



***Microtech's API***  
(Application Programming Interface)  
*User's Manual*

Revision 2

***Microtech Systems, Inc.***  
2 Davis Drive  
Belmont, CA 94002 USA

***Phone*** 650/596-1900  
***Fax*** 650/596-1915  
***Web*** [microtech.com](http://microtech.com)  
***email*** [info@microtech.com](mailto:info@microtech.com)

# CONTENTS

<b>Introduction</b> .....	<b>1</b>
<b>Overview</b> .....	<b>2</b>
<b>Objects</b> .....	<b>3</b>
<b>Single Server Interface</b> .....	<b>3</b>
<b>Multiple Server Interface</b> .....	<b>3</b>
<b>Listeners</b> .....	<b>4</b>
System (Searcher) Listener .....	4
Individual Server Listener.....	4
Job Listener.....	5
<b>Job Object</b> .....	<b>5</b>
<b>XML Communication</b> .....	<b>6</b>
<b>Sample API XML Strings</b> .....	<b>8</b>
<b>Server Requests</b> .....	<b>8</b>
Server Status .....	8
Server Settings .....	8
Printer Status .....	9
Drive Status.....	9
Input Status .....	9
Job Status.....	10
<b>Job Request</b> .....	<b>10</b>
Image Job.....	11
Audio Job.....	11
Premaster Job .....	11
<b>Job Update</b> .....	<b>12</b>
<b>Event Updates</b> .....	<b>12</b>
Pending Job Added.....	12
Pending Job Removed .....	12
Active Job Added .....	12
Active Job Cancelled .....	13
<b>Code Examples</b> .....	<b>14</b>
<b>C++</b> .....	<b>14</b>
Server and Job Requests.....	14
Multi-Server Discovery .....	16
<b>Java</b> .....	<b>17</b>
Server and Job Requests.....	17
Multi-Server Discovery .....	19
<b>C# .NET</b> .....	<b>20</b>
Server and Job Requests.....	20
Multi-Server Discovery .....	22
<b>VB.NET</b> .....	<b>23</b>
Server and Job Requests.....	23

Multi-Server Discovery .....	24
<b>Rimage (C++) .....</b>	<b>26</b>
<b><i>Building Your Application</i> .....</b>	<b>29</b>
<b>C++ .....</b>	<b>29</b>
<b>Java.....</b>	<b>30</b>
<b>C# .NET .....</b>	<b>31</b>
<b>VB.NET .....</b>	<b>32</b>
<b>Rimage .....</b>	<b>33</b>
<b><i>CD-Remote Emulator</i> .....</b>	<b>34</b>
<b>Configuration .....</b>	<b>34</b>
<b>Usage.....</b>	<b>35</b>
<b><i>What's Next?</i> .....</b>	<b>36</b>

# Introduction

This manual is intended for programmers who have a requirement to interact with Microtech's production system from their proprietary software. It is assumed that the reader is familiar with at least one of the programming languages that are supported by the Microtech API (Application Programming Interface) and has read the Microtech User's Guide or is familiar with the capabilities of the system. At this time the API includes support for C/C++, Java, Microsoft .Net (C#) and Microsoft Visual Basic.

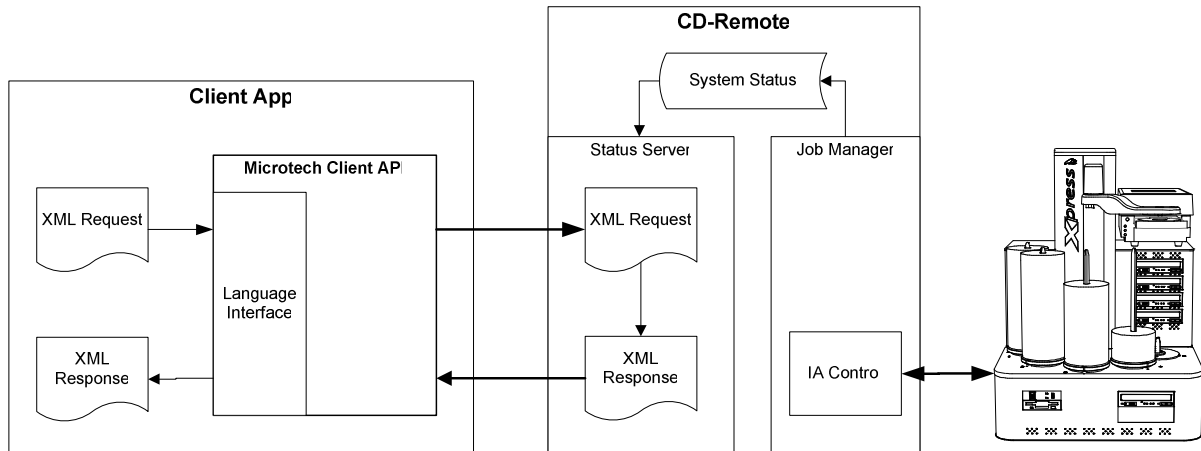
The Microtech API was designed to allow a software interface between a client application and the Microtech production system thus providing a means to create tightly integrated customized solutions that include disc creation and printing.

Before introducing the programmatic interface to the API, it should be noted that there is a text-file based job submittal interface to the Microtech production system. This is achieved by building .JOB text files and placing them into the job directory of the production system server. The .JOB file contains all the parameters for the job submission and can be created by an executable script file or manually in a text editor. The job directory is automatically scanned and any new .JOB files encountered will be processed by the system. The format and description for this text-base method of job submission is covered in depth in the ImageMaker Technical Reference User's Manual.

If server and/or job status and feedback are required, however, the API described in this manual should be used.

The chapters prior to the code sample sections are intended to be programming language independent. The objects and calls have common names regardless of language used and will be referenced without regard to specific language syntax. It is recommended that the sample code of the language of interest be looked at while reading this manual.

# Overview



The purpose of the Microtech API is to enable a user application to interact with the Microtech production system.

The interaction is achieved by interfacing with the CD-Remote program that is running on the Microtech server. The API consists of library routines that are linked into the client application. These routines perform all the networked communication between the client application and CD-Remote server.

A single application may communicate sequentially with multiple CD-Remote servers, or communicate with multiple servers simultaneously.

The client application first “*connects*” to a CD-Remote server. Then other actions may be performed such as server “*status*” and “*settings*” requests to retrieve server parameters, “*job status*” requests to retrieve job parameters, and “*job submittal*” requests which enter jobs into the system for processing. To receive “*real-time*” status from the server or from a specific job, the API allows the client application to attach a server “*event listener*” and a “*job listener*”. Finally, the client application can “*disconnect*” from the server.

The next section contains some of these details.

# Objects

The objects discussed here are a subset of the API. The entire API with syntax and calls are explained in the documentation that is distributed with the API, and will not be duplicated here. However, as an overview, here are some of the key components.

## ***Single Server Interface***

The starting point and main API object used for communicating with a single CD-Remote server is the **MicrotechCDRemoteManager**. This object is needed to connect to and disconnect from the CD-Remote server, request server and job status, submit jobs, set up log files, and attach event and job listeners.

There are a few calls that will allow the application to directly query the server (see the documentation for **MicrotechCDRemoteManager**). However, most of the information returned by the server is in response to requests sent by the client application, (see the section on XML Communication).

**MicrotechCDRemoteManager** Main single server API object, see documentation for (subset):

- connectTo() / disconnect()
- request()
- submitJob()
- logToFile()
- listenForEvents() / stopListeningForEvents()
- listenForJob() / stopListening()

This object can only connect to one server at a time. To connect to another server, first disconnect from the current server.

## ***Multiple Server Interface***

To communicate with multiple servers simultaneously, the API object to use is the **MicrotechSearcherManager**. This object is needed to find and manage all the existing CD-Remote servers on the network, attached server listeners, and return the **MicrotechProcessingServer** objects that are used connect, query, etc. each individual server.

**MicrotechSearcherManager** Main multi-server API object, see documentation for (subset):

- Search()
- NewServer() / RemoveServer()
- Find() / ContainsServer()
- ConnectAll() / DisconnectAll()
- ListenForEvents() / StopListeningForEvents()  
{system level}

The **MicrotechProcessingServer** object is used similarly to the **MicrotechCDRemoteManager** and is used to interact with individual servers. This object is returned by **MicrotechSearcherManager** before use and may not be instantiated directly.

## **Listeners**

To get information about events in the system, a server, or a specific job as they occur, you must attach “*listeners*” that you create from the appropriate interface.

