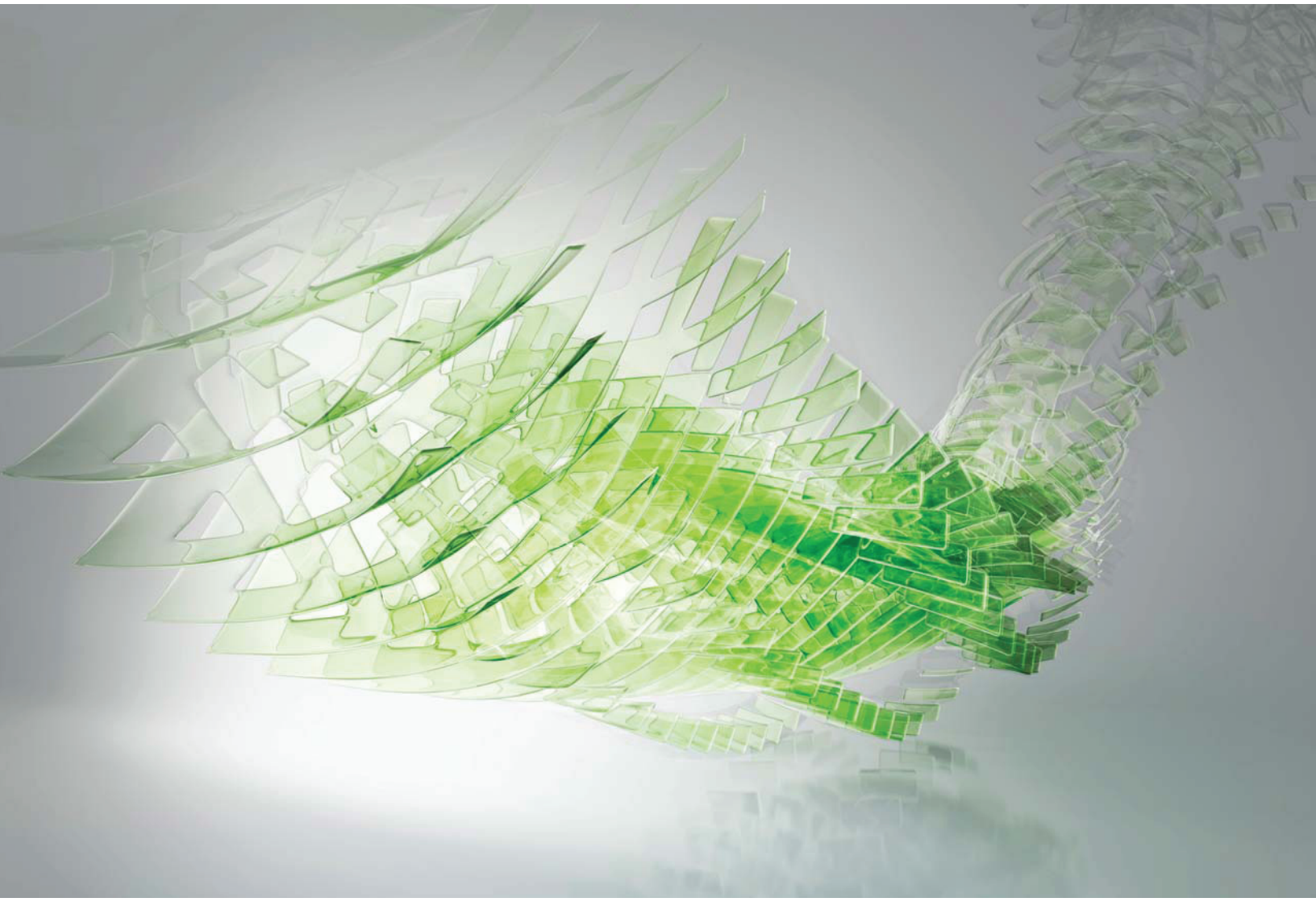
AUTODESK®

Wiretap SDK



Developer Guide

2016-05-24

AUTODESK.

Autodesk Legal Notice

f. "**Original Author**" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

g. "**Work**" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

h. "**You**" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

i. "**Publicly Perform**" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

j. "**Reproduce**" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

- a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;
- b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";
- c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,
- d. to Distribute and Publicly Perform Adaptations.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights described in Section 4(e).

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

- a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly

Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(d), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(d), as requested.

- b. You may Distribute or Publicly Perform an Adaptation only under: (i) the terms of this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-NonCommercial-ShareAlike 3.0 US) ("Applicable License"). You must include a copy of, or the URI, for Applicable License with every copy of each Adaptation You Distribute or Publicly Perform. You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License. You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.
- c. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
- d. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and, (iv) consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(d) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.
- e. For the avoidance of doubt:
  - i. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
  - ii. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for

a purpose or use which is otherwise than noncommercial as permitted under Section 4(c) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,

- iii. Voluntary License Schemes. The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(c).

- f. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING AND TO THE FULLEST EXTENT PERMITTED BY APPLICABLE LAW, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THIS EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

- a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
- b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

- a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
- b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
- c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
- d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
- e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be

bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

- f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at *http://creativecommons.org/*.

Certain materials included in this publication are reprinted with the permission of the copyright holder.

Creative Commons FAQ

Autodesk's Creative Commons FAQ can be viewed online at http://www.autodesk.com/company/creative-commons, and is reproduced here for offline users.

In collaboration with Creative Commons, Autodesk invites you to share your knowledge with the rest of the world, inspiring others to learn, achieve goals, and ignite creativity. You can freely borrow from the Autodesk Help, Support and Video libraries to build a new learning experience for anyone with a particular need or interest.

What is Creative Commons?

Creative Commons (CC) is a nonprofit organization that offers a simple licensing model that frees digital content to enable anyone to modify, remix, and share creative works.

How do I know if Autodesk learning content and Autodesk University content is available under Creative Commons?

All Autodesk learning content and Autodesk University content released under Creative Commons is explicitly marked with a Creative Commons icon specifying what you can and cannot do. Always follow the terms of the stated license.

What Autodesk learning content is currently available under Creative Commons?

Over time, Autodesk will release more and more learning content under the Creative Commons licenses.

Currently available learning content:

- Autodesk online help-Online help for many Autodesk products, including its embedded media such as images and help movies.
- Autodesk Learning Videos-A range of video-based learning content, including the video tutorials on the Autodesk YouTube™ Learning Channels and their associated iTunes® podcasts.

- Autodesk downloadable materials-Downloadable 3D assets, digital footage, and other files you can use to follow along on your own time.

Is Autodesk learning and support content copyrighted?

Yes. Creative Commons licensing does not replace copyright. Copyright remains with Autodesk or its suppliers, as applicable. But it makes the terms of use much more flexible.

What do the Autodesk Creative Commons licenses allow?

Autodesk makes some of its learning and support content available under two distinct Creative Commons licenses. The learning content is clearly marked with the applicable Creative Commons license. You must comply with the following conditions:

- **Attribution-NonCommercial-ShareAlike (CC BY-NC-SA)** This license lets you copy, distribute, display, remix, tweak, and build upon our work noncommercially, as long as you credit Autodesk and license your new creations under the identical terms.
- **Attribution-NonCommercial-No Derivative Works (CC BY-NC-ND)** This license lets you copy, distribute, and display only verbatim copies of our work as long as you credit us, but you cannot alter the learning content in any way or use it commercially.
- **Special permissions on content marked as No Derivative Works** For video-based learning content marked as No Derivative Works (ND), Autodesk grants you special permission to make modifications but only for the purpose of translating the video content into another language.

These conditions can be modified only by explicit permission of Autodesk, Inc. Send requests for modifications outside of these license terms to *creativecommons@autodesk.com*.

Can I get special permission to do something different with the learning content?

Unless otherwise stated, our Creative Commons conditions can be modified only by explicit permission of Autodesk, Inc. If you have any questions or requests for modifications outside of these license terms, email us at *creativecommons@autodesk.com*.

How do I attribute Autodesk learning content?

You must explicitly credit Autodesk, Inc., as the original source of the materials. This is a standard requirement of the Attribution (BY) term in all Creative Commons licenses. In some cases, such as for the Autodesk video learning content, we specify exactly how we would like to be attributed.

This is usually described on the video's end-plate. For the most part providing the title of the work, the URL where the work is hosted, and a credit to Autodesk, Inc., is quite acceptable. Also, remember to keep intact any copyright notice associated with the work. This may sound like a lot of information, but there is flexibility in the way you present it.

Here are some examples:

"This document contains content adapted from the Autodesk® Maya® Help, available under a Creative Commons Attribution-NonCommercial-Share Alike license. Copyright © Autodesk, Inc."

"This is a Finnish translation of a video created by the Autodesk Maya Learning Channel @ *www.youtube.com/mayahowtos*. Copyright © Autodesk, Inc."

"Special thanks to the Autodesk® 3ds Max® Learning Channel @ *www.youtube.com/3dsmaxhowtos*. Copyright © Autodesk, Inc."

Do I follow YouTube's standard license or Autodesk's Creative Commons license?

The videos of the Autodesk Learning Channels on YouTube are uploaded under YouTube's standard license policy. Nonetheless, these videos are released by Autodesk as Creative Commons Attribution-NonCommercial-No Derivative Works (CC BY-NC-ND) and are marked as such.

You are free to use our video learning content according to the Creative Commons license under which they are released.

Where can I easily download Autodesk learning videos?

Most of the Autodesk Learning Channels have an associated iTunes podcast from where you can download the same videos and watch them offline. When translating Autodesk learning videos, we recommend downloading the videos from the iTunes podcasts.

Can I translate Autodesk learning videos?

Yes. Even though our learning videos are licensed as No Derivative Works (ND), we grant everyone permission to translate the audio and subtitles into other languages. In fact, if you want to recapture the video tutorial as-is but show the user interface in another language, you are free to do so. Be sure to give proper attribution as indicated on the video's Creative Commons end-plate. This special permission only applies to translation projects. Requests for modifications outside of these license terms can be directed to *creativecommons@autodesk.com*.

How do I let others know that I have translated Autodesk learning content into another language?

Autodesk is happy to see its learning content translated into as many different languages as possible. If you translate our videos or any of our learning content into other languages, let us know. We can help promote your contributions to our growing multilingual community. In fact, we encourage you to find creative ways to share our learning content with your friends, family, students, colleagues, and communities around the world. Contact us at *creativecommons@autodesk.com*.

I have translated Autodesk learning videos into other languages. Can I upload them to my own YouTube channel?

Yes, please do and let us know where to find them so that we can help promote your contributions to our growing multilingual Autodesk community. Contact us at *creativecommons@autodesk.com*.

Can I repost or republish Autodesk learning content on my site or blog?

Yes, you can make Autodesk learning material available on your site or blog as long as you follow the terms of the Creative Commons license under which the learning content is released. If you are simply referencing the learning content as-is, then we recommend that you link to it or embed it from where it is hosted by Autodesk. That way the content will always be fresh. If you have translated or remixed our learning content, then by all means you can host it yourself. Let us know about it, and we can help promote your contributions to our global learning community. Contact us at *creativecommons@autodesk.com*.

Can I show Autodesk learning content during my conference?

Yes, as long as it's within the scope of a noncommercial event, and as long as you comply with the terms of the Creative Commons license outlined above. In particular, the videos must be shown unedited with the exception of modifications for the purpose of translation. If you wish to use Autodesk learning content in a commercial context, contact us with a request for permission at *creativecommons@autodesk.com*.

Can I use Autodesk learning content in my classroom?

Yes, as long as you comply with the terms of the Creative Commons license under which the learning material is released. Many teachers use Autodesk learning content to stimulate discussions with students or to complement course materials, and we encourage you to do so as well.

Can I re-edit and remix Autodesk video learning content?

No, but for one exception. Our Creative Commons BY-NC-ND license clearly states that "derivative works" of any kind (edits, cuts, remixes, mashups, and so on) are not allowed without explicit permission from Autodesk. This is essential for preserving the integrity of our instructors' ideas. However, we do give you permission to modify our videos for the purpose of translating them into other languages.

Can I re-edit and remix Autodesk downloadable 3D assets and footage?

Yes. The Autodesk Learning Channels on YouTube provide downloadable 3D assets, footage, and other files for you to follow along with the video tutorials on your own time. This downloadable material is made available under a Creative Commons Attribution-NonCommercial-ShareAlike (CC BY-NC-SA) license. You can download these materials and experiment with them, but your remixes must give us credit as the original source of the content and be shared under the identical license terms.

Can I use content from Autodesk online help to create new materials for a specific audience?

Yes, if you want to help a specific audience learn how to optimize the use of their Autodesk software, there is no need to start from scratch. You can use, remix, or enrich the relevant help content and include it in your book, instructions, examples, or workflows you create, then Share-Alike with the community. Always be sure to comply with the terms of the Creative Commons license under which the learning content is released.

What are the best practices for marking content with Creative Commons Licenses?

When reusing a CC-licensed work (by sharing the original or a derivative based on the original), it is important to keep intact any copyright notice associated with the work, including the Creative Commons license being used. Make sure you abide by the license conditions provided by the licensor, in this case Autodesk, Inc.

Trademarks

Disclaimer

Contents

Welcome

1

Welcome to the *Autodesk Wiretap SDK Developer Guide*. This guide describes Wiretap and explains how to use the Wiretap Client API to write client applications and modules that implement typical project/user and media management workflows.

Wiretap is a cross-platform client-server interoperability framework, providing high-performance access to remote media and metadata. Wiretap is used internally by Autodesk applications, and is available to third-party developers by way of the Wiretap client application programming interface (API).

The client API is packaged as a software developer kit (SDK). Its libraries and other tools allow developers to write stand-alone applications that communicate with the remote Wiretap servers in a facility, for the purpose of leveraging the server's functionality. The particular actions your application can perform depend on the type of Wiretap server being accessed. The following Wiretap server types are available.

Wiretap Server	Description
IFFFS	Exposes the database or clip libraries of Visual Effects and Finishing applications without the need for a running application. Exposed database objects include projects, libraries, clips, and media. Typical uses include creating and setting up new projects, creating users and setting user preferences, adding clips to clip libraries, and others. The SDK also includes a command-line tool for performing media transfer and conversion jobs in the background, moving media onto any machine running Autodesk Stone and Wire.
Gateway	The Wiretap Gateway is a universal media gateway, designed to read image media on standard FS filesystems, in any supported format, and stream it live as raw RGB to Wiretap clients. It provides client applications with ingest access to supported media. For example, RED REDCODE (.r3d) files on all available devices on the Wiretap network.
Backburner	Exposes the Backburner Manager's database of jobs, servers, server groups, and others. The Backburner Wiretap server makes it possible to create a client application to submit, monitor and control rendering jobs and I/O on the network.

A Wiretap server is provided with the following Autodesk applications:

- Flame
- Flame Assist
- Flare
- Lustre (Wiretap Gateway only)
- Backburner

The Wiretap SDK is available in C++ and Python, and supports Windows, Mac OS X and Linux platforms.

Supported Operating Systems and Platforms

You can build and run Wiretap clients on the following operating systems and platforms.

OS	Platform	Bits	Compiler	Compiler Version
Linux	x86-64	64	GCC	4.4.6
Mac OS X	Intel	64	GCC	5.1
Windows	x86	32	MSVC	9.0 / 10.0 / 14.0
Windows	x86-64	64	MSVC	9.0 / 10.0 / 14.0

Components of the Wiretap Client SDK

The components of the Wiretap Client SDK are contained in the following directories within the Wiretap SDK package:

Directory	Description
api	Contains the*WireTapClientAPI.h* and *WireTapTypes.h* header files. You must include these files when compiling code that uses the Wiretap Client API.
doc	Contains the reference documentation for the C++ version of the API, in HTML.
lib	Contains the static and dynamic versions of the API libraries. The library files in this directory depend on the platform-specific version of the Wiretap Client SDK.
samples	Contains sample programs that demonstrate the common applications of the Wiretap Client API.
tools	Contains command-line tools that call key functions of the client API. These tools allow you to experiment with the API and to troubleshoot the Wiretap clients during development. These are the same tools as those found in */usr/discreet/wiretap/tools/current*.

NOTE While we take steps to preserve the forward- and backward-compatibility of the API (and advertise changes and deprecation when required), we cannot take such steps with the command line tools provided here. As such, they should only be used for testing or prototyping. Do not design mission critical software around the tools. Use the API.

Understanding Wiretap 2

This section explains the Wiretap architecture including the IFFFS Wiretap server, Wiretap clients, and storage components.

Wiretap Terminology

You need to know the following terms to clearly understand the information in this guide.

Wiretap

Wiretap is a cross-platform client-server interoperability framework that provides high-performance access to remote media and metadata using the Wiretap API. For example, you can examine and manipulate the clip library metadata using a common interface in Wiretap. Frame data exchange services are also provided using optimized, high-performance communication protocols.

Wiretap Server

A Wiretap server is a service that exposes a proprietary or public database as a tree-like hierarchy of Wiretap nodes.

IFFFS Exposes the proprietary Creative Finishing clip library database as a hierarchy of projects, workspaces, libraries, clips, and others.

Gateway Exposes the contents of filesystems as a Wiretap hierarchy of directories, clips, files, and other objects.

Backburner Exposes the Backburner Manager's database of jobs, servers, server groups, and others.

In each case, the Wiretap server presents the database contents in a consistent and predictable manner. A Wiretap server is typically a daemon running on a host machine.

Wiretap Client

A Wiretap client is a program that uses the Wiretap Client API to communicate with a Wiretap server. A Wiretap client can be a desktop or web application, an adapter/plug-in for another application, a Python module, or even a batch file. Generally, a particular Wiretap client communicates with a particular Wiretap server implementation only. For example, the IFFFS Wiretap server, Wiretap Gateway server, or Backburner Wiretap server. Typically, the client uses the services offered by the server to push or pull media across the network. For Backburner, the client can initiate jobs, monitor jobs, and so on.

Nodes and Node IDs

A node is a single element in the hierarchy of a database exposed by a Wiretap server. For example, an IFFFS node can be a project, library, or clip contained in the clip library database of its associated Creative Finishing application. Each node is identified by a node ID (a string that uniquely and persistently identifies each node on a particular Wiretap host).

Frame

A frame is a data buffer representing a single image (for video frames) of a clip node. You can frames for any data type including audio and formatted image data. A frame can represent an individual image or a tile of a stream such as, an audio stream.

Frame Format

A frame format is a set of parameters needed to decode or interpret frame data. The parameters include a format tag used to identify the frame data encoding algorithm. For example, *rgb* or *aiff*.

Metadata

Metadata is a node-specific ASCII data stream. The metadata stream can be in EDL or XML. The stream syntax is dependent on the Wiretap server. See Media and Metadata Formats (page 37) for descriptions of the syntax of EDL and XML streams.

Roles of the Wiretap Server and Client

The function of a Wiretap server is to expose public and/or proprietary databases (sometimes representing storage) as a uniform structure, typically a hierarchy of node types encapsulating meta and media data. The server provides a common set of access methods, which allows Wiretap clients to access the local or remote server without knowledge of the server's underlying structure.

The Wiretap Client API defines a common interface and abstracts the underlying communication with Wiretap server. Similarly, the Wiretap server API abstracts connectivity with Wiretap clients.

As a general rule, the performance of a Wiretap server will not have a negative impact on the performance of any concurrently running applications on the Wiretap host machine. Specifically, a Wiretap server should defer all heavy processing (for example, rendering and file format conversions) to Wiretap clients so the server remains responsive to requests from clients.

The Wiretap infrastructure is responsible for all media and metadata exchange, cross-platform issues, and protocol versioning compatibility.

About Wiretap Gateway Supported Ingest File Formats

The image and audio file formats supported by the Wiretap Gateway server, for ingest, are available from the Flame user guide. Use *these tables in the Flame user guide* to determine if a particular digital image sequence or container format can be recognized by the Wiretap Gateway.

An image sequence is a series of sequentially numbered files, traditionally the result of scanning film stock at high resolution to produce a digital intermediate. Here, each file contains the digital scan of an individual frame. Common image sequence formats include Cineon®, DPX, OpenEXR, and Tiff. The type of image sequence file on hand is usually revealed by its extension.

In contrast, container formats, also called "wrapper" formats, can contain image sequences (commonly called *streams* or *essences*) and audio, compressed using a variety of compression algorithms (codecs) into a

single file. Container formats do not impose specific video or audio codecs upon the media they contain. Rather, a container format defines only how the video, audio and other data is stored within the container itself. Unlike image sequences, it is not possible to tell by looking at the extension what kind of video or audio is inside a container format. Common streamed formats are QuickTime, MXF, and RED (R3D).

The Wiretap Gateway server also provides basic support for sequence formats such as AAF and EDL.

Getting Started 3

You must install the Wiretap Client SDK on your computer by unzipping the SDK package to follow the instructions in this section.

NOTE The directory in which you have installed the Wiretap Client SDK is referred to as the *wiretap_install_dir* in this guide.

Three Ways to Use the Wiretap Client API

You can use the API in three ways:

- **Command line tools** – These tools are recommended for everyone because they allow you to see the API in action immediately. See Using the Command Line Tools (page 7).
- **C++ classes** – Experienced C++ developers can use these classes to program their own Wiretap clients. See Setting Up Your Environment for C++ Developers (page 10).
- **Python modules** – These modules are the basis for writing scripts that can be run immediately without compilation. See Setting Up Your Environment for Python Developers (page 11).

