

# Introduction to CloudTurbine



John P. Wilson, Erigo Technologies

April 9, 2018



## Contents

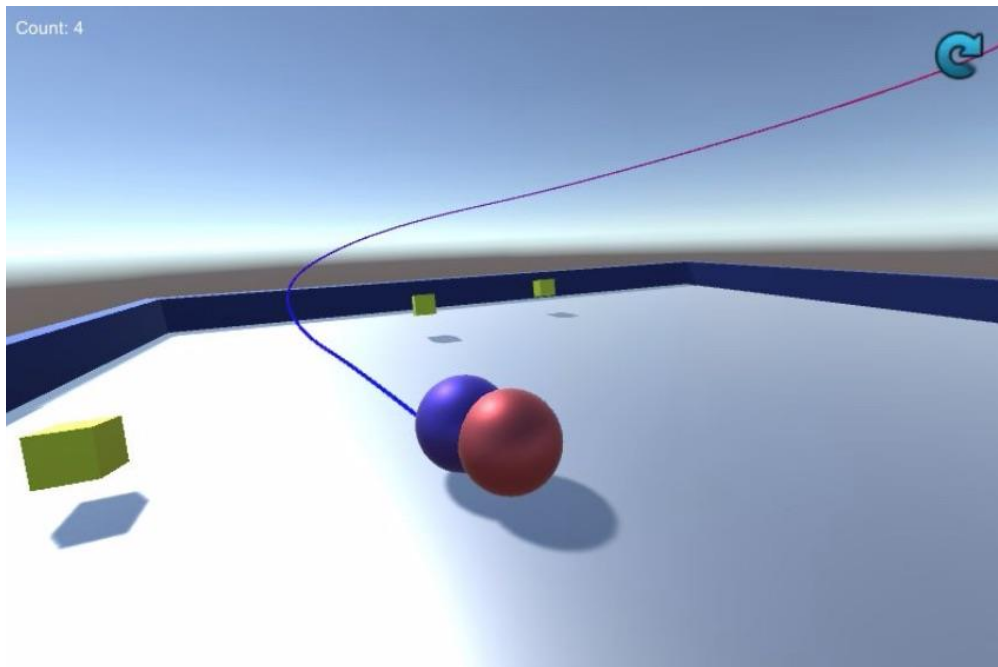
Contents .....	i
Goals .....	1
Introduction .....	2
Installation .....	3
Using CT source and sink applications .....	6
CTmousetrack demonstration .....	6
CTstream demonstration .....	8
Sharing data .....	11
Developing CT applications .....	14
Final notes .....	18



## Goals

CloudTurbine (CT) is an open source technology which saves data as incremental file “packets” to support archiving and streaming. It has been utilized in various data acquisition, video/audio and even multi-player gaming applications (Figure 1). This tutorial explains how to install CT, use source applications (apps such as CTstream that write data to CT format) and sink applications (apps such as CTweb that read CT-formatted data). We also provide a brief overview of developing CT apps. Throughout this tutorial, we will point to useful resources available at the CloudTurbine web site, <http://www.cloudturbine.com>.

In providing a broad overview of CT, there are many aspects of the technology which we won’t examine in detail. For further information, we invite you to explore the CT web site, which contains reference documents, white papers and demonstrations highlighting CT in action.



**Figure 1. CloudTurbine technology enabling real-world streaming data in a Unity-based demonstration game (<http://www.cloudturbine.com/cloudturbine-for-mixed-reality/>).**



## Introduction

CloudTurbine is a structured folder and file format that facilitates archiving and streaming any type of data. As open source software (published under the Apache 2.0 license), CT is freely available and can be extended or embedded in other applications (commercial or otherwise). For a brief introduction to CT, see the 1-page brochure available at the CloudTurbine website (<http://www.cloudturbine.com/ct-white-paper-doc/>).

CT's core innovation is to save data (of any type) as incremental files. CT formatted data files are useful as a local archive, but the real power of saving data in this format is to enable streaming via third-party file sharing services such as Dropbox or OneDrive.

The CT software distribution includes various “source” applications for saving data to CT format; you can also develop your own source apps using the Java or C# APIs<sup>1</sup>. CT supports viewing live and historical data in a browser with WebScan (we'll show how to do this in the next section); other viewer and utility applications are available online<sup>2</sup>, or you can develop your own using the Java API<sup>3</sup>.

Just as important as CT's core innovation (saving data to incremental files) is our core philosophy: ***with CT, you never lose control of your data***. There is no “central repository” storing your data. You control where you write and how you distribute your data. CloudTurbine also includes security features for data at rest and data in transit encryption.

Look at the introductory video at <http://www.cloudturbine.com/category/demos/>. The image and audio data for this video is stored in CT format, accessed from a cloud-based CT archive and played in an embedded WebScan frame in the browser. This brief video is presented by Matt Miller, the inventor of CloudTurbine.

While CT includes the capability to save data via FTP and HTTP as well as distribute data using an included web-server application, at its core CT focuses on reading and writing data files. Network services such as Dropbox are leveraged to create a complete data archiving, mirroring and distribution network<sup>4</sup>.

---

<sup>1</sup> <http://www.cloudturbine.com/writing-ct-source-apps/>

<sup>2</sup> <http://www.cloudturbine.com/applications/>

<sup>3</sup> <http://www.cloudturbine.com/writing-ct-sink-apps/>

<sup>4</sup> CT's “micro-kernel” approach may be contrasted with DataTurbine/RBNB middleware server software. DataTurbine users may be interested in the “OSDT vs. CT” FAQ page available at <http://www.cloudturbine.com/osdt-vs-ct/>.



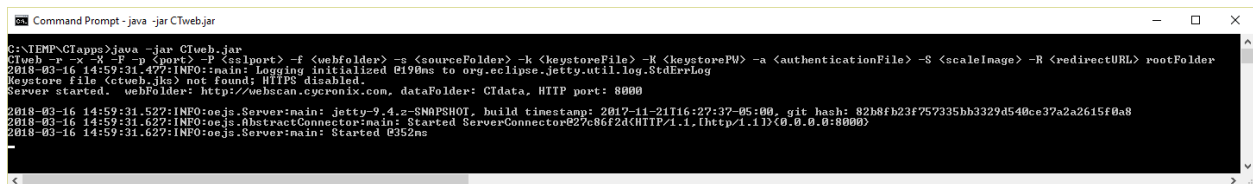
## Installation

CloudTurbine is hosted at the Cyclicronix page on GitHub, <https://github.com/cyclicronix/cloudturbine>. Binary applications are available with the CT releases on GitHub (download the *CTapps.zip* file) or you can compile apps from source. To compile CT code, you will need a Git client to clone the CT repository to your local machine and a JDK from Oracle (v1.8 or newer). CT uses a Gradle-based build system. Compile the apps using the supplied Gradle wrapper files (no need to locally install Gradle). Developers may want to peruse two resources at the CT website: “Using GitHub and Git” (<http://www.cloudturbine.com/using-github-and-git/>) and “Gradle-based CloudTurbine build system” (<http://www.cloudturbine.com/gradle/>).

