

Introduction to the

Cine-tal

**PhiloWebService Software
Developers Kit**

9/06/2007

Introduction

The public Cine-tal SDK, PhiloWebService, is web based and available on Cinemage and eL1000. It is implemented as a SOAP Web Service that runs on the Cine-tal product. The SDK requires the web service to be enabled on the server (the Cine-tal equipment). On eL1000, the web server is always enabled. On the Cinemage, the web service is disabled when it is shipped. For security reasons, changes to the state of the web server must be done from the Cinemage itself. Consult the Cinemage documentation for details.

To explore the web service on a Cine-tal product, you can access it from your browser with a URL like the one shown below.

`http://cinetal-XXXXXX/PhiloWebService`

The name of your machine should be substituted for cinetal-XXXXXX. To get your machine name on a Cinemage, look under Unit information. On an eL 1000, you can get your machine name from the Machine Name section of the Setup window. The XXXXXX represents the last 6 digits of the serial number.

PhiloWebService is used to control many of the the device's capabilities, such as routing, calibration, stills and LUTs. It's not a "real time" control, although the commands will be executed as quickly as possible. For some things, such as routing controls, this will take place in the next field. For other controls, such as those that need to parse complex file structures, a command may take as long as 5 seconds to execute.

PhiloWebService was named in honor of Philo T. Farnsworth, inventor of the CRT and television transmitter.

SOAP Introduction

SOAP is an XML-based communication protocol for exchanging information between applications over HTTP. SOAP uses the same ports and protocols as conventional web pages, so it's easy to use with existing web browsers or other applications. SOAP is platform and language independent.

For our purposes, we are interested in the ability of SOAP to facilitate communication between a client application (a developer application, SDK example, etc.) and a Web service (in this case PhiloWebService). The PhiloWebService in turn controls the hardware of the Cine-tal unit.

There are a variety of Web Services and SOAP frameworks available. Among the frameworks that have been successfully used with Cine-tal products are gSoap, Apache Axis, and Microsoft .NET.

WSDL Introduction

WSDL is an XML-based language for describing Web services and how to access them. When a developer adds a web reference to an application, a WSDL document that describes the web service is downloaded from the web service. The WSDL document tells the application the location of the web service and what it can do and how to get it to do it.

Most SOAP frameworks will import a WSDL file, and then automatically build code to access that framework. For example, gSOAP uses the command-line applications “wsdl2h” and “soap2cpp” to automatically generate C++ or C bindings and access code from a WSDL file. The generated code includes the documentation that’s part of the SOAP service.

Further Information

If you wish to learn more about SOAP, WSDL and a variety of related topics, the URLs shown below are a good place to start.

<http://www.w3schools.com>

<http://www.cs.fsu.edu/~engelen/soap.html>

<http://sourceforge.net/projects/gsoap2>

<http://ws.apache.org/axis/>

<http://www.ddj.com/cpp/184401909>

PhiloWebService Overview

PhiloWebService provides three separate services: Parameters, File Transfers and LUT Service. Note that the following documentation and current WSDL files are available directly on the units.

- [Parameters](#)

The SDK is build around the name-value pairs known as Parameters. The following operations are available on parameters:

GetParameterList

Provides the list of parameter names supported for this product at this time. Use the other methods provided to find out more about each parameter. Note that all products do not support all parameters, so it’s a good idea to get the list and validate that a parameter is available before use.

GetOptions

Returns the valid options for the given parameter. Sends a SOAP fault if the parameter name is invalid.

GetValue

Gets the current value of the parameter. Sends a SOAP fault if the parameter name is invalid. Returns string with value of the selected parameter.

SetValue

Sets the current value of the parameter. Sends a SOAP fault if the parameter name or value is invalid.

IncrementValue

Depending on the parameter type, this method increments numeric value (stopping at max), steps to the next option (looping back to the start), or triggers a change in a state machine.

GetMin

Returns the minimum for the given parameter. Sends a SOAP fault if the parameter name is invalid.

GetMax

Returns the maximum for the given parameter. Sends a SOAP fault if the parameter name is invalid.

GetDescription

Describes the parameter, may include caveats and the like. Can be null, obvious, or very long.