### **System (Searcher) Listener**

Use the **MicrotechSearcherManager** **ListenForEvents()** to attach a system listener. This will cause “call back” methods to be executed whenever certain system events occur as described in the documentation for the interface **MicrotechSearcherListener**.

**MicrotechSearcherListener** System event listener, see documentation for:

- onServerAdded()
- onServerRemoved()
- onServerConnected()
- onServerDisconnected()

### **Individual Server Listener**

Use the **MicrotechCDRemoteManager** (single server application) or **MicrotechProcessingServer** (multi-server application) **listenForEvents()** to attach the server event listener. This will cause “*call back*” methods to be executed whenever certain server events occur as described in the documentation for the interface **MicrotechEventListener**.

**MicrotechEventListener** Server event listener, see documentation for (subset):

- onActiveJobAdded()
- onPendingJobAdded()
- onError()
- onPause()
- onQuit()

## Job Listener

Use the `MicrotechCDRemoteManager` (single server application) or `MicrotechProcessingServer` (multi-server application) `listenForJob()` to attach a listener to a specific job. Each job should have its own listener. This will cause a “*call back*” method to occur about every two seconds until the job has completed. This is described in the documentation for the interface `MicrotechJobListener`. It is recommended that the listener process its `onJobUpdate()` method quickly so that the next “*call back*” is not missed.

**MicrotechJobListener** Job listener, see documentation for:

- `onJobUpdate()`

## Job Object

The final object that will be discussed in this section is the “*job*” object. This object is returned when the job is submitted, and can be used to query the job parameters and for attaching a job listener. This object can also be created and given a name of an existing job in order to attach a listener.

**MicrotechJob** Job object, see documentation for (subset):

- `getJobFile()`
- `getJobName() / setJobName()`
- `getJobType()`
- `getQuantity()`



## XML Communication

All communication between the API and the CD-Remote server is implemented by sending and receiving XML strings. The XML strings must be created to send requests to the server, and the returned response XML strings must be parsed by the application. If future enhancements are made to the XML content, no modification of the API will be required. See the following section for XML string examples.

Each XML string must comply with its corresponding DTD (Document Type Definition).

**Note:** If the reader is not familiar with the XML and DTD formats, spending some time with a tutorial is strongly recommended. Try the links:

<http://www.w3schools.com/xml/default.asp>

<http://www.w3schools.com/dtd/default.asp>

Or perform a search for “*xml tutorial*” and “*dtd tutorial*”.

The DTD files distributed with the API must be placed in a known directory on the system running the client application. The API uses the DTD for XML validation (when submitting a job, for example). There are five DTD formats that are used in the API. **Note:** In each description below, the object `MicrotechProcessingServer` may be substituted for `MicrotechCDRemoteManager` for multi-server applications.

**Server request:** Server request XML strings must be created by the application and sent to the server. This will be used to request status from the server. The API call used to send a request to the server is **`MicrotechCDRemoteManager.request()`**. The format of this XML can be found in the file **`MicrotechRequest.dtd`**.

**Server command:** Server command XML strings must be created by the application and sent to the server. This will be used to send server commands such as “*pause*”, “*resume*”, and “*cancel job(s)*”. The API call used to send a request to the server is the same as the for the server request: **`MicrotechCDRemoteManager.request()`**. The format of this XML can be found in the file **`MicrotechCommand.dtd`**.

**Job request:** A job request XML string must be created by the application and sent to the server to submit a job. The call used to submit the job request is **`MicrotechCDRemoteManager.submitJob()`**. The format of this XML can be found in the file **`MicrotechJobRequest.dtd`**. **Note:** The DTD path must be specified in the job request XML (see example job request XML in the next section).

*Server response:* The XML response sent back from the server will conform to the format in the file **MicrotechResponse.dtd**. This includes every server response except those received from the event listener.

*Server event:* The XML event response is sent back from the server via a call back routine whenever there is an event listener attached to the server. The format of this XML can be found in the file **MicrotechEvent.dtd**.

## Sample API XML Strings

As previously mentioned, all communication between the API and the CD-Remote server are implemented by sending and receiving XML strings. The XML strings must be created to send requests to the server, and the returned response XML strings must be parsed by the application. Each XML string must comply with its corresponding DTD (Document Type Definition). All the examples below contain line feeds and tabs for readability; these are not necessary for validation purposes.

There are several existing XML parsers that can be found on the web depending on the programming language used. Any of these may be used, but will not be discussed here. **Note:** If using C++, see the section on building your application later in this manual.

### Server Requests

The following information requests are sent to the server using the API call “MicrotechCDRemoteManager.request()” or “MicrotechProcessingServer.request()”. All these requests conform to the MicrotechRequest.dtd DTD. The responses that are returned by the server conform to the MicrotechResponse.dtd DTD.

### Server Status

The *GetServerStatus* request is sent to the server to get status information.

```
<?xml version='1.0'?>
  <MicrotechRequest>
    <GetServerStatus/>
  </MicrotechRequest>
```

An example response for this request would be:

```
<MicrotechResponse>
  <ServerStatus ServerId="CDRemote" ServerName="LAB2-DEV" ServerType="Xpress XL"
  SoftwareVersion="3.06.10" SoftwareRelease="3.11" ListeningPort="3519" Status="Running"
  OutOfMedia="false"/>
</MicrotechResponse>
```

### Server Settings

The *GetServerSettings* request is sent to the server to get the server settings information.

```
<?xml version='1.0'?>
  <MicrotechRequest>
    <GetServerSettings />
  </MicrotechRequest>
```

An example response for this request would be:

```
<MicrotechResponse>
  <ServerSettings EmailNotificationEnabled="false" PremasteringEnabled="true"
  NetworkingEnabled="true" MP3ConversionEnabled="true" BTWEnabled="false"
  SonicEngineEnabled="false">
    <DirectorySettings JobDirectory="\\LAB2-XP1\Drive-D\Jobs" ImageDirectory="\\LAB2-
  XP1\Drive-D\Images" LayoutDirectory="\\LAB2-XP1\Drive-D\Layouts" ResultDirectory="\\LAB2-
  XP1\Drive-D\Jobs">
      <TemporaryImageDirectory Path="D:\Images"/>
      <TemporaryImageDirectory Path="D:\TmpImgs"/>
    </DirectorySettings>
  </ServerSettings>
```

```

    </DirectorySettings>
    <DiscoveryServer Enabled="true" Port="3518"/>
  </ServerSettings>
</MicrotechResponse>

```

## Printer Status

The *GetPrinterStatus* request is sent to the server to get the printers' information.

```

<?xml version='1.0'?>
  <MicrotechRequest>
    <GetPrinterStatus />
  </MicrotechRequest>

```

An example response for this request would be:

```

<MicrotechResponse>
  <PrinterStatus Configured="2" Active="2">
    <Printer Name="Rimage Printer R" Number="1" Location="Right" Type="Thermal"
DiscsDone="0" Active="true" State="OK">
      <InkLevels Cyan="-1" Magenta="-1" Yellow="-1" Black="-1"/>
    </Printer>
    <Printer Name="Rimage Printer L" Number="2" Location="Left" Type="Thermal"
DiscsDone="0" Active="true" State="OK">
      <InkLevels Cyan="-1" Magenta="-1" Yellow="-1" Black="48"/>
    </Printer>
  </PrinterStatus>
</MicrotechResponse>

```

## Drive Status

The *GetDriveStatus* request is sent to the server to get the output drives' information.

```

<?xml version='1.0'?>
  <MicrotechRequest>
    <GetDriveStatus />
  </MicrotechRequest>

```

An example response for this request would be:

```

<MicrotechResponse>
  <DriveStatus TotalDrives="6" Active="6">
    <Drive Number="0" Id="D1" Type="DVD-ROM" Enabled="true" CompletedDiscs="0"
FailedDiscs="0" MaxConsecutiveFailedDiscs="0"/>
    <Drive Number="1" Id="D2" Type="CD/DVD" Enabled="true" CompletedDiscs="0"
FailedDiscs="0" MaxConsecutiveFailedDiscs="0"/>
    <Drive Number="2" Id="D3" Type="CD/DVD" Enabled="true" CompletedDiscs="0"
FailedDiscs="0" MaxConsecutiveFailedDiscs="0"/>
    <Drive Number="3" Id="D4" Type="CD/DVD" Enabled="true" CompletedDiscs="0"
FailedDiscs="0" MaxConsecutiveFailedDiscs="0"/>
    <Drive Number="4" Id="D5" Type="CD/DVD" Enabled="true" CompletedDiscs="0"
FailedDiscs="0" MaxConsecutiveFailedDiscs="0"/>
    <Drive Number="5" Id="D6" Type="CD/DVD" Enabled="true" CompletedDiscs="0"
FailedDiscs="0" MaxConsecutiveFailedDiscs="0"/>
  </DriveStatus>
</MicrotechResponse>

