An API for CBF/imgCIF
Crystallographic Binary Files with ASCII Support

Version 0.7.6
15 July 2006

by
Paul J. Ellis
Stanford Synchrotron Radiation Laboratory

and
Herbert J. Bernstein
Bernstein + Sons
yaya@bernstein-plus-sons.com

CBFlib Notices

COPYING

All of the CBFlib 0.7.5 package may be distributed under the terms of the GNU General Public License (the GPL), see

http://www.gnu.org/licenses/gpl.txt

Alternatively most of the CBFlib 0.7.5 package may be distributed under the terms of the GNU Lesser General Public License (the LGPL), see

http://www.gnu.org/licenses/lgpl.txt

The portions that may be distributed under the LGPL indentified as such in the comments of the relevant files, and include the portions constituting the API, but do not include the documentation nor does it include the example programs. The documentation and examples may only be distributed under the GPL.

THE FIRST ALTERNATIVE LICENSE FOR ALL OF CBFLIB (GPL)
(Valid for versions of CBFlib starting with release 0.7.5)

```
========================== GPL STARTS HERE ================================
                GNU GENERAL PUBLIC LICENSE
                   Version 2, June 1991

 Copyright (C) 1989, 1991       Free Software Foundation, Inc.
                          51 Franklin St, Fifth Floor,
                          Boston, MA  02110-1301  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                       Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
```

you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third

parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering

access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding

those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or

```
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA


Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General
Public License instead of this License.

  ============================ GPL ENDS HERE ===================================
```

THE SECOND ALTERNATIVE LICENSE FOR CERTAIN PORTIONS OF CBFLIB INCLUDING THE API ITSELF, BUT NOT THE DOCUMENTATION AND NOT THE EXAMPLES (LGPL)

(Valid for versions of CBFlib starting with release 0.7.5)

```
 ============================ LGPL STARTS HERE ================================
                GNU LESSER GENERAL PUBLIC LICENSE
                     Version 2.1, February 1999

 Copyright (C) 1991, 1999 Free Software Foundation, Inc.
     51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL.  It also counts
 as the successor of the GNU Library Public License, version 2, hence
 the version number 2.1.]

                          Preamble

  The licenses for most software are designed to take away your
```

freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

  This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

  When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

  To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

  For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

  We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

  To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

  Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

  Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

  When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

  We call this license the "Lesser" General Public License because it

does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does

and what the program that uses the Library does.

  1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

  You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

    a) The modified work must itself be a software library.

    b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

    c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

    d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

    (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

  3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so

that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by

this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

    a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

    b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

    c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

    d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

    e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

  For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

  It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

  7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

    a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

    b) Give prominent notice with the combined library of the fact

that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

    <one line to give the library's name and a brief idea of what it does.>
    Copyright (C) <year> <name of author>

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public

```
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the library, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the
  library `Frob' (a library for tweaking knobs) written by James Random Hacker.

  <signature of Ty Coon>, 1 April 1990
  Ty Coon, President of Vice

That's all there is to it!

========================== LGPL ENDS HERE ================================
```

The following notice applies to this work as a whole and to the works included within it:

- Creative endeavors depend on the lively exchange of ideas. There are laws and customs which establish rights and responsibilities for authors and the users of what authors create. This notice is not intended to prevent you from using the software and documents in this package, but to ensure that there are no misunderstandings about terms and conditions of such use.
- Please read the following notice carefully. If you do not understand any portion of this notice, please seek appropriate professional legal advice before making use of the software and documents included in this software package. In addition to whatever other steps you may be obliged to take to respect the intellectual property rights of the various parties involved, if you do make use of the software and documents in this package, please give credit where credit is due by citing this package, its authors and the URL or other source from which you obtained it, or equivalent primary references in the literature with the same authors.
- Some of the software and documents included within this software package are the intellectual property of various parties, and placement in this package does not in any way imply that any such rights have in any way been waived or diminished.
- With respect to any software or documents for which a copyright exists, **ALL RIGHTS ARE RESERVED TO THE OWNERS OF SUCH COPYRIGHT.**
- Even though the authors of the various documents and software found here have made a good faith effort to ensure that the documents are correct and that the software performs according to its documentation, and we would greatly appreciate hearing of any problems you may encounter, the programs and documents any files created by the programs are provided ****AS IS**** without any warranty as to correctness, merchantability or fitness for any particular or general use.
- **THE RESPONSIBILITY FOR ANY ADVERSE CONSEQUENCES FROM THE USE OF PROGRAMS OR DOCUMENTS OR ANY FILE OR FILES CREATED BY USE OF THE PROGRAMS OR DOCUMENTS LIES SOLELY WITH THE USERS OF THE PROGRAMS OR DOCUMENTS OR FILE OR FILES AND NOT WITH AUTHORS OF THE PROGRAMS OR DOCUMENTS.**