To use a binary distribution, go to the latest release (<https://github.com/cyclicronix/cloudturbine/releases/latest>), download *CTapps.zip* and unzip it on your computer. You must have a Java JRE or JDK (v1.8 or newer) installed on your system to run CT applications. Bring up a terminal or command prompt window in the directory where you unzipped the CT apps. Execute the web server application, *CTweb.jar*, as follows:

```
java -jar CTweb.jar
```

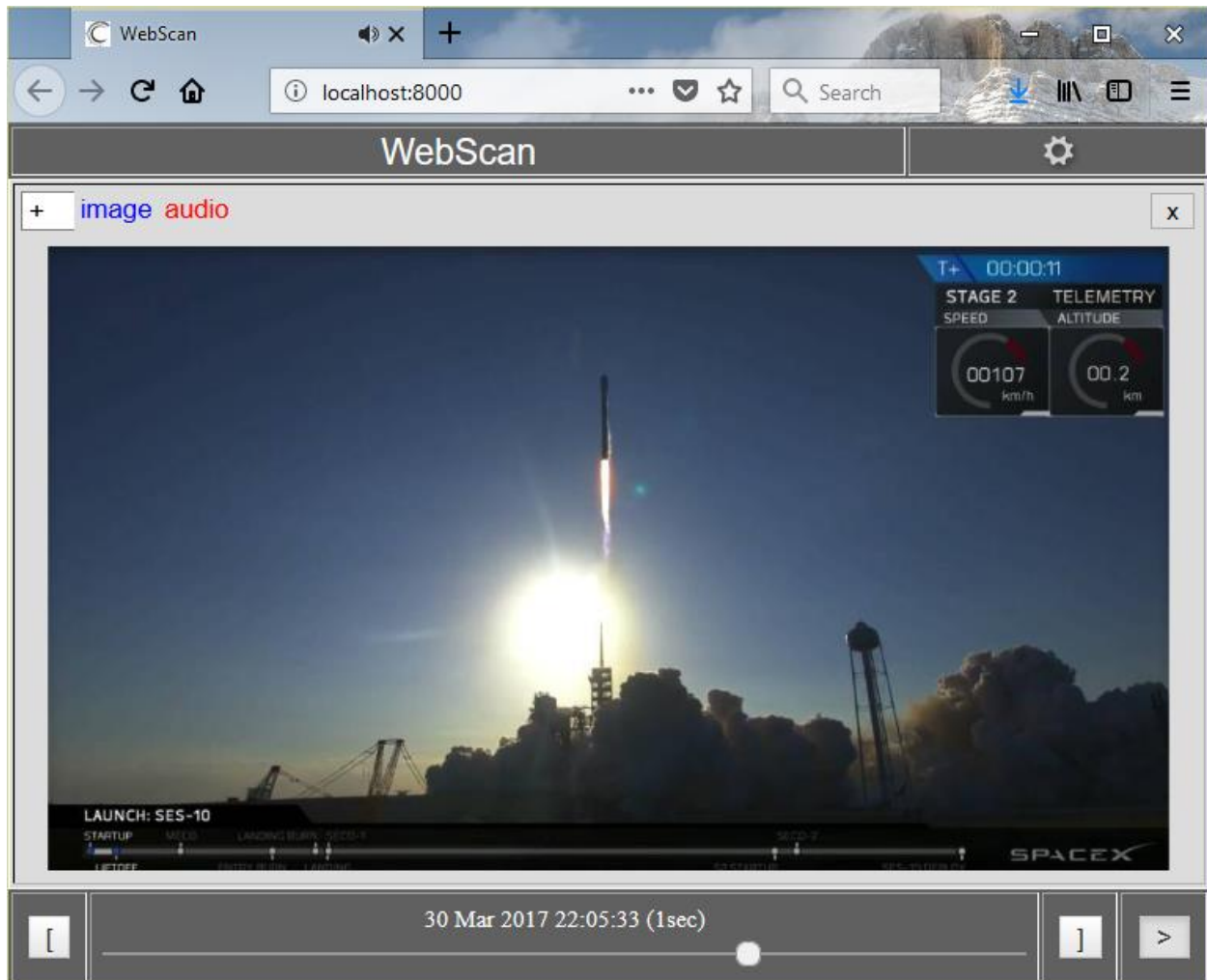
Figure 2 shows *CTweb* running in a Windows command prompt window.



```
Command Prompt - java -jar CTweb.jar
C:\TEMP\CTapps>java -jar CTweb.jar
CTweb -r -x -K -P -p <port> -P <sslport> -f <webfolder> -s <sourceFolder> -k <keyStoreFile> -K <keyStorePW> -a <authenticationFile> -S <scaleImage> -R <redirectURL> rootFolder
2018-03-16 14:59:31.477:INFO:main: Logging initialized @190ms to org.eclipse.jetty.util.log.StdErrLog
Keystore file <ctweb.jks> not found; HTTPS disabled.
Server started. webFolder: http://webscan.cyclicronix.com, dataFolder: C:\data, HTTP port: 8000
2018-03-16 14:59:31.522:INFO:oejs.Server:main: jetty-9.4.z-SNAPSHOT, build timestamp: 2017-11-21T16:27:37-05:00, git hash: 82b8fb23f757335bb3329d540ce37a2a2615f0a8
2018-03-16 14:59:31.627:INFO:oejs.AbstractConnector:main: Started ServerConnector@27c86f2d[HTTP/1.1,(http/1.1)<0.0.0.0:8000>]
2018-03-16 14:59:31.627:INFO:oejs.Server:main: Started @352ms
```

Figure 2. Running CT web server.