GetCategory

Describes the parameter's category in a language aimed at the developer, may include caveats and the like. Can be null, obvious, or very long. Current list of categories: routing, calibration, LUTs_3D, LUTs_input, framestore, test_pattern_generator, preferences, unit_info, display_process, housekeeping, presets

GetParameterType

Used to determine the type of parameter. There is only one type of parameter at the current time, option_list. Sends a SOAP fault if the parameter name is invalid. Current list of parameters: option_list, point, rect, mapped_address, number, boolean, text, percentage, state_machine,

GetFriendlyName

This is a version of the parameter name that can be used for display. Sends a SOAP fault if the parameter name is invalid.

GetDeveloperNotes

An extended description or special caveat about the parameter in a language aimed at the developer, may include caveats and the like. Can be null, obvious, or very long.

- [File Transfers.](#)

This is a set of file transfers for LUTS and images in standard file formats, directly to and from the hardware, bypassing local storage. The following operations are available for FileTransfers:

GetFramestoreContents

Retrieves the current contents of the current framestore, as a file of the

specified type, in raw base-64-encoded data. Supported types are bmp, tif, dpx, png, jpg, it will default to jpg if filetype is null. Decimation is a simple division, 1 is full-sized, 2 is half, 3 is third, etc.

SetInputLUT

The string should be a valid 1D LUT in a format that the unit understands. The file will not be saved. The LUT number is 1,2,3 or 4 for single-link, 1 or 2 for dual-link.

SetFramestoreContents

Sends the selected still to the framestore. Images larger than the current resolution will be resized down after arrival, smaller images will be matted on black. This uses base-64 encoding (MIME, DIME is not accepted). The image will not be saved.

Set3DLUT

The string should be a valid 3D LUT in a format that the unit understands. The file will not be saved.

- [LUT Service.](#)

This is an optimized method for transferring 3D LUTs to the hardware, bypassing local storage. It bypasses file handling and parsing, using instead raw arrays of floats or packed 10-bit integers. See the “*For Advanced Developers - Using the LUTService*” section of this document for more information about using the LUT Service.

CreateLUT

This method creates a 3D LUT using the specified parameters.

```
uint CreateLUT(
    LUT_Type LutType,
    LUT_EntryType EntryType,
    LUT_ComponentType ComponentType,
    uint LutRes,
    out uint pLUTID,
    uint Flags )
```

LUT_Type LutType - The underlying array type.

1 = Cine-tal hardware format [8][Res][Res][Res],

2 = master format [Res][Res][Res],

3 = linear input format [Res * Res * Res]

LUT_EntryType EntryType - Describes the format of an individual LUT entry.

LUT_ComponentType ComponentType - Describes the type of components that make up an individual LUT entry.

uint LutRes - size of one side of the 3D LUT cube. eg for a 65x65x65 LUT the LUT Resolution is 65.

out uint pLUTID - LUT identifier value assigned to the created LUT.

uint Flags - Flags field for future possible options. Set it to 0 if not used.

DeleteLUT

This method deletes the specified LUT.

```
uint DeleteLUT(  
    uint LUTID,  
    uint Flags )
```

uint LUTID - LUT identifier value that was assigned when the LUT was created.

uint Flags - Flags field for future possible options. Set it to 0 if not used.

InitLUTData

This method initializes the specified LUT data buffer. The type of initialization is specified with the InitType parameter.

```
uint InitLUTData(  
    uint LUTID,  
    uint InitType )
```

uint LUTID - LUT identifier value that was assigned when the LUT was created.

uint InitType - 1 = unity LUT, 2 = all zeros, 3 = all ones

LoadLUTFromArray

This method loads the specified LUT with LUT data from the specified array.

```
uint LoadLUTFromArray(  
    uint LUTID,  
    uint[] LutArray,  
    LUT_EntryType EntryType,  
    LUT_ComponentType ComponentType,  
    uint EntryCount,  
    uint Flags )
```

uint LUTID - LUT identifier value that was assigned when the LUT was created.

uint[] LutArray - Array of uints that should contain LUT data in the format specified with the EntryType and ComponentType.

LUT_EntryType EntryType - Describes the format of an individual LUT entry.

LUT_ComponentType ComponentType - Describes the type of components that make up an individual LUT entry.

uint EntryCount - Number of LUT entries in the specified array. Note: the number of array elements that make one LUT entry depends on the EntryType and ComponentType.

uint Flags - Flags field for future possible options. Set it to 0 if not used.