Using the Command Line Tools

The *tools* directory in the Wiretap Client SDK contains a number of executable programs that you can run from the command line. They are useful for becoming familiar with what the API can do. You can use them for troubleshooting as you develop your own Wiretap client, which can be a C++ application or a Python module. You can compare your results with the results returned by the tools. You can also use the command line tools if you want to access Wiretap servers without having to program or script your own Wiretap client.

This section contains some general information about the command line tools and recommends several tools that you can try to become familiar with Wiretap.

NOTE While we take steps to preserve the forward- and backward-compatibility of the API (and advertise changes and deprecation when required), we cannot take such steps with the command line tools provided here. As such, they should only be used for testing or prototyping. Do not design mission critical software around the tools. Use the API.

Location of Command Line Tools

You will find the tools in this directory:

wiretap_install_dir/tools/platform_dirs

where,

- *wiretap_install_dir* is the directory where you installed the Wiretap Client SDK.
- *platform_dirs* is one or more nested directories corresponding to the version of the SDK you installed (Linux, Mac OS X, Windows).

Options and Help

Most of the command line tools accept options. Help is available for all the command line tools.

The most common option is `-h` for host. This option is available if a tool connects to a particular Wiretap server. Note the following points:

- If you use the `-h` option, you can enter the name or IP address of a Wiretap host.
- You can specify the type of server to query, using the `Host:Server` syntax, such as `localhost:Backburner`. Possible server values are `IFFFS`, `Gateway`, `Backburner`. Default is `IFFFS`. Be careful with the Gateway, as it exposes the whole filesystem structure.
- If you do not use the `-h` option to specify a host, the tool attempts to connect to the *localhost*. This works if you are working on a computer that is running a Wiretap server.

To view help for a command line tool:

- Enter the `--h` option after the command.

  The help for a particular command lists the options for the command.

Pinging a Wiretap Server

As an initial check, to see if the Wiretap Client SDK is installed properly and that you can access a Wiretap server, you can run the command line tool `wiretap_ping`. You will find `wiretap_ping` in a platform-specific subdirectory of the tools directory under your *wiretap_install_dir*.

To run ping:

1. Determine the IP address or name of a Wiretap host.

   It can be your own computer or any computer on which a Visual Effects and Finishing application is installed. If a Visual Effects and Finishing application is installed, the computer usually runs a Wiretap server as a daemon.
2. Enter the following command in a shell or terminal window:

   ```
   pathToTools/wiretap_ping -h <host> [:<database>]
   ```

where,

- `pathToTools` is a platform-specific subdirectory of the tools directory under your *wiretap_install_dir* directory.
- `host` is the host name or IP address.
- `database` is the database type of the server (`IFFFS` is the default).

Listing Wiretap Servers on the Network

The command line tool `wiretap_server_dump` displays a list of all the Wiretap hosts on the network to which your development machine is connected.

To list servers:

- Enter this command in a shell or command prompt:

  ```
  pathToTools/wiretap_server_dump [-p] [-U] [-d <db type>]
  ```

where,

- `pathToTools` is a platform-specific subdirectory of the tools directory under your *wiretap_install_dir*.
- `-p` displays server ports.
- `-U` displays the Host UUIDs.
- `-d` filters server of a given database type.

A list of Wiretap servers is displayed. For each server, the following information is displayed.

Column	Description
Wiretap Server	The server's display name. For example:`reykjavik`
Host UUID	Displayed with the `-U` switch. Identifies which server runs on a machine. Use the host UUID in cases where the `hostname` can change as it identifies the machine without relying on network configuration. See `/usr/discreet/cfg/network.cfg` for additional details.
Storage ID	An identifier (unique in your network) for the storage device connected to the server. For example: `IFFFS-162`

Plug-In	This column lists three pieces of information about the server:
	■ vendor
	■ implementation (for example, IFFFS, Wiretap Gateway or Backburner)
	■ version
	
	For example:
	`Autodesk IFFFS 2017.0`
	`Autodesk Wiretap Gateway Server 2017.0`
	`Autodesk Backburner 2017.0`

Displaying a Wiretap Server's Node Hierarchy

A Wiretap server uses a hierarchy of nodes to represent the directory structure of an underlying database. The command line tool `wiretap_print_tree` displays a tree of the nodes on a Wiretap host in text format. The command accepts a number of options which are explained in the procedure below.

To view a Wiretap server's node hierarchy:

1. Enter this command:

   ```
   wiretap_print_tree [-h host] [-d depth] [-n nodeID] [-k]
   ```

   where,

   - `host` is the host name or IP address. If you do not specify a host, it will default to *localhost*. You can specify the server type. If you have trouble, see Pinging a Wiretap Server (page 8).
   - `depth` is the depth to which you want to display nodes. The default is 4.
   - `nodeID` is the unique ID of the node at which you want print tree to start. For example, you could enter `/projects/<projectName>` to see the nodes under a particular project. The first time you enter the command, you cannot enter a node ID because you do not know any. However, node IDs are displayed when you run the command for the first time, so you can use them in the next step.
   - `-k` can be used to continue on error. This option might be useful if some clip does not load correctly but you want to still print out the complete hierarchy.

Things to experiment:

1. Display a branch of the hierarchy by running the command with the `-n` option using one of the node IDs displayed when you ran the command before.
2. Run with different depths (the `-d` option) to view the hierarchy in more or less depth.

Setting Up Your Environment for C++ Developers

This section provides the information you need to get started developing C++ Wiretap client applications. It provides the names and locations of the header files and libraries needed, and provides sample compiler commands for Linux and Mac OS X. It also presents the names and locations of the predefined project files provided for Windows developers working in Microsoft® Visual Studio®.

Location of Wiretap C++ Header Files

These two header files are in the `api` subdirectory of your *wiretap_install_dir*:

- `WireTapClientAPI.h`
- `WireTapTypes.h`

Location of Wiretap C++ Library Files

The following table lists the names and paths of the library files for your platform.

OS	Platform (Bits)	Compiler/ Version	Install path to library files	Library files
Linux RHEL 6	x86 (64)	GCC 4.4.6	wiretap_install_dir/lib/opt/LINUX/ x86_64/RHEL6/GCC_4_4_6	libwiretapClientAPI.a libwiretapClientAPI.so
Mac OSX	Intel (64)	GCC 5.1	wiretap_install_dir/lib/opt/MA-COSX/ fat/10_9_5/GCC_5_1	libwiretapClientAPI.a libwiretapClientAPI.dylib
Windows	x86 (32)	MSVC 9.0	wiretap_install_dir/lib/opt/WINNT/ x86_32/WinXP/MSVC_90	libwiretapClientAPI.lib libwiretapClientAPI_dynamic.dll libwiretapClientAPI_dynamic.exp libwiretapClientAPI_dynamic.lib
Windows	x86 (32)	MSVC 10.0	wiretap_install_dir/lib/opt/WINNT/ x86_32/WinXP/MSVC_100	libwiretapClientAPI.lib libwiretapClientAPI_dynamic.dll libwiretapClientAPI_dynamic.exp libwiretapClientAPI_dynamic.lib
Windows	x86 (32)	MSVC 14.0	wiretap_install_dir/lib/opt/WINNT/ x86_32/WinXP/MSVC_140	libwiretapClientAPI.lib libwiretapClientAPI_dynamic.dll libwiretapClientAPI_dynamic.exp libwiretapClientAPI_dynamic.lib
Windows	x86 (64)	MSVC 9.0	wiretap_install_dir/lib/opt/WINNT/ x86_64/WinXP/MSVC_90	libwiretapClientAPI.lib libwiretapClientAPI_dynamic.dll

OS	Platform (Bits)	Compiler/ Version	Install path to library files	Library files
				libwiretapClientAPI_dynamic.exp libwiretapClientAPI_dynamic.lib
Windows	x86 (64)	MSVC 10.0	wiretap_install_dir/lib/opt/WINNT/x86_64/WinXP/MSVC_100	libwiretapClientAPI.lib libwiretapClientAPI_dynamic.dll libwiretapClientAPI_dynamic.exp libwiretapClientAPI_dynamic.lib
Windows	x86 (64)	MSVC 14.0	wiretap_install_dir/lib/opt/WINNT/x86_64/WinXP/MSVC_140	libwiretapClientAPI.lib libwiretapClientAPI_dynamic.dll libwiretapClientAPI_dynamic.exp libwiretapClientAPI_dynamic.lib

Command Line for Mac and Linux

The compiler command must be structured as follows:

```
g++ test.C -o -I /includePath <libPath>/ libwiretapClientAPI.a
```

where,

- `test.C` is a sample program in the `samples/cpp` subdirectory of your `wiretap_install_dir`.
- `includePath` is the path to the Wiretap header files. The default path would be:

  `wiretap_install_dir/api`
- `<libPath>` is the path to the static library file `libwiretapClientAPI.a`. Specify the path to the library file in the command line or add it to the system path, as you wish. The default paths for different operating system/compiler combinations are listed in the table above.

NOTE If you are developing for Mac OS X, you might encounter problems when you attempt to run your Wiretap Client. See Problems executing your application under Mac OS X (page 84).

Setting Up Your Environment for Python Developers

You can use Python to write a Wiretap client that can run immediately, without compiling.

If you need to become familiar with Python, visit the web site: *http://www.python.org/*

Your Wiretap client will be a Python module (a *.py* file). To run your module, you will need a dynamic library that provides Python bindings to the C++ version of the Wiretap Client API. Check the next section to see if a Python library is available for your platform. If there is, you can proceed to Setting Up the Python Environment (page 12).

Availability of Python API

There is a pre-compiled version of the Wiretap Client API that was compiled with Python 2.7. It is only available for these platforms:

- Linux
- Mac OS X

Location of the Python Libraries

You will find the appropriate version of the Wiretap dynamic library for Python in your *wiretap_install_dir*. The following table indicates the names of the library files and the path to these files.

There is no library file for Windows. You need to compile a `.dll` as described in Setting Up the Python Environment (page 12).

OS	Platform (Bits)	Compiler/ Version	Path to Library Files	Wiretap Library File for Python
Linux RHEL 6	x86-64(64)	GCC 4.4.6	wiretap_install_dir/lib/opt/LINUX/ x86_64/RHEL6/GCC_4_4_6/Python2.7	libwiretapPythonClientAPI.so
Mac OSX	Intel(64)	GCC 5.1	wiretap_install_dir/lib/opt/MACOSX/ fat/10_9_5/GCC_5_1/Python*	libwiretapPythonClientAPI.dylib

NOTE On the Mac, three versions of the Wiretap library file are available. Python2.7, Python2.7.system, and Python2.6.system. If you plan on building something that includes the Creative Finishing Python hooks, you must use the Python2.7 distribution. If not, use the .system distribution that matches you Mac OS X Python distribution.

Setting Up the Python Environment

To use a version of Python other than the recommended version 2.7, or if your OS does not come with Python (Windows), set up your development environment using the following steps:

1. Ensure a dynamic library for *boost* (C++ extensions) is installed on your system:
   - If you are working on Linux or Mac OS X, a boost dynamic library might have already been installed on your system.
   - If you do not have a boost library, you will need to build one from the sources available at:

   For more details, see *http://www.boost.org*.
2. Ensure Python is installed on your system. Preferably, it should be version 2.7, because the dynamic library (that defines Python bindings for Wiretap) works correctly with it. Python 2.6 might work, Python 3.0 will not.

   If you need to get Python, go to the Python web site:

   http://www.python.org/
3. If you want to use a version of Python other than the pre-compiled version, you must regenerate the Wiretap dynamic library for Python as follows:
   - Compile *wiretapPythonClientAPI.C* (in the `samples` directory of the Wiretap Client SDK).
   - Specify the boost library (from Step 1) in your compile command.
   - The resulting library must be named `libwiretapPythonClientAPI` (with a platform-appropriate extension: `.so` for Linux, `.dylib` for Mac OS X, `.dll` for Windows) and must be installed as explained in the rest of this procedure.
4. Check Location of Python Libraries (page 12) to determine the location of the appropriate version of the Wiretap dynamic library for Python.
5. Ensure that the library files (for *boost* and `libwiretapPythonClientAPI`) are found at runtime in either of these standard ways:
   - Add the paths of the library files to your system path

     or
   - Install the library files in Python's library directory (which is platform-dependent):

     - Linux 32-bit: */usr/lib/python2.7/lib-dynload*

- Linux 64-bit: */usr/lib64/python2.7/lib-dynload*
- Mac OS X: */usr/lib/python2.7/lib-dynload*
- Windows: *C:\Python27\DLLs*

Running Python Modules

Once you have ensured that Python is installed and can find the required libraries, you can run the sample Python modules (*.py* files) included in the Wiretap Client SDK.

- Open a shell or command prompt and enter a command like this:

```
python wiretap_install_dir/samples/python/moduleName.py
```

where,

- `wiretap_install_dir` is the directory where you installed Wiretap.
- `moduleName` is the name of the Python module that you want to run.

Accessing Documentation for the Python API

The Wiretap Client SDK does not include documentation specifically for the Python version of the API. However, you can view the list of classes and their member functions in the API by using the Python commands `dir` and `help` as shown below.

To get help for the Python API:

1. Start Python, or open a shell or command prompt and enter:

   ```
   python
   ```

   The python prompt (>>>) appears.
2. To import the Wiretap API (with an alias), enter:

   ```
   import libwiretapPythonClientAPI as wiretap
   ```
3. To view a list of the classes in the API, enter:

   ```
   dir(wiretap)
   ```
4. To view the members of a particular class in the API, enter:

   ```
   help (wiretap.WireTapServerHandle)
   ```

   For detailed information about the member functions of a class, you must use the Wiretap C++ API reference documentation.

To get more information about the methods of a class:

- Consult the C++ API reference documentation. When you read the C++ version of the documentation, you must be aware of the differences between the two versions of the API, which are explained below.

Differences between the Python API and the C++ API

The Python API is designed to resemble the C++ API as much as possible. However, there are a few differences between the C++ and Python versions of the API. Unlike C++, Python does not support pointers and references. Python uses objects in situations where C++ would use pointers and references.

In the Wiretap Client API, some C++ accessor methods have output parameters that pass references to integers. The equivalent Python methods pass an instance of `WireTapInt`, which is used to represent the int base type.

Affected Classes and Methods

These are the Python method declarations that differ from the equivalent C++ declarations:

```
class WireTapServerHandle
  bool getVersion( WireTapInt &major, WireTapInt &minor ) const
  bool getProtocolVersion( WireTapInt &major, WireTapInt &minor ) const

class WireTapNodeHandle
  bool getNumAvailableMetaDataStreams( WireTapInt &numStreams ) const
  bool getNumChildren( WireTapInt &numChildren ) const
  bool getNumFrames( WireTapInt &numFrames ) const
  bool getNodeType( WireTapInt &type ) const
  bool linkToFrames( python::list pathList )

class WireTapServerList
  bool getNumNodes( WireTapInt &numberOfNodes )
```

Using the Sample Programs

The *samples* directory in the SDK contains a number of simple programs, each of which demonstrates how to program some basic functionality using the Wiretap Client API. You can treat these programs as building blocks when you are developing your own Wiretap client: cut, paste, and adapt code from these samples into your own C++ program or Python module.

This section recommends two sample programs that you can read, build (for the C++ version), and run. They will help you become familiar with a few classes and ways of doing things using the Wiretap Client API. After you have done so, you can look at Programming Typical Workflows (page 17), which covers basic programming issues and goes into more depth on how to program a variety of workflows.

NOTE Most of the C++ sample programs look for environment variables on your system. The programs supply default values if the environment variables are not found. If a sample program does not work on your system, you might need to set some environment variables or hard-code suitable values in the sample program.

Trying the `listAllServers` Sample

The Wiretap Client SDK includes a sample C++ program (*listAllServers.C*) and a Python module (*listAllServers.py*) that displays a list of all the Wiretap hosts on the network to which your development machine is connected.

The functionality shown in *listAllServers.C* is almost identical to that of the command line tool `wiretap_server_dump`.

When you examine the code in *listAllServers*, you will notice that it does the following:

1. Initiates the Wiretap Client API.

   All Wiretap clients must start with a call to the global function, `WireTapClientInit` or instantiate a `WireTapClient` guard object.

2. Uses a `WireTapServerList` object to determine Wiretap servers that can be accessed from the client.

   Wiretap uses network multicast technology to broadcast Wiretap server nodes to each other, allowing any Wiretap client to obtain a list of active servers. The Wiretap Client API silently finds one server on startup through which it gains knowledge of all others.

3. Iterates through all the Wiretap servers that are accessible. For each Wiretap server, it obtains a `WireTapServerInfo` object which gives access to:
   - Properties of the server (the sample only gets its display name, IP address, and database)

Your Wiretap client can populate a browser with any or all of the available information.

- A `WireTapServerId` object

  In your Wiretap client, you can use this `WireTapServerId` to instantiate a `WireTapServerHandle` which is a live connection to a particular Wiretap server and gives access to its node hierarchy.

4 Uninitiates the Wiretap Client API.

  All Wiretap clients must end with a call to the global function, `WireTapClientUninit` or destroy `WireTapClient` guard object if it was explicitly instantiated.

Trying the *listChildren* Sample

The Wiretap Client SDK includes a sample C++ program (*listChilden.C*) and a Python module (*listChildren.py*) that display a list of the child nodes of a particular node on a particular Wiretap server.

The functionality demonstrated in *listChildren* is almost identical to that of the command line tool `wiretap_get_children`.

When you examine the code in *listChildren*, you will notice that it does the following:

1 (C++ sample only) Establishes a namespace for this Wiretap host using its host name.

  As explained earlier, if a Creative Finishing application is installed on your machine, the IFFFS Wiretap server is also installed, and usually it is running. In this case, the default *localhost* allows you to run *listChildren.C*. If the default value does not work, you can set the host as an environment variable or hard-code the name of a valid host.

2 Initiates the Wiretap Client API.

  All Wiretap clients must start with a call to the `WireTapClientInit` global function or instantiate a WireTapClient guard object.

3 Instantiates a `WireTapServerHandle` using the host name.

  An instance of `WireTapServerHandle` is an active connection to a particular Wiretap server and gives access to its node hierarchy.

4 Gets the root node of the Wiretap server and finds out how many child nodes it has.

  The root node of an IFFFS Wiretap server is named /.

5 Gets a `WireTapNodeHandle` object for each of the root node's children. For each child node, it displays:

  - Its display name
  - Its node type

    The children of the root node on an IFFFS Wiretap server are VOLUME type nodes. The default VOLUME node is usually named *AutodeskMediaStorage*. The sample program only gets one level of children.

    Your Wiretap client can use these `WireTapNodeHandle` objects to populate a browser of the server's node hierarchy. By nesting calls to `getNumChildren` and `getChildNode`, your client can drill down through the entire node hierarchy. You can allow the user of your Wiretap client to select the branch to view. For each node, you can display as much information as useful for the user.

6 Uninitiates the Wiretap Client API.

  All Wiretap clients must end with a call to the `WireTapClientUninit` global function or destroy the WireTapClient guard object..

Basic Programming Issues

This section explains how Wiretap handles some basic programming issues. You may also want to look at the FAQs in General API Issues (page 81).

Namespace

All internal symbols of the Wiretap Client and Server APIs are protected by the WireTap prefix. Both the C++ and Python versions of the API use the prefix WireTap in the names of all classes and global functions.

Errors

Error messages generated by Wiretap are intended to be viewed by end users. Each Wiretap server implementation generates its own errors. Often, the error strings are generated dynamically and include contextual information.

Three classes fetch and manipulate data on a Wiretap server:

- `WireTapNodeHandle`
- `WireTapServerHandle`
- `WireTapServerList`

An error message is issued if a failure occurs during calls to any of their member functions that interact with a Wiretap server. Each of the above-mentioned classes has a member function called `lastError()` that looks for the error message. The error string must be used or stored immediately, since it will be overwritten the next time a member function is called.

All member functions that communicate with a Wiretap server, return a boolean value: true on success, false on failure. As shown in the sample programs (for example, *listChildren.C* and `wiretap_get_children.py`), this return value should be checked and, if it is false, `lastError()` must be called.

Strings

String Return Values – Wiretap guarantees that all methods that return string pointers will return a valid string (never a null pointer).

Custom String Class – API libraries often redefine *standard* library data types (like `std::string`) to avoid problems when a host application chooses a different standard C library from the one used by the API library. The Wiretap API includes the `WireTapStr` class for this reason. This means that Wiretap clients must duplicate strings when converting from `WireTapStr` to their own string class for internal use. `WireTapStr` is also needed for the Python version of the Wiretap API. Python does not allow basic types like string to be passed by reference. A `WireTapStr` object can be passed to methods that need to return strings in an output parameter.

Threads

Wiretap node handles and server handles can exist in different threads, but a single handle object cannot be used simultaneously in several threads. For example, multiple threads can be used to traverse a node hierarchy, but they cannot share the same node or server handles.

The one exception to this rule is the `WireTapNodeHandle.stop` function, which is intended to be used in an asynchronous thread to halt a pending request on a handle.

Localization

The Wiretap server API is internationalized so that your Wiretap client can be compiled for any language or locale. Error messages received from the operating system will be localized. Only the English version of the IFFFS Wiretap server is installed with Visual Effects and Finishing applications so its errors are in English.

Programming Typical Workflows

4

This section explains how to write Wiretap clients that implement typical project, user, and media management workflows.