In a browser window, go to <http://localhost:8000>. This connects to the web server and runs a WebScan data window in your browser. Click on the small “+” popup menu in the upper left corner of the window and add the *SpaceX/image.jpg* channel to the window; click on “+” again and add the *SpaceX/audio.wav* channel. Click the “>” button (in lower-right corner) to play the video. This video of a SpaceX launch (saved as CT image and audio channels) is included as sample data with the CT distribution. Try moving the scrollbar manually back and forth to play forward and backward through the locally-hosted CT data. Figure 3 shows the data in a browser window.



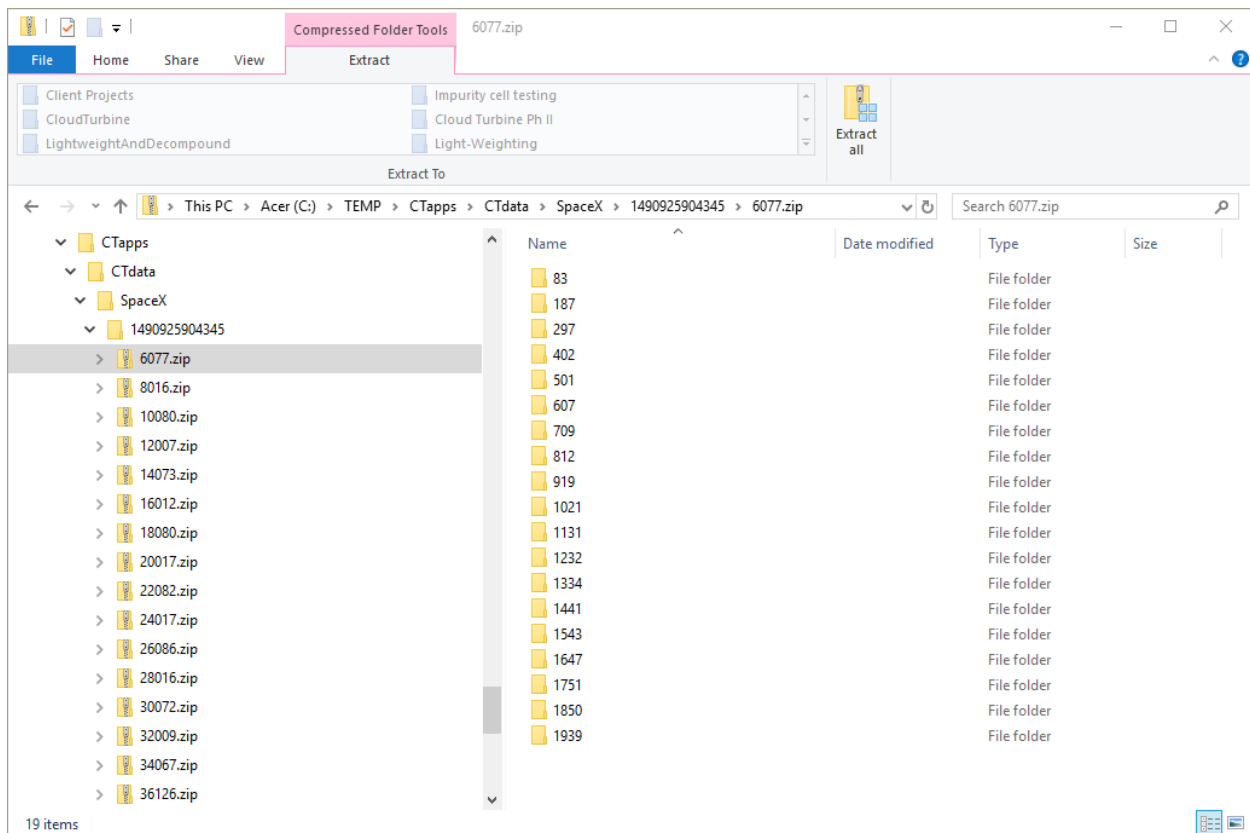
**Figure 3. Playing pre-recorded CT data in a WebScan window in a browser.**

Use your system's file explorer to examine the files in the *CTdata* folder from the distribution (Figure 4). *SpaceX* (the top folder) is the name of the CT source. Under *SpaceX* is a folder named *1490925904345*, which represents the data start time in milliseconds since January 1, 1970 (epoch time). This is called the "session" level folder in CT parlance. Within this CT session folder is a series of ZIP files, called data "blocks".

CT block folders are typically ZIP'ed for enhanced compression and organization. The numbers used for the names of the ZIP files indicate a relative offset (in milliseconds) from the top session time. Looking at the files in this example folder, you can see that each ZIP file holds approximately 2 seconds worth of data; that is, the names of the ZIP files step up in approximately 2000 millisecond increments. Unzipping one of these files reveals another folder layer, called the "point" folders.



The name of each point folder indicates a relative offset (in milliseconds) from the parent block folder time. Channel data files are stored in the point folders; for our SpaceX example, you will see *image.jpg* and *audio.wav* files. You can double-click on any of these files to see the data snapshot at that point in time. The video we watched in the browser is made up of a sequence of these files; in this case, images which were saved at 10 frames/sec and audio as a series of 16-bit 8kHz “.wav” files.



**Figure 4. Examining the folder hierarchy of a sample CT source.**

As an optional installation step, it may be convenient to define a *ctdir* environment variable which points to your unpacked CTapps folder. Using this, you can run CT apps from any location on your computer; for example, in Windows, you could run:

```
java -jar %ctdir%\CTweb.jar
```



## Using CT source and sink applications

In this tutorial we will introduce using CloudTurbine *source* applications (which produce CT data) and continue to use *CTweb* to fetch/view CT data (a CT *sink* application).