LoadLUTFromFloatArray

This method loads the specified LUT with LUT data from the specified array.

```
uint LoadLUTFromFloatArray(  
    uint LUTID,  
    float[] LutArray,  
    LUT_EntryType EntryType,  
    LUT_ComponentType ComponentType,  
    uint EntryCount,  
    uint Flags )
```

uint LUTID - LUT identifier value that was assigned when the LUT was created.

float[] LutArray - Array of floats that should contain LUT data in the format specified with the EntryType. ComponentType must be LUT_ComponentType_FLOAT.

LUT_EntryType EntryType - Describes the format of an individual LUT entry.

LUT_ComponentType ComponentType - Describes the type of components that make up an individual LUT entry.

uint EntryCount - Number of LUT entries in the specified array. Note: the number of array elements that make one LUT entry depends on the EntryType and ComponentType.

uint Flags - Flags field for future possible options. Set it to 0 if not used.

ResetLUTs

This method resets the LUT list stored on the server.

```
uint ResetLUTs(  
    uint Flags )
```

uint Flags - Flags field for future possible options. Set it to 0 if not used.

WriteLUTArrayToHardware

This method writes LUT data from the specified array directly to the hardware.

```
uint WriteLUTArrayToHardware(  
    uint[] LutArray,
```

LUT_EntryType EntryType,
LUT_ComponentType ComponentType,
uint EntryCount)

uint[] LutArray - Array of uints that should contain LUT data in the format specified with the EntryType and ComponentType.

LUT_EntryType EntryType - Describes the format of an individual LUT entry.

LUT_ComponentType ComponentType - Describes the type of components that make up an individual LUT entry.

uint EntryCount - Number of LUT entries in the specified array. Note: the number of array elements that make one LUT entry depends on the EntryType and ComponentType.

uint Flags - Flags field for future possible options. Set it to 0 if not used.

WriteLUTFloatArrayToHardware

This method writes LUT data from the specified array directly to the hardware.

uint **WriteLUTFloatArrayToHardware**(
float[] LutArray,
LUT_EntryType EntryType,
LUT_ComponentType ComponentType,
uint EntryCount)

float[] LutArray - Array of floats that should contain LUT data in the format specified with the EntryType. ComponentType must be LUT_ComponentType_FLOAT.

LUT_EntryType EntryType - Describes the format of an individual LUT entry.

LUT_ComponentType ComponentType - Describes the type of components that make up an individual LUT entry.

uint EntryCount - Number of LUT entries in the specified array. Note: the number of array elements that make one LUT entry depends on the EntryType and ComponentType.

uint Flags - Flags field for future possible options. Set it to 0 if not used.

WriteLUTToHardware

This method writes the contents of the LUT buffer to the Cinetal hardware.

uint **WriteLUTToHardware**(
uint LUTID,
uint Flags)

uint LUTID - LUT identifier value that was assigned when the LUT was created.

uint Flags - Flags field for future possible options. Set it to 0 if not used.

get_LUTComponentType

This method gets the current LUT Component type for the specified LUT.

```
LUT_ComponentType get_LUTComponentType(  
    uint LUTID )
```

uint LUTID - LUT identifier value that was assigned when the LUT was created.

get_LUTCount

This method gets the current LUT count.

```
int get_LUTCount()
```

get_LUTEntriesUsed

This method gets the current LUT Entries Used for the specified LUT.

```
uint get_LUTEntriesUsed (  
    uint LUTID )
```

uint LUTID - LUT identifier value that was assigned when the LUT was created.

get_LUTEntrySize

This method gets the Entry size in bits that corresponds to the specified LUT Entry Type and Component Type.

```
uint get_LUTEntrySize(  
    LUT_EntryType LutEntryType,  
    LUT_ComponentType ComponentType )
```

LUT_EntryType EntryType - Describes the format of an individual LUT entry.

LUT_ComponentType ComponentType - Describes the type of components that make up an individual LUT entry.

get_LUTEntryType

This method gets the current LUT Entry type for the specified LUT.

```
LUT_EntryType get_LUTEntryType(  
    uint LUTID )
```

uint LUTID - LUT identifier value that was assigned when the LUT was created.

get_LUTResolution

This method gets the current LUT Resolution for the specified LUT.

```
uint get_LUTResolution(  
    uint LUTID )
```

uint LUTID - LUT identifier value that was assigned when the LUT was created.

get_LUTSizeInBytes

This method gets the current LUT Size In Bytes for the specified LUT.

```
uint get_LUTSizeInBytes(  
    uint LUTID )
```

uint LUTID - LUT identifier value that was assigned when the LUT was created.

get_LUTSizeInEntries

This method gets the current LUT Size In Entries for the specified LUT.

```
uint get_LUTSizeInEntries(  
    uint LUTID )
```

uint LUTID - LUT identifier value that was assigned when the LUT was created.

get_LUTType

This method gets the current LUT type for the specified LUT.

```
LUT_Type get_LUTType (  
    uint LUTID )
```

uint LUTID - LUT identifier value that was assigned when the LUT was created.

