

UltraXML:Server

- **Scaleable blackbox server solution**
- **Fully automated and distributed production**
- **Runs on Windows 2000 and XP**
- **Single or multiple composition engines**
- **Allows distributed production**
- **Schedules, prioritises and monitors work flow**
- **Runs as a Windows service**

Introduction

UltraXML:Server is designed to run as a Windows service. It uses the standard SOAP protocol to listen for commands. This allows clients to be multi platform, sending XML SOAP messages over HTTP to the Windows Service.

Any number of instances of UltraXML Server can be installed on any number of machines located anywhere which can work in parallel, processing the jobs sent to it by the client application. Each running server instance will batch queue any number of jobs that any number of concurrent clients send it. Each job can be given a priority in by the client application. Jobs with lower priority numbers will jump any job queues on that server instance.

Server Installation

UltraXML:Server requires one dongle key per machine. You can install and run as many instances of UltraXML:Server as required on that machine. Multiple instances can be useful on a single processor server machine to help spread the processing load equally between concurrent jobs. When only one instance is used then client jobs will be batch queued and processed in FIFO (first in first out) order, unless a job is prioritized by the client whereupon it can jump the queue if other jobs are queued before it is processed.

The installation can be performed on multiple server machines (by purchasing additional dongle keys) if it is expected that the job load will be high. This enables your solution to be fully scaleable, with each machine processing different jobs. Since client/server communication is done via SOAP, the servers can be located anywhere on an Intranet or Internet connection.

When solutions use multiple instances of the server, it is the responsibility of the client software to decide which server should process the job. This can be configured in many ways. For example, the client can hold an array of server locations and port numbers and send the next job to each one in turn, thereby spreading the load equally. Alternatively one server instance can be assigned to certain users of the client software or a combination of exclusive user servers and equal load spreading for other users could be setup.

After installing the software package and inserting the server dongle, the Windows Services must be configured.

To install a Windows Service instance of UltraXML:Server start a DOS command prompt and navigate to the installation directory:

Eg.

```
cd \program files\webx\ultraxml server
```

Then install the service as follows:

```
uxmlserver install 1 8000
```

This will then install an UltraXML Server Service with label 1 which will listen on localhost port 8000

You could then install a second instance of the server to listen on a different port (eg. 8001) as follows:

```
uxmlserver install 2 8001
```

Labels can also be names (but no spaces) eg. to label the server dedicated to run on port 8002

```
uxmlserver install dedicated 8002
```

You should not install 2 instances of the server on the same machine with the same port number. Each instance must be given a unique port number for it to work correctly.

To remove a service, just use the remove parameter and specify the label, eg.

```
uxmlserver remove 1
```

Or

```
uxmlserver remove 2
```

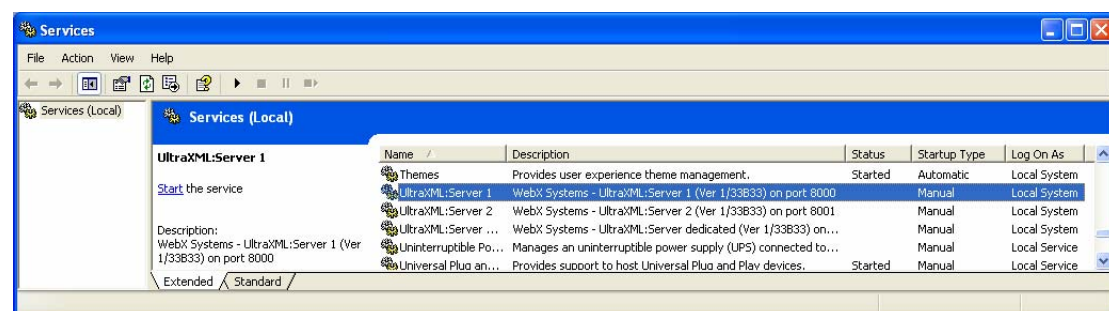
Or

```
uxmlserver remove dedicated
```

Note that for the server to operate the UltraXML:Server WIBU hardware dongle key must be installed on the machine at all times.

To see your installed UltraXML web services, go to the Windows 2000/XP control panel and select Administrative Tools -> Services and scroll down to the UltraXML:Server entries.

