:

    Windows: `c:/myScriptsFolder/myScript.jsx`

    Mac OS: `/myScriptsFolder/myScript.jsx`
  - A relative path to either the InDesign Server application scripts folder or the InDesign Server user's scripts folder. You must use a colon (:) as the path separator, on both Windows and Mac OS.

    The path prefix `Scripts:Application:` represents the application scripts folder:

    Windows: `C:\Program Files\Adobe\Adobe InDesign CS5 Server\Scripts\`

    Mac OS: `/Applications/Adobe InDesign CS5 Server/Scripts/`

    The path prefix `Scripts:User:` represents the user's scripts folder:

    Windows:
    ```
    C:\Documents and Settings\myName\Application Data\Adobe\
    InDesign Server\Version 7.0\en_US\configuration_12345\Scripts\
    ```

    Mac OS:
    ```
    /Users/myName/Library/Preferences/Adobe InDesign Server/
    Version 7.0/en_US/configuration_12345/Scripts/
    ```

    Use:
    ```
    Scripts:Application:myScriptsFolder:myScript.jsx
    Scripts:User:myScriptsFolder:myScript.jsx
    ```

`scriptArgs` — A series of elements, where each element contains `name` and `value` elements. ScriptArgs are stored by InDesign Server. They are accessed from within your script by accessing the InDesign Server `scriptArgs` object. Each of the following examples assigns a local variable to the value of a `scriptArg` named `argOne`.

JavaScript:
```
if (app.scriptArgs.isDefined("argOne")) {
    var myArgValue = app.scriptArgs.getValue("argOne");
}
```

Applescript:
```
tell script args
    if is defined name "argOne" then
        set myArgValue to get value name "argOne"
    end if
end tell
```

VBScript:
```
if myApp.ScriptArgs.IsDefined("argOne") then
    myArgValue = myApp.ScriptArgs.GetValue("argOne")
end if
```

Example The following is an example of a SOAP request that tells InDesign Server to run a JavaScript located at `c:\examplefiles\test.jsx`. Two `scriptArgs` are passed: arg0 = 88, and arg1 = "some text."

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:IDSP="http://ns.adobe.com/InDesign/soap/">

<SOAP-ENV:Body>
    <IDSP:RunScript>
        <IDSP:runScriptParameters>
            <IDSP:scriptText></IDSP:scriptText>
            <IDSP:scriptLanguage>javascript</IDSP:scriptLanguage>
            <IDSP:scriptFile>c:\examplefiles\test.jsx</IDSP:scriptFile>
            <IDSP:scriptArgs>
                <IDSP:name>arg0</IDSP:name>
                <IDSP:value>88</IDSP:value>
            </IDSP:scriptArgs>
            <IDSP:scriptArgs>
                <IDSP:name>arg1</IDSP:name>
                <IDSP:value>some text</IDSP:value>
            </IDSP:scriptArgs>
        </IDSP:runScriptParameters>
    </IDSP:RunScript>
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

The InDesign Server SOAP response envelope

The response envelope contains the result of the SOAP call. This response can be the return value of the script passed to InDesign Server, the value of an error caused by the script passed to InDesign Server, or the value of a SOAP fault.

If the result is a SOAP fault, the Body element contains one `Fault` element, which contains `faultcode` and `faultstring` elements.

If there was no SOAP fault, the Body of the response envelope contains one element, `RunScriptResponse`. The `RunScriptResponse` element contains the following elements:

- *`errorNumber`* — The value of the error. If no error occurred, the value is 0.
- *`errorString`* — A string containing the error message returned from InDesign Server. This element is present only if errorNumber is not 0.

- *`scriptResult`* — The data returned from the script run by InDesign Server. In JavaScript and AppleScript, the script's return value is the last value encountered in the script. In VBScript, you set a variable named `returnValue` to the return value for the script. Each of the following examples returns the name of the document at the first index.