### ***CTmousetrack demonstration***

Open a command prompt/terminal window in your CTapps installation folder. Run the *CTmousetrack* source app, which captures your mouse cursor position and stores it in CT format as “x” and “y” channels:

```
java -cp CTexample.jar CTmousetrack CTdata/CTmousetrack
```

Next, we’ll run the CT web server to view this data. (If you still have the server running from the *Installation* tutorial above, skip this step.) Open another command prompt/terminal window in your CTapps installation folder. Execute the CT web server application, *CTweb.jar*, as follows:

```
java -jar CTweb.jar
```

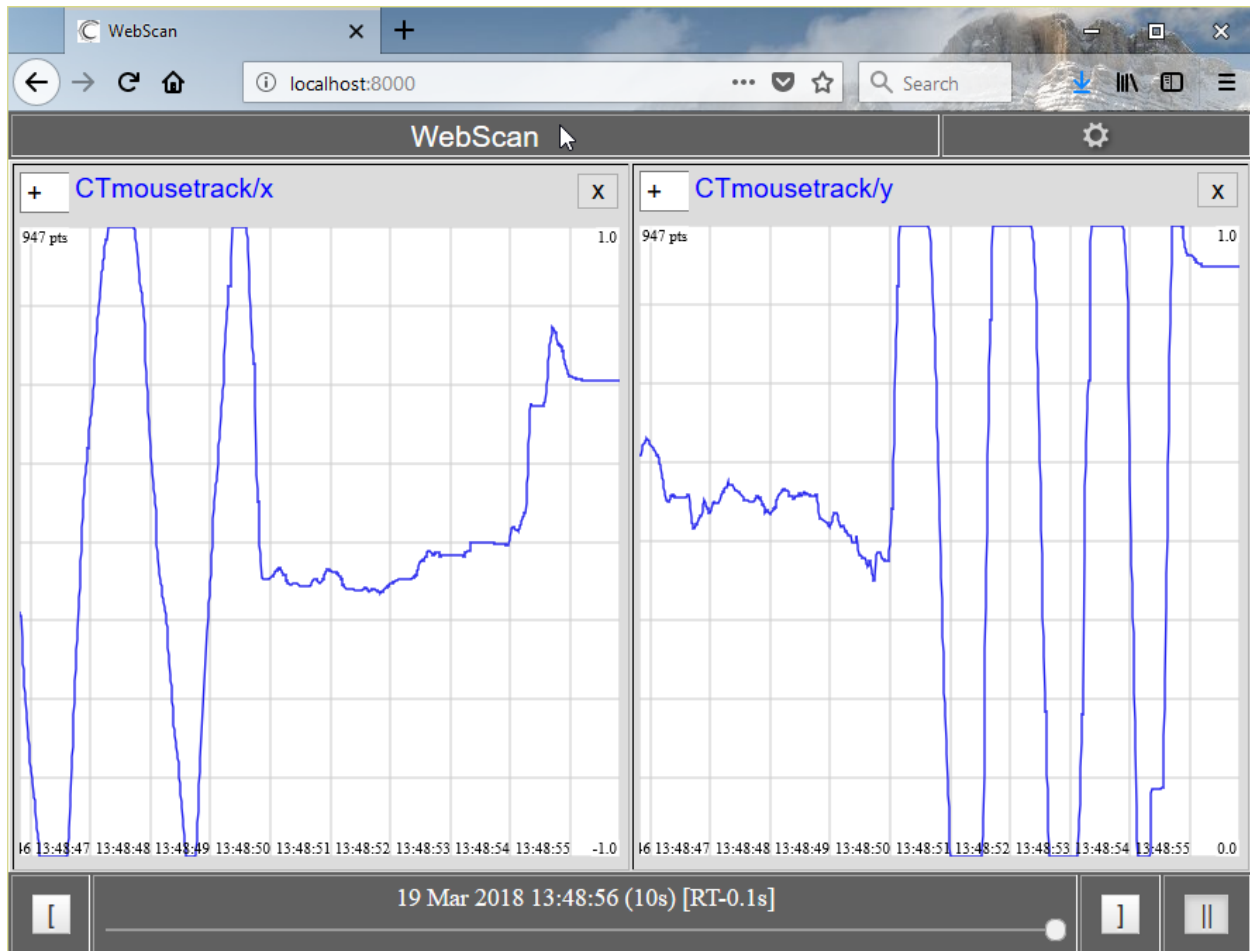
To view the mouse position data, go to <http://localhost:8000> in a browser to bring up the WebScan interface (if you are connecting to the same server from the *Installation* tutorial above, you might have to Shift+reload this URL to refresh the page to pick up the new channels). Configure the WebScan view as follows:

- Click the “gear” icon in the upper right corner of the WebScan window; select a 10 second *View* duration and 2 *Plots*. Click on the gear icon again to close the configuration menu.
- Click the “+” channel selector in one of the plots; you will notice that channels from the SpaceX and CTmousetrack sources are available.
- In one plot, select the CTmousetrack/x channel and in the other plot select CTmousetrack/y.
- In the lower-right corner of the WebScan interface, click the “go to end” button (“J”) and then the “real-time” button (“RT”).

As you move the mouse around on your desktop, notice how the x and y channel data changes (Figure 5). You may notice a short delay between moving your mouse and seeing the data change in the WebScan plot. Two sources contribute to this delay:

- Data is collected for a short time (10 samples at 10 msec period, thus about 100msec) before being written out to disk as a ZIP file. Packing data in this manner helps to constrain the number of files being created.
- There is also a small delay for data to be fetched by the WebScan web application and displayed in the plot. In its “normal” refresh mode, WebScan fetches data updates every second. To speed this up, open the WebScan configuration menu (click the gear icon in the upper right corner of the window) and change *Refresh* to “Fast”.

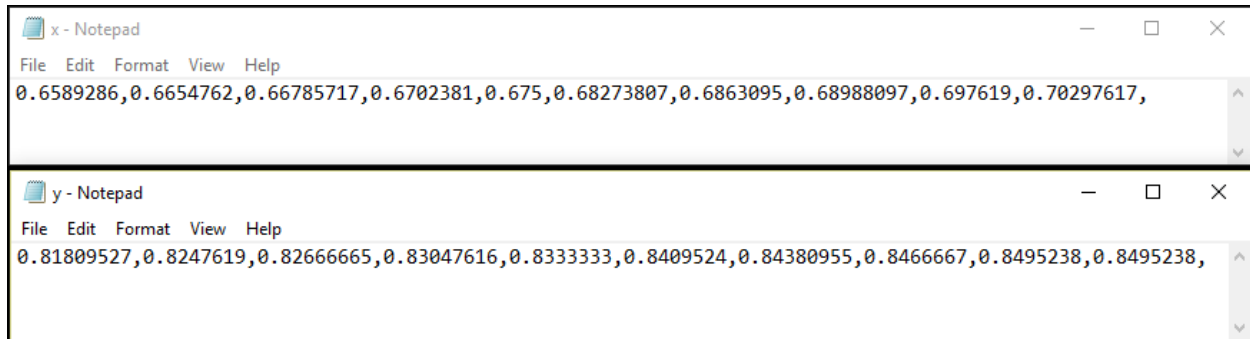




**Figure 5. Data from CTmousetrack displayed using the WebScan interface.**

Observe the CT files as they are being written to disk by launching a file explorer window on your system and navigating to the CTdata/CTmousetrack folder. At the top level you will see a single “session” level folder whose name is a 13-digit number, representing the epoch timestamp (milliseconds since January 1, 1970) of the start of the data source. Under this session folder you will find a series of “segment” level folders whose names represent a relative offset (in milliseconds) from the top session folder timestamp. CTmousetrack is configured to store 1000 block-level ZIP files in each segment folder. Navigate into the newest segment folder and watch as it is populated with ZIP files. Each ZIP file contains 10 mouse cursor x,y position data points; after collecting 10 data points, CTmousetrack “flushes” the data to write it out as a block-level ZIP file.