Description of examples

A growing number of SDK examples are provided to assist developers. Most of the examples are available for the developer to download, build and run on their development machine.

Online Examples

The online examples are included as part of PhiloWebService and are available to run on any Cine-tal product with the web service enabled. The machine name embedded as part of the URL will need to be changed to match your machine name.

- [Web-based Parameter Example](#). A web-based example that shows all the parameters and allows the user to inspect the properties and change the values. This is a handy way for developers to experiment with the different parameters.

Downloadable Examples

The downloadable examples are stored as compressed .zip files on a server on the main Cine-tal web site. The .zip files contain the source code and project files needed to build the example. Also included in the .zip files for each example is a release executable that can be run on the appropriate platform. Typically the executable will be in the bin\release folder within the .zip file. Each example has a readme.txt file that contains any example-specific information.

Java Examples

- [Java Example](#). A Java/ant example that demonstrates both the FileTransfer and Parameter services. Apache Axis is the framework used to build this example.

Mac Examples

- [Mac OS X Example](#). A Native OS X example, written in Cocoa, that demonstrates both the FileTransfer and Parameter services. The gSOAP framework was used to construct this example.

Windows .NET Examples

- [PhiloWebService FileTransferExample](#) Demonstrates how to use the FileTransfer service to upload 3D or Input LUT files and upload or download still image files. Written for Windows .NET 1.1 in C#.
- [PhiloWebService LUTClientExample](#) Example that shows three different ways to interact with the 3D LUTs on a Cine-tal device. Uses the Parameters, LutService and FileTransfers services. Written for Windows .NET 1.1 in C#.
- [PhiloWebService Pick3DLUT Example](#) Example that shows how to load existing 3D LUT files on a Cine-tal device using the Pick3DLUT parameter. Written for Windows .NET 1.1 in C#.
- [PhiloWebService PickInputLUT Example](#) Example that shows how to load existing Input LUT files on a Cine-tal device using the PickInputLUT parameters. Written for Windows .NET 1.1 in C#.
- [PhiloWebService PickStill Example](#) Example that shows how to load existing still image files on a Cine-tal device using the PickStill parameter. Written for Windows .NET 1.1 in C#.
- [PhiloWebService SimplerLUTClientExample](#) This example shows how to load a 3D LUT from an array in memory to a Cine-tal device using the LutService WriteLUTArrayToHardware call. Written for Windows .NET 1.1 in C#.

How to run examples

After you start running an example program, you will need to provide it with your Cine-tal machine name (so it knows which machine to connect to). The name of your machine should be substituted for Cinetal-example. To get your machine name on a Cinemage, look under Unit information. On an eL 1000, you can get your machine name from the Machine Name section of the Setup window.

For many of the examples, you may need to do some setup of your Cine-tal unit in order to see the program results. E.g. If you are loading 3D LUTs, you need to have the Video Output configured to display the 3D LUT Output. Each example has a readme.txt file that contains any example-specific setup information. The Video Output can be configured on a Cinemage using the Front Panel and on an eL 1000 using the Operate screen.

Application Development

There are a variety of ways to develop an application that works with the Cine-tal products. At a basic level, there are two main areas of functionality that are needed for any application that uses the SDK.

1. Some way to connect to the web server.
2. Some code that does something with one or more of the web services.

.NET Application Development Steps

The steps to create a very simple Parameter-based application like the SDK example PhiloWebService_PickStill_Example are outlined below.

- open Visual Studio 2003 and select File/New/Project
- Select Visual c# projects and use the Windows Application template
- Add Web reference(s) to PhiloWebService services you want to use
- If working on remote machine type the URL for your Cine-tal machine (eg. <http://cintal-123456/PhiloWebService>). If working on the local machine: Select Web services on the local machine.
- Click on Parameters (or FileTransfers or LutService if needed)
- You should get 1 service found. Choose a descriptive Web reference name (eg Parameters) and click on Add Reference
- Add desired controls etc. to your application
- You will need some way for the user to specify the machine path unless you are absolutely positive you will never need your application to work with any Cine-tal except your particular device. The SDK examples include a GroupBox that contains the controls and code to take care of this task.
- Add any other controls and code you desire.

