F

WSF and WSC File Format Reference

XML Conformance

Windows Script File (WSF) files encapsulate one or more script programs that can be run with the `wscript` or `cscript` programs. Windows Script Component (WSC) files define scriptable automation (COM) objects written in a Windows Script Host (WSH)-supported language. Both file types are formatted with Extensible Markup Language (XML) tags that define the structure of the program or component.

By default:

- tag and attribute names are case insensitive.
- attribute values need not be enclosed in quotes, if the value contains no spaces.
- the `<script>` element is "opaque" to the parser, so script code inside the `<script>` element can use XML's special characters (`<`, `>`, and `&`) without special consideration.

If the WSF or WSC file begins with the line:

```
<?XML version="1.0" ?>
```

then strict XML syntax compliance is required of the file. When strict compliance is in effect:

- tag and attribute names are case sensitive.
- all attribute values must be enclosed in quotes.
- `<script>` elements are not opaque, so script code must be enclosed in `<![CDATA[ … ]]>` tags. Without the `CDATA` markup, XML special characters in scripts, such as `&`, `<` and `>`, would have to be entered as entities `&amp;`, `&lt;` and `&gt;`, respectively. This applies also to the descriptive text in `<description>`, `<example>`, and `<usage>` elements. If the text contains the special characters, you must either put it inside `<![CDATA[` and `]]>` tags or use the corresponding entities instead of the literal characters.

XML elements with no end tag must use the following format:

```
<tagname [attributes] />
```

The special tag indicator `/>` is *required* to show the element contains no content and no end tag follows. (This applies only to element tags and not to the special `<!>` and `<?>` directive formats.)

Structure of a WSF

The following listing shows the general order of elements in a WSF; however, not all the elements need to be used. Parameters and most end tags are not shown; see the tag syntax listing for each tag's syntax definition. Ellipses (…) indicate elements that can be repeated.

```
<?xml?>
<package>
    <comment>
    <job>…
        <?job?>
        <runtime>
            <description>
            <named>…
            <unnamed>…
            <example>
            <usage>
        </runtime>
        <object>
        <reference>
        <resource>
        <script>…
    </job>…
</package>
```

Structure of a WSC File

The following listing shows the general order of elements in a WSC file; however, not all the elements need to be used. Parameters and most end tags are not shown; see the tag syntax listing for each tag's syntax definition. Ellipses (…) indicate elements that can be repeated.

```
<?xml?>
<package>
    <comment>
    <component>...
        <?component?>
        <public>
            <property><get><put></property>
            <method><parameter>…</method>
            <event>
        </public>
        <registration>
        <object>
        <reference>
        <resource>
        <script>…
    </component>…
</package>
```

Tag Syntax

This section briefly describes the syntax for each tag. For more information, see the reference lists in Appendix G, "Creating Your Own Scriptable Objects" (included with the PDF version of this book and available free on the web at www.helpwin7.com/scripting), Chapter 9, "Deploying Scripts for Computer and Network Management," or online documentation at www.microsoft.com.

You can also read Microsoft's documentation. I recommend getting the downloadable version, as it's easier to read than the online version. To get it, go to www.microsoft.com/downloads and search for "Windows script documentation." The document you want, at the time this was written, is titled "Windows Script 5.6 Documentation." Download and save the file script56.chm. Double-click to open it. It contains reference information for JScript, VBScript, scripting objects, script components, and more.

REFERENCE LIST F.1 WSC Tag Syntax

<?XML Version="1.0" [**standalone="yes"**]**?>**

Requests strict XML conformance. Must be the first line of the file.