Unpack one of the ZIP files. Within the unpacked block folder, you will see a point folder which contains the “x” and “y” channel files. Each of these channel files contain 10 comma-separated value (CSV) samples (Figure 6). In CT-parlance, these are referred to as “packed” data files.



**Figure 6. Sample data from CTmousetrack, saved as comma-separated value files.**

For this CTmousetrack source, step back and notice the hierarchy of timestamped folders with channel data files at the lowest level. At the top level (under CTdata/CTmousetrack) we find the “session” folder with its absolute epoch timestamp, followed by an (optional) segment level folder, followed by ZIP files at the block level. Within each ZIP file is an additional folder level (point folder) which contains the channel files. Additional information about the CT folder/file hierarchy can be found in the “structure” document at the CT web portal, <http://www.cloudturbine.com/structure/>.

As a side note, let’s think about the CT “flush” rate (the rate at which ZIP data blocks are written to disk) for a moment. If you are distributing CT ZIP files in real time using a file sharing service, the rate at which these files are generated (the “flush” rate) can be optimized based on what the underlying network service can support. For example, Dropbox can share about 2 files per second. Thus, it would be best to write at most 2 ZIP block files per second when using this file sharing service, otherwise your shared data will get further and further behind. As a rule of thumb, many file sharing services can support at least one file update per second, thus writing out one new CT ZIP file per second is a good rate.

We’ll run another CT source demonstration next using CTstream. But first, to avoid filling your disk with data, terminate the CTmousetrack application.

## ***CTstream demonstration***

Bring up a command prompt/terminal window in your CTapps folder and start CTstream as follows:

```
java -jar CTstream.jar
```

Click on the “screen” and “text” checkboxes and click the “Start” button. Text messages you write in the multiline text area are saved to CT format, as are screen captures from the region within the red capture box. As shown in Figure 7, we are capturing a web browser displaying the current time.

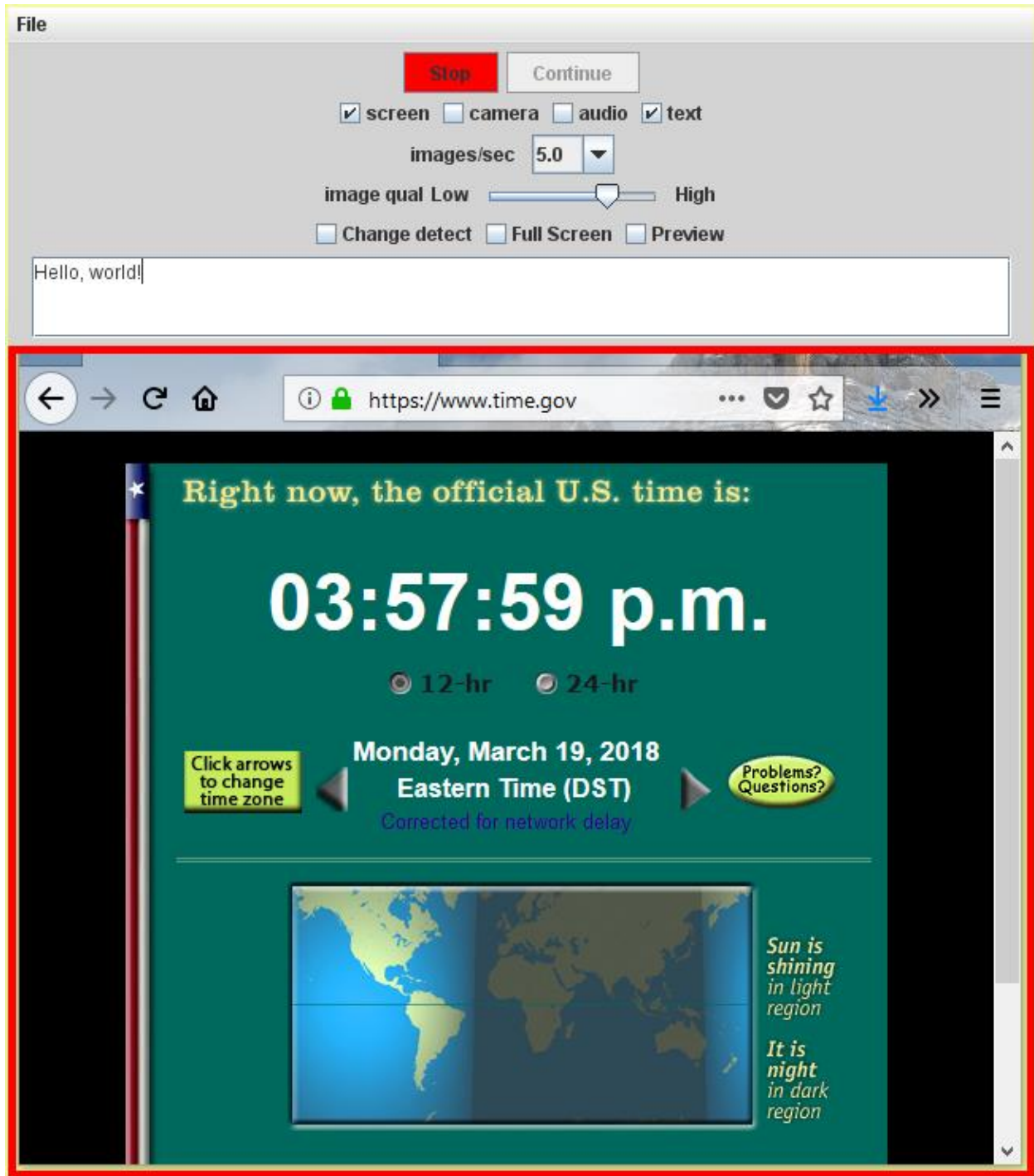


Figure 7. CTstream saving text and screen capture images to CT file format.