  JavaScript:
  ```
  var documentName = app.documents.item(0).name;
  documentName;
  ```

  AppleScript:
  ```
  tell application "InDesignServer"
      set documentName to name of document 1
  end tell
  documentName
  ```

  VBScript:
  ```
  Set myApp = CreateObject("InDesignServer.Application.CS5")
  documentName = myApp.Documents.Item(1).Name
  returnValue = documentName
  ```

The following are simple examples of each of these types of response envelopes.

Example The following is an example of a `RunScriptResponse` that contains a return value. In this case, the JavaScript passed to InDesign Server returned an array containing these elements: `"1"`, `"2"`, `10`, `12`.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:IDSP="http://ns.adobe.com/InDesign/soap/">

<SOAP-ENV:Body>
    <IDSP:RunScriptResponse>
        <errorNumber>0</errorNumber>
        <scriptResult>
            <data xsi:type="IDSP:List">
                <item><data xsi:type="xsd:string">1</data></item>
                <item><data xsi:type="xsd:string">2</data></item>
                <item><data xsi:type="xsd:long">10</data></item>
                <item><data xsi:type="xsd:long">12</data></item>
            </data>
        </scriptResult>
    </IDSP:RunScriptResponse>
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Example The following is an example of a `RunScriptResponse` containing an error (25) caused by a parse error ("Expected: ;") in the JavaScript:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:IDSP="http://ns.adobe.com/InDesign/soap/">
<SOAP-ENV:Body>
    <IDSP:RunScriptResponse>
        <errorNumber>25</errorNumber>
        <errorString>Expected: ;</errorString>
        <scriptResult></scriptResult>
    </IDSP:RunScriptResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example The following is an example of a `RunScriptResponse` containing a SOAP fault caused by a file-not-found error:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:IDSP="http://ns.adobe.com/InDesign/soap/">
<SOAP-ENV:Body>
    <SOAP-ENV:Fault>
        <faultcode>SOAP-ENV:Server</faultcode>
        <faultstring>The script file specified can not be found</faultstring>
        <detail>None</detail>
    </SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Message serialization and deserialization

Most Web Service toolkits provide message serialization and deserialization. This allows you to represent your request and response messages using objects that are native to the programming language, rather than XML strings.

For example, the `RunScript` method accepts a parameter of type `RunScriptParameters`. To call `RunScript`, you first set up the data required to initialize the `RunScriptParameters` structure. The way you set up this data depends on the programming language and toolkit you use. In general, though, you set up `RunScriptParmeters` as an object that is native to the language, like an array, structure, or class. When a call to `RunScript` is made, the toolkit serializes `RunScriptParameters` and uses the resulting XML string to create the request message.

Likewise, the `RunScriptResponse` result of the `RunScript` method is returned to you as an object that is native to the programming language, not an XML string. The toolkit takes care of the deserialization from a response message XML string to an object.

SOAP client implementations

The InDesign Server installation includes a sample SOAP client application named `sampleclient`. This command-line application allows you to easily send a script to InDesign Server via SOAP, and receive the result returned from the script. `sampleclient` is written in C++, and the source code is included in the InDesign products SDK.

The InDesign Server SDK contains samples demonstrating how to create a similar sample client using a variety of technologies, including Java, C#.NET, ASP.NET, PHP, and Flex.

The following sections contain brief descriptions of the technology behind the sample clients. For more detailed information, see the samples in the SDKs and *Adobe InDesign CS5 Server SDK Samples*, located in the *IDS_SDK*`/docs/guides` folder.

Java

The Java sample is a command-line application that employs the Apache Axis Web Service framework. The Axis framework provides the `wsdl2java` tool, used to generate source code based on a WSDL. The generated code is then packaged into a `.jar` file for use by the client application. The client application can be built using Eclipse or Ant, on Windows® or Mac OS®.

For more information on Apache Axis, go to http://ws.apache.org/axis/.

The sample project is located at *IDS_SDK*`/samples/sampleclient-java-soap`.