A Wiretap server presents the database it exposes as a navigable hierarchy of nodes. The IFFFS Wiretap server exposes an IFFFS database, which is a clip library as a collection of project nodes, library nodes, and other related objects. By operating upon IFFFS Wiretap server nodes and node metadata, your client application can create, copy and populate projects and user setups, clips, libraries and reels. The Wiretap Gateway server presents the filesystem as directory nodes, clip nodes, timeline nodes, and so on. The nodes of the Wiretap Gateway server provide additional media ingest functionality.

This section covers workflows related to the IFFFS Wiretap server and the Wiretap Gateway server. The workflows are grouped by object type such as, projects, users, audio or video clips, and so on.

For the Backburner Wiretap server, see Backburner Wiretap Server (page 57).

Discovering Wiretap Servers

The first step in any workflow is locating the desired Wiretap server. To assist in this task, Wiretap is designed to automatically discover all the Wiretap servers available on the local network segment, using multicast addressing. You obtain the list of discovered servers using the `WireTapServerList` class. Upon instantiation of an object of this class, Wiretap silently and automatically finds one server through which it gains knowledge of all the others.

For more information on the Wiretap server list, see the FAQs:

- Why do I see a Backburner server in my server list? (page 82)
- Why can't I see all Wiretap servers? (page 82)

Viewing Every Wiretap Server on the Network

Related C++ Sample

- `listAllServers.C`

Related Python Sample

- `listAllServers.py`

Related Command Line Tool

- `wiretap_server_dump`

See also Trying the listAllServers Sample (page 14).

Understanding Server IDs

Each Wiretap server is identified by a unique a server ID, used by the client application to gain access to its associated database. For example, an IFFFS Wiretap server running on a host named *montreux* is discovered as *montreux:IFFFS*. This combination of host and database type results in unique server IDs, and is especially helpful when Wiretap servers for different Autodesk products are running on the same host.

When specifying the ID for a server already known to you, the API offers flexibility in how you construct it. You can specify the first component of the ID (host machine) by either its name or IP address. Similarly, you can specify the database type by its name such as IFFFS, Gateway, and Backburner, or by its associated TCP port.

Wiretap Server TCP Ports

Each Wiretap server type makes use of two TCP ports. The first is for the high-bandwidth frame I/O activities, and is reserved for the exclusive use of Wiretap itself, internally. The second is reserved for the low-bandwidth data associated with metadata, and is used by your client application for all operations including traversing the node hierarchy. Different ports are used by each server type. The IFFFS, Wiretap Gateway, and Backburner wiretap servers all use different ports.

The ports are set in the Wiretap configuration associated with the Wiretap server of interest by the person responsible for installing or maintaining the servers. No configuration is required on the client side. When you know the server's low-bandwidth TCP port, you can simply use that instead of the database name, when specifying the Server ID.

For example, consider a host workstation named *cardigan* with an IP address of 192.168.1.5 that is running both an IFFFS and a Wiretap Gateway server. The following table illustrates the variety of ways you can gain access to the servers.

Wiretap Server ID Specification	Description
cardigan:IFFFS	The IFFFS Wiretap server database on cardigan whose name is `IFFFS`.
cardigan:7549	The Wiretap server on port 7549 on cardigan. This is the default port for the IFFFS Wiretap server. In this case, the server's database name is not needed.
192.168.1.5:7549	As above, using the host's IP address instead of its name.
cardigan:Gateway	The Wiretap Gateway server on cardigan.
cardigan:7183	The Wiretap server on port 7183, the default port for the Wiretap Gateway server.
cardigan	The IFFFS Wiretap server on cardigan.

These classes/methods give access to the server ID of a Wiretap server:

- `WireTapServerId.getId()` – Returns a string containing the persistent ID of a Wiretap server.
- `WireTapServerInfo.getId()` – The sample program `listAllServers.C` shows how to use this class to get the properties of Wiretap servers discovered on the network (by the `WiretapServerList` class). The method returns a `WireTapServerId` object.
- `WireTapServerHandle.getId()` – Returns a `WireTapServerId` object. This method returns same as `WireTapServerInfo.getId()`, but is used in different contexts.

Storage ID

Related C++ Sample

- `resolveStorageId.C`

Related Python Sample

- `resolveStoragageId.py`

Related Command Line Tool

- `wiretap_client_tool`

Server Storage ID can also be used to connect to a server. The Storage ID is the persistent identifier of a storage device currently connected to the Wiretap server.

Storage ID is a preferred way of connecting since it is independent of network topology and will work, in conjunction with self discovery, after network changes. If you want users of your Wiretap client to be able to regain access to nodes from one work session to another, you must retain the storage ID. Knowing the storage ID, look for the Wiretap server connected to that storage.

Server Handles

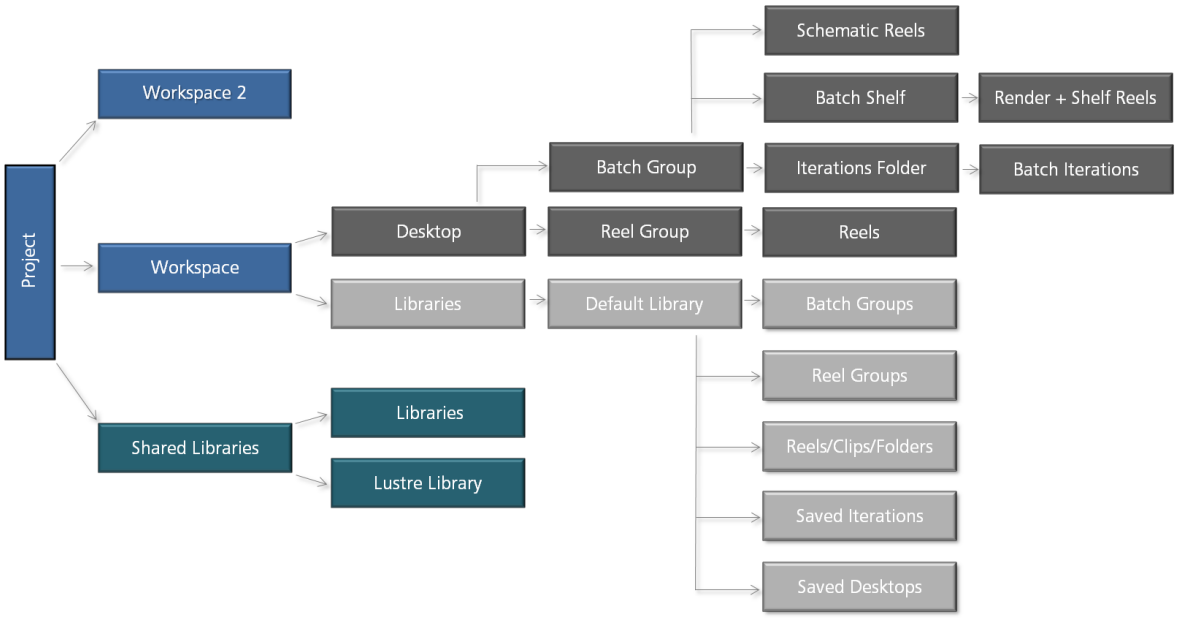
The `WireTapServerHandle` class represents a connection to a Wiretap server. It is the entry point for accessing the node hierarchy of the server. The `WireTapServerHandle` object is not automatically updated when there is a change on the server to which it points. For example, the storage device connected to a Wiretap server can be changed, but this change is not reflected automatically in the `WireTapServerHandle` object.

Understanding the IFFFS Wiretap Server Node Hierarchy

As noted in the introduction, a Wiretap server uses a navigable hierarchy of nodes to represent the structures it exposes. The IFFFS Wiretap server exposes the structure of an IFFFS database, which is a clip library as a collection of project nodes, library nodes, and other related objects. The following diagram shows its node hierarchy.

Below is a diagram showing the hierarchy of components in the Flame project structure. The example below is shown without any changes to the User Interface preferences and with a Lustre project automatically created. For the purposes of illustration, a second Workspace, additional shared libraries, and sample contents of the Default Library have been added.

Each project could contain an unlimited number of Workspaces, which each comprise a protected hierarchy of components accessible to a single user.



IFFFS Node Types

All nodes have a type. Node types are case-sensitive strings. Each Wiretap server implementation defines its own node types using a set of string constants. The following table describes the node types of the IFFFS Wiretap server.

IFFFS Node Type	Description
NODE	Generic node type. Mostly used as container nodes. The diagram above shows a number of container nodes that are of type NODE: setups, users, editing, effects, and preferences.
VOLUME	VOLUME nodes are the first level below the root node of the server. An IFFFS Wiretap server can have several VOLUME nodes. A VOLUME node has the following child nodes: ■ multiple PROJECT nodes (described below) ■ multiple USER nodes (a generic container NODE)
PROJECT	A PROJECT node contains the entire creative finishing project containing all Workspaces and shared libraries, including clip libraries and setups for the various Creative Finishing modules that will be used for the project. It also exposes metadata describing the project. Project metadata can be accessed as explained in Getting and Setting Metadata on a Project Node (page 27). PROJECT nodes are children of a VOLUME node. A PROJECT node has the following child nodes: ■ multiple WORKSPACE nodes ■ a Shared Library List node, which is a node of the type LIBRARY_LIST.
WORKSPACE	A new Workspace is automatically generated upon creating a new project. If the project is already created and in use and another artist connects to that project

IFFFS Node Type	Description
	from another system, another Workspace will be generated to avoid conflicting saves and operations. Workspaces can never be explicitly created by a user, but existing Workspaces can be selected between at the start-up screen. A WORKSPACE node has the following child nodes: ■ a DESKTOP node ■ a Libraries node, which is a node of the type LIBRARY_LIST.
DESKTOP	A DESKTOP node contains: ■ One or more Batch group for compositions, containing a Batch Shelf and an Iterations folder (if shown). The Batch Shelf contains the Batch Renders reel and all Shelf reels. The Iterations folder contains all saved iterations and their associated reels. ■ One or more group (if created), containing reels for storing media on the Desktop outside of Batch. A DESKTOP can be saved to a library and later restored, or copied to a shared library to be shared across Workspaces. A DESKTOP node has the following child nodes: ■ multiple BATCH_CONTAINER ■ multiple REEL_GROUP
BATCH_CONTAINER	A BATCH_CONTAINER contains: ■ a SAVED_BATCH ■ a REEL_LIST ■ a SAVED_BATCH_LIST
SAVED_BATCH	A SAVED_BATCH contains multiple REEL nodes.
REEL_LIST	A REEL_LIST contains multiple REEL nodes.
SAVED_BATCH_LIST	A SAVED_BATCH_LIST contains multiple SAVED_BATCH nodes.
SETUP	A SETUP node gives access to a setup file for a particular Creative Finishing module to be used for the project to which it belongs. The content of a setup file is accessed as explained in Getting and Setting Setups (page 27). A generic NODE named *setups* contains the entire setups branch for a project. Each SETUP node is a child of a generic NODE named for the Visual Effects and Finishing module in which it is used (Paint, CC, Keyer, and so on). SETUP nodes are the leaf nodes in the setups branch. They have no child nodes.
LIBRARY_LIST	A LIBRARY_LIST contains multiple LIBRARY nodes.
LIBRARY	A LIBRARY node is a container for the following child nodes: ■ multiple REEL nodes ■ multiple CLIP nodes ■ multiple FOLDER nodes

IFFFS Node Type	Description
REEL	A REEL node is a container which can only contain multiple CLIP nodes. REEL nodes can be used to organize CLIP nodes. Note that CLIP nodes can be stored directly in LIBRARY nodes.
CLIP	A CLIP node exposes media in the form of frames. Its child nodes can be of the following types: ■ HIRES ■ LOWRES ■ SLATE ■ AUDIOSTREAM ■ VERSION
HIRES	A HIRES node contains high-resolution video frames.
LOWRES	A LOWRES node contains low-resolution video frames.
SLATE	A SLATE node is an alias for the lowest resolution video clip node available (either a HIRES node or a LOWRES node).
AUDIOSTREAM	An AUDIOSTREAM node represents a block of audio media.
VERSION	A VERSION node represents a version, in the context of multi-version clips. There is one VERSION node for each version in the multi-version clip.
USER	A USER node contains application preferences for a particular user and exposes metadata describing the user. User metadata can be accessed as explained in Getting and Setting Metadata on a User Node (page 28). A USER node is a container that only contains multiple USER_PREFERENCE nodes.
USER_PREFERENCE	A USER_PREFERENCE node exposes a preferences file for a particular user for a particular Visual Effects and Finishing module. The content of a preferences file can be accessed as explained in Getting and Setting User Preferences (page 29). A generic NODE named *preferences* contains the entire preferences branch for a user. Each USER_PREFERENCE node is a child of a generic NODE named for the Creative Finishing module in which it is used (Paint, CC, Keyer, and so on). USER_PREFERENCE nodes are the leaf nodes in a user branch. They have no child nodes.

Understanding the Wiretap Gateway Server Node Hierarchy

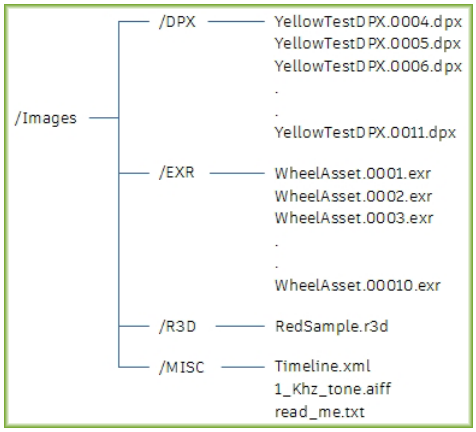
Similarly to the IFFFS Wiretap server, the Wiretap Gateway uses a hierarchy of nodes to expose the structure of a public filesystem, intelligently presenting supported media as clips.

The gateway sever is a multi-process system that consists of 3 processes:

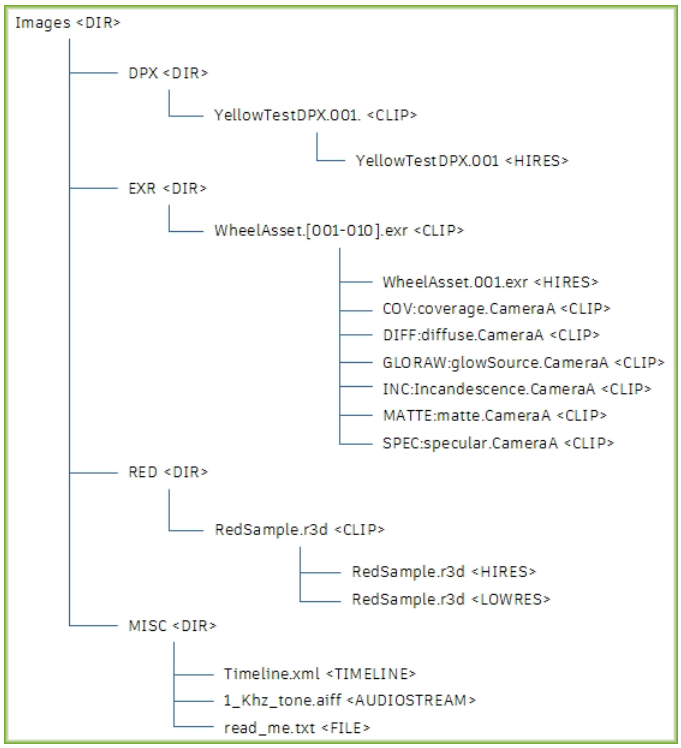
■ The *master* process, launched at startup, dispatches calls to the other two processes.

- The *session* processes are created upon connection from client that maintain a consistent and isolated state for each client. They are tied to client process and all server handles created by a client process will use the same sessions.
- The *slave* processes are spawned by the master process on startup and are shared between session processes to offload image processing of high latency codec formats like red.

The Wiretap Gateway exposes the directories, clips and files of a filesystem. The following diagram shows a directory structure containing DPX, OpenEXR and RED raw image files, plus audio and a timeline in XML format.



The following diagram shows the same directories as seen through the Wiretap Gateway.



The following table presents the node types exposed through the Wiretap Gateway.

Node Type	Description
NODE	Generic node type. Used mainly to contain directory (DIR) nodes.
DIR	DIR nodes are the first level below the root node of the server.
CLIP	A CLIP node exposes media in the form of frames. For details, see Structure of a Wiretap Gatweay Server Clip Node (page 30).
HIRES	A HIRES node contains high-resolution video frames.
LOWRES	A LOWRES node contains low-resolution video frames. **NOTE** Only selected CLIP nodes have low-resolution representations. For example, RED (*.r3d*) CLIP nodes.
AUDIOSTREAM	An AUDIOSTREAM node represents a block of audio media

Traversing and Modifying a Node Hierarchy

A Wiretap server uses a hierarchy of nodes to represent the directory structure of an underlying database. These workflows are related to navigating the node hierarchy and to modifying it in generic ways (copy, modify, delete). The `WireTapNodeHandle` class is important in these workflows.

Viewing the Node Hierarchy of a Wiretap Server

Related C++ Sample

- `listChildren.C`

Related Python Sample

- `listChildren.py`

Related Command Line Tools

- `wiretap_get_root_node`
- `wiretap_get_children`
- `wiretap_print_tree`

See Understanding the IFFFS Wiretap Server Node Hierarchy (page 19) for the structure and description of the IFFFFS Wiretap server hierarchy.

Node Handles

The `WireTapNodeHandle` class represents a node in a Wiretap server hierarchy.

The fact that a Wiretap client can manipulate objects on a Wiretap server resembles Microsoft's Component Object Model (COM). Unlike COM objects, Wiretap node handles are not automatically updated when there is a change in the object they point to on the server. If the Wiretap client requires the latest state of an object

on the server, it must update the information explicitly by calling the appropriate accessor method on the node handle.

Node IDs

Each node has a node ID. These methods give access to the node IDs of a Wiretap server:

- `WireTapNodeHandle.getNodeId()` – Returns a `WireTapNodeId` object.
- `WireTapNodeId.id()` – Returns a string containing the persistent ID of a node.

Responsibility of Wiretap Servers with Respect to IDs

In addition to node IDs, other Wiretap entities have IDs: servers, storage devices, and frames. Wiretap servers are responsible for ensuring the persistence and uniqueness of:

- Server IDs (network-based server identification)
- Node IDs
- Storage IDs (persistent and network-independent server identification)
- Host UUID

Wiretap servers do not guarantee the uniqueness and persistence of frame IDs.

You should not persist connection information based on the server ID: this ID can change over time, such as when the network uses DHCP. Instead, use the server ID for the first connection and get either Storage ID or Host UUID, and then use either for the following connections.

Creating Nodes

Related C++ Samples

- `createClip.C`
- `createProject.C`
- `createUser.C`

Related Python Sample

- `createClip.py`

Related Command Line Tool

- `wiretap_create_node`
- `wiretap_duplicate_node`

The `WireTapNodeHandle` class includes a number of methods for creating nodes and clip nodes. Use `WireTapNodeHandle.createNode` to create all types of nodes (except clip nodes).

Getting and Setting Node Metadata

Related C++ Sample

- `createProject.C`

Related Command Line Tools

- `wiretap_get_metadata`
- `wiretap_set_metadata`

To get and set the metadata for a node, you call the `getMetaData` and `setMetaData` methods on the `WireTapNodeHandle` object. Metadata is a stream that can be in any format (for example, XML).

NOTE The node hierarchy is automatically saved 2 seconds after the last operation in Wiretap. To force a save and avoid having to wait before reading back the information, perform a `wiretap_set_metadata -commit` on the parent catalog node.

For more information about the metadata that can be set on a particular type of node, see:

- Getting and Setting Metadata on a Project Node (page 27)
- Getting and Setting Metadata on a User Node (page 28)
- Getting and Setting Clip Format Metadata (page 32)
- Getting Library Metadata (page 35)
- Getting Volume Metadata (page 35)
- Getting Source Metadata

Deleting Nodes

Related C++ Sample

- `destroyNodeHierarchy.C`

Related Command Line Tool

- `wiretap_destroy_node`

To delete a node, you call the `destroyNode` method on the `WireTapNodeHandle` object.

Depending on the type of node, there may be some conditions on deleting a node. For more information, see:

- Deleting a Project Node (page 28)
- Deleting a User Node (page 29)

Managing Projects and Setups

These workflows create, copy, and modify project nodes, and access project setup files. They apply to the IFFFS Wiretap server only. The class `WireTapNodeHandle` is important in these workflows.

Creating/Copying Project Nodes

Related C++ Samples

- `createProject.C`: Creates a new empty project node and sets its metadata.
- `copyProject.C`: Copies the setups content of a project node recursively. Libraries are not copied.

When a node of type PROJECT is created, nothing is created beneath it. The Wiretap client must populate the entire setups branch under the project node.

The easiest way for a Wiretap client to create a new project containing an IFFFS project hierarchy is to copy an existing project created in a Creative Finishing application.

For information on copying libraries, see Copying Clips (page 34).

Getting and Setting Metadata on a Project Node (XML Format)

Related C++ Sample

- `createProject.C`: Creates a new empty project node and sets its metadata.

Related Command Line Tools

- `wiretap_get_metadata`
- `wiretap_get_available_metadata`
- `wiretap_is_metadata_available`

Project nodes usually have metadata associated with them. The IFFFS Wiretap server expects project node metadata to be in XML format. The sample `createProject.C` shows how to prepare and set the XML metadata stream. For detailed information on the format, see Project Node Metadata (XML) (page 43).