Assuming you still have the CTweb server running, go to <http://localhost:8000> in a browser to view the captured data (you might need to Shift+reload to refresh the WebScan page to have it pick up the new channels). Click on the “gear” icon in the upper right corner of the WebScan window to pop open the



options dialog. Select a 1 second duration and chose to view 2 plots. Click on the gear icon again to hide the configuration dialog box. Click on the “+” menus in the upper left corner of each empty plot and select the “screencap.jpg” or “text.txt” channels. In the lower right corner of the WebScan page, click the “go to end” button (“J”) and then click the “real-time” button (“RT”). WebScan will display new data as it is captured and saved to CT format by CTstream. For example, change the text message in the CTstream text area or move/resize CTstream’s screen capture region and observe the data updates in WebScan. By default, CTstream flushes data to CT-formatted files once every second; thus, there is a small delay between CTstream capturing the data and that new data showing up in WebScan. If you capture an image of current wall clock time (as shown in Figure 7), you will get a good idea of the delay between capturing the image in CTstream and displaying it in WebScan. CTstream’s flush rate can be changed via the File/Settings dialog.

As we did with the CTmousetrack source, you can explore the hierarchy of time folders saved in the CTstream source data. Navigating to the lowest level, you will see the saved channel files: “screencap.jpg” and “text.txt”.

Before proceeding to the next part of the tutorial, shut down CTstream and the CTweb server.



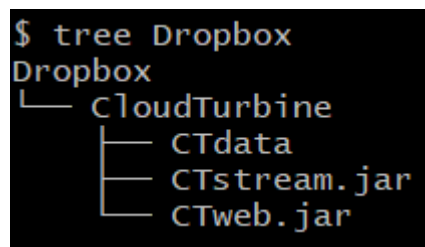
## Sharing data

CloudTurbine is a technology which writes data as incremental, time-stamped files. As such, it is uniquely positioned to leverage third-party file sharing services to distribute and mirror data. Any service that supports file sharing can be used to “stream” CloudTurbine data; for example, Dropbox, Box, Google Drive, or Microsoft OneDrive. Users can take advantage of the built-in capabilities file sharing services offer, including performance, flexibility, supported tools and security features (these capabilities will, of course, differ widely across services).

Suppose you’d like to provide users access to your CT data. There are two basic ways to establish a CT network to accomplish this: First, run CTweb as a hub-and-spoke server on a LAN. This simple network design is effective for providing low-latency access to CT data for a small number of users. Second, CT files can be widely distributed to a group of users you specify via a file sharing service such as Dropbox or Google Drive. Across this WAN, users can access their own local copy of your CT data, viewing it with a locally running CTweb server and WebScan. Data distribution using a third-party service introduces additional latency but has the potential to support many users.

Let’s try the “WAN” network mode described above. For this test, you will need two computers, each of which have the selected file sharing service (such as Dropbox) installed. For ease of use, you may choose to run this demonstration with both computers side-by-side on your desk.

Make a new shared folder called CloudTurbine. Copy the CTstream.jar and CTweb.jar apps to this shared folder. Within the shared folder, make another folder called CTdata. See the folder hierarchy shown in Figure 8 (we are using Dropbox for this demonstration, but you can use any file sharing service). These folders and files should automatically be copied to your remote system.



```
$ tree Dropbox
Dropbox
├── CloudTurbine
│   ├── CTdata
│   ├── CTstream.jar
│   └── CTweb.jar
```

**Figure 8. Setup for sharing data using a third-party file sharing service.**

On one system, we’ll call it the “local” system, bring up a command prompt/terminal window, change to the shared CloudTurbine folder and run CTstream:

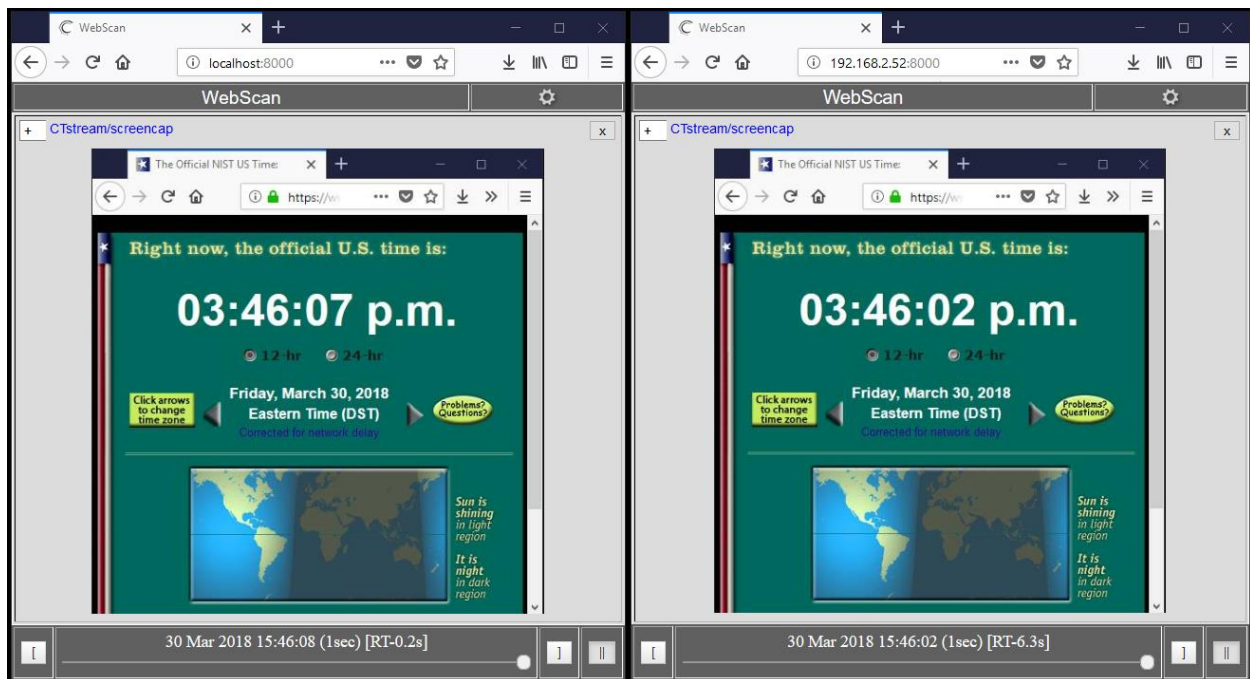
```
java -jar CTstream.jar
```

Enable any of the desired capture streams (screen, camera, etc.) and click the Start button. (See the prior section, “Using CT source and sink applications”, for details on running this application.) CT data will be saved in the CTdata/CTstream folder.