```

## Input Status

The *GetInputStatus* request is sent to the server to get the input drives' information.

```

<?xml version='1.0'?>
  <MicrotechRequest>

```

```
<GetInputStatus />
</MicrotechRequest>
```

An example response for this request would be:

```
<MicrotechResponse>
  <InputStatus Active="2">
    <Input Number="1" Station="1" Status="Active" Type="CD" MediaSize="Standard"/>
    <Input Number="3" Station="2" Status="Active" Type="DVD" MediaSize="Standard"/>
  </InputStatus>
</MicrotechResponse>
```

## Job Status

The *GetJobList* request is sent to the server to get the status of all jobs on the server.

```
<?xml version='1.0'?>
  <MicrotechRequest>
    <GetJobList />
  </MicrotechRequest>
```

A shortened example response for this request would be:

```
<MicrotechResponse>
  <JobList>
    <Job Name="Administ141007" DiscName="MyJob.iso" Requested="1" Completed="0"
Failed="0" State="PENDING" Priority="4" TotalPercentComplete="0" DateTime="01/16/07 13:42"/>
    <Job Name=" Administ105416" DiscName="MyJobName" Requested="10" Completed="4"
Failed="0" State="PROCESSING" Priority="5" TotalPercentComplete="44" DateTime="01/16/07
13:47"/>
    <Job Name="Administ154127" DiscName="YourJobName" Requested="5" Completed="5"
Failed="0" State="COMPLETE" TotalPercentComplete="100" DateTime="01/16/07 13:26"/>
    <Job Name="Administ291913" DiscName="5OMB.iso" Requested="2" Completed="2"
Failed="0" State="COMPLETE" TotalPercentComplete="100" DateTime="01/16/07 13:17"/>
  </JobList>
</MicrotechResponse>
```

To request the status of a single job(s):

```
<?xml version="1.0"?>
<!DOCTYPE MicrotechRequest SYSTEM "C:\Icdr-MJ\XML\MicrotechRequest_1.1.dtd">
<MicrotechRequest>
  <GetJobStatus>
    <GetSingleJobStatus Name="Administ154126" />
    <GetSingleJobStatus Name="Administ154127" />
  </GetJobStatus>
</MicrotechRequest>
```

And the response might be:

```
<MicrotechResponse>
  <JobStatus>
    <MissingJob Name="Administ154126"/>
    <Job Name="Administ154127" DiscName="YourJobName" Requested="5" Completed="5"
Failed="0" State="COMPLETE" Priority="5" TotalPercentComplete="100" DateTime="01/16/07
13:26"/>
  </JobStatus>
</MicrotechResponse>
```

## Job Request

The following job requests are sent to the server using the API call “MicrotechCDRemoteManager.submitJob()” or “MicrotechProcessingServer.submitJob()” to

submit a job to the server. These requests conform to the MicrotechJobRequest.dtd DTD.  
**Note:** The DTD path must be present in the XML. The specified DTD file path should be the path on the server where the DTD files reside.

## Image Job

A simple image job request might be:

```
<?xml version="1.0"?>
<!DOCTYPE MicrotechJobRequest SYSTEM "C:\Icdr-Mj\XML\MicrotechJobRequest_1.0.dtd">
<MicrotechJobRequest>
  <Options Quantity="10">
    <GeneralOptions ImageName="MyJobName" MediaType="CD" IncludeFailures="TRUE"
Priority="4"/>
  </Options>
  <ImageJob>
    <ImageFile Path="\\LAB2-XL-1\Drive-D\Images\MyJob.iso" />
  </ImageJob>
</MicrotechJobRequest>
```

## Audio Job

An example audio disc job might be:

```
<?xml version="1.0"?>
<!DOCTYPE MicrotechJobRequest SYSTEM "C:\Icdr-Mj\XML\MicrotechJobRequest_1.0.dtd">
<MicrotechJobRequest>
  <Options JobName="SampleAudioJob" Quantity="5">
    <GeneralOptions IncludeFailures="TRUE" Copy="TRUE" LeaveBsyFiles="TRUE"/>
  </Options>
  <LayoutFile LayoutFilePath="D:\Layouts\SampleLay.lay" />
  <AudioDiscJob>
    <AudioFile FullPath="C:\WINDOWS\Media\Chimes.wav" Pause="150"/>
    <AudioFile FullPath="C:\WINDOWS\Media\Chord.wav" Pause="250"/>
    <AudioFile FullPath="C:\WINDOWS\Media\Ding.wav" Pause="200"/>
  </AudioDiscJob>
</MicrotechJobRequest>
```

## Premaster Job

An example premaster job might be:

```
<?xml version="1.0"?>
<!DOCTYPE MicrotechJobRequest SYSTEM "C:\Icdr-Mj\XML\MicrotechJobRequest_1.0.dtd">
<MicrotechJobRequest>
  <Options JobName="SamplePremaster" Quantity="2">
    <GeneralOptions IncludeFailures="FALSE" MediaType = "DVD" ImageName="MyDataJob"
LeaveBsyFiles="TRUE"/>
    <WritingOptions TestWrite="TRUE"/>
    <VerifyOptions Verify="TRUE" />
  </Options>
  <DataDiscJob PremasterOnly="FALSE">
    <MMOptions Joliet="TRUE" UDF="TRUE" ISOLevel="2" VolumeName="SamplePremaster" />
    <IncludeFiles>
      <IncludeFolder FullPath = "C:\WINDOWS\Cursors" LocalPath="\Cursors">
        <Exclude Path = "3dgn0.cur" Type = "File" />
      </IncludeFolder>
      <IncludeFolder FullPath = "C:\system32\" LocalPath="\System">
        <PathChange Path ="DirectX" LocalPath ="DX" />
        <Exclude Path = "Microsoft" Type = "Folder" />
      </IncludeFolder>
      <IncludeFile FullPath = "C:\Windows\Clock.avi" LocalPath="\Winn\clock.avi" />
    </IncludeFiles>
  </DataDiscJob>
```

```
</MicrotechJobRequest>
```

## **Job Update**

When a job listener is attached to a job, the `MicrotechJobListener.onJobUpdate()` call back method will be executed about every two seconds. (See also the `MicrotechResponse.dtd` DTD.)

```
<MicrotechResponse>
  <JobStatus>
    <Job Name="Administ141007" DiscName="MyJob.iso" Requested="1" Completed="0"
Failed="0" State="PENDING" Priority="4" TotalPercentComplete="0" DateTime="01/16/07 13:42"/>
  </JobStatus>
</MicrotechResponse>
```

## **Event Updates**

Event call backs are invoked whenever an event listener has been attached to the server by a “`MicrotechCDRemoteManager.listenForEvents()`” or “`MicrotechProcessingServer.listenForEvents()`”. All the following event call back response XML strings conform to the `MicrotechEvent.dtd` DTD.

### **Pending Job Added**

When an event listener is attached to the server, the `MicrotechEventListener.onPendingJobAdded()` call back method will be executed whenever a job is added to the pending queue.

```
<?xml version = "1.0"?>
<!DOCTYPE MicrotechEvent SYSTEM "C:\Icdr-Mj\XML\MicrotechEvent_1.0.dtd">
  <MicrotechEvent>
    <PendingJobAdded Name="Administ154127" Type="Write"/>
  </MicrotechEvent>
```

### **Pending Job Removed**

When an event listener is attached to the server, the `MicrotechEventListener.onPendingJobRemoved()` call back method will be executed whenever a job is removed from the pending queue. This would occur when the job is aborted before it begins.

```
<?xml version = "1.0"?>
<!DOCTYPE MicrotechEvent SYSTEM "C:\Icdr-Mj\XML\MicrotechEvent_1.0.dtd">
  <MicrotechEvent>
    <PendingJobRemoved Name="Administ154127"/>
  </MicrotechEvent>
```

### **Active Job Added**

When an event listener is attached to the server, the `MicrotechEventListener.onActiveJobAdded()` call back method will be executed whenever a job is added to the active queue.

```
<?xml version = "1.0"?>
<!DOCTYPE MicrotechEvent SYSTEM "C:\Icdr-Mj\XML\MicrotechEvent_1.0.dtd">
```

```
<MicrotechEvent>  
  <ActiveJobAdded Name="Administ154127" Type="Write"/>  
</MicrotechEvent>
```

## Active Job Cancelled

When an event listener is attached to the server, the `MicrotechEventListener.onActiveJobCanceled()` call back method will be executed whenever an active job is aborted.

```
<?xml version="1.0"?>  
  <MicrotechEvent>  
    <ActiveJobRemoved Name="Administ154127"/>  
  </MicrotechEvent>
```



# Code Examples

This section shows some simple code examples using the API. These examples are simple “linear” code streams with emphasis on demonstration rather than elegance. The larger code samples that are distributed with this API will be most useful for first time developers. After reading through these simple examples, it is recommended that the code samples be further studied.