Stanford University Notices
for the CBFlib software package that incorporates SLAC software on which copyright is disclaimed

This software

The term 'this software', as used in these Notices, refers to those portions of the software package CBFlib that were created by employees of the Stanford Linear Accelerator Center, Stanford University.

Stanford disclaimer of copyright

Stanford University, owner of the copyright, hereby disclaims its copyright and all other rights in this software. Hence, anyone may freely use it for any purpose without restriction.

Acknowledgement of sponsorship

This software was produced by the Stanford Linear Accelerator Center, Stanford University, under Contract DE-AC03-76SFO0515 with the Department of Energy.

Government disclaimer of liability

Neither the United States nor the United States Department of Energy, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any data, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

Stanford disclaimer of liability

Stanford University makes no representations or warranties, express or implied, nor assumes any liability for the use of this software.

Maintenance of notices

In the interest of clarity regarding the origin and status of this software, this and all the preceding Stanford University notices are to remain affixed to any copy or derivative of this software made or distributed by the recipient and are to be affixed to any copy of software made or distributed by the recipient that contains a copy or derivative of this software.

Based on SLAC Software Notices, Set 4 OTT.002a, 2004 FEB 03

The IUCr Policy
for the Protection and the Promotion of the STAR File and
CIF Standards for Exchanging and Archiving Electronic Data

Overview

The Crystallographic Information File (CIF)[1] is a standard for information interchange promulgated by the International Union of Crystallography (IUCr). CIF (Hall, Allen & Brown, 1991) is the recommended method for submitting publications to Acta Crystallographica Section C and reports of crystal structure determinations to other sections of Acta Crystallographica and many other journals. The syntax of a CIF is a subset of the more general STAR File[2] format. The CIF and STAR File approaches are used increasingly in the structural sciences for data exchange and archiving, and are having a significant influence on these activities in other fields.

Statement of intent

The IUCr's interest in the STAR File is as a general data interchange standard for science, and its interest in the CIF, a conformant derivative of the STAR File, is as a concise data exchange and archival standard for crystallography and structural science.

Protection of the standards

To protect the STAR File and the CIF as standards for interchanging and archiving electronic data, the IUCr, on behalf of the scientific community,

* holds the copyrights on the standards themselves,

* owns the associated trademarks and service marks, and

* holds a patent on the STAR File.

These intellectual property rights relate solely to the interchange formats, not to the data contained therein, nor to the software used in the generation, access or manipulation of the data.

Promotion of the standards

The sole requirement that the IUCr, in its protective role, imposes on software purporting to process STAR File or CIF data is that the following conditions be met prior to sale or distribution.

* Software claiming to read files written to either the STAR File or the CIF standard must be able to extract the pertinent data from a file conformant to the STAR File syntax, or the CIF syntax, respectively.

* Software claiming to write files in either the STAR File, or the CIF, standard must produce files that are conformant to the STAR File syntax, or the CIF syntax, respectively.

* Software claiming to read definitions from a specific data dictionary approved by the IUCr must be able to extract any pertinent definition which is conformant to the dictionary definition language (DDL)[3] associated with that dictionary.

The IUCr, through its Committee on CIF Standards, will assist any developer to verify that software meets these conformance conditions.