Since the CT data is saved in a shared folder, it will be automatically copied to the second computer (what we'll call the "remote" system). Examine the incoming block data (ZIP) files under the CTdata/CTstream folder on the remote system. Depending on the sharing service, it may not be able to keep up with CTstream's default one-zip-file-per-second rate. If you notice that the incoming files on the remote system are getting further and further behind the files being saved on the local system, try increasing CTstream's "Flush interval" (available from the File menu, Settings item); for example, change from 1sec to 2sec or 5sec interval. As flush interval is increased, the block/ZIP file generation rate will decrease (more data will be packed into each ZIP file). A downside of increasing flush interval is that data latency also increases (more delay before the files are saved to disk).

Run CTweb server on the local system and another CTweb server on the remote system; view the data on each system in a browser at <http://localhost:8000>. If the local and remote systems used in this demonstration are on the same LAN, you can also view the CT data in a browser on a third computer using the local IP address of either the "local" or "remote" system. Figure 9 shows a screen capture of data from the "local" system (left panel) and the "remote" system (right panel) at one instant of time. The difference in time between the clock images in each browser screen gives an indication of the latency in copying data from the "local" to the "remote" system; about 5 seconds in this case.



**Figure 9. Viewing shared data from a "local" system (left browser window) and the "remote" system (right browser window).**

We can run this demonstration on a LAN with "local" and "remote" computers side-by-side on a desk, but bear in mind that (based on the capabilities of the third-party file sharing system) these two machines could just as easily be half-way around the world from each other. In this manner, CT can leverage third-party file sharing services to support global data WANs.



This tutorial has focused on writing local CT data files and using a third-party file sharing service to distribute these files. CT also supports writing data to remote FTP or HTTP servers. You can *HTTP PUT* data to the CTweb server (must run CTweb with the “-W <num\_sources>” option). CTstream supports writing out CT data to the local file system (as we did above) and also to FTP or HTTP servers; see the CTstream app page for details (<http://www.cloudturbine.com/ctstream/>).

The following white papers (available on the CloudTurbine website) provide further information and options to consider when sharing CT data:

“Scalable CloudTurbine Networks” (<http://www.cloudturbine.com/scalable-cloudturbine-networks/>) describes CT networking topologies.

“CloudTurbine Security White Paper” (<http://www.cloudturbine.com/ct-security-white-paper-pdf/>) discusses various aspects of CloudTurbine security, including data-at-rest, data-in-use, data in transit, and security use-cases.



## Developing CT applications

CloudTurbine technology is focused on writing data to and reading data from a well-defined folder/file hierarchy.<sup>5</sup> By default, CT writes to the local file system, but support is included for writing via FTP or HTTP/S. CT includes a Java API library<sup>6</sup> (“CTlib.jar” in the binary distribution) for developing your own CT source or sink/viewer apps. A more limited C# library is also available which supports writing data to CT format<sup>7</sup>.

Writing custom CT applications is straightforward using one of these APIs, but it isn’t the only way to create CT data. Consider the article “How to get your data into CT”<sup>8</sup> which discusses 4 ways to save data to CT format. This article’s “Shell script” tab includes a simple Linux shell script that fetches live internet-accessible webcam images and saves them in CT format. The script is shown in Figure 10. Try this script by saving it as a bash file, “save\_webcam.bash”, in the same folder as your other CT JAR files (see the Installation section above for setup information).

NOTE: If you are running this script under Cygwin, save the content from Figure 10 in a file called “save\_webcam.txt” then run the following command to convert Windows line endings to Linux line endings:

```
tr -d '\15\32' < save_webcam.txt > save_webcam.bash
```

Give the script file executable permission and run it. Data will be saved under the CTdata directory in the “Dartmouth” source folder. Open file explorer on your computer and look through the folder/file hierarchy (for example, see Figure 11). You can double-click on channel files (“webcam.jpg”) to view the image at that corresponding time snapshot.

Start CTweb server from the CTapps folder:

```
java -jar CTweb.jar
```

To view the CT data in real-time, open the WebScan interface at <http://localhost:8000>, add channel “Dartmouth/webcam.jpg”, view the newest data by clicking “J” and enter real-time mode by clicking “RT” (Figure 12).

---

<sup>5</sup> <http://www.cloudturbine.com/structure/>, PDF version at [http://www.cloudturbine.com/wordpress/wp-content/uploads/2018/02/CT\\_structure2.pdf](http://www.cloudturbine.com/wordpress/wp-content/uploads/2018/02/CT_structure2.pdf)

<sup>6</sup> <https://github.com/cycronix/cloudturbine/tree/master/JavaCode/CTlib>

<sup>7</sup> [https://github.com/jpw-erigo/CTlib\\_csharp](https://github.com/jpw-erigo/CTlib_csharp)

<sup>8</sup> <http://www.cloudturbine.com/get-data-into-ct/>





```
#!/bin/bash

CTdata="CTdata"
if [ "$#" -eq 1 ]; then
    CTdata=$1
fi

CTsource="Dartmouth"
start_time=`date +%s`
CTsession=$CTdata/$CTsource/$start_time
mkdir -p $CTsession

webcamURL="http://wc2.dartmouth.edu/jpg/image.jpg"

while :
do
    current_time=`date +%s`
    block_time=$((current_time-start_time))
    CTpoint=$CTsession/$block_time/0
    mkdir -p $CTpoint
    curl -s -S -o temp.jpg "$webcamURL"
    mv temp.jpg "$CTpoint/webcam.jpg"
    printf "."
    sleep 1
done
```