## C++

### Server and Job Requests

```
/**
 * CodeExample.cpp
 *
 * Simple code example to demonstrate the Microtech API. The following API
 * tasks are performed:
 *
 *     Connect to a fixed server.
 *     Request server status.
 *     Submit a fixed job.
 *     Attach a job listener.
 *     Stop the job listener.
 *     Disconnect from the server.
 */
#include <iostream>           // for cin, cout
#include <string>             // for string
#include <MicrotechCDRemoteManager.h> // for Microtech API

using namespace std;        // for string, cin, cout

/**
 * Most basic possible job listener. Just accept the XML response (which
 * conforms to the MicrotechResponse.dtd DTD), and display it.
 */
class JobListener : public MicrotechJobListener
{
public:
    /**
     * This is the lone method of the job listener interface that must
     * be implemented. This call will occur about every two seconds until
     * the listener is stopped, or the job completes (possibly because of
     * an error).
     *
     * @param jobXml The job update XML response from the server.
     */
    void OnJobUpdate(LPCTSTR jobXml)
    {
        /*
         * Output the response - this will not be parsed.
         */
        cout << jobXml;
    }
};

/**
 * Run the Microtech Code Example program.
 */
void main(int argc, char *argv[])
{
    try
    {
        /*
         * Connect to the server using a fixed name and port number.

```

```

    * MicrotechCDRemoteManager is implemented as a "singleton" object
    * that can be accessed via the getInstance() method.
    */
MicrotechCDRemoteManager::GetInstance()->ConnectTo(_T("lab2-xl-1"), 3519);
cout << "Successfully connected to the server.\n";

/*
 * Request server status. The XML string for the request is hard
 * coded here and conforms to the MicrotechRequest.dtd DTD. The
 * response will also be an XML string conforming to the
 * MicrotechResponse.dtd DTD.
 */
LPCTSTR serverStatusXml =
    _T("<?xml version=\"1.0\"?><MicrotechRequest><GetServerStatus
/></MicrotechRequest>");
LPCTSTR xmlResponse = MicrotechCDRemoteManager::GetInstance()-
>Request(serverStatusXml);
cout << "Successfully requested server status.\n";
/*
 * Output the response - this will not be parsed.
 */
cout << xmlResponse << "\n";

/*
 * Submit a fixed job. This is basic image job where the image file
 * must already exist on the server. The XML string argument conforms
 * to the MicrotechJobRequest.dtd DTD.
 */
string xmlJob = "<?xml version=\"1.0\"?>";
xmlJob.append( "<!DOCTYPE MicrotechJobRequest SYSTEM \"C:\\Icdr-
Mj\\XML\\MicrotechJobRequest_1.0.dtd\">");
xmlJob.append( "<MicrotechJobRequest>");
xmlJob.append( "    <Options Quantity=\"10\">");
xmlJob.append( "        <GeneralOptions ImageName=\"MyJobName\"
IncludeFailures=\"TRUE\" Priority=\"4\"/>");
xmlJob.append( "    </Options>");
xmlJob.append( "    <ImageJob>");
xmlJob.append( "        <ImageFile Path=\"\\\\\\LAB2-XL-1\\Drive-
D\\Images\\MyJob.iso\" />");
xmlJob.append( "    </ImageJob>");
xmlJob.append( "</MicrotechJobRequest>");
MicrotechJob job = MicrotechCDRemoteManager::GetInstance()->SubmitJob(xmlJob.c_str() );
cout << "Successfully submitted a job.\n";

/*
 * Attach a job listener. The job object was returned by the
 * submitJob() method above. Alternatively, the job listener could
 * be attached during the submitJob() call above. The job listener
 * object is defined at the top of this source file.
 */
JobListener jobListener;
MicrotechCDRemoteManager::GetInstance()->ListenForJob(job, &jobListener);
cout << "Successfully attached a job listener.\n";

/*
 * Now the job listener should get a call back about every two
 * seconds until it completes or the user presses return below.
 */
cout << "Press return to quit:\n";
string textLine;
getline(cin, textLine);

/*
 * Stop the job listener.
 */
MicrotechCDRemoteManager::GetInstance()->StopListening(job);
cout << "Successfully stopped job listener.\n";

/*
 * Disconnect from the server.
 */

```

```

        MicrotechCDRemoteManager::GetInstance()->Disconnect();
        cout << "Successfully disconnected from the server.\n";
    }
    catch ( MicrotechBaseException b )
    {
        /*
         * Just catch and display any Microtech exceptions.
         */
        cout << "Error communicating with CDRemote server.\n";
        cout << "    Message: " << b.GetExceptionMessage() << "\n";
    }
};

```

## Multi-Server Discovery

```

#include <iostream>          // cout
#include <MicrotechSearcherManager.h>    // MicrotechSearcherManager
                                         // MicrotechProcessingServer

using namespace std;      // cout

/* The default CD-Remote discovery port is configured at 3518 */
static int SEARCH_PORT = 3518;

/* Depending on the system and network, the search timeout may have to
 * be extended in order to find all the available servers. The search
 * will continue until the time out duration is complete.
 */
static int SEARCH_TIMEOUT = 5000;    // 5 seconds * 1000 counts/second

/* A true value will clear any servers found from a previous search.
 * A false value will append newly discovered servers to the existing
 * list of servers.
 */
static bool CLEAR_CURRENT = true;

void main(int argc, char *argv[])
{
    /* perform server "discovery" */
    MicrotechSearcherManager::GetInstance()->Search(SEARCH_PORT,
                                                    SEARCH_TIMEOUT,
                                                    CLEAR_CURRENT);

    /* display number of servers found */
    size_t serverCount = MicrotechSearcherManager::GetInstance()->Count();
    cout << "Number of servers found: " << serverCount << "\n";

    /* display each server by name */
    for (size_t i = 0; i < serverCount; i++)
    {
        MicrotechProcessingServer *server =
            MicrotechSearcherManager::GetInstance()->Get(i);
        cout << "    " << i << ": " << server->GetServerName() << "\n";
    }

    /* At this point, an individual server may be interacted with using
     * the appropriate MicrotechProcessingServer object.
     */
}