Glossary of terms

[1] CIF:

is a data file conformant to the file syntax defined at http://www.iucr.org/iucr-top/cif/spec/index.html

[2] STAR File:

is a data file conformant to the file syntax defined at http://www.iucr.org/iucr-top/cif/spec/star/index.html

[3] DDL:

is a language used in a data dictionary to define data items in terms of "attributes". Dictionaries currently approved by the IUCr, and the DDL versions used to construct these dictionaries, are listed at http://www.iucr.org/iucr-top/cif/spec/ddl/index.html

Last modified: 30 September 2000

CBFlib V0.1 Notice

The following Diclaimer Notice applies to CBFlib V0.1, from which this version is derived.

- The items furnished herewith were developed under the sponsorship of the U.S. Government. Neither the U.S., nor the U.S. D.O.E., nor the Leland Stanford Junior University, nor their employees, makes any warranty, express or implied, or assumes any liability or responsibility for accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use will not infringe privately-owned rights. Mention of any product, its manufacturer, or suppliers shall not, nor is it intended to, imply approval, disapproval, or fitness for any particular use. The U.S. and the University at all times retain the right to use and disseminate the furnished items for any purpose whatsoever.
- Notice 91 02 01

CIFPARSE notice

Portions of this software are loosely based on the CIFPARSE software package from the NDB at Rutgers university (see http://ndbserver.rutgers.edu/NDB/mmcif/software). CIFPARSE is part of the NDBQUERY application, a program component of the Nucleic Acid Database Project [H. M. Berman, W. K. Olson, D. L. Beveridge, J. K. Westbrook, A. Gelbin, T. Demeny, S. H. Shieh, A. R. Srinivasan, and B. Schneider. (1992). The Nucleic Acid Database: A Comprehensive Relational Database of Three-Dimensional Structures of Nucleic Acids. Biophys J., 63, 751-759.], whose cooperation is gratefully acknowledged, especially in the form of design concepts created by J. Westbrook.

Please be aware of the following notice in the CIFPARSE API:

- This software is provided WITHOUT WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR ANY OTHER WARRANTY, EXPRESS OR IMPLIED. RUTGERS MAKE NO REPRESENTATION OR WARRANTY THAT THE SOFTWARE WILL NOT INFRINGE ANY PATENT, COPYRIGHT OR OTHER PROPRIETARY RIGHT.

MPACK notice

Portions of this library are adapted from the "mpack/munpack version 1.5" routines, written by John G. Myers. Mpack and munpack are utilities for encoding and decoding (respectively) binary files in MIME (Multipurpose Internet Mail Extensions) format mail messages. The mpack software used is (C) Copyright 1993,1994 by Carnegie Mellon University, All Rights Reserved, and is subject to the following notice:

- Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Carnegie Mellon University not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Carnegie Mellon University makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.
- **CARNEGIE MELLON UNIVERSITY DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.**

MD5 Notice

The following notice applies to the message digest software in md5.h and md5.c which are optionally used by this library. To that extent, this library is a work "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm".

The software in md5.h and md5.c is Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved, and is subject to the following notice:

- License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function.
- License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work.
- RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind.
- These notices must be retained in any copies of any part of this documentation and/or software.