C++

This sample is located in the InDesign Products SDK. The C++ sample is a command-line application that employs the gSOAP Web Service framework. The framework provides the `wsdl2h` and `soapcpp2` tools to generate source code based on a WSDL. The generated code is then included in the client application's project file. The project is built using Visual Studio on Windows and XCode on Mac OS.

For more information on gSOAP, go to http://gsoap2.sourceforge.net/.

The project is located at:

Windows: *SDK*`\build\win\prj\SampleClient.vcproj`

Mac OS: *SDK*`\build\mac\prj\SampleClient.xcodeproj`

Instructions on how to use the sample are in *Introduction to Adobe InDesign CS5 Server*, located at *IDS_SDK*`/docs/guides/intro-to-indesign-server.pdf`.

C# .NET

The C# sample is a command-line application that employs the .NET framework, which is a Windows-only technology. The .NET framework provides Web Reference technology to generate a proxy class representing the WSDL. This proxy class is instantiated by the client, allowing access to the types and methods within the WSDL. The client application is built using Visual Studio 2008.

For more information on C# .NET, go to http://msdn.microsoft.com/en-us/vcsharp/default.aspx.

The project is located at *IDS_SDK*`/samples/sampleclient-csharp-soap`.

ASP.NET

ASP.NET is a server-side scripting technology requiring IIS (Internet Information Services) Web server. An ASP.NET project is developed and hosted using only Windows, but the client Web page can be accessed from any platform. The .NET framework provides Web Reference technology to generate a proxy class representing the WSDL. This generated file is added to the client Web page project by the Web Reference. The sample client is built using Visual Studio .NET and accessed from a Web browser.

For more information on ASP.NET, go to http://www.asp.net/.

For more information on IIS, go to http://www.microsoft.com/WindowsServer2003/iis/default.mspx.

The project is located at *IDS_SDK*/samples/sampleclient-aspnet-soap.

PHP

PHP is a server-side scripting technology requiring a Web Server (IIS or Apache). PHP can be written using any text editor, and it is cross platform. This sample employs two of the major frameworks use to develop for SOAP in PHP: PHP:SOAP and NuSoap. Each framework provides an API for accessing a WSDL and generating a client object based on the WSDL. The PHP sample client is accessed from a Web browser.

For more information on:	Go to:
PHP	http://www.php.net/
PHP:SOAP	http://www.php.net/soap
NuSOAP	http://www.sourceforge.net/projects/nusoap/
IIS	http://www.microsoft.com/WindowsServer2003/iis/default.mspx
Apache Web Server	http://httpd.apache.org/

The project is located at *IDS_SDK*/samples/sampleclient-php-soap.

Flex

The Flex sample client is an Adobe Flex Builder 3 project. It uses the Flex WebService API (mx.rpc.soap.WebService) to load the InDesign Server WSDL at runtime. The WebService API handles all serialization and deserialization of the SOAP packets that are sent to and received from InDesign Server. This sample is deployed in a browser, and has a simple User Interface allowing you to configure all RunScript parameters.

For more information on Flex and the Flex WebService, go to:

http://www.adobe.com/devnet/flex/

http://livedocs.adobe.com/flex/3/

http://livedocs.adobe.com/flex/3/langref/mx/rpc/soap/mxml/WebService.html

The project is located at *IDS_SDK*/samples/sampleclient-flex-soap.

Other technologies

There are even more languages and frameworks available for developing SOAP clients for InDesign Server, including the following:

- Mac OS X Developer Tools CD — `WSMakeStubs`. Located in the `/Developer/Tools/` folder. Generates stubs based on the WebServicesCore.framework for AppleScript, C++ and Objective C.
- Java™ — IBM Web Services and Sun™ Microsystems Web Services. Contains tools and APIs for use with Java.
- Perl — SOAPLite for Perl. Provides Perl modules for writing SOAP client scripts.