```

# Java

## Server and Job Requests

```
/**
 * CodeExample.java
 *
 * Simple code example to demonstrate the Microtech API. The following API
 * tasks are performed:
 *
 *     Connect to a fixed server.
 *     Request server status.
 *     Submit a fixed job.
 *     Attach a job listener.
 *     Stop the job listener.
 *     Disconnect from the server.
 */
import MicrotechObjects.*;           // Microtech classes - MicrotechCDRemoteManager
import MicrotechException.*;        // Microtech classes - MicrotechBaseException
import MicrotechInterfaces.*;       // Microtech classes - IMicrotechJobListener
import MicrotechJobs.MicrotechJob;  // Microtech classes - MicrotechJob

import java.io.*;                   // IOException

public class CodeExample
{
    /**
     * Most basic possible job listener. Just accept the XML response (which
     * conforms to the MicrotechResponse.dtd DTD), and display it.
     */
    public static class JobListener implements IMicrotechJobListener
    {
        /**
         * This is the lone method of the job listener interface that must
         * be implemented. This call will occur about every two seconds until
         * the listener is stopped, or the job completes (possibly because of
         * an error).
         *
         * @param jobXml    The job update XML response from the server.
         */
        public void onJobUpdate(String jobXml)
        {
            /**
             * Output the response - this will not be parsed.
             */
            System.out.println(jobXml);
        }
    }

    /**
     * Run the Microtech Code Example program.
     */
    public static void main(String[] args)
    {
        try
        {
            /**
             * Connect to the server using a fixed name and port number.
             * MicrotechCDRemoteManager is implemented as a "singleton" object
             * that can be accessed via the getInstance() method.
             */
            MicrotechCDRemoteManager.getInstance().connectTo("lab2-x1-1", 3519);
            System.out.println("Successfully connected to the server.");

            /**
             * Request server status. The XML string for the request is hard
             * coded here and conforms to the MicrotechRequest.dtd DTD. The
             * response will also be an XML string conforming to the
             * MicrotechResponse.dtd DTD.
             */
        }
    }
}
```

```

        */
        String xmlResponse = MicrotechCDRemoteManager.getInstance().request(
            ("<?xml version=\"1.0\"?><MicrotechRequest><GetServerStatus
/></MicrotechRequest>");
        System.out.println("Successfully requested server status.");
        /*
        * Output the response - this will not be parsed.
        */
        System.out.println(xmlResponse);

        /*
        * Submit a fixed job. This is basic image job where the image file
        * must already exist on the server. The XML string argument conforms
        * to the MicrotechJobRequest.dtd DTD.
        */
        String xmlJob =
            "<?xml version=\"1.0\"?>" +
            "<!DOCTYPE MicrotechJobRequest SYSTEM \"C:\\Icdr-
Mj\\XML\\MicrotechJobRequest_1.0.dtd\">" +
            "<MicrotechJobRequest>" +
            "  <Options Quantity=\"10\">" +
            "    <GeneralOptions ImageName=\"MyJobName\" IncludeFailures=\"TRUE\"
Priority=\"4\"/>" +
            "  </Options>" +
            "  <ImageJob>" +
            "    <ImageFile Path=\"\\\\\\LAB2-XL-1\\Drive-D\\Images\\MyJob.iso\" />" +
            "  </ImageJob>" +
            "</MicrotechJobRequest>";
        MicrotechJob job = MicrotechCDRemoteManager.getInstance().submitJob(xmlJob);
        System.out.println("Successfully submitted a job.");

        /*
        * Attach a job listener. The job object was returned by the
        * submitJob() method above. Alternatively, the job listener could
        * be attached during the submitJob() call above. The job listener
        * object is defined at the top of this source file.
        */
        JobListener jobListener = new JobListener();
        MicrotechCDRemoteManager.getInstance().listenForJob(job, jobListener);
        System.out.println("Successfully attached a job listener.");

        /*
        * Now the job listener should get a call back about every two
        * seconds until it completes or the user presses return below.
        */
        System.out.println("Press return to quit:");
        while (System.in.read() != '\n');

        /*
        * Stop the job listener.
        */
        MicrotechCDRemoteManager.getInstance().stopListening(job);
        System.out.println("Successfully stopped job listener.");

        /*
        * Disconnect from the server.
        */
        MicrotechCDRemoteManager.getInstance().disconnect();
        System.out.println("Successfully disconnected from the server.");
    }
    catch ( MicrotechBaseException b )
    {
        /*
        * Just catch and display any Microtech exceptions.
        */
        System.out.println("Error communicating with CDRemote server.");
        System.out.println("  Message: " + b.getMessage() );
        System.out.println("  Cause: " + b.getCause().toString() );
    }
    catch ( IOException io )
    {

```

```

        /*
        * Catch any console read errors.
        */
        System.out.println("I/O Error.");
        System.out.println("    Message: " + io.getMessage() );
        System.out.println("    Cause: " + io.getCause().toString() );
    }
}

```

## Multi-Server Discovery

```

import MicrotechInterfaces.*;           // IMicrotechProcessingServer
import MicrotechObjects.*;             // MicrotechSearcherManager

public class Discovery
{
    /* The default CD-Remote discovery port is configured at 3518 */
    private static final int SEARCH_PORT = 3518;

    /* Depending on the system and network, the search timeout may have to
    * be extended in order to find all the available servers. The search
    * will continue until the time out duration is complete.
    */
    private static final int SEARCH_TIMEOUT = 5000; // 5 secs * 1000 counts/sec

    /* A true value will clear any servers found from a previous search.
    * A false value will append newly discovered servers to the existing
    * list of servers.
    */
    private static final boolean CLEAR_CURRENT = true;

    public static void main(String[] args)
    {
        /* perform server "discovery" */
        MicrotechSearcherManager.getInstance().search(SEARCH_PORT,
                                                    SEARCH_TIMEOUT,
                                                    CLEAR_CURRENT);

        /* display number of servers found */
        int serverCount = MicrotechSearcherManager.getInstance().count();
        System.out.println("Number of servers found: " + serverCount);

        /* display each server by name */
        for (int i = 0; i < serverCount; i++)
        {
            IMicrotechProcessingServer server =
                MicrotechSearcherManager.getInstance().get(i);
            System.out.println("    " + i + ": " + server.getServerName());
        }

        /* At this point, an individual server may be interacted with using
        * the appropriate MicrotechProcessingServer object.
        */
    }
}

```

# C#.NET

## Server and Job Requests

```
/**
 * CodeExample.cs
 *
 * Simple code example to demonstrate the Microtech API. The following API
 * tasks are performed:
 *
 *     Connect to a fixed server.
 *     Request server status.
 *     Submit a fixed job.
 *     Attach a job listener.
 *     Stop the job listener.
 *     Disconnect from the server.
 */
using System;
using System.Collections.Generic;
using System.Text;

using MicrotechApi.Objects;    // MicrotechCDRemoteManager, MicrotechJob
using MicrotechApi.Interfaces; // MicrotechJobListener
using MicrotechApi.Exceptions; // MicrotechBaseException

namespace CodeExample
{
    public class Program
    {
        /**
         * Most basic possible job listener. Just accept the XML response (which
         * conforms to the MicrotechResponse.dtd DTD), and display it.
         */
        public class JobListener : MicrotechJobListener
        {
            /**
             * This is the lone method of the job listener interface that must
             * be implemented. This call will occur about every two seconds until
             * the listener is stopped, or the job completes (possibly because of
             * an error).
             *
             * @param jobXml The job update XML response from the server.
             */
            public void OnJobUpdate(string jobXml)
            {
                /*
                 * Output the response - this will not be parsed.
                 */
                Console.WriteLine(jobXml);
            }
        }

        /**
         * Run the Microtech Code Example program.
         */
        public static void Main(string[] args)
        {
            try
            {
                /*
                 * Connect to the server using a fixed name and port number.
                 * MicrotechCDRemoteManager is implemented as a "singleton" object
                 * that can be accessed via the GetInstance() method.
                 */
                MicrotechCDRemoteManager.GetInstance().connectTo("lab2-x1-1", 3519);
                Console.WriteLine("Successfully connected to the server.");

                /*
                 * Request server status. The XML string for the request is hard

```

```

        * coded here and conforms to the MicrotechRequest.dtd DTD. The
        * response will also be an XML string conforming to the
        * MicrotechResponse.dtd DTD.
        */
        string xmlResponse = MicrotechCDRemoteManager.GetInstance().Request(
            "<?xml version='1.0'?'><MicrotechRequest><GetServerStatus
/></MicrotechRequest>");
        Console.WriteLine("Successfully requested server status.");
        /*
        * Output the response - this will not be parsed.
        */
        Console.WriteLine(xmlResponse);

        /*
        * Submit a fixed job. This is basic image job where the image file
        * must already exist on the server. The XML string argument conforms
        * to the MicrotechJobRequest.dtd DTD.
        */
        string xmlJob =
            "<?xml version='1.0'?'>" +
            "<!DOCTYPE MicrotechJobRequest SYSTEM 'C:\\Icdr-
Mj\\XML\\MicrotechJobRequest_1.0.dtd'>" +
            "<MicrotechJobRequest>" +
            "    <Options Quantity='10'>" +
            "        <GeneralOptions ImageName='MyJobName' IncludeFailures='TRUE'
Priority='4' />" +
            "    </Options>" +
            "    <ImageJob>" +
            "        <ImageFile Path='\\\\\\LAB2-XL-1\\Drive-D\\Images\\MyJob.iso'
/>" +
            "    </ImageJob>" +
            "</MicrotechJobRequest>";
        MicrotechJob job = MicrotechCDRemoteManager.GetInstance().SubmitJob(xmlJob);
        Console.WriteLine("Successfully submitted a job.");

        /*
        * Attach a job listener. The job object was returned by the
        * submitJob() method above. Alternatively, the job listener could
        * be attached during the submitJob() call above. The job listener
        * object is defined at the top of this source file.
        */
        JobListener jobListener = new JobListener();
        MicrotechCDRemoteManager.GetInstance().ListenForJob(job, jobListener);
        Console.WriteLine("Successfully attached a job listener.");

        /*
        * Now the job listener should get a call back about every two
        * seconds until it completes or the user presses return below.
        */
        Console.WriteLine("Press return to quit:");
        while (Console.Read() != '\n') ;

        /*
        * Stop the job listener.
        */
        MicrotechCDRemoteManager.GetInstance().StopListening(job);
        Console.WriteLine("Successfully stopped job listener.");

        /*
        * Disconnect from the server.
        */
        MicrotechCDRemoteManager.GetInstance().Disconnect();
        Console.WriteLine("Successfully disconnected from the server.");
    }
    catch (MicrotechBaseException b)
    {
        /*
        * Just catch and display any Microtech exceptions.
        */
        Console.WriteLine("Error communicating with CDRemote server.");
        Console.WriteLine("    Message: " + b.Message);
    }
}