The metadata can be accessed using the `getMetaData` and `setMetaData` methods on the `WireTapNodeHandle` object for the project node.

Setting project metadata is done as follows:

```
projectNode.setMetaData("XML", metadata);
```

where,

- `XML` is a tag that specifies the format of the metadata stream. When getting and setting project node metadata, the metadata tag must be set to the string *XML*.
- `metadata` is a `WireTapStr` object that contains the XML stream. See Project Node Metadata (XML) (page 43).

Getting metadata is done as follows:

```
projectNode.getMetaData("XML","",1, metadata);
```

where,

- `XML` is the metadata format tag that must be set to the string *XML*.
- "" (empty string) is a filter that specifies an element in the metadata stream. This parameter is ignored for project metadata.
- `1` is the depth to which the metadata must be obtained. This parameter is ignored for project metadata.
- `metadata` is a `WireTapStr` object that is an output parameter in which the XML stream is returned. For details of the XML format for a project, see Project Node Metadata (XML) (page 43).

Getting and Setting Setups

Related C++ Sample

- `createProject.C`: Creates a new empty project node and sets its metadata.

Related Command Line Tool

- `wiretap_rw_file`

In an IFFFS node hierarchy, a project branch usually includes many setup nodes. A setup node gives access to the content of a setup file for a particular Creative Finishing module. These files can be in ASCII or binary format. They can be quite large. The file contents can be read and written with the `pullStream` and `pushStream` methods of the `WireTapServerHandle` class as shown in the `copyProject.C` sample program.

Deleting a Project Node

Related Command Line Tool

- `wiretap_destroy_node`

To delete a project node, call the `destroyNode` method on the `WireTapNodeHandle` object for the project.

If a project contains any libraries, it cannot be deleted. The libraries must be deleted before the project can be deleted.

When `destroyNode` is called on a project node (that contains no libraries):

- The project and all its setups are removed from the IFFFS database.
- The project node and all its children are no longer available on the Wiretap server.

Managing Users and Preferences

These workflows create, copy, and modify user nodes. In addition, they can access user preference files. They apply to the IFFFS Wiretap server only. The class `WireTapNodeHandle` is important in these workflows.

Creating/Copying User Nodes

Related C++ Samples

- `createUser.C`: Creates an empty user node.
- `copyUser.C`: Copies an entire user node hierarchy recursively.

When a node of type USER is created, nothing is created beneath it. The Wiretap client must populate the entire preferences branch under the user node.

The easiest way for a Wiretap client to create a new user containing an IFFFS user hierarchy is to copy an existing user created in a Creative Finishing application.

Getting and Setting Metadata on a User Node

Related Command Line Tool

- `wiretap_get_metadata`

User nodes can have metadata associated with them. The IFFFS Wiretap server expects user node metadata to be in XML format. The sample `createUser.C` shows how to prepare and set the XML metadata stream. For detailed information on the format, see User Node Metadata (XML) (page 47).

The metadata can be accessed using the `getMetaData` and `setMetaData` methods on the `WireTapNodeHandle` object for the user node.

Setting metadata is done as follows:

```
userNode.setMetaData("XML", metadata);
```

where,

- `XML` is a tag that specifies the format of the metadata stream. When getting and setting user node metadata, the metadata tag must be set to the string *XML*.
- `metadata` is a `WireTapStr` object that contains the XML stream. See User Node Metadata (XML) (page 47).

Getting metadata is done as follows:

```
userNode.getMetaData("XML", "", 1, metadata);
```

where,

- `XML` is the metadata format tag.
- "" is a filter that specifies an element in the metadata stream. This parameter is ignored for user metadata.
- `1` is the depth to which the metadata must be obtained. This parameter is ignored for user metadata.
- `metadata` is a `WireTapStr` object that is an output parameter in which the XML stream is returned. For details of the user XML format, see User Node Metadata (XML) (page 47).

Getting and Setting User Preferences

Related C++ Sample

- `copyUser.C`: Copies an entire user node hierarchy recursively.

In an IFFFS node hierarchy, a user branch usually includes many USER_PREFERENCE nodes. A USER_PREFERENCE node gives access to the content of a preferences file for a particular Creative Finishing module. These files can be in ASCII or binary format. The file content can be read and written with the `pullStream` and `pushStream` methods of the `WireTapServerHandle` class as shown in `copyUser.C`.

Deleting a User Node

Related Command Line Tool

- `wiretap_destroy_node`

To delete a user node, call `WireTapNodeHandle.destroyNode`. The user and all its preferences are removed from the IFFFS database. The user node and its preferences hierarchy are no longer available on the Wiretap server.

Managing Clips

These workflows create, copy, and modify video and audio clip nodes. For the most part, Wiretap handles audio clips and video clips in the same way. The main difference is how the clip format is defined.

The following classes are important in these workflows:

- `WireTapNodeHandle`
- `WireTapClipFormat`
- `WireTapAudioClipFormat`

About Clip Nodes and Formats

The following sections contain information regarding clip nodes and formats.

Structure of an IFFFS Wiretap Server Clip Node

On an IFFFS Wiretap server, a CLIP node is a container for child nodes of the following types.

Node Type	Description
HIRES	Represents high-resolution video media.

Node Type	Description
ALPHA_HIRES	Represents the alpha of the high-resolution video media.
LOWRES	Represents low-resolution video media. Wiretap supports the use of low-resolution proxy versions of video media to increase the speed at which video clips are transferred and displayed.
ALPHA_LOWRES	Represents low-resolution alpha of the video media.
SLATE	Represents the lowest resolution video media available. If a LOWRES node exists, the SLATE node will be equivalent to the LOWRES node, and otherwise the SLATE node will be equivalent to the hires node.
AUDIOSTREAM	Represents an audio clip.
VERSION	Represents a version, in the context of multi-version clips. One VERSION node for each version.

For video, the parent CLIP node is normally a shortcut to the HIRES node. A video clip node has at least two child nodes: the HIRES and SLATE nodes. It may also have a LOWRES child node if the clip has proxies. The resolution of a video clip node is stored in its metadata. The proxy version of a clip is stored on the Wiretap server like the high-resolution original.

An audio clip consists of a CLIP node (with zero frames) with an AUDIOSTREAM child node.

Structure of a Wiretap Gateway Server Clip Node

On the Wiretap Gateway server, a CLIP node is a container for child nodes of the following types.

Node Type	Description
HIRES	Represents highest-resolution or highest-quality video media available for the file type.
LOWRES	Represents low-resolution video media for file types such as, Red (*.r3d*) files that support multiple qualities in the same file.
AUDIOSTREAM	Represents an audio clip.
CLIP	For particular media types, a CLIP node can act as a container of other CLIP nodes: ■ OpenEXR: Each channel is represented as a CLIP child node. ■ Red (*.r3d*) files: Each quality is a separate CLIP child node.

Clip Formats

Wiretap supports the same formats as the ones supported by the Flame Family.

The IFFFS Wiretap server allows the media files to be read in their native format with no conversion by directly accessing the file paths, or as raw RGB by reading through the server. For more information, see Importing Clips.

The`WireTapClipFormat` class is used to define the format of a video clip node. The `WireTapAudioClipFormat` class is used to define the format of an audio clip node. When instantiating a new clip node, a clip format object must be supplied as an input parameter. `WireTapClipFormat` also supplies a large number of constants for specifying industry-standard formats.

For more information on cached clip formats, see:

- Raw Video Frame Buffer Format (RGB) (page 37)
- 12-bit Packed RGB Format (page 40)
- Raw Audio Frame Buffer Format (DL) (page 42)

Frame IDs

Each frame in a clip node has a frame ID. Frame IDs are unique for a particular instance of a particular Wiretap server. These classes/methods give access to the IDs of the frames on a Wiretap server:

- `WireTapFrameId.id()` – Returns a string containing the persistent ID of a frame.
- `WireTapNodeHandle.getFrameId()` – Returns the ID of a frame associated with a clip node. The frame is specified by its index in the set of frames associated with the clip node.

Limitations on Clip Nodes

The following limitations apply to clip nodes:

- Like the Creative Finishing applications, Wiretap does not permit overwriting the frames of a clip. Your Wiretap client must create a new clip and write the required frames to it.
- When creating or destroying a clip node, or setting its metadata, the library (to which the clip belongs) must not be in use or opened for reading and writing by a Creative Finishing application.

Limitations on Audio Clips

- Multi-channel audio in a single track/stream is not supported on write. To write multi-channel audio, use a separate AUDIOSTREAM child node for each channel.
- EDL metadata is not supported for audio.

Getting Frame ID Strings or the Paths to Frame Files.

Related C++ Sample

- `listFrames.C`

Related Python Sample

- `listFrames.py`

Related Command Line Tool

- `wiretap_get_frames`

Getting Frames into Clips

Frames can be written or linked to a clip in three ways:

- **By writing the frames to the clip** – Only cached frames can be written to a clip node. This involves the method `writeFrame` of `WireTapNodeHandle`. See Copying Clips (page 34).

- **By providing source metadata** – This applies to frames that are not cached, and involves getting a source data definition from the Gateway and using it to create a source clip in the IFFFS wiretap server. See Soft-importing Clips (page 33).
- **By accessing frames directly on the storage device using file paths** – This can be done for frames in any standard format. This technique does not actually involve the Wiretap server when writing frames. However, the Wiretap server does supply the paths to the frames. The sample program *listFrames.C* shows how to obtain frame paths. This technique can improve performance when reading and writing frames. See the FAQ How do I read standard-formatted frames from a network-mounted standard FS? (page 85).

Creating Clip Nodes

Related C++ Samples

- `createClip.C`: Defines a video clip format, create a new clip node, and write frames to it.
- `createAudio.C`: Defines an audio clip format, create a new clip node, and writes audio samples to it.

Related Python Sample

- `createClip.py`

Related Command Line Tool

- `wiretap_create_clip`

The method `WireTapNodeHandle.createClipNode` is used to create video and audio clip nodes.

Getting and Setting Clip Format Metadata

An instance of the `WireTapClipFormat` class can have metadata associated with it. The clip format metadata is used to describe the media in the clip. Its content is similar to the content of an image file header.

The IFFFS Wiretap server expects clip node metadata to be in XML format. For detailed information on the format and content of the metadata, see Clip Format Metadata (XML) (page 47).

Clip format metadata and its metadata tag can be set when constructing an instance of `WireTapClipFormat` or `WireTapClipFormat`. The metadata tag must be *XML*.

The metadata can also be set and get using the `getmetaData` and `setMetaData` methods on an instance of `WireTapClipFormat` or `WireTapClipFormat`.

Setting metadata and the metadata tag is done as follows:

```
clipFormat.setMetaDataTag("XML");
clipFormat.setMetaData(metadata);
```

where,

- `XML` is a tag that specifies the format of the metadata stream.
- `metadata` is a `WireTapStr` object that contains the XML stream. See Clip Format Metadata (XML) (page 47).

Getting metadata is done as follows:

```
clipFormat.metaData( );     // this returns a string
                            // that contains the XML stream
clipFormat.metaDataTag( ); // this returns "XML"
```

Getting and Setting Metadata on a Clip Node

Related Command Line Tools

- `wiretap_get_metadata`
- `wiretap_set_metadata`

An instance of the `WireTapNodeHandle` class whose type is CLIP can have one or more metadata streams associated with it.

The metadata can be get and set by calling the `getMetadata` and `setMetaData` methods on the `WireTapNodeHandle` object representing the clip node.

An Edit Decision List (EDL) is an example of metadata that can be set on a clip node. An EDL describes how the media in a clip are assembled. See Creating a Clip from an EDL Timeline (page 33).

Importing Clips

Related C++ Sample

- `importOpenClips.C`: Creates a CLIP node and uses an Open Clip located within a directory as the XML data source.

Related Python Sample

- `importOpenClips.py`: Creates a CLIP node and uses an Open Clip located within a directory as the XML data source.

The IFFFS Wiretap server allows media files in standard formats (for example, DPX, QuickTime, and so on) to be imported. Importing means *referencing* rather than *copying* media to the IFFFS Wiretap server. Once created, the imported clip will be treated like any other clip, as though it were imported by a user from a Creative Finishing workstation.

Importing frames involves fetching a source data definition from a Wiretap Gateway server and forwarding it in a `createClip` call to the IFFFS wiretap server, as shown in the above program samples.

Creating a Clip from an EDL Timeline

Related C++ Sample

- `createTimeline.C`

Related Python Sample

- `createTimeline.py`

A timeline consists of video elements, audio elements, and transitions placed together chronologically on one or more parallel tracks. An Edit Decision List (EDL) is the standard format used for timelines. A Wiretap client can construct a new clip from frames in several existing clips based on an EDL that specifies frame IDs in those clips.

Wiretap treats an EDL as metadata associated with a clip node. EDL metadata can be get and set on an instance of `WireTapNodeHandle` whose type is CLIP by calling the `getMetadata` and `setMetaData` methods. When setting or getting EDL metadata, *DMXEDL* must be specified as the metadata format tag.

For detailed information on EDL format, see Clip Node Metadata (EDL) (page 48).

Setting EDL Metadata on a Clip Node (Assembling the Clip)

When you are creating a new clip based on a timeline, the call to `setMetaData` actually carries out the assembly of the frames in the new clip. The frames are soft-imported or linked to the new clip. The sample *createTimeline.C* shows how to prepare and set EDL metadata.

Getting EDL Metadata on a Clip Node

When getting EDL metadata, an optional *filter* parameter can be used. The filter parameter is used to specify the resolution of the frames. It can be set to the following values:

- `high` to fetch metadata about the high-resolution frames in the clip
- `low` to fetch metadata about the low-resolution frames in the clip
- `all` to fetch metadata about both high-resolution and low-resolution frames in the clip

Limitations on Assembling Clips from Timelines

Assembling clips from timelines is limited in the following ways:

- An EDL cannot be applied to a clip that already contains frames.
- All source nodes and the resulting new node based on the timeline must be located in the same reel (the parent reel).
- The tape names in the metadata of the source clip nodes must match the tape names in the EDL.
- Only cut and dissolve timeline effects are supported.

Copying Clips

Related C++ Samples

- `readFrames.C`
- `duplicateNode.C`

Related Python Samples

- `readFrames.py`
- `duplicateNode.py`

Related Command Line Tools

- `wiretap_rw_frame`
- `wiretap_client_tool`

There are two methods that you can use to copy a clip.

The first method involves reading the frames of an existing clip and writing them to a new clip. The sample program *readFrames.C* shows how to read frames, but it does not show writing frames to a new clip.

Here are the steps that would be involved in implementing the full workflow of copying a clip:

1. Identify the clip node to be copied (the source clip node).
2. Identify the parent node of the new clip node to be created (the destination clip node).
3. Copy the clip format of the source clip node.
4. Create the destination clip node under the parent node using the copied clip format.
5. Copy the frames from the source clip node to a buffer by calling `readFrame` on the source clip node.

6 Copy the frames from the buffer to the destination clip node (by calling `writeFrame` on the destination clip node).

The second method consists of duplicating the node itself, and is shown in the sample programs `duplicateNode.C` and `duplicateNode.py`.

Here are the steps that would be involved in implementing the full workflow of duplicating a clip:

1 Identify the clip node to be copied (the source clip node).
2 Identify the parent node of the new clip node to be created (the destination clip node).
3 Use the duplicateNode method to create the duplicated copy under the parent node.

See also:

- FAQs related to Reading and Writing Video Media (page 85)
- FAQs related to Reading Audio Media (page 85)

Managing Containers

Containers are nodes used to organize the clips stored in a project. The available containers are objects such as Desktop, Library, Reel, Batch, and Folders.

Containers can be managed in the generic ways described in Traversing and Modifying a Node Hierarchy (page 24).

Limitations on Containers

Containers Reels and libraries are limited in the following ways:

- Most container nodes do not have metadata associated with them.
- Wiretap needs to acquire exclusive access to modify project, workspace or libraries. If this is attempted while the object is being used by a Creative Finishing application, the Wiretap method fails and the server returns an error message. The reverse is also true.
- The only container that can be destroyed while not being empty is the Project node. See Managing Projects and Setups (page 26)

Managing Volumes

A volume node represents a storage partition.

Note that an IFFFS Wiretap server can have more than one volume node. Each volume node can have project nodes.

Getting Volume Metadata

Related Command Line Tool

- `wiretap_get_metadata`

Volume metadata can be accessed using the `getMetaData` method on the `WireTapNodeHandle` object for the library node. Returned metadata includes the volume name, state (mounted/unmounted), capacity, and available space. For detailed information on the metadata format, see Volume Node Metadata (XML) (page 42).

Getting metadata is done as follows:

```
volumeNode.getMetaData("XML", "", 1, metadata);
```

where,

- `XML` is the metadata format tag.
- "" is a filter that specifies an element in the metadata stream. This parameter is ignored for volume metadata.
- `1` is the depth to which the metadata must be obtained. This parameter is ignored for volume metadata.
- `metadata` is a `WireTapStr` object that is an output parameter in which the XML stream is returned.

Media and Metadata Formats

5

This section describes the metadata and frame formats supported by Wiretap.

Raw Video Frame Buffer Format (RGB)

Video frames can be stored on the Autodesk Media Storage as raw uncompressed RGB data with no file header, and are stored under frame IDs. Frame format information is stored separately. This section describes this raw RGB video format.

Wiretap allows you to access format information through the `WireTapClipFormat` object associated with a video clip node.

NOTE In the sections below, raw RGB refers specifically to the raw RGB file format stored on the Autodesk Media Storage.

Key Attributes of the Raw RGB Format

This section describes key attributes of the raw RGB video format.

Image Orientation – Indicates the orientation of the image data for display. Raw RGB frames are oriented as follows:

- Line direction = Left to right
- Frame direction = Top to bottom

Number of Components or Channels – Defines the number of color components per pixel. Raw RGB pixels have three color components.

Bit Size – Defines the number of bits used for each color component (or channel) in a pixel. All components have the same bit size. Raw RGB frames can have the following bit sizes:

- 8-bit integer
- 10-bit integer
- 12-bit integer (can be packed or unpacked/filled. See the next section)
- 16-bit float
- 32-bit float

Bit size is also known as frame depth.

Component Data Packing Method – Indicates whether the components of a pixel are *packed* or *filled/unpacked* into 32-bit words. *Packed* means pixel components are stored contiguously, regardless of whether they form complete 32-bit words. With this method, every bit contains image data. *Filled* means the last few bits for a pixel component are skipped or set to 0 so that the pixel component corresponds to a certain number of words. Filling makes each pixel component (and therefore each pixel) start at a word boundary. Filling is only necessary if the data does not fit evenly into words. The number of bits used as filler depends on the bit size. The last few bits can usually be ignored when reading the pixel from memory. Raw RGB frames might be packed or unpacked/filled depending on the bit size. For details, see the table in Memory Required Per Pixel (page 38). The Component Packing Diagrams (page 39) show how component data is laid out for various bit sizes.

End-of-line Padding – Specifies the number of bits required to make each scan line of a frame end on a 32-bit boundary. Padding makes each scan line correspond to a round number of 32-bit words. The raw RGB format uses end-of-line padding. The number of bits of padding required depends on the frame width and the number of bits per pixel.

Encoding – Defines whether the element is run-length encoded. No encoding is applied to raw RGB data.

Memory Required Per Pixel

The memory required for each pixel of a raw RGB frame depends on three factors:

- Number of color components or channels per pixel (for raw RGB this is 3)
- Bit size (in *bits per component* or *bits per channel*)
- Data packing method

The following table indicates the memory required for raw RGB frames of various bit sizes.

Number of Components	Bits per Component	Data Type	Bits per Pixel	Memory Required per Pixel
3	8	integer	24	3 bytes per pixel. No filling necessary
3	10	integer	32	4 bytes per pixel in both cases. The last byte is filled. Its last 2 bits can be ignored.
3	12 (packed)	integer	36	4.5 bytes per pixel. This is a non-standard, Visual Effects and Finishing application-specific format. No filler bits are used. For more information on this format and how to unpack it, see 12-bit Packed RGB Format (page 40).
3	12 (filled/unpacked)	integer	48	6 bytes per pixel. The last byte is filled. Its last 4 bits can be ignored.
3	16	float	48	6 bytes per pixel. No filling necessary
3	32	float	96	12 bytes per pixel. No filling necessary

Memory Required per Frame

Pixels are arranged in scan lines. The memory required for one scan line is the width of the frame multiplied by the number of bytes per pixel. Each line is padded to a double word length by adding the necessary number of bits set to 0.

Memory required per frame can be calculated as follows:

bitsPerFrame = ((bitsPerPixel * frameWidth) + endOfLinePadding) * frameHeight

Component Packing Diagrams

The diagrams in this section show how component data is packed and filled for different bit sizes. Diagrams are not provided for all bitsize/packing combinations. A few representative diagrams are provided to illustrate the principle. For formats that use filling, the diagrams demonstrate the fact that pixel component boundaries do not coincide with byte boundaries.

How to Read the Diagrams

The following table explains how to interpret the diagrams.