- Compile and run your application

gSOAP Application Steps

gSOAP is a generic cross-platform SOAP client that's used for C++ and C development. gSOAP is an open source Web services development toolkit that offers an XML to C/C++ language binding to simplify the development of SOAP/XML Web services in C and C/C++. The gSOAP source code is available from SourceForge.

[This section currently in progress]

Application Development Notes

Not all units will support all parameters, for example a Cinemage does not always have a framestore or 3D LUT, and will never have calibration on the outputs. So be sure to verify that a parameter is present before using it.

Note that parameter option lists can change dynamically in response to changes in related parameters. E.g. When changing from Dual link to Single link the *Input_3DLUT* Parameter options in a fully-featured Cinemage will change from:

- SDI 1
- SDI 2
- SDI 3
- SDI 4
- TEST PATTERN GENERATOR
- FRAMESTORE
- DVI INPUT

To

- SDI 1&2
- SDI 3&4
- TEST PATTERN GENERATOR
- FRAMESTORE
- DVI INPUT

When dealing with these sort of dynamic changes, the PhiloWebServices makes an attempt to do the “right thing” when parameters have changed. For example, in the above case if the user sends down the routing choice of “SDI 1” when that input is in Dual-link mode then the 3D LUT input will switch to “SDI 1&2” and will return that value when queried.

There may be subtle differences in the way that parameters are presented between different units or code revisions; typically this is done to add features or clarify the options. As an example, with a code revision an existing feature may get a new option (such as Cinemage 2.2 adding DVI input to the above case). Or an option may change when new options are available, for example from “YCbCr 4:2:2” to “YCbCr 4:2:2 SMPTE range” when a new “YCbCr 4:2:2 full range” selection is added.. In this case the unit will attempt to “do the right thing” and support the older parameter.

For the above reasons it’s best to code defensively and expose the raw parameter options to end users, rather than wrapping them. This of course is not possible in all cases.

Parameter Notes

When you run the online Parameters example, one of the first things you notice is the large number of parameters. The relevance of the exposed parameters ranges from generally very useful for most developers to downright obscure. Some of the more useful parameters are listed below.

- **Pick3DLUT** - Lists all 3D LUT files in the current storage location and loads the one selected. *Get* returns the currently active LUT or **RESET** if a passthrough LUT is loaded. *Options* returns all the LUTs in the current storage location, and *set* sets the LUT.
- **PickInputLUT{1-5}** - Lists all input LUT files in the current storage location and loads whichever one is chosen. There is a separate **PickInputLUT** parameter for each input. The input number {usually 1-5} is added to the end of the parameter name. *Get* returns the currently active LUT. *Options* returns all the LUTs in the current storage location, and *set* selects a 3 channel 1D LUT for Input {1-5}.
- **PickPreset** - Select one preset file, it’s loaded on selection. This can be any subset name/parameter in an **ArrayOfParamState** wrapper. Look at an existing preset file for an example. . *Get* returns the currently active preset. *Options* returns a list all the preset files in the current storage location, and *set* selects and loads a preset file.
- **PickStill** - Lists all still image files in the current storage location and loads the one selected. *Get* returns the currently active Still in the current *UserFramestoreNumber* or **RESET** if a passthrough LUT is loaded. *Options* returns all the stills in the current storage location, and *set* loads the framestore. Will return that the storage location is **unavailable** if it is, or null or an empty string if nothing is loaded. May also return a temp file name if *UserFramestorePersistence* is available and set.
- **Input_3DLUT** - Controls input to 3D LUT. *Get* returns the currently selected input, *Options* returns a list all the possible inputs, and *Set* selects an input.

Introduction to the Cine-tal Software Developers Kit

- **Input_Display** - Controls input to display and/or DVI. *Get* returns the currently selected input, *Options* returns a list all the possible inputs, and *Set* selects an input.
- **Input_Framestore** - Controls input to the Framestore. *Get* returns the currently selected input, *Options* returns a list all the possible inputs, and *Set* selects an input.

For Advanced Developers - Using the LUTService

[This section currently in progress]