```



```

    }
}
}
}

```

## Multi-Server Discovery

```

using System; // Console
using MicrotechApi.Objects; // MicrotechSearcherManager,
// MicrotechProcessingServer

namespace MTApiDiscovery
{
    class Discovery
    {
        /* The default CD-Remote discovery port is configured at 3518 */
        const int SEARCH_PORT = 3518;

        /* Depending on the system and network, the search timeout may have to
        * be extended in order to find all the available servers. The search
        * will continue until the time out duration is complete.
        */
        const int SEARCH_TIMEOUT = 5000; // 5 seconds * 1000 counts/second

        /* A true value will clear any servers found from a previous search.
        * A false value will append newly discovered servers to the existing
        * list of servers.
        */
        const bool CLEAR_CURRENT = true;

        static void Main(string[] args)
        {
            /* perform server "discovery" */
            MicrotechSearcherManager.GetInstance().Search(SEARCH_PORT,
                SEARCH_TIMEOUT,
                CLEAR_CURRENT);

            /* display number of servers found */
            uint serverCount = MicrotechSearcherManager.GetInstance().Count;
            Console.WriteLine("Number of servers found: " + serverCount);

            /* display each server by name */
            for (uint i = 0; i < serverCount; i++)
            {
                MicrotechProcessingServer server =
                    MicrotechSearcherManager.GetInstance().Get(i);
                Console.WriteLine(" " + i + ": " + server.ServerName);
            }

            /* At this point, an individual server may be interacted with using
            * the appropriate MicrotechProcessingServer object.
            */
        }
    }
}
}

```

# VB.NET

## Server and Job Requests

```
' CodeExample.vb
'
' Simple code example to demonstrate the Microtech API. The following API
' tasks are performed:
'
'     Connect to a fixed server.
'     Request server status.
'     Submit a fixed job.
'     Attach a job listener.
'     Stop the job listener.
'     Disconnect from the server.

Imports MicrotechApi.Exceptions
Imports MicrotechApi.Interfaces
Imports MicrotechApi.Objects

Module CodeSample
    ' Most basic possible job listener. Just accept the XML response (which
    ' conforms to the MicrotechResponse.dtd DTD), and display it.
    Public Class JobListener
        Implements MicrotechJobListener

        ' This is the lone method of the job listener interface that must
        ' be implemented. This call will occur about every two seconds until
        ' the listener is stopped, or the job completes (possibly because of
        ' an error).
        '
        ' @param jobXml The job update XML response from the server.
        Public Sub OnJobUpdate(ByVal jobXml As String) Implements
MicrotechApi.Interfaces.MicrotechJobListener.OnJobUpdate
            ' Output the response - this will not be parsed.
            '
            Console.WriteLine(jobXml)
        End Sub
    End Class

    '
    ' Run the Microtech Code Example program.
    '
    Sub Main()
        Dim xmlResponse, xmlJob As String
        Dim job As MicrotechJob
        Dim jobListener As JobListener

        Try
            ' Connect to the server using a fixed name and port number.
            ' MicrotechCDRemoteManager is implemented as a "singleton" object
            ' that can be accessed via the getInstance() method.
            MicrotechCDRemoteManager.GetInstance.ConnectTo("lab2-xl-1", 3519)
            Console.WriteLine("Successfully connected to the server.")

            ' Request server status. The XML string for the request is hard
            ' coded here and conforms to the MicrotechRequest.dtd DTD. The
            ' response will also be an XML string conforming to the
            ' MicrotechResponse.dtd DTD.
            xmlResponse = MicrotechCDRemoteManager.GetInstance().Request( _
                ("<?xml version=\"1.0\"?><MicrotechRequest><GetServerStatus
/></MicrotechRequest>")
            Console.WriteLine("Successfully requested server status.")
            ' Output the response - this will not be parsed.
            Console.WriteLine(xmlResponse)

            ' Submit a fixed job. This is basic image job where the image file
            ' must already exist on the server. The XML string argument conforms
```

```

' to the MicrotechJobRequest.dtd DTD.
xmlJob = _
  "<?xml version=""1.0""?>" & _
  "<!DOCTYPE MicrotechJobRequest SYSTEM ""C:\\Icdr-
Mj\\XML\\MicrotechJobRequest_1.0.dtd"">" & _
  "<MicrotechJobRequest>" & _
    "<Options Quantity=""10"">" & _
      "<GeneralOptions ImageName=""MyJobName"" IncludeFailures=""TRUE""
Priority=""4"" />" & _
    "</Options>" & _
    "<ImageJob>" & _
      "<ImageFile Path=""\\\\\\LAB2-XL-1\\Drive-D\\Images\\ MyJob.iso"" />" & _
    "</ImageJob>" & _
  "</MicrotechJobRequest>"
job = MicrotechCDRemoteManager.GetInstance.SubmitJob(xmlJob)
Console.WriteLine("Successfully submitted a job.")

' Attach a job listener. The job object was returned by the
' submitJob() method above. Alternatively, the job listener could
' be attached during the submitJob() call above. The job listener
' object is defined at the top of this source file.
jobListener = New JobListener()
MicrotechCDRemoteManager.GetInstance.ListenForJob(job, jobListener)
Console.WriteLine("Successfully attached a job listener.")

' Now the job listener should get a call back about every two
' seconds until it completes or the user presses return below.
Console.WriteLine("Press return to quit:")
Console.ReadLine()

'
' Stop the job listener.
'
MicrotechCDRemoteManager.GetInstance.StopListening(job)
Console.WriteLine("Successfully stopped job listener.")

'
' Disconnect from the server.
'
MicrotechCDRemoteManager.GetInstance.Disconnect()
Console.WriteLine("Successfully disconnected from the server.")
Catch b As MicrotechBaseException
'
' Just catch and display any Microtech exceptions.
'
Console.WriteLine("Error communicating with CDRemote server.")
Console.WriteLine("    Message: " & b.Message)
Catch e As Exception
'
' Catch any console read errors.
'
Console.WriteLine("I/O Error.")
Console.WriteLine("    Message: " + e.Message)
End Try
End Sub

End Module

```

## Multi-Server Discovery

```

Imports MicrotechApi.Objects
Module Discovery

    Sub Main()

        ' The default CD-Remote discovery port is configured at 3518
        Dim SEARCH_PORT As Integer = 3518

        ' Depending on the system and network, the search timeout may have to
        ' be extended in order to find all the available servers. The search

```

```

' will continue until the time out duration is complete.
Dim SEARCH_TIMEOUT As Integer = 5000      ' 5 seconds * 1000 counts/second

' A true value will clear any servers found from a previous search.
' A false value will append newly discovered servers to the existing
' list of servers.
Dim CLEAR_CURRENT As Boolean = True

' perform server "discovery"
MicrotechSearcherManager.GetInstance().Search(SEARCH_PORT, _
                                              SEARCH_TIMEOUT, _
                                              CLEAR_CURRENT)

' display number of servers found
Dim serverCount As UInteger
serverCount = MicrotechSearcherManager.GetInstance().Count
Console.WriteLine("Number of servers found: " + serverCount.ToString())

' display each server by name
Dim i As UInteger
For i = 1 To serverCount
    Dim server As MicrotechProcessingServer
    server = MicrotechSearcherManager.GetInstance().Get(i - 1)
    Console.WriteLine("    " + i.ToString() + ": " + server.ServerName)
Next

' At this point, an individual server may be interacted with using
' the appropriate MicrotechProcessingServer object.
End Sub

End Module