Version History

Version	Date	By	Description
0.1	Apr. 1998	PJE	This was the first CBFlib release. It supported binary CBF files using binary strings.
0.2	Aug. 1998	HJB	This release added ascii imgCIF support using MIME-encoded binary sections, added the option of MIME headers for the binary strings was well. MIME code adapted from mpack 1.5. Added hooks needed for DDL1-style names without categories.
0.3	Sep. 1998	PJE	This release cleaned up the changes made for version 0.2, allowing multi-threaded use of the code, and removing dependence on the mpack package.
0.4	Nov. 1998	HJB	This release merged much of the message digest code into the general file reading and writing to reduce the number of passes. More consistency checking between the MIME header and the binary header was introduced. The size in the MIME header was adjusted to agree with the version 0.2 documentation.
0.5	Dec. 1998	PJE	This release greatly increased the speed of processing by allowing for deferred digest evaluation.
0.6	Jan. 1999	HJB	This release removed the redundant information (binary id, size, compression id) from a binary header when there is a MIME header, removed the unused repeat argument, and made the memory allocation for buffering and tables with many rows sensitive to the current memory allocation already used.
0.6.1	Feb. 2001	HP (per HJB)	This release fixed a memory leak due to misallocation by size of cbf_handle instead of cbf_handle_struct
0.7	Mar. 2001	PJE	This release added high-level instructions based on the imgCIF dictionary version 1.1.
0.7.1	Mar. 2001	PJE	The high-level functions were revised to permit future expansion to files with multiple images.
0.7.2	Apr. 2001	HJB	This release adjusted cbf_cimple.c to conform to cif_img.dic version 1.1.3
0.7.2.1	May 2001	PJE	This release corrected an if nesting error in the prior mod to cbf_cimple.c.
0.7.3	Oct 2002	PJE	This release modified cbf_simple.c to reorder image data on read so that the indices are always increasing in memory (this behavior was undefined previously).
0.7.4	Jan 2004	HJB	This release fixes a parse error for quoted strings, adds code to get and set character string types, and removes compiler warnings
0.7.5	Apr 2006	HJB	This release cleans up some compiler warnings, corrects a parse error on quoted strings with a leading blank as adds the new routines for support of aliases, dictionaries and real arrays, higher level routines to get and set pixel sizes, do cell computations, and to set beam centers, improves support for conversion of images, picking up more data from headers.
0.7.6	Jul 2006	HJB	This release reorganizes the kit into two pieces: CBFlib_0.7.6_Data_Files and CBFlib_0.7.6. An optional local copy of getopt is added. The 1.4 draft dictionary has been added. cif2cbf updated to support vcif2 validation. convert_image and cif2cbf updated to report text of error messages. convert_image updated to support tag and category aliases, default to adxv images. convert_image and img updated to support row-major images. Support added for binning. API Support added for validation, wide files and line folding. Logic changed for beam center reporting. Added new routines: cbf_validate, cbf_get_bin_sizes, cbf_set_bin_sizes, cbf_find_last_typed_child, cbf_compose_itemname, cbf_set_cbf_logfile, cbf_make_widefile, cbf_read_anyfile, cbf_read_widefile, cbf_write_local_file, cbf_write_widefile, cbf_column_number, cbf_blockitem_number, cbf_log, cbf_check_category_tags, cbf_set_beam_center

Known Problems

This version does not have support for byte-offset or predictor compression. Code is needed to support array sub-sections.

Foreword

In order to work with CBFlib, you need:

the source code, in the form of a "gzipped" tar, CBFlib_0.7.6.tar.gz; and
the test data, in the form of a "gzipped" tar CBFlib_0.7.6_Data_Files.tar.gz

Uncompress both of these files, and unpack them with tar:

gunzip < CBFlib_0.7.6.tar.gz | tar xvf -
gunzip < CBFlib_0.7.6_Data_Files.tar.gz | tar xvf –

The data files are compressed with bzip2. **Do not "bunzip2" the files in Place them in an otherwise empty directory, and unpack it with tar.** As in the past you will also need Paul Ellis's sample MAR345 image, example.mar2300 and Chris Nielsen's sample ADSC Quantum 315 image, mb_LP_1_001.img as sample data. The Makefile will extract decompress these files from the CBFlib_0.7.6_Data_Files directory.

Adjust the definition of CC and C++ in Makefile to point to your C compiler, and then

make all
make tests

We have included examples of CBF/imgCIF files produced by CBFlib, the current best draft of the CBF Extensions Dictionary, and of Andy Hammersley's CBF definition, updated to become a DRAFT CBF/ImgCIF DEFINITION.

Contents

4. Installation
5. Example programs

1. Introduction

CBFlib is a library of ANSI-C functions providing a simple mechanism for accessing Crystallographic Binary Files (CBF files) and Image-supporting CIF (imgCIF) files. The CBFlib API is loosely based on the CIFPARSE API for mmCIF files. Like CIFPARSE, If a dictionary is provided, CBFlib checks data values for type and against limits and enumerations. It provides functions to create, read, modify and write CBF binary data files and imgCIF ASCII data files.