**Figure 10.** Script which fetches images from a live internet webcam and saves them in CT format.

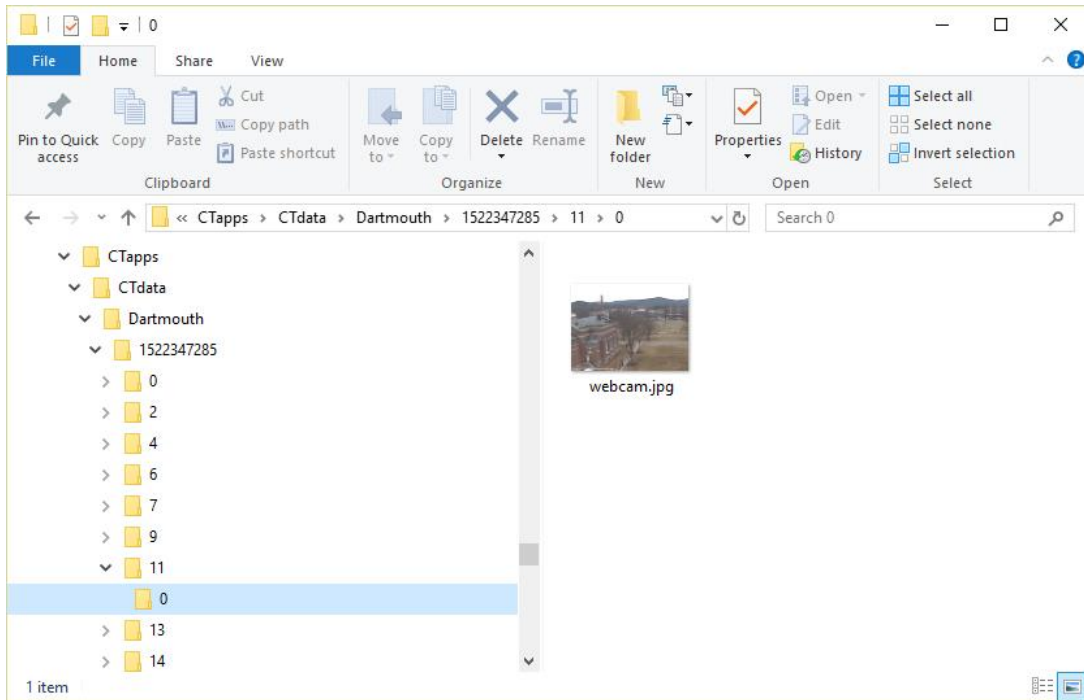


Figure 11. Image output from the script shown in Figure 10.

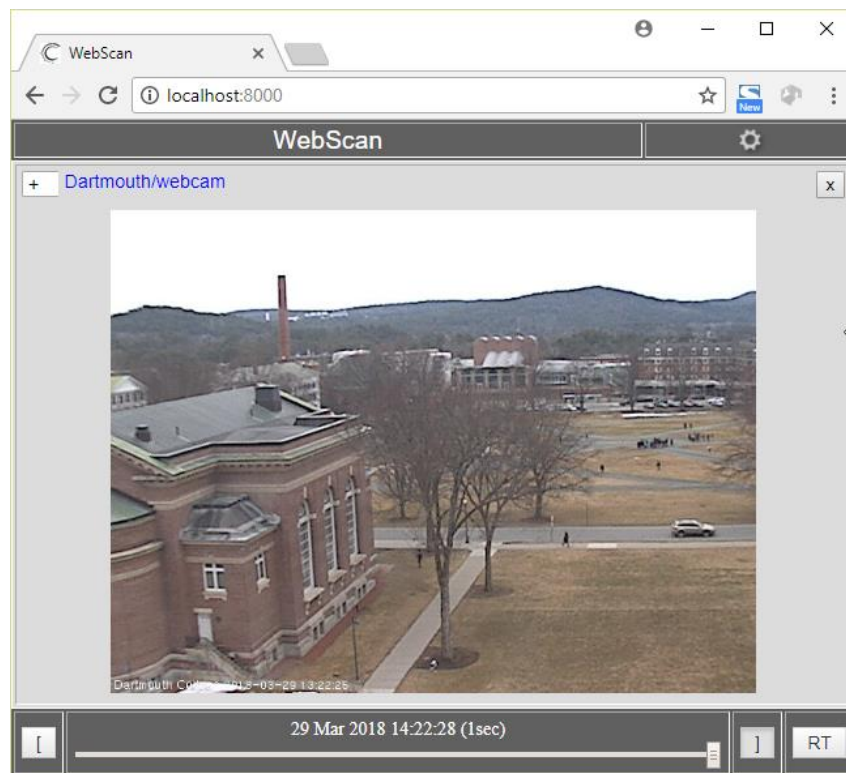


Figure 12. Viewing images saved via the script shown in Figure 10 in WebScan.



This scripting example demonstrates how easy it can be to get data into CT format; once data is saved in this format, other tools (such as CTweb server) can be used to fetch and view the data. More substantial source or sink/viewer applications can be developed using one of the CT APIs. The Java API library, “CTlib.jar”, is included with the CT distribution (you’ll find this file in your CTapps folder). A section of the CloudTurbine website is devoted to CT development: go to <http://www.cloudturbine.com/documents/> and click on the Developer tab. Two good resources to get you started with Java development are:

- “Writing CT source apps”, <http://www.cloudturbine.com/writing-ct-source-apps/>
- “Writing CT sink apps”, <http://www.cloudturbine.com/writing-ct-sink-apps/>

Also, JavaDoc documentation for the Java API is available at <http://cloudturbine.com/javadoc/>.

While the CloudTurbine Java API is the officially supported programming interface for developing CT apps, a more limited Windows .NET API written in C# is available for saving data to CT format (i.e., it supports a CT-writer capability but not CT-reader). The GitHub page for this library ([https://github.com/jpw-erigo/CTlib\\_csharp](https://github.com/jpw-erigo/CTlib_csharp)) includes a description of the library’s capabilities, documentation and a sample application.

A great way to participate in CT development is through the officially sponsored “App-Kits” (<http://www.cloudturbine.com/applications/>, click on the App-Kits tab). Each App-Kit has well-defined goals, a list of resources, and “Developer Updates” where team members can write blog-like posts. Registered members of the CT website can request to join App-Kit teams or submit an idea to start their own App-Kit.



## Final notes

Make sure all CT source applications from the tutorials are terminated; delete unneeded data from the CTdata folder.

We invite you to explore the CT website, <http://www.cloudturbine.com>, for further information about CloudTurbine. Our web portal contains introductory information, video demonstrations, white papers, app and developer documentation, news updates and forums for posting questions or sharing ideas.

If you have an idea for CloudTurbine development or would like help integrating CT into your project, please email us at [info@cycronix.com](mailto:info@cycronix.com).

Finally, we would like to express our appreciation to NASA Armstrong Flight Research Center for supporting the development of CloudTurbine.