```

## Rimage (C++)

This code example demonstrates how to connect to the Microtech API using Rimage API calls. This would most likely be used if an existing Rimage API application is required to connect to a Microtech production system. This example uses C++, but the other languages could be used instead.

```
/**
 * CodeExample.cpp
 *
 * Simple code example to demonstrate the Rimage interface to the Microtech API.
 * The following API tasks are performed:
 *
 *     Connect to a fixed server.
 *     Request server status.
 *     Submit a fixed job.
 *     Attach a job listener.
 *     Stop the job listener.
 *     Disconnect from the server.
 */
#include <fstream>           // for ofstream
#include <iostream>         // for cin, cout
#include <string>           // for string
#include <ClientApiInclude.h> // Rimage API

using namespace std;       // for string, cin, cout

#define SERVER_ID    _T("lab2-x1-1")
#define SERVER_PORT  _T("3519")
#define CLIENT_ID    _T("MyClientID")

/**
 * Most basic possible job listener. Just accept the XML response (which
 * conforms to the ProductionOrderStatus.dtd DTD), and display it.
 */
class RimageOrderListener : public OrderStatusListener
{
public:
    /**
     * This is the lone method of the order status listener interface that
     * must be implemented. This call will occur about every two seconds until
     * the listener is stopped, or the job completes (possibly because of
     * an error).
     *
     * @param orderStatusXml The order status XML response from the server.
     */
    void onStatus(LPCTSTR xmlOrderStatus)
    {
        /*
         * Output the response - this will not be parsed.
         */
        cout << _T("Order Status: ");
        cout << xmlOrderStatus << _T("\n");
    }
};

/**
 * Run the Microtech Code Example program.
 */
void main(int argc, char *argv[])
{
    try
    {
        /*
         * Connect to the server using a fixed name and port number.
         * MicrotechCDRemoteManager is implemented as a "singleton" object
         * that can be accessed via the getInstance() method.
         */
        SystemManager::getInstance()->connect(CLIENT_ID, SERVER_ID, SERVER_PORT);
    }
}
```

```

cout << _T("Successfully connected to the server.\n");

/*
 * Request server status. The XML string for the request is hard
 * coded here and conforms to the ProductionServerRequest.dtd DTD.
 * The response will also be an XML string conforming to the
 * MicrotechResponse.dtd DTD.
 */
string serverStatusXml = _T("<?xml version=\"1.0\"?>\n");
serverStatusXml.append( _T("<!DOCTYPE ProductionServerRequest SYSTEM
\"C:\\Program Files\\RimageSDK\\ApiSdk\\XML\\ProductionServerRequest_1.0.dtd\">\n"));
serverStatusXml.append( _T("<ProductionServerRequest ServerId=\"\"");
serverStatusXml.append( SERVER_ID);
serverStatusXml.append( _T("< ClientId=\"\""));
serverStatusXml.append( CLIENT_ID);
serverStatusXml.append( _T(">\n"));
serverStatusXml.append( _T("<GetServerStatus GetAutoloaderStatus=\"true\"
/>\n"));
serverStatusXml.append( _T("</ProductionServerRequest>"));
LPTSTR xmlResponse = ServerManager::getInstance()->executeServerRequest(SERVER_ID,
serverStatusXml.c_str());
cout << _T("Successfully sent server status request.\n");
/*
 * Output the response - this will not be parsed.
 */
if (xmlResponse == NULL)
{
    cout << _T("Error requesting server status.\n");
}
else
{
    cout << _T("Response:\n") << xmlResponse << _T("\n\n");
}

/*
 * Submit a fixed job. This is basic print job. The XML string argument
 * conforms to the ProductionOrder.dtd DTD.
 */
string xmlJob = _T("<?xml version=\"1.0\"?>\n");
xmlJob.append( _T("<!DOCTYPE ProductionOrder SYSTEM \"C:\\Program
Files\\RimageSDK\\ApiSdk\\XML\\ProductionOrder_1.3.dtd\">\n"));
xmlJob.append( _T("<ProductionOrder Copies=\"1\" OrderId=\"Norm-Print-Only2\"
ClientId=\"\""));
xmlJob.append( CLIENT_ID);
xmlJob.append( _T("< Priority=\"Normal\" Originator=\"\""));
xmlJob.append( CLIENT_ID);
xmlJob.append( _T(">\n"));
xmlJob.append( _T("<Media Size=\"120mm\" Type=\"CDR\"/>\n"));
xmlJob.append( _T("<Target Cluster=\"DefaultProductionCluster\"/>\n"));
xmlJob.append( _T("<Action>\n"));
xmlJob.append( _T("<Label Filename=\"C:\\Icdr-MJ\\30MinPhoto.lay\">\n"));
xmlJob.append( _T("<BTW SaveAfterRendering=\"default\"/>\n"));
xmlJob.append( _T("</Label>\n"));
xmlJob.append( _T("</Action>\n"));
xmlJob.append( _T("</ProductionOrder>\n"));

cout << "Submitting job:\n" << xmlJob << "\n\n";

RimageOrderListener orderListener;
OrderDescription *orderDescription =
    OrderManager::getInstance()->submitOrder(xmlJob.c_str(), &orderListener);
cout << "Successfully submitted a job.\n";
cout << "Successfully attached a job listener.\n";

/*
 * Now the job listener should get a call back about every two
 * seconds until it completes or the user presses return below.
 */
cout << "Press return to quit:\n";
string textLine;
getline(cin, textLine);

```

```

    /*
     * Stop the job listener.
     */
    OrderManager::getInstance()->stopListeningForOrder(orderDescription);
    cout << "Successfully stopped job listener.\n";

    /*
     * Disconnect from the server.
     */
    SystemManager::getInstance()->disconnect();
    cout << "Successfully disconnected from the server.\n";
}
catch ( BaseException be )
{
    /*
     * Catch and display any Rimage exceptions.
     */
    cout << "API exception.\n";
    cout << "    Message: " << be.getMessage() << "\n";
}
};

```

# Building Your Application

## C++

When building an application using C++, the API objects and functions are accessed using the Microtech API library.

The following Microtech API provided directories contain files required to build an application. The API “...\include” directory must be accessible to the compiler, the API library directory (“...\lib”) must be accessible to the linker, and the API “...\bin” directory containing the API dll (Dynamic Link Library) file(s) must be accessible at run time.

**...\include** The include file `MicrotechCDRemoteManager.h` will usually be sufficient to make the API classes accessible. Use the “`#include`” statement in your source file(s).

**...\lib** The library file `Microtech_Client_Api.lib` or `Microtech_Client_Api_Unicode.lib` will usually be sufficient for linking the API application. Add this library directory to your linker invocation. Depending on your C++ development system, you may be required to add this library as a dependency.

**...\bin** The dll file `Microtech_Client_Api.dll` or `Microtech_Client_Api_Unicode.dll` will usually be sufficient for running the API application. Make this directory available in the PATH environment variable or put the dll file(s) into an accessible directory.

The Microtech C++ API uses MicroSoft’s MSXML4 parser internally to interpret the XML text strings that are used for communication between the API and CDRemote. The parser dll file is required for use by the API and is also used by the sample programs supplied by Microtech.

This parser should be downloaded and installed by visiting this site:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=3144b72b-b4f2-46da-b4b6-c5d7485f2b42&DisplayLang=en>

(Or perform a web search using “microsoft msxml4”.)

The MSXML parser uses the COM library, so use `CoInitialize()` at the beginning and `CoUninitialize()` at the end of each thread of your application using the parser. This includes the callback thread that is implicitly created when attaching a job listener object. The one exception is when the server listener callback thread is created. This thread will already have called `CoInitialize()` because of internal parsing requirements from CDRemote. Of course you can always check the results back from the `CoInitialize()` call and compare it against `S_OK` for success. Only call `CoUninitialize()` if the result from `CoInitialize()` was `S_OK`. See the msdn Microsoft documentation for more details.



## **Java**

When building an application using Java, the API objects and methods are accessed using the following JAR (Java ARchive) library files:

<b>MicrotechApiJ.jar</b>	Contains classes for the CD-Remote Manager and event and job listeners. (See packages MicrotechObjects and MicrotechInterfaces.)
<b>MicrotechApiJGlobals.jar</b>	Contains classes for the various Microtech exceptions, job objects, and log manager. (See packages MicrotechException, MicrotechJobs, and MicrotechLogger.)
<b>MicrotechApiJInternal.jar</b>	Contains classes for internal Microtech use.