2. Function descriptions

2.1 General description

Almost all of the CBFlib functions receive a value of type cbf_handle (a CBF handle) as the first argument. Several of the high-level CBFlib functions dealing with geometry receive a value of type cbf_goniometer (a handle for a CBF goniometer object) or cbf_detector (a handle for a CBF detector object).

All functions return an integer equal to 0 for success or an error code for failure.

2.1.1 CBF handles

CBFlib permits a program to use multiple CBF objects simultaneously. To identify the CBF object on which a function will operate, CBFlib uses a value of type cbf_handle.

All functions in the library except cbf_make_handle expect a value of type cbf_handle as the first argument.

The function **cbf_make_handle** creates and initializes a new CBF handle.

The function **cbf_free_handle** destroys a handle and frees all memory associated with the corresponding CBF object.

2.1.2 CBF goniometer handles

To represent the goniometer used to orient a sample, CBFlib uses a value of type cbf_goniometer.

A goniometer object is created and initialized from a CBF object using the function **cbf_construct_goniometer**.

The function **cbf_free_goniometer** destroys a goniometer handle and frees all memory associated with the corresponding object.

2.1.3 CBF detector handles

To represent a detector surface mounted on a positioning system, CBFlib uses a value of type cbf_detector.

A goniometer object is created and initialized from a CBF object using the function **cbf_construct_detector**.

The function **cbf_free_detector** destroys a detector handle and frees all memory associated with the corresponding object.

2.1.4 Return values

All of the CBFlib functions return 0 on success and an error code on failure. The error codes are:

CBF_FORMAT	The file format is invalid
CBF_ALLOC	Memory allocation failed
CBF_ARGUMENT	Invalid function argument
CBF_ASCII	The value is ASCII (not binary)
CBF_BINARY	The value is binary (not ASCII)
CBF_BITCOUNT	The expected number of bits does not match the actual number written
CBF_ENDOFDATA	The end of the data was reached before the end of the array
CBF_FILECLOSE	File close error
CBF_FILEOPEN	File open error
CBF_FILEREAD	File read error
CBF_FILESEEK	File seek error
CBF_FILETELL	File tell error
CBF_FILEWRITE	File write error
CBF_IDENTICAL	A data block with the new name already exists
CBF_NOTFOUND	The data block, category, column or row does not exist
CBF_OVERFLOW	The number read cannot fit into the destination argument. The destination has been set to the nearest value.
CBF_UNDEFINED	The requested number is not defined (e.g. 0/0; new for version 0.7).
CBF_NOTIMPLEMENTED	The requested functionality is not yet implemented (New for version 0.7).

If more than one error has occurred, the error code is the logical OR of the individual error codes.

2.2 Reading and writing files containing binary sections

2.2.1 Reading binary sections

The current version of CBFlib only decompresses a binary section from disk when requested by the program.

When a file containing one or more binary sections is read, CBFlib saves the file pointer and the position of the binary section within the file and then jumps past the binary section. When the program attempts to access the binary data, CBFlib sets the file position back to the start of the binary section and then reads the data.

For this scheme to work:

1. The file must be a random-access file opened in binary mode (fopen (," rb")).
2. The program *must not* close the file. CBFlib will close the file using fclose ( ) when it is no longer needed.

At present, this also means that a program cant read a file and then write back to the same file. This restriction will be eliminated in a future version.

When reading an imgCIF vs a CBF, the difference is detected automatically.

2.2.2 Writing binary sections

When a program passes CBFlib a binary value, the data is compressed to a temporary file. If the CBF object is subsequently written to a file, the data is simply copied from the temporary file to the output file.

The output file can be of any type. If the program indicates to CBFlib that the file is a random-access and readable, CBFlib will conserve disk space by closing the temporary file and using the output file as the location at which the binary value is stored.