Row Label	Comment
Byte	■ The pipe character (|) represents a byte boundary.
Comp	■ The values indicate the pixel component. For example, P1/C2 refers to Pixel 1/Component 2. ■ The pipe character (|) represents a component boundary. ■ The word *fill* indicates that part of the section is filled (does not contain component data).
Bit	■ Values are indices for the bits within a particular byte.
Data	The values indicate the type of data that is contained in a particular bit, namely: ■ R (red) ■ G (green) ■ B (blue) ■ 0 (filler bit) Set to 0 when writing. It can be ignored when reading.

8-bit Integer Diagram

(packed, no filling necessary)

```
Byte |      Byte0      |      Byte1      |      Byte2      |      Byte3      |
Comp.|      P1/C0      |      P0/C2      |      P0/C1      |      P0/C0      |
Bit  | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 |
Data | R R R R R R R R | B B B B B B B B | G G G G G G G G | R R R R R R R R |
```

10-bit Integer Diagram

(filled to 32-bit word boundaries)

```
Byte |      Byte0      |      Byte1      |      Byte2      |      Byte3      |
Comp.|fill|        P0/C2        |        P0/C1        |        P0/C0        |
Bit  | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 |
Data | 0 0 B B B B B B | B B B B G G G G | G G G G G G R R | R R R R R R R |
```

12-bit Integer Packed Diagram

```
Byte |      Byte0      |      Byte1      |      Byte2      |      Byte3      |
Comp.|      P0/C2      |          P0/C1          |          P0/C0          |
Bit  | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 |
Data | B B B B B B B B | G G G G G G G G | G G G G R R R R | R R R R R R R R |
Byte |      Byte4      |      Byte5      |      Byte6      |      Byte7      |
Comp.| P1/C2|           P1/C1          |          P1/C0          |  P0/C2 |
Bit  | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 |
Data | B B B B G G G G | G G G G G G G G | R R R R R R R R | R R R R B B B B |
Byte |      Byte8      |      Byte9      |      Byte10     |      Byte11     |
Comp.|          P2/C1          |          P2/C0          |  P1/C2 |
Bit  | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 |
Data | G G G G G G G G | G G G G R R R R | R R R R R R R R | B B B B B B B B |
```

12-bit Integer Unpacked Diagram

(filled/unpacked to 16-bit word boundaries)

```
Byte |      Byte0      |      Byte1      |      Byte2      |      Byte3      |
Comp.|  fill  |          P0/C1          |  fill  |          P0/C0          |
Bit  | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 |
Data | 0 0 0 0 G G G G | G G G G G G G G | 0 0 0 0 R R R R | R R R R R R R R |
Byte |      Byte4      |      Byte5      |      Byte6      |      Byte7      |
Comp.|  fill  |          P1/C0          |  fill  |          P0/C2          |
Bit  | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 |
Data | 0 0 0 0 R R R R | R R R R R R R R | 0 0 0 0 B B B B | B B B B B B B B |
Byte |      Byte8      |      Byte9      |      Byte10     |      Byte11     |
Comp.|  fill  |          P1/C2          |  fill  |          P0/C1          |
Bit  | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 |
Data | 0 0 0 0 B B B B | B B B B B B B B | 0 0 0 0 G G G G | G G G G G G G G |
```

12-bit Packed RGB Format

The 12-bit filled or unpacked (48 bit) integer RGBA format is a standard video format in which each pixel component consumes 12 bits for a total of 48. The alpha channel is effectively ignored. This method of pixel packing wastes 25% of storage, but requires less CPU to manipulate once in memory. On systems where storage is at a premium, the Visual Effects and Finishing applications allow the artist to work in 12-bit packed RGB format (also known as 36-bit format), which is a non-standard Visual Effects and Finishing application-specific format. The alpha component is not present or used in this format, so the pixels are packed together to fit in a 36-bit space. This saves 25% of storage at the expense of increased CPU time required to extract the components into proper memory-aligned integers.

The table below describes the pixel component layout over 8 words (32 bytes), where R, G, and B represent the pixel components, and the integers represent the zero-based pixel index. For example, B2 is the blue component of the third pixel.

```
             | +0  |   +1  |  +2  |  +3 |
w0 byte 0  : | G0    |B0|    R0      |B1|
w1 byte 4  : | G1    |B0|    R1      |B1|
w2 byte 8  : | G2    |B0|    R2      |B1|
w3 byte 12 : | G3    |B2|    R3      |B3|
w4 byte 16 : | G4    |B2|    R4      |B3|
w5 byte 20 : | G5    |B2|    R5      |B3|
w6 byte 24 : | G6    |B4|    R6      |B5|
w7 byte 28 : | G7    |B4|    R7      |B5|
w8 byte 32 : | B6    |B4|    B7      |B5|
```

Unpacking Algorithm

The algorithm required to unpack 12-bit packed RGB format can be seen in the (un-optimized) utility macro below, where `src` and `dst` are the source and destination buffers, respectively.

```
#define UNPACK36GL(src, dst)                    \
{                                               \
 /* Unpacking : 6.625 cycles per pixel */      \
 UInt32 w0, w1, w2, w3, w4, w5, w6, w7, w8;    \
 UInt32 b0, b2, b4, b6;                         \
                                              \
 /* 9 cycles */                                 \
 w0 = *((UInt32*)((src) + 0));                  \
 w1 = *((UInt32*)((src) + 4));                  \
 w2 = *((UInt32*)((src) + 8));                  \
 w3 = *((UInt32*)((src) + 12));                 \
 w4 = *((UInt32*)((src) + 16));                 \
 w5 = *((UInt32*)((src) + 20));                 \
 w6 = *((UInt32*)((src) + 24));                 \
 w7 = *((UInt32*)((src) + 28));                 \
 w8 = *((UInt32*)((src) + 32));                 \
                                              \
 /* 24 cycles */                                \
 b0 = (((w0 & 0x000F000F) << 4) |               \
  ((w1 & 0x000F000F) << 8) |                    \
  ((w2 & 0x000F000F) << 12));                   \
 b2 = (w3 & 0x000F000F) << 4) |                 \
  ((w4 & 0x000F000F) << 8) |                    \
  ((w5 & 0x000F000F) << 12));                   \
 b4 = (w6 & 0x000F000F) << 4) |                 \
  ((w7 & 0x000F000F) << 8) |                    \
  ((w8 & 0x000F000F) << 12));                   \
 b6 = w8;                                       \
                                              \
 /* 20 cycles */                                \
 *((UInt32*)((dst) + 0)) = b0 >> 16;            \
 *((UInt32*)((dst) + 4)) = w0;                  \
 *((UInt32*)((dst) + 8)) = b0;                  \
 *((UInt32*)((dst) + 12)) = w1;                 \
 *((UInt32*)((dst) + 16)) = b2 >> 16;           \
 *((UInt32*)((dst) + 20)) = w2;                 \
 *((UInt32*)((dst) + 24)) = b2;                 \
 *((UInt32*)((dst) + 28)) = w3;                 \
```

```
    *((UInt32*)((dst) + 32)) = b4 >> 16;        \
    *((UInt32*)((dst) + 36)) = w4;              \
    *((UInt32*)((dst) + 40)) = b4;              \
    *((UInt32*)((dst) + 44)) = w5;              \
    *((UInt32*)((dst) + 48)) = b6 >> 16;        \
    *((UInt32*)((dst) + 52)) = w6;              \
    *((UInt32*)((dst) + 56)) = b6;              \
    *((UInt32*)((dst) + 60)) = w7;              \
}
```

Raw Audio Frame Buffer Format (DL)

An audio stream is a sequence of samples of a specified data type and endianness.

Raw audio frames are identified by the dlaudio_XXXXX format tag. Audio clips will have one of the following format tags.

Format Type	Permitted Tag
Big-endian	■ dlaudio_int16 ■ dlaudio_int24 ■ dlaudio_float
Little-endian	■ dlaudio_int16_le ■ dlaudio_int24_le ■ dlaudio_float_le

For audio streams that contain 24-bit samples, the samples are stored as 32-bit integers. Samples are not packed. Instead, eight filler bits are appended to the sample, so each sample corresponds to a 32-bit word. They can be little-endian or big-endian.

Multi-channel (interlaced) audio appears as samples grouped into channels. For example, a three-channel floating point sample consists of a stream of sample triplets.

NOTE A Wiretap client can obtain the format tag for an audio clip as follows:

```
WireTapAudioClipFormat formatObject;
clipNode.getClipFormat(formatObject);
char* formatTag = formatObject.formatTag();
```

Volume Node Metadata (XML)

Volume node metadata is returned by the IFFFS Wiretap server when you call `getMetaData` on the `WireTapNodeHandle` object for the node. The metadata is read-only.

Valid Values for Volume Metadata

The following table shows valid values for each element in the volume XML stream.

Element	Type	Valid Range/Value	Comment
<Volume>	complex	n/a	Container for other elements. Required.
<Name>	string	No length restrictions	Storage partition name.
<State>	string	mounted/unmounted	–
<CapacityMB>	integer	megabytes	Total capacity of the volume
<FreeMB>	integer	megabytes	Available space

Sample XML for Volume Metadata

The following is the metadata for a typical volume:

```
<Volume>
    <Name>AutodeskMediaStorage</Name>
    <State>mounted</State>
    <CapacityMB>1144220</CapacityMB>
    <FreeMB>853560</FreeMB>
</Volume>
```

Project Node Metadata (XML)

Project nodes have XML metadata associated with them. You can gain access to it using the `getMetaData` and `setMetaData` methods on the `WireTapNodeHandle` object for the project node.

Valid Values for Project Metadata

The following table shows valid values for each element in a project XML stream. Some elements are read-only. Values are returned by the IFFFS Wiretap server when you call `getMetaData` on the `WireTapNodeHandle` object for the project node.

Element	Type	Valid Range/Values	Comments
<Project>	complex		Contains all other elements. Required.
<Name>	string		Read-only, optional
<Nickname>	string		Read-only, optional
<Description>	string	No length restrictions	Optional
<CreationDate>	string		Read-only, optional

Element	Type	Valid Range/Values	Comments
<SetupDir>	string	No length restrictions	Optional IFFFS Wiretap server returns the full path
<Partition>	string		Read-only, optional
<Version>	integer		Read-only, optional
<FrameWidth>	integer	24 to 8192	Optional
<FrameHeight>	integer	24 to 8192	Optional
<FrameDepth>	string	8-bit 10-bit 12-bit 12-bit u 16-bit fp	Optional The *u* in *12-bit u* means unpacked. *fp* stands for floating point.
<AspectRatio>	decimal	0.001 to 100.0	Optional
<FieldDominance>	string	FIELD_1 FIELD_2 PROGRESSIVE (case-sensitive)	Optional
<ProxyWidth>	integer	Width of the proxies. Calculated: <FrameWidth> x <ProxyWidthHint>.	Read-only.
<ProxyWidthHint>	float	Used to compute the size of proxies. Must be one of the following: ■ 0.5 ■ 0.25 ■ 0.125	Optional
<ProxyMinFrameSize>	integer	The threshold height over which proxies are generated when importing from MediaHub. ■ 24 to 8192. Must be an integer multiple of 4.	Optional
<ProxyAbove8bits>	string	true/false (case-sensitive)	Optional
<ProxyQuality>	string	■ Lanczos ■ Shannon ■ Gaussian	Optional

Element	Type	Valid Range/Values	Comments
		■ Quadratic ■ Bicubic ■ Mitchell ■ Triangle ■ Impulse ■ Draft (case-sensitive)	
<ProxyRegen-State>	integer		Read-only, optional
<ProxyDepth-Mode>	string	8-bit SAME_AS_HIRES 8-BITS (deprecated)	Optional
<ProxyDepth>	string	8-bit 10-bit 12-bit 12-bit u 16-bit fp	Optional The *u* in *12-bit u* means unpacked. *fp* stands for floating point.
<BatchShader-Path>	string	Path to the default Matchbox shaders	Optional
<ActionLightOper-atorPath>	string	Path to the default Lightbox shaders	Optional
<ActionShader-Path>	string	Path to the default Matchbox for Action shaders	Optional
<InputCol-ourSpaceRules-File>	string	Path to the input colour space rules file	Optional
<VisualDepth>	string	8-bit 12-bit unknown 8bits (deprecated) 12bits (deprecated) (case-sensitive)	Optional
<ProcessMode>	integer	Rendering engine used, 1 for Flame Re-actor, 0 for Classic Engine	Optional
<IntermediatesPro-file>	integer	Dictionary key defining the intermediate format used for caching. See the table below.	Optional

Element	Type	Valid Range/Values	Comments
<FrameRate>	string	Default frame rate for menus that require a frame rate.	Optional

NOTE Proxies are no longer enabled by the project, but by each clip individually. There is no public method to generate proxies.

<IntermediatesProfile> Supported Values

Compressed Intermediates	<IntermediatesProfile> Value
Uncompressed / RAW	0
Uncompressed	2
Legacy Configuration (deprecated)	1
DNx Compressed Intermediates (Linux-only)	
DNxHR/DNxHD 444 / RAW	512
DNxHR/DNxHD HQ / RAW	513
DNxHR/DNxHD SQ / RAW	514
DNxHR/DNxHD LB / RAW	515
DNxHR/DNxHD 444	640
DNxHR/DNxHD HQ	641
DNxHR/DNxHD SQ	642
DNxHR/DNxHD LB	643
QuickTime ProRes (Linux and Mac)	
ProRes 422 Proxy	65536
ProRes 422 LT	65537
ProRes 422	65538
ProRes 422 HQ	65539
ProRes 4444	65540

Compressed Intermediates	<IntermediatesProfile> Value
ProRes 4444 XQ	65541
ProRes 422 Proxy / RAW	65792
ProRes 422 LT / RAW	65793
ProRes 422 / RAW	65794
ProRes 422 HQ / RAW	65795
ProRes 4444 / RAW	65796
ProRes 4444 XQ / RAW	65797

NOTE DNx-type compressed intermediates are not supported for projects opened on Mac OS X. Use QuickTime ProRes instead.

User Node Metadata (XML)

The metadata (user settings or properties) for a user node is in XML format. It can be accessed using the `getMetaData` and `setMetaData` methods on the `WireTapNodeHandle` object for the user node.

Valid Values for User Metadata

Element	Type	Valid Range / Values	Comments
<User>	complex		Container for other elements. Required.
<Name>	string	No length restrictions	–
<PreferenceDir>	string	No length restrictions	IFFFS Wiretap server returns full path
<Default>	string	True/False (case-sensitive)	–

Clip Format Metadata (SourceData)

The classes `WireTapClipFormat` and `WireTapAudioClipFormat` are used to specify a number of properties of a clip. Instances of these classes can also have XML metadata associated with them. The metadata is used for free-form or specialized data.

The representation of a clip closely matches that of an Open Clip.

```
wiretap_get_metadata -s SourceData -n <path_to_your_CLIP_node_>
```

displays all the metadata available, of which the most relevant parts are described in the *Open Clip description*.

The XML stream of a CLIP node returns the sequence structure and characteristics of that node.

Clip Node Metadata (EDL)

The IFFFS Wiretap EDL stream syntax follows the CMX 3600 format, augmented to include Autodesk's own edit codes. These are indicated by the keyword DLEDL, and are considered comments in the CMX 3600 format. They are used to indicate information necessary for EDL completeness, but that is not part of the original CMX 3600 standard.

The overall form of the Wiretap EDL stream is similar to the Autodesk EDL file generated by performing a *flatten* publish in a Visual Effects and Finishing application. It is also similar to the EDL file you can generate using the Export EDL menu by selecting the CMX 3600 format. Before extracting EDLs programmatically, it can be helpful to view one or two within the application.

This section examines the syntax of the Wiretap EDL stream through a series of sample outputs. Its purpose is to explain the Autodesk edit codes (the comment lines prefaced by `DLEDL`). However, not all Autodesk edit codes appearing in the Wiretap EDL stream are documented. In addition, some information is provided on the CMX 3600 format as background information.

In the sample outputs presented in this section, lines have been numbered for ease of reference in the explanatory discussions. The clip node EDL metadata stream is not itself numbered. Also, in some cases spacing has been altered and line breaks inserted for clarity.

Basic Case

The following sample output shows the EDL stream for a simplified timeline containing a single segment.

```
1. TITLE: BASIC_CASE
2. FCM: NON-DROP FRAME

3. TITLE: ASSEMBLY RESOLUTION: 720:486:32:3:0.899998:1399696:BE:P:29.97
4. FCM: NON-DROP FRAME

5. 001 PBS7890 V C 00:00:00:00 00:00:00:08 00:00:00:00 00:00:00:08
6. DLEDL: SOURCEID: H_279746176_S_1210191224_U_166040
7. DLEDL: SEGMENTID: H_279746176_S_1210191248_U_10962
8. Sepia Tone Waterfall
9. FROM CLIP NAME: BASIC_CASE
10. DLEDL: EDIT:0 RESOLUTION: 720:486:32:3:0.899998:1399696:BE:P:29.97
11. DLEDL: EDIT:0 FRAME: 0x258193a806c28855
12. DLEDL: EDIT:0 FRAME: 0x258193a906c28855
13. DLEDL: EDIT:0 FRAME: 0x258193aa06c28855
14. DLEDL: EDIT:0 FRAME: 0x258193ab06c28855
15. DLEDL: EDIT:0 FRAME: 0x258193ac06c28855
16. DLEDL: EDIT:0 FRAME: 0x258193ad06c28855
17. DLEDL: EDIT:0 FRAME: 0x258193ae06c28855
18. DLEDL: EDIT:0 FRAME: 0x258193af06c28855
19. DLEDL: START TC: 00:00:00:00
20. DLEDL: REEL:PBS7890 PBS1234567890
```

Line #	Element	Comment
1	Title: BASIC_CASE	Name of the clip represented by the timeline.

Line #	Element	Comment
2	FCM: NON-DROP FRAME	Timeline frame timecode mode (DROP FRAME or NON-DROP FRAME)
3	TITLE: ASSEMBLY RESOLUTION 720:486: 32: 3: 0.899998: 1399696: BE: P: 29.97	Timeline resolution and other basic information, in the following format (line-breaks have been added for clarity): *width:height:* *bits/pixel:* *number of channels:* *pixel ratio:* *frame buffer size:* *byte order:* *scan mode:* *frames per second*
4	FCM: NON-DROP FRAME	Clip frame timecode mode (DROP FRAME or NON-DROP FRAME)
5	001 PBS7890 V C 00:00:00:00 00:00:00:08 00:00:00:00 00:00:00:08	The first edit event uses source material from tape PBS7890. In edit events, long tape names are reduced to 7 characters. The full tape name is presented at the end of the edit. See the description for line 20. It is of type V (video). A (audio) is the other possibility. The dissolve is of type C (cut). D (dissolve) is the other possibility, which is followed by a three digit number indicating the length of the dissolve, in frames. The final four numbers indicate the source material start and end times and the placement of the clip in the timeline.
6 & 7	DLEDL: SOURCEID: H_279746176_S_ 1210191224_U_166040 DLEDL: SEGMENTID: H_279746176_S_ 1210191248_U_10962	Source clips and timeline segments are given unique identifiers for the purposes of tracking media and media history. Each source clip, whether captured, imported, or rendered is assigned a unique identifier. Similarly, each segment or span of a segment within the timeline is assigned a unique segment identifier. The combination of source ID and segment ID allows third-party applications to reference clips when timecodes and/or tape names are unavailable, unreliable, or not unique. More simply, the source ID can be used to distinguish between source clips with the same name. Neither the source ID nor the segment ID must be considered globally unique identifiers (GUIDs). Neither the source nor segment ID persist when copied to another system or project, nor when

Line #	Element	Comment
		restored from archive. In addition, segment IDs change each time work is performed on the segment. If the same source clip is used in different places, it will have a different source ID with each use.
8	Sepia Tone Waterfall	This line is a comment associated with the clip
9	FROM CLIP NAME: BASIC_CASE	Name of the segment
10	DLEDL: EDIT:0 RESOLUTION: 720:486:32:3:0.899998: 1399696:BE:P:29.97	Segment resolution and other basic information See description of line 3 for details
11–18	DLEDL EDIT:0 FRAME 0x1f03ad2006b01d99	The frame IDs associated with the clip
20	DLEDL REEL:PBS7890 PBS1234567890	The tape/reel name in both its abbreviated form as used on line 5, and its full form as used in the timeline. This full form of the tape/reel name is helpful when creating a new clip based on the EDL, since it enables you to assemble the timeline from its constituent parts automatically. The full forms of all tape/reel names that have been abbreviated are always provided at the end of the EDL.