These JAR files must be accessible from your development environment (such as the CLASSPATH environment variable in Windows).

Within the application source code, the following packages may be imported to access the corresponding pieces of the API:

<b>import MicrotechObjects.*;</b>	Contains the CD-Remote manager.
<b>import MicrotechInterfaces.*;</b>	Contains the event and job listeners.
<b>import MicrotechException.*;</b>	Contains all the Microtech specific exceptions.
<b>import MicrotechJobs.*;</b>	Contains the job and job creator objects.
<b>import MicrotechLogger.*;</b>	Contains the log file manager.

## **C# .NET**

When building an application using C# .NET, the API objects and functions are accessed using the Microtech API .NET libraries. The .NET libraries also use the C++ run-time libraries, so the section on C++ application building and run-time .dll files should also be referenced.

The .NET libraries are distributed in the “..\bin” directory.

<b>MicrotechApi.Net.dll</b>	This is the primary library that must be added as a reference to the application (using the referenced name of MicrotechApi.Net).
<b>Microtech_Client_Api_Unicode.dll</b>	This library is used by MicrotechApi.Net.dll and must be present, though not directly referenced by the application.

Once the above library is referenced by the application, the API calls may be used by specifying the appropriate classes either directly or through “using” statements.

<b>using MicrotechApi.Objects;</b>	Contains the CD-Remote manager and the job and logger objects.
<b>using MicrotechApi.Interfaces;</b>	Contains the event and job listeners.
<b>using MicrotechApi.Exceptions;</b>	Contains all the Microtech specific exceptions.

Note: The .NET security permissions must be correctly set on the client machine running the Microtech API application. These settings are accessed using the Microsoft .NET Framework Configuration tool (found in the Control Panel – Administration Tools). It is beyond the scope of this document to cover this topic, so further familiarity of this tool may be required. A symptom of a problem might be a run-time error referencing: “System.Security.Permissions.SecurityPermission”.

## **VB.NET**

When building an application using Visual Basic .NET, the API objects and functions are accessed using the Microtech API .NET libraries (these are the same libraries that are used for C#). The .NET libraries also use the C++ run-time libraries, so the section on C++ application building and run-time .dll files should also be referenced.

The .NET libraries are distributed in the “...\bin” directory.

<b>MicrotechApi.Net.dll</b>	This is the primary library that must be referenced by the application (using the referenced name of MicrotechApi.Net).
<b>Microtech_Client_Api_Unicode.dll</b>	This library is used by MicrotechApi.Net.dll and must be present, though not directly referenced by the application.

Once the above library is referenced by the application, the API calls may be used by specifying the appropriate classes either directly or through “imports” statements.

<b>Imports MicrotechApi.Objects</b>	Contains the CD-Remote manager and the job and logger objects.
<b>Imports MicrotechApi.Interfaces</b>	Contains the event and job listeners.
<b>Imports MicrotechApi.Exceptions</b>	Contains all the Microtech specific exceptions.

Note: The .NET security permissions must be correctly set on the client machine running the Microtech API application. These settings are accessed using the Microsoft .NET Framework Configuration tool (found in the Control Panel – Administration Tools). It is beyond the scope of this document to cover this topic, so further familiarity of this tool may be required. A symptom of a problem might be a run-time error referencing: “System.Security.Permissions.SecurityPermission”.

## **Rimage**

The Rimage interface to the Microtech API has been designed to be as transparent as possible to the developer. It is the intent that little or no source code modification would be required to rebuild an existing Rimage API application to become a Microtech API application. The only build requirement is that the Microtech libraries must replace the Rimage SDK libraries.

Because of the architectural differences between the Rimage and Microtech systems, some parameters to statuses and jobs will not be meaningful. The Microtech API will accept all the parameters from a Rimage command, but may ignore them if they don't have a comparable translation on the Microtech production system.

The Rimage SDK must be installed on the development system (this should already be the case for an existing application). The existing application still uses all the include files and all the DTD files from the Rimage SDK, so these files must be accessible to the compiler (include files) and run time environment (DTD files). The language being used will determine which Microtech API library should replace the existing Rimage library.

**C++**                    Link with the Microtech *RmClient\_7\_0\_n\_2.lib* and *RmMsg\_7\_0\_n\_2.lib* (or the Unicode equivalents).

**Java**                    Add the Microtech *ClientApi.jar* file to the CLASSPATH.

**C#.NET, VB.NET**            Reference the Microtech *Rimage.Client.Api.dll* file. The Microtech C++ run time libraries are also used in .NET applications.

It would also be useful to read the previous sections pertaining to each specific language for relevant details.

## CD-Remote Emulator

To assist the API developer, Microtech has included in the API package a CD-Remote emulator. This stand alone program can be run on any PC and serves to act as a virtual CD-Remote. The emulator will appear to the API program as a true CD-Remote executing program and will mimic all its functionality including server status and response, job submission, attaching event and job listeners, etc.

The emulator does not interact with any Microtech production system hardware but only simulates the CD-Remote program. The simplest use of the emulator would be to execute it concurrently on the development PC and connect to it using “*localhost*” as the server name.

The “...\emulator” directory contains the files required to run the emulator and these file should always be kept together.

<b>CDRemoteEmulator.exe</b>	Emulator executable which will run in a command shell.
<b>CDRemoteEmulator.config.xml</b>	The configuration file for the emulator. This may be edited to define the environment to emulate.
<b>*.dll</b>	Library files used by the emulator.

### Configuration

The configuration XML file can be modified to simulate a specific Microtech production system. The default configuration file distributed with the API contains all the valid parameters which should be used as the starting point template for any developer. The following configuration XML elements are described.

<b>&lt;Server&gt;</b>	Contains the emulated server status settings. These are the values that would be returned in response to a <i>GetServerStatus</i> request.
<b>&lt;Jobs&gt;</b>	Contains the emulated system’s jobs that already exist in the system. These will always be added to the system with a “PENDING” state status. The <i>PercentIncrementRate</i> value indicates the rate at which the percent complete value will be incremented (during a job listener call back for example) to simulate job progression. See below for more details.
<b>&lt;Printers&gt;</b>	Contains the emulated system’s printer definitions for simulated printing.
<b>&lt;Drives&gt;</b>	Contains the emulated system’s output drive definitions for simulated disc writing.

<b>&lt;Inputs&gt;</b>	Contains the emulated system's input drive definitions for simulated disc reading.
<b>Directories</b>	Contains the emulated system's directory path definitions. The Jobs directory will be used for newly submitted jobs (*.job files).

## **Usage**

The emulator will execute in its own command console shell (via “*start*” – “*run*” or other). While running, status messages will be displayed in this window. There are just a few commands that can be invoked in the emulator command shell. These are displayed in a simple menu (pause the server, resume the server, raise an error).

While running, the emulator will receive and respond to commands from the API program. All server requests and job requests will be handled appropriately including event and job listener call backs. All responses will be returned with the appropriate data conforming to the relevant DTD.

When invoked, the emulator will load simulated jobs into the system that are defined in the configuration file and any \*.job files that appear in the Jobs directory. Jobs submitted from the API also cause a \*.job file to be created. To remove a job from the emulated system, delete the \*.job file and then restart the emulator.

All jobs will begin with a state of “PENDING”. Each job's state will automatically transition to “PROCESSING” and then to “COMPLETE”. These are the only job state values that the emulator supports. While processing, the job's percent complete will increment based on the configuration *PercentIncrementRate* value (see above).

## What's Next?

This manual is intended to be a “quick start” for a Microtech API programmer to become familiar with the interface. The basic overview of a program is discussed with some very basic code examples provided. For the programmer to be successful, the recommended next steps are to study the remaining API pieces including:

**Class Reference**        The class reference files are distributed in multiple formats. These should appear as standardized documentation describing each class along with its methods and calling arguments.

**Sample Code**         The provided sample code should provide the programmer more examples of API calls and usage. These may be used as a template or code snippets to be used during development.

**DTD Files**            These files are essential for use when coding the creation and parsing of the request and response XML strings. As noted in this manual, these files must also be used by the API libraries.

**Emulator**            The emulator will be very useful for trying out API code while avoiding interacting with hardware. See the prior section of this manual for an introduction to the emulator.