For this option to work:

1. The file must be a random-access file opened in binary update mode (fopen (, "w+b")).

2. The program *must not* close the file. CBFlib will close the file using fclose ( ) when it is no longer needed.

If this option is not used:

1. CBFlib will continue using the temporary file.
2. CBFlib *will not* close the file. This is the responsibility of the main program.

2.2.3 Summary of reading and writing files containing binary sections

1. Open disk files to read using the mode "rb".
2. If possible, open disk files to write using the mode "w+b" and tell CBFlib that it can use the file as a buffer.
3. Do *not* close any files read by CBFlib or written by CBFlib with buffering turned on.
4. Do *not* attempt to read from a file, then write to the same file.

2.3 Low-level function prototypes

2.3.1 cbf_make_handle

PROTOTYPE

#include "cbf.h"

int cbf_make_handle (cbf_handle **handle*);

DESCRIPTION

cbf_make_handle creates and initializes a new internal CBF object. All other CBFlib functions operating on this object receive the CBF handle as the first argument.

ARGUMENTS
handle Pointer to a CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.2 cbf_free_handle

2.3.2 cbf_free_handle

PROTOTYPE

#include "cbf.h"

int cbf_free_handle (cbf_handle *handle*);

DESCRIPTION

cbf_free_handle destroys the CBF object specified by the *handle* and frees all associated memory.

ARGUMENTS
handle CBF handle to free.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.1 cbf_make_handle

2.3.3 cbf_read_file

PROTOTYPE

#include "cbf.h"

int cbf_read_file (cbf_handle *handle*, FILE **file*, int *headers*);
int cbf_read_widefile (cbf_handle *handle*, FILE **file*, int *headers*);

DESCRIPTION

cbf_read_file reads the CBF or CIF file *file* into the CBF object specified by *handle*, using the CIF 1.0 convention of 80 character lines. cbf_read_widefile reads the CBF or CIF file *file* into the CBF object specified by *handle*, using the CIF 1.1 convention of 2048 character lines. A warning is issued to stderr for ascii lines over the limit. No test is performed on binary sections.

Validation is performed in three ways levels: during the lexical scan, during the parse, and, if a dictionary was converted, against the value types, value enumerations, categories and parent-child relationships specified in the dictionary.

headers controls the interprestation of binary section headers of imgCIF files. MSG_DIGEST: Instructs CBFlib to check that the digest of the binary section matches any header value. If the digests do not match, the call will return CBF_FORMAT. This evaluation and comparison is delayed (a "lazy" evaluation) to ensure maximal processing efficiency. If an immediately evaluation is required, see MSG_DIGESTNOW, below. MSG_DIGESTNOW: Instructs CBFlib to check that the digest of the binary section matches any header value. If the digests do not match, the call will return CBF_FORMAT. This evaluation and comparison is performed during initial parsing of the section to ensure timely error reporting at the expense of processing efficiency. If a more efficient delayed ("lazy") evaluation is required, see MSG_DIGESTNOW, below. MSG_NODIGEST: Do not check the digest (default).

CBFlib defers reading binary sections as long as possible. In the current version of CBFlib, this means that:

1. The file must be a random-access file opened in binary mode (fopen (, "rb")).
2. The program *must not* close the file. CBFlib will close the file using fclose ( ) when it is no longer needed.

These restrictions may change in a future release.

ARGUMENTS

handle CBF handle.
file Pointer to a file descriptor.
headers Controls interprestation of binary section headers.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.4 cbf_write_file

2.3.4 cbf_write_file

PROTOTYPE

#include "cbf.h"

int cbf_write_file (cbf_handle *handle*, FILE **file*, int *readable*, int *ciforcbf*, int *headers*, int *encoding*);
int cbf_write_widefile (cbf_handle *handle*, FILE **file*, int *readable*, int *ciforcbf*, int *headers*, int *encoding*);

DESCRIPTION

cbf_write_file writes the CBF object specified by *handle* into the file *file*, following CIF 1.0 conventions of 80 character lines. cbf_write_widefile writes the CBF object specified by *handle* into the file *file*, following CIF 1.1 conventions of 2048 character lines. A warning is issued to stderr for ascii lines over the limit, and an attempt is made to fold lines to fit. No test is performed on binary sections.