Debugging tips

You may find the following tips helpful when debugging your client:

- Use a packet sniffer to monitor the XML data sent to and from InDesign Server. There are many available; for example, Charles (http://www.xk72.com/charles/).
- When writing PHP code, use print statements to trace information to the Web page.
- Have your script print output to the InDesign Server console. The following are examples of how to write to the InDesign Server console:

JavaScript:
```
app.consoleout("my message");
```

AppleScript:
```
tell application "InDesignServer"
    consoleout message "my message"
end tell
```

VBScript:
```
Set myApp = CreateObject("InDesignServer.Application.CS5")
myApp.ConsoleOut("my message");
```

Frequently asked questions

What can I do in the script that I pass into the sample client?

Basically, anything you can do in a regular InDesign Server script. If you have an InDesign Server plug-in that provides specific features, you need to provide scripting support so you can automate your feature using InDesign Server. For more details on how to add scripting support for your plug-in, see *Making Your Plug-in Scriptable*, a technical note included with the InDesign Products SDK.

When I send a script to InDesign Server using the sample client, I get a message containing an error code. What does this mean?

The error displayed in the sample client comes from InDesign Server after calling the InDesign Server command RunScript. The response usually contains a string associated with the error, giving more details; sometimes, however, the error string is not passed back to the client, but is written to the InDesign Server console.

For more information on InDesign errors, look at the error code listing in the InDesign Products SDK at *SDK*`/docs/references/errorcodes.htm`.

What happens to InDesign Server when a client terminates?

InDesign Server continues to run, waiting for more instructions.

Can a client communicate across multiple instances of InDesign Server?

A client communicates with only one instance of InDesign Server at a time; however, this does not mean that you cannot develop an application that communicates across multiple instances of InDesign Server. Keep in mind that each instance of InDesign Server (distinguished by TCP/IP port number) has its own set of "InDesign Defaults" and "InDesign Saved Data" files. These correspond to the databases represented by kWorkspaceBoss and kSessionBoss, respectively. If your client application works with multiple instances of InDesign Server, and it depends on this data, consider employing a strategy to keep these databases synchronized. For details, see the *Adobe InDesign CS5 Server Plug-in Techniques* technical note.

References

Publications

- "Scriptable Plug-in Fundamentals" in *Adobe InDesign CS5 Products Programming Guide*, Adobe Systems Incorporated
- *Adobe InDesign CS5 Server Plug-in Techniques*, Technical Note #10093, Adobe Systems Incorporated
- *Adobe InDesign CS5 Scripting Guide*, Adobe Systems Incorporated
- *Adobe InDesign Server CS5 Scripting Guide*, Adobe Systems Incorporated
- Apple® Developer Connection, *Web Services,* http://developer.apple.com/mac/library/navigation/index.html
- Cover Pages Technology Reports, Web Services Description Language (WSDL), http://xml.coverpages.org/wsdl.html
- Microsoft® Developer Network, Web Services Developer Center, http://msdn.microsoft.com/webservices/
- Microsoft Developer Network, *Understanding SOAP*, http://msdn2.microsoft.com/en-us/library/ms995800.aspx
- Microsoft Developer Network, *Understanding WSDL*, http://msdn2.microsoft.com/en-us/library/ms996486.aspx
- W3C SOAP technical reports, http://www.w3.org/TR/soap
- W3Schools SOAP Tutorial, http://www.w3schools.com/soap/

Tools

- Apache Axis, http://ws.apache.org/axis/
- gSOAP, http://sourceforge.net/projects/gsoap2/

- IBM Web Services (r)evolution, http://www-106.ibm.com/developerworks/webservices/library/ws-peer4/
- Microsoft Visual Studio, wsdl.exe, http://msdn.microsoft.com/en-us/library/7h3ystb6.aspx
- SOAPLite for Perl, http://www.soaplite.com/
- Sun Microsystems Web Services, http://java.sun.com/webservices/index.jsp