Simple Transition

The following sample output shows the EDL stream for a timeline containing two segments, BEACH and WATERFALL with a dissolve over four frames between them.

```
1. TITLE: SIMPLE_TRANSITION
2. FCM: NON-DROP FRAME

3. TITLE: ASSEMBLY RESOLUTION: 640:480:24:3:1.000000:921616:BE:F1:29.97
4. FCM: NON-DROP FRAME

5. 001 BBC1 V C 00:00:00:17 00:00:00:23 00:00:00:00 00:00:00:06
6. DLEDL: SOURCEID: H_279746176_S_1210195077_U_621587
7. DLEDL: SEGMENTID: H_279746176_S_1210252986_U_7794
8. FROM CLIP NAME: BEACH
9. DLEDL: EDIT:0 RESOLUTION: 640:480:24:3:1.000000:921616:BE:F1:29.97
10. DLEDL: EDIT:0 FRAME: 0x258195de06c2bc6e
11. DLEDL: EDIT:0 FRAME: 0x258195df06c2bc6e
12. DLEDL: EDIT:0 FRAME: 0x258195e006c2bc6e
13. :
14. DLEDL: EDIT:0 FRAME: 0x258195fb06c2bc6e
15. DLEDL: START TC: 00:00:00:00
```

```
16. 002 BBC1 V C 00:00:00:23 00:00:00:23 00:00:00:06 00:00:00:06
17. 002 PBS7890 V D 004 00:00:00:02 00:00:00:08 00:00:00:06 00:00:00:12
18. DLEDL: SOURCEID: H_279746176_S_1210254513_U_730359
19. DLEDL: SEGMENTID: H_279746176_S_1210254513_U_730374
20. Sepia Tone Waterfall
21. FROM CLIP NAME: BEACH
22. TO CLIP NAME: WATERFALL
23. DLEDL: EDIT:0 RESOLUTION: 640:480:24:3:1.000000:921616:BE:F1:29.97
24. DLEDL: EDIT:0 FRAME: 0x258195de06c2bc6e
25. DLEDL: EDIT:0 FRAME: 0x258195df06c2bc6e
26. DLEDL: EDIT:0 FRAME: 0x258195fb06c2bc6e
27. :
28. DLEDL: EDIT:1 FRAME: 0x2581966b06c39c34
29. DLEDL: START TC: 00:00:00:02
30. DLEDL: FOCUS_DESCR CENTERED
31. DLEDL: REEL:PBS7890 PBS1234567890
```

Line #	Element	Comment
1–4	Title, FCM, ASSEMBLY RESOLUTION	Name of the clip represented by the timeline, frame timecode mode, resolution, and other information.
5	001 BBC1 V C 00:00:00:17 00:00:00:23 00:00:00:00 00:00:00:06	The first edit event uses source material from tape BBC1, V (video), C (cut), source material start and end timecode, placement in timeline start and end timecodes.
10–14	DLEDL: EDIT:0 FRAME: 0x258195de06c2bc6e	Frame IDs. The colon (:) indicates material omitted from the example.
16–17	002 BBC1 V C 00:00:00:23 00:00:00:23 00:00:00:06 00:00:00:06 002 PBS7890 V D 004 00:00:00:02 00:00:00:08 00:00:00:06 00:00:00:12	Transitions are indicated by two sequential lines with the same edit event numbers. In this case, the transition is between the source material from tape BBC1 to the source material from tape PBS7890. The second line indicates the transition is of type D (dissolve), taking place over four frames.
21–22	FROM CLIP NAME: BEACH TO CLIP NAME: WATERFALL	The names of the clips involved in the dissolve.
30	DLEDL: FOCUS_DESCR CENTERED	The dissolve is centred with respect to the cut.
31	DLEDL: REEL:PBS7890 PBS1234567890	The tape/reel name in both its abbreviated form as used on line 17, and its full form as used in the timeline. Since, the other tape/reel name (BBC1) is not abbreviated, it goes unreported in this area of the EDL.

Colour Sources and Virtual Tape Names

Colour sources are virtual sources that contain frames generated within the application. The following sample output shows the EDL stream for a timeline containing four colour source segments: colour noise, a solid colour, SMPTE colour bars, and PAL colour bars. Its purpose is to present the *virtual* tape names automatically assigned to colour sources.

```
1. TITLE: VIRTUAL_TAPE NAMES
2. FCM: NON-DROP FRAME

3. TITLE: ASSEMBLY RESOLUTION: 720:486:24:3:0.899998:1049776:BE:F1:29.97
4. FCM: NON-DROP FRAME

5. 001 COLOUR V C 00:00:00:00 00:00:00:04 00:00:00:00 00:00:00:04
6. DLEDL: SOURCEID: H_279746176_S_1210260387_U_813790
7. DLEDL: SEGMENTID: H_279746176_S_1210260388_U_97758
8. FROM CLIP NAME: COL_NOISE
9. DLEDL: EDIT:0 RESOLUTION: 720:486:24:3:0.899998:1049776:BE:F1:29.97
10. DLEDL: EDIT:0 FRAME: 0x258190bf06bfdbd1
11. DLEDL: EDIT:0 FRAME: 0x258190c006bfdbd1
12. DLEDL: EDIT:0 FRAME: 0x258190c106bfdbd1
13. DLEDL: EDIT:0 FRAME: 0x258190c206bfdbd1
14. DLEDL: START TC: 00:00:00:00

15. 002 GREEN V C 00:00:00:01 00:00:00:03 00:00:00:04 00:00:00:06
16. DLEDL: SOURCEID: H_279746176_S_1210260459_U_265566
17. DLEDL: SEGMENTID: H_279746176_S_1210260579_U_1346
18. FROM CLIP NAME: GREEN
19. DLEDL: EDIT:0 RESOLUTION: 720:486:24:3:0.899998:1049776:BE:F1:29.97
20. DLEDL: EDIT:0 FRAME: 0x2581977406c3b91a
21. DLEDL: EDIT:0 FRAME: 0x2581977406c3b91a
22. DLEDL: START TC: 00:00:00:00
23. M2 GREEN 000.0 MSTR I +00:00:00:02

24. 003 SMPE_75 V C 00:00:00:00 00:00:00:02 00:00:00:06 00:00:00:08
25. DLEDL: SOURCEID: H_279746176_S_1210260492_U_164759
26. DLEDL: SEGMENTID: H_279746176_S_1210260584_U_1749
27. FROM CLIP NAME: SMPTE_75
28. DLEDL: EDIT:0 RESOLUTION: 720:486:24:3:0.899998:1049776:BE:F1:29.97
29. DLEDL: EDIT:0 FRAME: 0x2581977506c3b93b
30. DLEDL: EDIT:0 FRAME: 0x2581977506c3b93b
31. DLEDL: START TC: 00:00:00:00
32. M2 SMPE_75 000.0 MSTR I +00:00:00:02

33. 004 PAL_75 V C 00:00:00:00 00:00:00:02 00:00:00:08 00:00:00:10
34. DLEDL: SOURCEID: H_279746176_S_1210260509_U_942020
35. DLEDL: SEGMENTID: H_279746176_S_1210260587_U_924
36. FROM CLIP NAME: PAL_75
37. DLEDL: EDIT:0 RESOLUTION: 720:486:24:3:0.899998:1049776:BE:F1:29.97
38. DLEDL: EDIT:0 FRAME: 0x2581977606c3b94c
39. DLEDL: EDIT:0 FRAME: 0x2581977606c3b94c
40. DLEDL: START TC: 00:00:00:00
```

```
41. DLEDL: REEL:SMPE_75 SMPTE_75
42. M2 PAL_75 000.0 MSTR I +00:00:00:02
```

Line #	Element	Comment
1–4	Title, FCM, ASSEMBLY RESOLUTION	Name of the clip represented by the timeline, frame timecode mode, resolution and other information.
5	001 COLOUR V C 00:00:00:00 00:00:00:04 00:00:00:00 00:00:00:04	The first edit event represents a colour noise segment of four frames duration. It is automatically assigned a source clip from virtual tape COLOUR, of type V (video), with a dissolve of type C (cut).
8	FROM CLIP NAME: COL_NOISE	Name of the segment
10–13	DLEDL: EDIT:0 FRAME: 0x258190bf06bfdbd1etc.	Frame IDs for the colour noise segment. Each ID is different because the colour noise frames are distinct.
15	002 GREEN V C 00:00:00:01 00:00:00:03 00:00:00:04 00:00:00:06	The second edit event represents a solid colour segment of two frames duration. It is automatically assigned the virtual tape name GREEN.
18	FROM CLIP NAME: GREEN	Name of the segment
23 32 42	M2 GREEN 000.0 MSTR I +00:00:00:02 M2 SMPE_75 000.0 MSTR I +00:00:00:02 M2 PAL_75 000.0 MSTR I +00:00:00:02	Some virtual sources such as colour bars and solid colours contain repeated frames, hence have motion events associated with them. The colour noise segment (line 5) does not have a motion event, since each frame in colour noise is distinct. Motion events are also used to report timewarps.
24	003 SMPE_75 V C 00:00:00:00 00:00:00:02 00:00:00:06 00:00:00:08	Event for the SMPTE colour bars, reported as coming from virtual tape SMPE_75. Even virtual tape names can be abbreviated. In this case, the full virtual tape name is SMPTE_75. See line 41.
27	FROM CLIP NAME: SMPTE_75	Name of the segment
29–30	DLEDL: EDIT:0 FRAME: 0x2581977506c3b93b	Frame IDs for the SMPTE colour bars Both IDs are identical because the colour bar frames do not change.
33	PAL_75 V C 00:00:00:00 00:00:00:02 00:00:00:08 00:00:00:10	PAL colour bars event

Line #	Element	Comment
41	DLEDL: REEL:SMPE_75 SMPTE_75	The tape/reel name in both its abbreviated form as used on line 24, and its full form as used in the timeline.

Imports

The following sample output shows the EDL stream for a timeline containing two segments, with a dissolve between them. Because the clips were imported but not cached, the EDL reports the path to the original (full resolution) media.

```
1. TITLE: Soft Imports
2. FCM: NON-DROP FRAME

3. TITLE: ASSEMBLY RESOLUTION: 720:486:24:3:0.899998:0:BE:F1:29.97
4. FCM: NON-DROP FRAME

5. 001 PBS15 V C 00:00:00:00 00:00:00:06 00:00:00:02 00:00:00:08
6. DLEDL: SOURCEID: H_279746176_S_1210108314_U_403593
7. DLEDL: SEGMENTID: H_279746176_S_1210108466_U_3126
8. FROM CLIP NAME: Wally's_material_parachute
9. DLEDL: PATH: /magma/people/sl/MEDIA_SERVER/images/TIF/parachute/
10. DLEDL: EDIT:0 FILENAME: Wally's_material_parachute.(0001@0010).tif
11. DLEDL: START TC: 00:00:00:00

12. 002 PBS15 V C 00:00:00:06 00:00:00:06 00:00:00:08 00:00:00:08
13. 002 1237890 V D 004 00:00:00:00 00:00:00:10 00:00:00:08 00:00:00:18
14. DLEDL: SOURCEID: H_279746176_S_1210108372_U_88045
15. DLEDL: SEGMENTID: H_279746176_S_1210108466_U_3127
16. FROM CLIP NAME: Wally's_material_parachute TO CLIP NAME: HAND3
17. DLEDL: PATH: /magma/people/sl/MEDIA_SERVER/images/TIF/parachute/
18. DLEDL: EDIT:0 FILENAME: Wally's_material_parachute.(0001@0010).tif
19. DLEDL: PATH: /magma/people/sl/MEDIA_SERVER/images/TIF/
20. DLEDL: EDIT:1 FILENAME: HAND3.(0001@0010).tga
21. DLEDL: START TC: 00:00:00:00
22. DLEDL: FOCUS_DESCR CENTERED
23. DLEDL: REEL:1237890 1234567890
```

Line #	Element	Comment
1–4	Title, FCM, ASSEMBLY RESOLUTION	Name of the clip represented by the timeline, frame timecode mode, resolution and other information.
5	001 PBS15 V C 00:00:00:00 00:00:00:06 00:00:00:02 00:00:00:08	The first edit event uses a source clip from tape PBS15, of type V (video), with a dissolve of type C (cut).
9	DLEDL: PATH: /magma/people/sl/MEDIA_SERVER/images/TIF/parachute/	Path to the original, full-resolution material

Line #	Element	Comment
10	DLEDL: EDIT:0 FILENAME: Wally's_material_parachute.(0001@0010).tif	Frame IDs of the full-resolution material The <*start_frame*>@<*end_frame*> means of referring to frames that are sequentially numbered.
12	002 PBS15 V C 00:00:00:06 00:00:00:06 00:00:00:08 00:00:00:08	These two lines indicate a dissolve between two segments, from a clip on tape *PBS15* to a clip on tape *1237890*.
13	002 1237890 V D 004 00:00:00:00 00:00:00:10 00:00:00:08 00:00:00:18	In particular, line 13 indicates the presence of a segment of type V (video) with an edit of type D (dissolve).
16	FROM CLIP NAME: Wally's_material_parachute TO CLIP NAME: HAND3	Details on the dissolve from clip *Wally's_material_parachute* to clip *HAND3*.
22	DLEDL: FOCUS_DESCR CENTERED	The dissolve is centred with respect to the cut.
23	DLEDL: REEL:1237890 1234567890	When a tape name is abbreviated, the abbreviation and full name are presented at the end of the EDL stream.

Backburner Wiretap Server

6

This section explains how to develop a Wiretap client that can communicate with the Backburner managers and render nodes to view and manage jobs on the Backburner network in a custom manner.

Autodesk Backburner is a product-agnostic background job management system that allows multiple jobs such as, I/O operations, composites, and animation scenes to be processed by many computers working collectively on the same network. It is provided with many Autodesk creative applications such as, Flame, 3ds Max®, Maya, and so on. One of its major components, the Backburner Manager is actually a Wiretap server, so you can create a client application to submit, monitor and control rendering jobs on the network using the Wiretap SDK.

Similar to other Wiretap servers in the Wiretap network, the Backburner Manager exposes its database to Wiretap client applications as a tree-like hierarchy of nodes. In this case, the node types relate to background processing. These include the slave (render) servers, server groups, jobs and other elements that make up the Backburner network. By interacting with the metadata in these nodes, your Wiretap client can monitor and alter job status. You can easily suspend, restart or delete jobs, or re-assign them to a different slave server or server group.

The Backburner Manager is well-known to Backburner users because it distributes and manages the jobs on the network amongst other tasks. You can interact with the Backburner Manager using the monitoring functionality built in to the Autodesk creative applications or Backburner Web Monitor. Using the Wiretap SDK, you can create your own client with similar or extended functionality.

This section begins with a few definitions, then examines the Backburner network architecture. Next, it presents the Backburner Manager hierarchy of nodes. It then presents the nodes that contain metadata by which you can monitor and alter job status including server, server group, job and job archive nodes. It explains how to list the Backburner servers on the network. Finally, it outlines how to create and submit a job. Sending attachments is also covered for jobs where large amounts of data are needed by the Backburner Manager and renderer.

Backburner Terminology

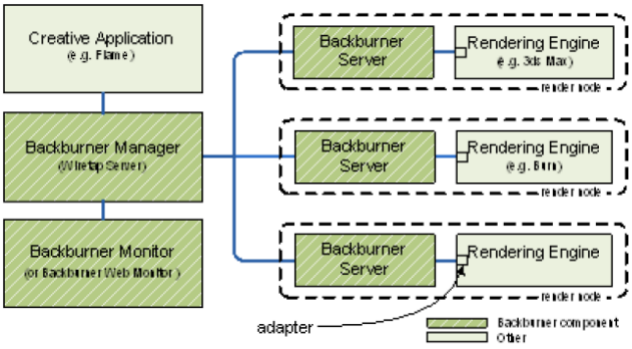
You need to know the following terms that are related to the Backburner Wiretap API to understand the information in this section. For definitions of general Wiretap terms such as, *node* and *metadata*, see Wiretap Terminology (page 3).

Term	Definition
Backburner	Autodesk's distributed job management system for executing rendering and I/O jobs in the background.
Job	A set of one or more tasks submitted to Backburner for processing such as, a 3ds Max scene, Flame Batch setup, or background I/O.

Term	Definition
Backburner Manager	Coordinates jobs submitted by Wiretap clients and delegates them to the Wiretap servers on the Wiretap network.
Backburner Web Monitor	Front-end interfaces for management and control of the Backburner Manager.
Backburner Server	The slave job-processing component (daemon) of Backburner that invokes the rendering or processing engine.
Renderer Rendering Engine	The server-side process responsible for rendering frames.
Processing Engine	Similar to a rendering engine for non-scene processing (such as background I/O).
Plug-in/Adapter	The mechanism by which renderers and processing engines integrate themselves with the Backburner Manager.

Backburner Network Architecture

As illustrated in the following diagram, Backburner consists of the Backburner Manager, Backburner Monitor, and Backburner (slave) servers. These operate in the greater context of creative applications (such as, the Autodesk Creative Finishing applications), adapters/plug-ins, and rendering engines like Burn.



At the centre of Backburner is the Backburner Manager. It receives jobs from render clients, which it then distributes to the render nodes on the network. The Backburner Manager maintains status information about its network of Backburner (slave) servers. It also maintains a database of submitted, active, and completed jobs.

End-user interaction with the Backburner Manager, or jobs on the network is through the Backburner Monitor or Backburner Web Monitor. You can use these interfaces to monitor the progress of a job.

Generally, render nodes consist of a Backburner (slave) server, adapters/plug-ins, and rendering engines. The Backburner server performs the jobs assigned to it by the Backburner Manager. It does so by passing the jobs on to the rendering engine through the plugin/adapter. The adapter is furnished by the render client for the purpose of receiving instructions from the Backburner server and controlling the rendering engine.

The kinds of jobs a server can process depends on the adapter/plug-ins installed on it. Some Autodesk applications such as, 3ds Max have their own rendering engine. Others, such as, Creative Finishing applications share the Burn rendering engine and Wire® processing engine.

Backburner Node Hierarchy

The Backburner Manager maintains a hierarchy of nodes representing the servers and jobs in the system.

The function of each node type is summarized in the following table. A check-mark in the metadata column indicates the node has metadata that can be used by a Wiretap client to view, manage, and control rendering jobs.

Node Type (CAPS) Node ID	Metadata	Description
MANAGER /		Root node of the Backburner hierarchy This node can neither be created nor destroyed by a Wiretap client. See Manager Metadata (page 61).
SERVERLIST /servers		Parent of all Backburner slave servers registered with the Backburner Manager This node can neither be created nor destroyed by a Wiretap client.
SERVER <*server name*>		A slave server (render node) registered with the Backburner Manager. The node ID of a server node is its host name as registered with the Backburner Manager. A Wiretap client cannot create a server node. However, it can delete a server node that has the status of *absent*. A server is automatically marked as absent when it fails to respond to the regular *pings* sent to it by the Backburner Manager. Deleting the node only removes it from the Backburner Manager's node hierarchy. It has no effect upon the slave server itself. See Server Metadata (page 63).
SERVERGROUPLIST /servergroups		Parent of all the server groups handled by the Backnburner Manager. This node can neither be created nor destroyed by a Wiretap client.
SERVERGROUP /servergroups/<*group name*>		Represents a group of Backburner slave servers. A server group is a collection of slave servers. You can create a server group containing all of your facility's fastest render nodes. A Wiretap client can both create and destroy *servergroup* nodes. See Servergroup Metadata (page 67).
JOBLIST /jobs		A container for the current jobs, both active and inactive overseen by Backburner Manager. It can only contain nodes of type *job*. No other type of node can exist under this parent.

Node Type (CAPS) Node ID	Metadata	Description
		This node can neither be created nor destroyed by a Wiretap client. See Joblist Metadata (page 67).
JOB *<alphanumeric>*		A job being handled by the Backburner Manager. A Wiretap client can both create and destroy job nodes. See Job Metadata (page 70).
JOBARCHIVE /archive		A job archive is a compressed file maintained by the Backburner Manager containing information pertaining to archived jobs. This node can neither be created nor destroyed by a Wiretap client. Read-only. See Jobarchive Metadata (page 77).

Workflow, Samples and Tools

Workflow	Related sample and tool	For more information
List all Backburner Wiretap servers on the network.	C++: ■ *listAllservers.C* Python: ■ *listAllservers.py* Command line tool: ■ *wiretap_server_dump*	See: ■ Trying the listAllServers Sample (page 14) ■ Listing Backburner Wiretap Servers (page 78)
List servergroups, (slave) servers	C++: ■ *listChildren.C* Python: ■ *listChildren.py* Command line tools: ■ *wiretap_get_root_node* ■ *wiretap_get_children*	See: Trying the listChildren Sample (page 15)
List jobs List jobs in queue List archived jobs	See above cell, plus: ■ *wiretap_get_metadata*	See: ■ Listing Jobs (page 79) ■ Joblist Metadata (page 67) ■ Jobarchive Metadata (page 77)

Workflow	Related sample and tool	For more information
Get/set individual job metadata Get manager metadata Get/set server metadata	C++: ■ *submitJob.C* Command line tools: ■ *wiretap_get_metadata* ■ *wiretap_set_metadata*	See: ■ Getting and Setting Node Metadata (page 25) ■ Manager Metadata (page 61) ■ Server Metadata (page 63) ■ Job Metadata (page 70)
Delete a job	Command line tools: ■ *wiretap_destroy_node*	See Deleting Nodes (page 26)
Create a job **NOTE** The server-side (renderer) SDK is not yet available. It is therefore not possible to completely implement the full workflow of submitting a job to be executed by your own renderer.	C++: ■ *submitJob.C* Command line tools: ■ *wiretap_create_node*	See: ■ Creating and Submitting a Job (page 79) ■ Node Handles (page 24)

Manager Metadata

Manager nodes contain an XML metadata stream called *info*, which you can use to monitor and set Backburner Manager behaviour related to job processing. It can be obtained using the `getMetaData` method on the `WiretapNodeHandle` object for the manager node, or using the `wiretap_get_metadata`. It is set using the `setMetaData` method and `wiretap_set_metadata` command.