<?component error="*value***" debug="***value***" ?>**

Enables error messages and debugging of a component (WSC only).

```
<?job error="value" debug="value" ?>
```
Enables error messages and debugging of a script job (WSF only).

```
<! [CDATA[
    protected text
    ⋮
]]>
```
Encloses the contents of the `<script>` tag when using strict XML conformance. This is not to be used when strict XML conformance is not specified. You can also use this to surround text in the `<description>`, `<example>`, and `<usage>` that is described shortly, if the text contains the characters `<`, `>` or `&`.

```
<!-- any text
    ⋮
-->
```
Indicates comment text that is to be ignored by the parser.

```
<comment>
    any text
    ⋮
</comment>
```
Indicates comment text that is to be ignored by the parser.

```
<component [id="componentID"]>
    component markup: registration, public,
    implements and script elements
    ⋮
</component>
```
Encloses the definition of an object (WSC only).

```
<description>any text</description>
```
Describes the purpose of the script for `ShowUsage` (WSC only).

```
<event name="name" [dispid="id"]/>
```
Declares an event the component might fire.

```
<example>example text</example>
```
Lists a sample command line (WSF only).

```
<get [name="functionname"]/>
```
Indicates that a property is readable and optionally specifies the name of the function that returns the property value.

```
<job [id="jobid"]>
    job content: <?job?>, <script>, other elements
    ⋮
</job>
```
Encapsulates one self-contained script program (WSF only). A WSF file might contain a single job or multiple jobs enclosed in a `<package>` element. The job or jobs to be executed can be specified on the command line when the WSF file is run using a command line of the form `cscript somescript.wsf //job:jobid`.

```
<method name="methodname" [internalName="functionname"] [dispid="dispID"]>
    [<parameter name="paramname"/>…]
</method>
```

Declares a method and associated function (WSC only). If no parameters are defined, the `method` tag can be closed with `/>` and the `</method>` end tag omitted.

```
<named name="argname" helpstring="description" type="argtype" required="boolean"/>
```

Defines a named argument's name and type (WSF only).

```
<object id="name" {classid="clsid:GUID" | progid="progid"}
[events="{true|false}"]/>
```

Creates an object variable with global scope.

```
<package>
    one or more job or component elements
    ⋮
</package>
```

Encloses one or more jobs or components in a WSF or WSC file, respectively.

```
<property name="propertyname" [internalname="varname"]/>
```

or

```
<property name="propertyname" [get] [put]>
    [<get [internalName="functionname"]/>]
    [<put [internalName="subroutinename"]/>]
</property>
```

Declares a read/write property and an associated variable by the first form. The second form declares a property and an associated function or functions (WSC only).

```
<public>
    Interface definition: property, method, event elements
</public>
```

Encloses the definition of an object's interface (WSC only).

```
<put [name="subroutinename"]/>
```

Indicates a property is writeable (WSC only).

```
<reference {object="progid" | guid="GUID"}
    [version="version"]/>
```

Imports a type library and imports the library's constants.

```
<registration progid="progid" [classid="GUID"]
     [description="text"] [version="number"]
     [remotable="boolean"]>
  [<script>
      optional registration and unregistration script
  </script>]
</registration>
```

Sets object registration information and procedure (WSC only). If no associated `script` is required, the `registration` tag can be closed with `/>` and the end tag omitted.

```
<resource id="resourceid">
    text or number
</resource>
```

Defines a named resource value.

```
<runtime>
    argument definitions: named, unnamed, example elements
    ⋮
</runtime>
```

Encloses argument syntax declarations (WSF only).

```
<script [language="name"]>
<![CDATA[
    script code
    ⋮
]]>
</script>
```

Encloses script code in the named language (for example, VBScript or JScript). The `<![CDATA[` and `]]>` tags are used if, and only if, strict XML conformance is in effect.

```
<script language="name" src="location">
```

Imports an external file containing script procedures. The file's location can be specified as filename with a drive and full path; a UNC-formatted shared file name; or a URL. The file cannot contain any XML markup.

```
<unnamed name="argname" helpstring="description"
    many="boolean" required="{boolean|number}"/>
```

Defines an unnamed command-line parameter (WSF only). The `named` and `unnamed` specifications are used by the `ShowUsage` method to construct a syntax-definition string.

```
<usage>
    descriptive text
</usage>
```

Provides syntax description for the `ShowUsage` method and overrides the default self-generated text. Be careful with the content in these text sections. The `&`, `<` and `>` characters have special meaning to XML, and if you want them to appear as literal characters in your text, you must use the entities **`&`**, **`>`**, and **`<`**, respectively.