You should see something like this:



Note that none of the installed services have been started. You need to click on [Start](#) to start each one manually. If you go to the properties dialog of each service then you set the startup type to be automatic, manual or disabled. Automatic means that the service will start automatically when the system is booted. You can also set recovery options in the property dialog. Each service can be manually stopped or restarted at any time.

Client Software

The client software can be written in any language and platform that can support the SOAP standard.

The UltraXML Web Service Description Language file (WSDL) can be found here: www.webxsystems.com/UltraXML.wsd

Many client programming systems (such as Microsoft Visual Studio dot net) are capable of reading a WSDL file and automatically producing the required functions.

For example, in Visual Studio 2003 Dot Net:

Go to Project->Add Web Reference and enter the URL to the UltraXML WSDL file www.webxsystems.com/UltraXML.wsd This will instantly generate a new class for you which you can then use to develop your client software, which may look something like this:

```
WebReference1.UltraXML uxml = new WebReference1.UltraXML();
uxml.Url = "http://localhost:8000/";
int jobid = uxml.Open("c:/test.uxd",1);
if (jobid > 0) {
    if (uxml.Set(jobid,"xml1","c:/test.xml") == 0) {
        uxml.Publish(jobid,"c:/test.ps",1,2,"",false,false,"PScript");
    }
    uxml.Close(jobid);
}
```

The above will switch to the installed UltraXML server located at localhost port 8000, open the UltraXML document at c:/test.uxd. It then sets the contents of the UltraXML tag, 'xml1' to the content of the XML located at c:/test.xml. It then publishes (prints) pages 1 to 2 using the Postscript driver (indicated by "PScript") to c:/test.ps.

Note that the file locations could have been Internet URLs, allowing UXD and XML files to be picked up from any remote system and the Postscript file created on any remote system.

SOAP Methods

The following is a list of all available SOAP methods (defined by the WSDL)

```
/*
UltraXML Web Service
*/
```

METHODS

```
/*
Opens an UltraXML UXD template ready for XML input and/or publishing. Assigns
```

a job id. The job is queued in the assigned formatting engine. Returns job id immediately.

**/*

[rpc/encoded]

int Open

(
 string url,
 int priority
)

*/**

Requests that the specified job id is closed, which in turn will close any open UltraXML templates. Returns immediately. Job id is made available for reuse

**/*

[rpc/encoded]

int Close(int id)

*/**

Queues printing of the current job id to the specified location in the specified format. Any queued macros or commands are executed first. Returns immediately

**/*

[rpc/encoded]

int Publish

(
 int id,
 string url,
 int start_page,
 int end_page,
 string print_ctrl,
 boolean cutting_guides,
 boolean color_guides,
 string format // PScript, EPSF, Windows or TIFF
)

*/**

Queues an UltraXML macro to the specified job id. Returns immediately

**/*

[rpc/encoded]

int Macro

(
 int id,
 string macro
)

*/**

Queues the setting of the UltraXML XML tag to the contents of the specified URL for a job id. Returns immediately./*

[rpc/encoded]

int Set

(
 int id,

```

    string tag,
    string url
)

/*
Closes any current open job, even if it was created by a different client. Returns
response after closing.
*/

[rpc/encoded]
int CloseCurrent
(
    void
)

/*
Closes the current job, removes all pending jobs and resets (does not restart the
service). Returns response code when done.
*/

[rpc/encoded]
int Reset
(
    void
)

/*
Returns the number of pending jobs including any currently open one.
*/

[rpc/encoded]
int GetJobCount
(
    void
)

/*
Restarts the service. (Effectively Reboots the UltraXML Server). Returns response
before attempting restart – NOTE: You must STOP the service select 'Allow
service to interact with desktop' and START the service in order for this function
to work. You should also allow time for the service to restart before calling further
methods.
*/

[rpc/encoded]
int RestartService
(
    void
)

```

NOTES:

Note, you can queue any number of UltraXML macros for a job (using the Macro() method). You can also Set() any number of times and Publish() any number of times for a current open template (useful for multiple jobs where the template is large or contains many images), but you MUST Close() before the next job queued on that server can be done.

Each method will return an integer value. This will be 0 if there were no errors, otherwise it will return an error code. The only exception to this is the `Open()` method which will return a job id or 0 if no job id could be assigned and the `GetJobCount()` method which returns a count.