The returned metadata is structured as follows:

```
<info>
  <administrators>root,admin1,admin2,...,adminn</administrators>
  <config>
    <retryCount>server_retries</retryCount>
    <timeBetweenRetriesMS>retry_delay</timeBetweenRetriesMS>
    <maxConcurrentJobs>jobs_on_network</maxConcurrentJobs>
    <forceSingleTaskProcessing>bool</forceSingleTaskProcessing>
    <onJobCompletion>action</onJobCompletion>
    <archiveDays>num_days</archiveDays>
    <deleteDays>num_days</deleteDays>
    <mailServer>smtp_server</mailServer>
    <logLevel>level</logLevel>
  </config>
</info>
```

The following table explains the elements in the XML stream.

Element	Type	Comments
<info>	complex	Container for the other elements.

Element	Type	Comments
<administrators>	string	A comma-separated list of users with *admin* privileges. Read-only.
<config>	complex	Container for elements to configure the behaviour of the Backburner Manager.
<retryCount>	integer	Number of times the Backburner Manager attempts to restart a job on a server that has failed to complete its processing. A failed job might be returned by Backburner to the job processing queue. Set to zero (0) to have job processing halted on the server after its first failure.
<timeBetweenRetriesMS>	integer	The time (in milliseconds) before the Backburner Manager attempts to re-start a job on a server that has failed. Works in conjunction with <retryCount>. Default is 30,000 milliseconds (30 seconds).
<maxConcurrentJobs>	integer	Maximum number of jobs that Backburner sends out for processing on the render farm at the same time.
<forceSingleTaskProcessing>	boolean	By default, the Backburner Manager can assign multiple tasks or *blocks* of tasks to a single server. To force the Backburner Manager to assign each server with only one task at a time, set this boolean to 1 (true).
<onJobCompletion>	string	A token specifying what happens to job metadata once the job has successfully completed. Works in conjunction with <archiveDays> and <deleteDays>. Valid values: ■ leave: Leave in the job list. ■ delete: Remove from the job list after the number of days specified by the <deleteDays> element. ■ archive: Remove from the job list and place in the job archive after the number of days specified by the <archiveDays> element.
<archiveDays>	integer	Number of days a completed job is left in the job list before being archived. Set to 0 (zero) to archive the job immediately after its completion.
<deleteDays>	integer	Number of days a completed job is left in the job list before being deleted. Set to 0 (zero) to delete the job immediately after its completion.

Element	Type	Comments
<mailServer>	string	The Backburner Manager can send job success/failure notifications to the addresses submitted with the job (see Job Metadata (page 70)). This element indicates the server where the smtp mailer daemon is running.
<logLevel>	string	Determines the level of information collected and written to the log file maintained by the Backburner Manager (*backburner.log*). Valid values: ■ error: Records fatal operation failures. ■ warning: Operations that complete with non-fatal errors. ■ info: Successful operations, possibly with minor faults or caveats. ■ debug: Detailed state information includng TCP/IP packet information that is helpful in tracking down bugs. ■ debugX: Extended debug information. For more information, see the *Backburner User Guide*.

Example

The following use of the `wiretap_get_metadata` command returns the metadata for the Backburner manager (always named `/`) on a host called *rossendale:Backburner*.

In this case, the Backburner Manager is set to retry a failed job on the same server three times, waiting 30 seconds before reinitializing the job. The Backburner Manager is configured to issue a maximum of 20 jobs onto the render farm at once, and to archive completed jobs after five days. Any mail notifications are sent out through a daemon running on host *copenhagen*. The log level is *error*.

```
wiretap_get_metadata -h rossendale:Backburner -n / -s info
<info>
  <administrators>root,m_flinders</administrators>
  <config>
    <retryCount>3</retryCount>
    <timeBetweenRetriesMS>30000</timeBetweenRetriesMS>
    <maxConcurrentJobs>20</maxConcurrentJobs>
    <forceSingleTaskProcessing>0</forceSingleTaskProcessing>
    <onJobCompletion>archive</onJobCompletion>
    <archiveDays>5</archiveDays>
    <deleteDays>5</deleteDays>
    <mailServer>copenhagen</mailServer>
    <logLevel>error</logLevel>
  </config>
</info>
```

Server Metadata

Server nodes contain two XML metadata streams: *info* and *schedule*. Some stream elements are read-only, while others are writable. They can be obtained using the `getMetaData` method on the `WiretapNodeHandle`

object for the server node, or using the `wiretap_get_metadata` command-line tool. They are set using the `setMetaData` method and `wiretap_set_metadata` command.

The following table summarizes the purpose of the server node metadata streams

Metadata Stream	Description
info	Basic information relating to server identification, available adapters/plug-ins, job processing activity, and others.
schedule	Get or set a schedule indicating when the node is available to process jobs.

Server info Metadata

The Server node *info* metadata stream allows you to retrieve basic server information including its state, available adapters/plug-ins, current job (if any), and so on. Most elements are read-only. The returned metadata is structured as follows:

```
<info>
  <name>hostname</name>
  <id>server ID</id>
  <state>server state</state>
  <currentJobId>job ID/currentJobId>
  <perfIndex>performance index</perfIndex>
  <description>description</description>
  <plugins>
    <plugin>
      <name>name</name>
      <version>version</version>
      <description>description</description>
    </plugin>
  </plugins>
</info>
```

The following table explains the elements in the *info* metadata stream:

Element	Type	Comments
<info>	complex	Container for the other elements
<name>	string	Server name (host name) Read-only.
<server ID>	integer	Node ID of the server Read-only.
<server state>	string	A token indicating the current activity of the server: ■ absent: Server is no longer seen by the manager, possibly down. ■ active: Server is currently working on a job. ■ suspended: Server has been put on hold. ■ idle: Server is inactive.

Element	Type	Comments
		■ error: Problem on the server Read-only.
<currentJobId>	integer	The job's id as assigned by the Backburner Manager. Read-only.
<perfIndex>	decimal	A value in the range [0–1] indicating the performance level of the server, relative to the other servers on the same job. A score of 1 indicates this is the best-performing server. Read-only.
<description>	string	A short description of the server. Writable.
<plugins> <plugin>	complex	Containers for elements detailing the installed adapters/plug-ins.
<name>	string	The adapter/plug-in name, for example: ■ Burn: The Burn renderer. ■ Command Line Tool: The Backburner cmdjob command-line adapter allows you to submit batch, executable, or script files to Backburner as *custom* jobs. For more information, see the *Autodesk Backburner User Guide*. ■ Wire: Installed with Stone and Wire. Can be used to import/export media, perform Wire transfers, and so on. Used by the Wiretap SDK's background I/O tool, *wiretap_bgio_tool*. For more information see Wiretap Background I/O Tool. Read-only.
<version>	string	Adapter/plug-in version number. Read-only.
<description>	string	A short description of the adapter/plug-in. Read-only.

Example

The following use of the `wiretap_get_metadata` command returns the metadata for a server named *slave1* on a host called *rossendale:Backburner*.

```
wiretap_get_metadata -h rossendale:Backburner -n slave1  -s info
```

```
<info>
  <name>slave1</name>
  <id>slave1</id>
  <state>absent</state>
  <currentJobId>1318676208</currentJobId>
  <perfIndex>1.0</perfIndex>
  <description>Burn Node #10</description>
  <plugins>
    <plugin>
      <name>burn</name>
      <version>2010.1</version>
      <description>description1</description>
    </plugin>
    <plugin>
      <name>Wire</name>
      <version>version1</version>
      <description>description1</description>
    </plugin>
  </plugins>
</info>
```

Server schedule Metadata

The Server node *schedule* metadata stream is a writable stream for viewing and setting server availability. The stream is formatted as follows:

```
<schedule>sun,mon,tue,...,sat</schedule>
```

The following table explains the elements in the *schedule* metadata stream:

Element	Type	Comments
<schedule>	integer	A comma-separated list, one entry for each day of the week indicating server availability. Each entry is the integer representation of a 24-bit binary value. There is one bit for each hour in the day. When the bit is *on*, the server is available for that hour. For example: ■ 0 = 000000000000000000000000 = never available ■ 16777215 = 111111111111111111111111 = always available Days begin at midnight. First day of the week is Sunday. This element is writable.

Example

The following use of the `wiretap_get_metadata` command returns the *schedule* metadata for a server named *slave1* on a host called *rossendale:Backburner*.

In this example, the server is scheduled for availability between Monday to Friday from 7 p.m to 7 a.m, and all of Saturday. It is unavailable on Sundays. This might be the case for a creative workstation used as a render node after-hours.

```
wiretap_get_metadata -h rossendale:Backburner -n slave1  -s schedule
```

```
<schedule>0,16711711,16711711,16711711,
          16711711,16711711,16777215</schedule>
```

Servergroup Metadata

Servergroup nodes contain an XML metadata stream called *info*. You can obtain it using the `getMetaData` method on the `WiretapNodeHandle` object for the `servergroup` node, or using the `wiretap_get_metadata` command-line tool. It is set using the `setMetaData` method and `wiretap_set_metadata` command.

The returned metadata is structured as follows:

```
<info>
  <name>groupname</name>
  <servers>server1,server2,...,servern</servers>
</info>
```

The following table explains the elements in the metadata stream.

Element	Type	Comments
<info>	complex	Container for the other elements
<name>	string	The server group
<servers>	string	A comma-separated list of servers in the group

Example

The following use of the `wiretap_get_metadata` command returns the metadata for a `servergroup` node named *TestGroup* on a host called *rossendale:Backburner*.

```
wiretap_get_metadata -h rossendale:Backburner  -n /servergroups/TestGroup -s info
```

```
<info>
  <name>TestGroup</name>
  <servers>rossendale,belize,new-york</servers>
</info>
```

Joblist Metadata

Joblist nodes contain a read-only XML metadata stream called *info*. It can be obtained using the `getMetaData` method on the `WiretapNodeHandle` object for the `joblist` node, or using the `wiretap_get_metadata` command-line tool.

The joblist node *info* stream contains the metadata for all jobs on the server. This approach is considerably more efficient than retrieving the job node IDs (using the `listChildren` or `wiretap_get_children` command-line tool) then querying each job node for its metadata individually.

The returned metadata contains the info element and attributes related to the current jobs. It is structured as follows (line-breaks have been inserted between attributes for clarity):

```
<info>
  <job  id="jobid"
        name="jobname"
        user="userid"
        description="description"
        host="hostname"
        pluginName="renderername"
        state="jobstate"
        priority="jobpriority"
        submittedTime="YYYY-MM-DD HH:MM:SS"
        startTime="YYYY-MM-DD HH:MM:SS"
        endTime="YYYY-MM-DD HH:MM:SS"
        numTasks="totaltasks"
        numTasksCompleted="completedtasks">
  </job>
</info>
```

The following table explains the elements in the metadata stream.

Element/Attribute	Type	Comments
<info>	complex	Container for other elements
<job </job>	start tag end tag	Container for job attributes
id	integer	The job's ID as assigned by the Backburner Manager.
name	string	The job display name. Unique within current set of Backburner jobs, and set when the job is first created.
user	string	The name of the user that submitted the job. This element is set automatically by Backburner when the job is first created. This value is also returned by `WireTapOS::getUserId`.
description	string	The description as provided by the processing client.
host	string	The name of the host from which the job was submittedThis value is also returned by `WireTapOs::getHostName`.
pluginName	string	The Backburner renderer adapter/plug-in needed for the job. Backburner jobs are always assigned to a specific renderer type. This value is used by the Backburner Manager to determine which servers are eligible to handle a given job.
state	string	The current state of the job: ■ complete: Completed successfully. ■ active: Currently being serviced. ■ suspended: On hold. ■ idle: Not scheduled for service.

Element/Attribute	Type	Comments
		■ waiting: Ready, and waiting to be serviced.
priority	integer	The job priority, from 0 to 100. Zero is the highest priority. 100 means the job is suspended. Default is 0.
submittedTime	string	The time at which the job was originally submitted.
startTime	string	The time at which the job started in the following format: YYYY-MM-DD HH:MM:SS If the job has not started, zeros appear for the values.
endTime	string	The time at which the job completed. When not completed, the time appears as a series of zeros: 0000-00-00 00:00:00.
numTasks	integer	Backburner jobs are broken down into tasks or sub-jobs that define the granularity of the job. Blocks of tasks are delegated to multiple servers by the Backburner Manager in any order. This value is set by the Wiretap client when the job is created. Default is 1.
numTasksCompleted	integer	Set by the Backburner Manager as each task is completed.

Example

The following use of the `wiretap_get_metadata` command returns the metadata for a joblist node on a Backburner Wiretap server called *camiri:Backburner*. The colons (*:*) indicate material omitted from the example.

```
wiretap_get_metadata -h camiri:Backburner  -n /jobs -s info

<info>
  <job   id="1318676208"
         name="Burn_camiri_070620_12.41.07"
         user="root"
         description="Test"
         host="camiri"
         pluginName="burn 2008.0"
         state="suspended"
         priority="50"
         submittedTime="2008-01-16 12:15:18"
         startTime="2008-01-16 12:22:23"
         endTime="0000-00-00 00:00:00"
         numTasks="3"
         umTasksCompleted="0">
  </job>
  :
  :
</info>
```

Job Metadata

Job nodes contain the following metadata streams: `info`, `details`, `xmlDetails`, `state`, and `tasks`. Some streams and elements are read-only and are for the purposes of monitoring jobs already submitted. Others are read-write and can be used to create and control jobs. Like manager and server metadata, job node metadata can be obtained programmatically using the `getMetaData` method on the `WiretapNodeHandle` object for the server node, or using the `wiretap_get_metadata` command-line tool. It is set using the `setMetaData` method and `wiretap_set_metadata` command.

The following table summarizes the purpose of the job node metadata streams.

Metadata Stream	Description
info	Basic information needed to create, monitor and/or control a job.
details and xmldetails	Job-specific or renderer-specific metadata for a job.
state	Current state of activity for the job such as, *waiting*, and *active*. Can be used to restart a job.
tasks	Read-only information about the individual tasks making up a job.

NOTE To retrieve selected metadata for all jobs on a server at once, use the *joblist* node instead. For all archived jobs, use the *jobarchive* node. See Jobarchive Metadata (page 77).

Job info Metadata

The `info` metadata stream contains basic job information common to all jobs. Some elements are read-only, and are automatically assigned by the system when you create the job. Others are writable, allowing you to control or modify the job.

Job node `info` metadata is formatted as follows:

```
<info>
  <name>jobname</name>
  <id>jobid</id>
  <description>description</description>
  <tasknameList>tskname1,tskname2,...,tsknamen</tasknameList>
  <user>userid</user>
  <host>hostname</host>
  <pluginName>renderername</pluginName>
  <state>jobstate</state>
  <priority>jobpriority</priority>
  <numTasks>totaltasks</numTasks>
  <numTasksCompleted>completedtasks</numTasksCompleted>
  <percentTasksCompleted>percentage</percentTasksCompleted>
  <serverGroup>groupname</serverGroup>
  <servers>server1,server2,...,servern</servers>
  <serverCount>count</serverCount>
  <submittedTime>YYYY-MM-DD HH:MM:SS</submittedTime>
  <startTime>YYYY-MM-DD HH:MM:SS</startTime>
  <endTime>YYYY-MM-DD HH:MM:SS</endTime>
  <dependencies>jobid1,jobid2,...,jobidn</dependencies>
```

```
    <emailDest>name@host.com</emailDest>
    <emailFrom>name@ host.com</emailFrom>
    <emailCompletion>name@host.com</emailCompletion>
    <emailFailure>name@host.com</emailFailure>
    <emailProgress>name@host.com</emailProgress>
    <lastError>errorstring</lastError>
  </info>
```

The following table explains the elements in the metadata stream.

Element	Type	Comments
<info>	complex	Container for other elements
<name>	string	The job display name. Must be unique within current set of Backburner jobs, and set when the job is first created. Required. Read-only.
<id>	integer	The job's ID as assigned by the Backburner Manager. Read-only.
<description>	string	The description as provided by the processing client.
<tasknameList>	string	A comma-separated list naming each task in the job. If you do not set this element explicitly, a default name is given to each task. The default name has the form: task **X**-**Y** Where X is the current task number, and Y is the total number of tasks in the job. For example, a job with only one task has a default task name of *task 1-1*. Optional. Writable. **NOTE** The task names you set here become read-only elements in the job node's *tasks* metadata stream. See Job tasks Metadata (page 75).
<user>	string	The name of the user that submitted the job. This element is set automatically by Backburner when the job is first created. This value is also returned by `WireTapOS::getUserId`. Read-only.
<host>	string	The name of the host from which the job was submitted. This value is also returned by `WireTapOs::getHostName`. Read-only.
<pluginName>	string	The Backburner renderer adapter/plug-in needed for the job. Backburner jobs are always assigned to a specific renderer type. This value is used by the Backburner Manager to determine which servers are eligible to handle a given job.
<state>	string	The current state of the job: ■ complete: Completed successfully.

Element	Type	Comments
		■ active: Currently being serviced. ■ suspended: On hold. ■ idle: Not scheduled for service. ■ waiting: Ready, and waiting to be serviced.
<priority>	integer	The job priority from 0 to 100. Zero is the highest priority. 100 means the job is suspended. Default is 0.
<numTasks>	integer	Backburner jobs are broken down into tasks or sub-jobs that define the granularity of the job. Blocks of tasks are delegated to multiple servers by the Backburner Manager in any order. This value is set by the Wiretap client when the job is created. Default is 1. Mandatory.
<numTasksCompleted>	integer	Set by the Backburner Manager as each task is completed. Read-only.
<percentTasks Completed>	integer	Set by the Backburner Manager as each task is completed. Read-only.
<serverGroup>	string	The server group to which the job will be submitted. Only servers in the specified server group work on the given job, unless the group is set to use idle non-group servers.
<servers>	string	A comma-separated list of servers to which the job's tasks are submitted. When no servers are specified, the Backburner Manager is free to assign the job to any server with the appropriate adapter/plug-in. This metadata is ignored if a server group is specified.
<submittedTime>	string	The time stamp at which the job is submittted (YYYY-MM-DD HH:MM:SS).
<startTime>	string	The time stamp at which the job is started (YYYY-MM-DD HH:MM:SS).
<endTime>	string	The time stamp at which the job is completed (YYYY-MM-DD HH:MM:SS).
<dependencies>	string	A comma-separated list of job node IDs on which the current job is dependent. The current job is started only when all the dependent jobs are complete.

Element	Type	Comments
<serverCount>	integer	The maximum number of servers that can work on this job at any one time. Set this to zero to run on all servers.
<emailDest>	string	Email address to which notification of job completion or failure is sent.
<emailFrom>	string	When email is generated by the Backburner Manager, this is the notification sender's email address.
<emailCompletion>	string	Turns on/off email notification on job completion: ■ 0: Email on completion is *off*. ■ 1 (default): Email on completion is *on*.
<archived>	integer	Turns on/off email notification on job completion: ■ 0: Email on completion is *off*. ■ 1 (default): Email on completion is *on*.
<emailProgress>	integer	Trigger email notification on the completion of every Nth task. Set to zero (default) to disable.
<lastError>	string	The last error message for the most recent task executed by the Backburner Manager. Read-only.

Example

The following use of the `wiretap_get_metadata` command returns the `info` metadata for a job with ID `1318676208` on a Backburner Wiretap server called *camiri:Backburner*. As illustrated by the example, not all elements are returned for each job. For example, if you did not provide values for email-related elements when creating the job, these are not returned with the metadata stream.

```
wiretap_get_metadata -h camiri:Backburner  -n 1318676208 -s info

<info>
  <name>Burn_camiri_070620_12.41.07</name>
  <id>1318676208</id>
  <description>Clip: COL_NOISE</description>
  <user>root</user>
  <host>camiri</host>
  <pluginName>burn 2008.0</pluginName>
  <state>suspended</state>
  <priority>50</priority>
  <numTasks>3</numTasks>
  <numTasksCompleted>3</numTasksCompleted>
  <percentTasksCompleted>100</percentTasksCompleted>
  <serverGroup></serverGroup>
  <submittedTime>2009-02-09 07:12:21</submittedTime>
  <startTime>2009-02-09 07:12:21</startTime>
```

```
    <endTime>2009-02-09 07:19:00</endTime>
    <dependencies><dependencies/>
    <servers>camiri</servers>
    <serverCount>4</serverCount>
    <lastError>camiri</lastError>
  </info>
```

Job details and xmlDetails Metadata

The job node `details` and `xmlDetails` metadata streams facilitate the transmission of custom information in ASCII format from the Wiretap client to the Backburner server for processing. The `details` stream is converted into XML automatically. For information that is already XML compliant, there is the `xmlDetails` stream. When using these data streams, take care to consider the amount of data being transmitted, and backwards compatibility for future adapters/plug-ins.

An adapter is a plugin furnished by the rendering client. It receives job requests from Backburner Manager through the Backburner server and is responsible for launching a rendering engine to carry out the requests.

Both the `details` and `xmlDetails` metadata streams serve the purpose of supplying the adapter with custom information. For example, you can use the `details` and `xmlDetails` metadata streams to specify instructions specific to the renderer launched by the adapter, or to provide tailored instructions for a particular job. The job node `details` and `xmlDetails` metadata streams are free-form and there is no structure imposed. It is up to the Wiretap client and renderer to agree on a metadata format for the custom information.

The following table shows the encoding that automatically takes place in the details stream for the less-than (<), greater-than (>) and percent (%) characters. It is up to your adapter/plug-in to decode these appropriately. Null characters are not permitted in the details stream:

Character	Encoding
<	%3C
>	%3E
%	%25
null	*not permitted*

For metadata that is already XML-compliant use the `xmlDetails` stream, which you can use to monitor and control a job after it is submitted. Make sure that the stream is encoded correctly. Using the `xmlDetails` stream with non-XML encoded data can yield unexpected results including Backburner malfunction.

Note the following points related to the `details` or `xmlDetails` metadata stream.

- Do not use the `details` or `xmlDetails` streams to transmit binary data or large amounts of non-binary metadata (more than 1-5 MB). Send these as an attachment. See Sending Attachments to the Backburner Manager (page 80).
- Consider identifying the metadata stream using a version number so that future renderers can accommodate submissions from older Wiretap clients.

Job State Metadata

Job nodes contain a metadata stream called `state` that is useful to ascertain or modify a job's state of activity. It can be obtained using the `getMetaData` method on the `WiretapNodeHandle` object for the manager node, or using the `wiretap_get_metadata` command. It can be set using the `setMetaData` method or the `wiretap_set_metadata` command.

NOTE State information is also included as a read-only value in the job node's info metadata.

The job state metadata is returned as a string as shown in the following table.

String Value	Meaning
waiting	Ready, and waiting to be serviced.
active	Currently being serviced.
idle	Not scheduled for service.
suspended	On hold. This is the default state for a newly created job.
complete	Completed successfully.
restarting	Returning the job to its initial sate; setting the state to *waiting*.

The following table provides examples of specific state transitions.

Start State	End State	Result
complete	waiting/restarting	Restarts the job.
suspended/waiting/active/idle	restarting	Restarts the job.
waiting/active/idle	suspended	Suspends execution of the job.
suspended	waiting	Resumes a suspended job.

Job Tasks Metadata

The job node `tasks` metadata stream contains read-only information about the individual tasks that make up a job. It can be obtained using the `getMetaData` method on the `WiretapNodeHandle` object for the manager node, or using the `wiretap_get_metadata` command. It is set and updated by the Backburner manager.

Job node tasks metadata is formatted as follows:

```
<tasks>
  <task>
    <number>tasknumber</number>
    <name>task start-end</name>
    <startTime>YYYY-MM-DD HH:MM:SS.MS</startTime>
    <elapsedTime>HH:MM:SS.MS</elapsedTime>
    <elapsedTimeMsec>MS</elapsedTimeMsec>
    <server>servername</server>
    <state>taskstate</state>
    <lastError>message</lastError>
  </task>
</tasks>
```

The following table explains the elements in the metadata stream.

Element	Type	Comments
<number>	string	The task number for this task.
<name>	string	Name of the current task. By default, the task name has the form: task X-Y Where *X* is the is the current task, and *Y* is the total number of tasks in the job. For example, task 5-7 indicates this is the 5th task in a job consisting of 7 tasks in total. You can set task name information explicitly in the job node *info* metadata stream. See Job info Metadata (page 70).
<startTime>	string	The time stamp of the task assignment (YYYY-MM-DD HH:MM:SS.MS).
<elapsedTime>	string	The time duration consumed by the task (HH:MM:SS.MS).
<elapsedTimMsec>	integer	The time duration consumed by the task in milliseconds only.
<server>	string	The name of the server where the task is being executed.
<state>	string	The state of the task (active, complete, waiting, error).
<lastError>	string	The last execution error message.

Example

The following use of the `wiretap_get_metadata` command returns the `tasks` metadata for a job with ID `1318676208` on a Backburner Wiretap server called *cardigan:Backburner*.

```
wiretap_get_metadata -h cardigan:Backburner -n 1318676208 -s tasks

<tasks>
  <task>
    <number>1</number>
    <name>task 1-2</name>
    <startTime>2008-11-18 14:53:25:011</startTime>
    <elapsedTime>00:16:13.721</elapsedTime>
    <server>burn03</server>
    <state>error</state>
    <lastError>Launching renderer failed </lastrror>
  </task>
  <task>
    <number>2</number>
    <name>task 2-2</name>
    <startTime>2008-11-18 14:53:27:0705</startTime>
    <elapsedTime>00:00:03.216</elapsedTime>
    <server>burn04</server>
    <state>complete</state>
```

```
    <lastError></lastError>
  </task>
</tasks>
```

Jobs Events

Related Command Line Tool

- `wiretap_event_listener`

The client can monitor jobs using events. An event is sent for each Job node added or removed. An event is also sent every time a Job node's state changes.

To receive event you need to implement a `WiretapEventHandler` object and register it using `WireTapNodeHandler::registerEventHandler()` on the node you want to monitor.

Jobarchive Metadata

Jobarchive nodes contain a read-only metadata stream called `info`. Like all metadata streams, it can be obtained using the `getMetaData` method on the `WiretapNodeHandle` object for the joblist node, or using the `wiretap_get_metadata` command-line tool.

The jobarchive node `info` stream contains the metadata for all archived jobs on the server. This approach is considerably more efficient than retrieving the archived job node IDs (using the `listChildren` or `wiretap_get_children` command-line tool) and then querying each job node for its metadata individually.

The returned metadata contains the `info` element and attributes related to archived jobs. It is structured as follows (line-breaks are inserted between attributes for clarity):

```
<info>
  <job id="jobid"
       name="jobname"
       user="userid"
       description="description"
       submittedTime="timesent"
       endTime="timecompleted"
       pluginName="renderername"
  </job>
</info>
```

The following table explains the elements in the metadata stream.

Element/Attribute	Type	Comments
<info>	complex	Container for other elements
<job </job>	start tag end tag	Container for attributes
id	integer	The job's ID as assigned by the Backburner Manager.
name	string	The job display name. Unique within current set of Backburner jobs, this is set when the job is first created.

Element/Attribute	Type	Comments
user	string	The name of the user that submitted the job. This element is set automatically by Backburner when the job is first created. This value is also returned by `WireTapOS::getUserId`.
description	string	The description as provided by the processing client.
submittedTime	string	The time at which the job is submitted in the following format: YYYY-MM-DD HH:MM:SS.
endTime	string	The time at which the job is completed. When there is no end-time, or the job is archived before completion, the time appears as a series of zeros: 0000-00-00 00:00:00.
pluginName	string	The Backburner renderer adapter/plug-in needed for the job. Backburner jobs are always assigned to a specific renderer type. This value is used by the Backburner Manager to determine which servers are eligible to handle a given job.

Example

The following use of the `wiretap_get_metadata` command returns the `info` metadata for jobs archived on a Backburner Wiretap server called *santos:Backburner*. The colons (:) indicate material omitted from the example.

```
wiretap_get_metadata -h santos:Backburner -n /archive -s info

<info>
  <job  id="1040720950"
        name="job4"
        user="nelsonr"
        description="/tmp/dst"
        submittedTime="2008-01-08 15:31:13"
        endTime="0000-00-00 00:00:00"
        pluginName="Wire"
  </job>
:
:
</info>
```

Listing Backburner Wiretap Servers

As explained in Discovering Wiretap Servers (page 17), to distinguish between the multiple Wiretap servers that often co-exist on the same host, Wiretap server IDs consist of two parts: the host machine and the type of database it exposes. Backburner Wiretap servers follow the same naming convention.

Since Backburner databases are always named *Backburner*, the full specification for a Backburner server is *hostname:Backburner*. For example, if the Wiretap server is *montreux*, the associated Backburner server is discoverable as *montreux:Backburner*.

For example, issuing the following command from the command line lists only the Backburner Wiretap servers on the network:

```
wiretap_server_dump -d Backburner
```

Listing Jobs

The list of jobs on a server is obtained using the `getChild` method on the `WiretapNodeHandle` object for the server's job node. It can be obtained at the command-line using the `wiretap_get_children` command.

For example, the following command returns the jobs on a Backburner Wiretap server called *montreux*:

```
wiretap_get_children -h montreux:Backburner -n /jobs
```

Creating and Submitting a Job

In addition to monitoring and controlling jobs already deployed on the Backburner network, you can use the Backburner Client API for creating and submitting new jobs.

The following procedure outlines the steps in job creation. For details, see the code sample *submitJob.C*.

To create and submit a job:

1. Create the job node.

   Call `WireTapNodeHandle::createNode()` on the joblist node (`/jobs`), specifying the node type and display name.

   - The node type must be of type `JOB`.
   - The display name appears in the Backburner Monitor as the job name. The display name must be unique across all jobs.
   - The job node is automatically populated with the current local user, host, date, and time.
   - The resulting job node is empty and in the suspended state.

2. Specify basic job information.

   Call `WireTapNodeHandle::setMetadata()` on the new job node, specifying the `info` metadata stream. This sets the information needed by all jobs. The `pluginName` and `numTasks` fields are mandatory.

3. Specify job- or renderer-specific information (optional).

   Call `WireTapNodeHandle::setMetadata()` on the new job node, setting the `details` or `xmlDetails` metadata stream as desired to set job or renderer-specific data.

4. Attach a file (optional).

   Call `WireTapServerHandle::pushStream` to attach any file for transfer. The stream ID for the attachment for a given job node is simply the job's node ID. See Sending Attachments to the Backburner Manager (page 80).

5. Set the job state to waiting.

   By default, jobs are in a *suspended* state when created. The last step in job creation is to submit the job for processing by changing its state to *waiting*.

   Call `WireTapNodeHandle::setMetadata()` on the job node's state metadata stream, setting the state to *waiting*.

Sending Attachments to the Backburner Manager

Backburner jobs can sometimes require that large amounts of auxiliary data be transferred to the render node. For example, jobs can have setups or LUTs that are needed by the render node for correct processing. You can use the job node `details` and `xmlDetails` metadata streams to submit smaller amounts of custom text or XML-based data, but for sending larger amounts of data including binary data as attachments, use the Wiretap stream API.

The stream API provides a high-bandwidth out-of-phase connection to the Backburner Manager. The connection does not block concurrent metadata or node hierarchy requests. You can gain access to the stream API using the `pushStream` method on the `WireTapServerHandle` object for the Backburner Manager node.

Usually, you can store the attachment data in a local file from which the stream API reads the data. To reduce transfer time and storage requirements on the manager and improve scalability, it is recommended that you compress the local file prior to sending.

FAQs and Troubleshooting 7

This section provides a list of frequently asked questions and troubleshooting techniques.

General API Issues

This section includes questions on various topics such as errors, threads, server availability, and more.

How does Wiretap handle errors?

Functions that fetch and manipulate data on a Wiretap server might fail. These functions all return a boolean value: true on success, false on failure. Each of the three classes that interact with Wiretap servers (`WireTapNodeHandle`, `WireTapServerHandle`, and `WireTapServerList`) have a member function called `lastError()`, which looks for the error message arising from calls to other member functions that interact with Wiretap servers. If a function call fails, the error string obtained from `lastError()` must be used or stored immediately, since it will be overwritten the next time a method is called on the object that experienced the failed call.

Can I search for a specific error code?

There is no way to check for a specific error code. Error strings are often generated dynamically from contextual information, and are subject to change by the server implementation. Error messages returned by Wiretap are intended to be viewed by the end-user.

Is Wiretap thread-safe?

Wiretap nodes and server handles can exist in different threads, but a single handle cannot be used simultaneously from each thread. For example, multiple threads can be used to traverse a node hierarchy, but they cannot share the same node or server handles.

The one exception to this rule is the `WireTapNodeHandle.stop` function, which is intended to be used by an asynchronous thread to halt a pending request on a handle.

Can I increase performance using multiple threads for frame I/O?

Yes, but because each thread is contending for the same network pipe, there is not much benefit in performing multi-threaded I/O. It would be better to serialize the requests from each thread.

NOTE The Wiretap server automatically performs read-ahead when accessing frames of a clip sequentially.

How long can handle objects remain open?

Node and server handle objects can remain open indefinitely. A `WireTapNodeHandle` object will cache information from previous calls. You might want to recreate the node handle to refresh its contents in case the node is modified on the server by someone or some program other than your Wiretap client.

The `WireTapServerHandle` object caches a connection to a server, as well as other server information. Instances of `WireTapNodeHandle` connect to the server through the server handle. The server handle automatically reconnects to the server if the connection is broken (for example, if the server is restarted).

All wiretap object should be destroyed however before `WireTapClientUninit()` is called or WireTapClient guard object is destroyed.

Why do I see a Backburner server in my server list?

The Backburner Manager is a Wiretap server and can publish jobs and servers (slaves) as nodes in a Wiretap hierarchy. The Backburner Wiretap server's database ID is *Backburner*, which makes it easily identified and/or skipped, as required.

Generally, Wiretap allows more than one Wiretap server to run on the same host machine. Each server exposes its database as a Wiretap node hierarchy. Each server is tagged with a machine-unique database identifier, as well as its own set of TCP port numbers configurable by the given Wiretap server using the `WireTapServerId` class.

Why I cannot see all Wiretap servers?

Wiretap uses network multicast technology to broadcast the identities of all Wiretap servers on the network to each other. The Wiretap Client API silently finds one server on startup through which it gains knowledge of all others. The Wiretap Client API includes the `WireTapServerList` class, which allows clients to obtain a list of the Wiretap servers that are active on the network.

The most common reason for a server not appearing in the list is that the client or server is not on the same network switch. Wiretap transmits large frames across the network and by default, does not discover servers on other network switches to limit bandwidth bottlenecks across the network bridge.

Another common reason for not seeing servers is that the router/switch can be set to disable multicast requests. You can verify this by running ping `224.0.0.1` from the command-line of the client machine. All machines on the same subnet must respond almost instantly. If not, the router likely has disabled this functionality. See `/usr/discreet/cfg/network.cfg` on Linux and Mac OS X installations to properly configure your network.

If required, you can manually specify the host name or IP address when specifying a server for a Wiretap client.

Why is there a WireTapStr class?

When distributing a library, it is not uncommon to have to wrap standard library data types (such as `std::string`) to ensure that there are no issues when a host application chooses a different standard C library other than the one chosen by Wiretap.

This forces the caller to duplicate strings when converting from Wiretap strings to their own string class for internal use. The `WireTapStr` class is just a cut-down version of `std::string`.

Who defines node types?

Wiretap does not define node types beyond a generic node and a clip node. It is up to each Wiretap server to define extended node types to represent components such as, projects, libraries, reels, partitions, and so on. Node types are represented by strings and are unique within a given Wiretap server implementation.

Once defined, string node type values are permanent and can never change between versions of a particular server. They are not guaranteed to be identical across vendors of Wiretap servers.

Why do I get broken pipe signals when Wiretap connections are severed?

On UNIX-based systems (including Mac OS X), broken pipe signals are triggered when a socket connection to a Wiretap server is broken. The default signal handler set up by the operating system causes the host application to exit the moment the signal is triggered, which is undesirable.

The solution is to override the default signal handler to ignore the signal and allow the socket error to be propagated to the Wiretap client. The following sample code is provided for UNIX-based systems:

```
#include <signal.h>
struct sigaction sa;
sigemptyset( &sa.sa_mask );
sa.sa_flags = 0;
sa.sa_handler = SIG_IGN;
sigaction( SIGPIPE, &sa, 0 );
```

Signal handler set up must be done before Wiretap is initialized, ideally, as early as possible on application startup. All of the Wiretap tools shipped with the SDK perform this by default. The sample programs do not.

The rationale for forcing this upon the host application is that signal handlers are global to an application, and must not be tweaked by an API library such as, Wiretap. Most applications perform this operation to trap this and other signals on startup.

Why do I crash when trying to use a WireTapServerHandle objects?

`WireTapServerHandle` object be used within a `WireTapClientInit()` and `WireTapClientUninit()` call pair. Using a server handle before `WireTapClientInit()` or after `WireTapClientUninit()` result in undefined beheavior. One can also use the WireTapClient guard object to simplify return paths handling instead of calling `WireTapClientUninit()` everywhere.

IFFFS Wiretap Server Issues

These questions are related to the IFFFS Wiretap server.

Why is media unlinked in Smoke if I create a timeline through the IFFFS Wiretap Server?

When a timeline is created through Wiretap, the sources are searched for within the reel. Hence, the timeline must be created within the same reel as the sources. If the sources are not in the reel or the tape names of the sources do not match the tape names specified in the Wiretap DMXEDL, the media will be unlinked.

Version Compatibility

These questions are related to versions of Wiretap servers and Wiretap Client API.

Do I need to rebuild tools when a new Wiretap server and API is released?

The Wiretap API is backward-compatible with new servers, so there is no need to recompile. The only time you need to update is to get new API features or bug fixes.

How can I tell which version of the server is running? Why do I need to know?

Each version of a Wiretap server enables/extends its feature set. When talking to a specific server, the API might need to know what functionality is available, so a Wiretap server provides vendor, product, and version information to the client.

Compiling, Linking, and Executing

These questions are related to the build and runtime issues on Mac OS X.

Problems executing your application under Mac OS X

Mac OS X dynamic libraries (*.dylib*) embed the full path to their installed location. This is different from dynamic libraries on Unix (*.so* and *.dll*), where the path is set at link time by the executable application.

The Mac OS X dynamic library supplied with Wiretap uses a preset hard-coded path that might not match your desired install location. By default, the Wiretap dynamic library (`libwiretapClientAPI.dylib`) sets its install path to */Library/Frameworks*, which is the standard install location for 3rd-party (non-OS) libraries available to all users.

To work around this issue, you can use the environment variable (`DYLD_LIBRARY_PATH`), provided by Mac OS X, to have the executable application override the embedded path. Alternatively, you can use the command-line tool `install_name_tool` to reset the install path of the dynamic library and replace the contents (in place) within the library. The Wiretap dynamic library is built with 1024 bytes of reserved install path.

To change the library install path:

1. From the command line, run `install_name_tool`:

   ```
   install_name_tool -id <new_path> <current_install_location>
   ```

   where, `current_install_location` is where you installed the library, and `new_path` is the desired new path for the library. For example, suppose you installed the library to */tmp*. To change the path to */usr/lib*, run the `install_name_tool` as follows:

   ```
   install_name_tool -id /usr/lib/libwiretapClientAPI.dylib /tmp/libwiretapClientAPI.dylib
   ```

2. From a command-line, run `otool` to verify the path change:

   ```
   otool -L libwiretapClientAPI.dylib
   ```

   For example, if you set the new path to */usr/lib/libwiretapClientAPI.dylib*, `otool` returns something similar to the following:

   ```
   libwiretapClientAPI.dylib:
   /usr/lib/libwiretapClientAPI.dylib
   (compatibility version 0.0.0, current version 0.0.0)
   /usr/lib/libstdc++.6.dylib
   (compatibility version 7.0.0, current version 7.4.0)
   /usr/lib/libgcc_s.1.dylib
   (compatibility version 1.0.0, current version 1.0.0)
   /usr/lib/libSystem.B.dylib
   (compatibility version 1.0.0, current version 88.1.2)
   ```

3. Compile your executable with the changed library.

Reading and Writing Video Media

These questions address how to use the Wiretap Client API to read and write video media.

The WireTapNodeHandle and WireTapServerHandle classes both have methods to read frames. Which should I use?

The `WireTapNodeHandle` class provides a `readFrame` method that obtains a frame at a specified index on the total number of frames, which is the simplest method to use and appropriate in most cases. To interpret a frame, you also need format information, which can be obtained from a node handle (if the node is a clip node) by calling the `WireTapNodeHandle.getClipFormat` method.

The `WireTapServerHandle` class provides a `readFrame` method which can be used if you have access to a frame ID only (without the node ID). For example, this situation can arise when you are interpreting timeline metadata that contains frame IDs only, rather than node ID and frame indexes.

Unless you do not have access to the node handle, you must always use the `WireTapNodeHandle` methods to read and write frames. The `WireTapServerHandle` method is for random frame access.

How do I read frames from a network-mounted framestore?

There are two options available for reading/writing frames (such as DPX, OpenEXR, and so on). You can read frames through the Wiretap server: Wiretap converts the frames into raw RGB. Or you can read them directly from the storage device:

- **Read the frame through Wiretap** – The Wiretap server reads and transmits the data from the filesystem. This might involve an extra network hop from the network drive to the server of the client. However, media is automatically converted by the Wiretap server from the format on the filesystem into raw uncompressed RGB data with no header. The sample program `readFrames.C` (included in the Wiretap SDK) shows how to do this.
- **Read the media directly from the network storage device** – Wiretap can provide a network file path representation of the frame ID for each frame. Using the frame file path, the Wiretap client can read the formatted file directly off the storage device. The sample program *listFrames.C* (included in the Wiretap SDK) shows how to do this.

The second option has two important advantages:

- It reduces network usage, since there is no hop through the Wiretap server.
- It has no impact on the Wiretap server. This can be important if the Wiretap server is located on a Creative Finishing workstation. The workstation's CPU is protected from concurrent Wiretap server activity, but competition for its resources will occur.

Reading Audio Media

These questions are related to how Wiretap handles audio media.

How do I read audio?

Audio and video media are quite different from an I/O standpoint. Audio samples are extremely small. One second of uncompressed audio at 44kHz requires no more than 172KB, while an uncompressed video media requires 30MB for each second of 8-bit NTSC.

Wiretap is a client-server architecture operating across a network. You must be aware of the size of the I/O being performed so that the network is not clogged with small audio media requests. For this, the Wiretap frame API is currently used to read blocks of audio samples. Each *frame* of audio represents a block of samples,

the size of which is decided by the Wiretap server using the number of frames and the `WireTapClipFormat` object of the audio clip node in question. For more information, see Memory Required per Frame (page 39).