If a dictionary has been provided, aliases will be applied on output.

Unlike cbf_read_file, the *file* does not have to be random-access.

If the file is random-access and readable, *readable* can be set to non-0 to indicate to CBFlib that the file can be used as a buffer to conserve disk space. If the file is not random-access or not readable, *readable* must be 0.

If *readable* is non-0, CBFlib will close the file when it is no longer required, otherwise this is the responsibility of the program.

ciforcbf selects the format in which the binary sections are written: CIF Write an imgCIF file. CBF Write a CBF file (default). *headers* selects the type of header used in CBF binary sections and selects whether message digests are generated. The value of *headers* can be a logical OR of any of:

MIME_HEADERS Use MIME-type headers (default).
MIME_NOHEADERS Use a simple ASCII headers.
MSG_DIGEST Generate message digests for binary data validation.
MSG_NODIGEST Do not generate message digests (default). *encoding* selects the type of encoding used for binary sections and the type of line-termination in imgCIF files. The value can be a logical OR of any of:

ENC_BASE64 Use BASE64 encoding (default).
ENC_QP Use QUOTED-PRINTABLE encoding.
ENC_BASE8 Use BASE8 (octal) encoding.
ENC_BASE10 Use BASE10 (decimal) encoding.
ENC_BASE16 Use BASE16 (hexadecimal) encoding.
ENC_FORWARD For BASE8, BASE10 or BASE16 encoding, map bytes to words forward (1234) (default on little-endian machines).
ENC_BACKWARD Map bytes to words backward (4321) (default on big-endian machines).
ENC_CRTERM Terminate lines with CR.
ENC_LFTERM Terminate lines with LF (default).

ARGUMENTS

handle CBF handle.
file Pointer to a file descriptor.

readable If non-0: this file is random-access and readable and can be used as a buffer.
ciforcbf Selects the format in which the binary sections are written (CIF/CBF).
headers Selects the type of header in CBF binary sections and message digest generation.
encoding Selects the type of encoding used for binary sections and the type of line-termination in imgCIF files.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.3 cbf_read_file

2.3.5 cbf_new_datablock, cbf_new_saveframe

PROTOTYPE

#include "cbf.h"

int cbf_new_datablock (cbf_handle *handle*, const char **datablockname*);
int cbf_new_saveframe (cbf_handle *handle*, const char **saveframename*);

DESCRIPTION

cbf_new_datablock creates a new data block with name *datablockname* and makes it the current data block.
cbf_new_saveframe creates a new save frame with name *saveframename* within the current data block and makes the new save frame the current save frame.

If a data block or save frame with this name already exists, the existing data block or save frame becomes the current data block or save frame.

ARGUMENTS
handle CBF handle.
datablockname The name of the new data block.
saveframename The name of the new save frame.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.6 cbf_force_new_datablock, cbf_force_new_saveframe
2.3.7 cbf_new_category
2.3.8 cbf_force_new_category
2.3.9 cbf_new_column
2.3.10 cbf_new_row
2.3.11 cbf_insert_row
2.3.12 cbf_set_datablockname, cbf_set_saveframename
2.3.17 cbf_remove_datablock, cbf_remove_saveframe
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.6 cbf_force_new_datablock, cbf_force_new_saveframe

PROTOTYPE

#include "cbf.h"

int cbf_force_new_datablock (cbf_handle *handle*, const char **datablockname*);
int cbf_force_new_saveframe (cbf_handle *handle*, const char **saveframename*);

DESCRIPTION

cbf_force_new_datablock creates a new data block with name *datablockname* and makes it the current data block. Duplicate data block names are allowed. cbf_force_new_saveframe creates a new savew frame with name *saveframename* and makes it the current save frame. Duplicate save frame names are allowed.

Even if a save frame with this name already exists, a new save frame is created and becomes the current save frame.

ARGUMENTS

handle CBF handle.
datablockname The name of the new data block.
saveframename The name of the new save frame.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.5 cbf_new_datablock, cbf_new_saveframe
2.3.7 cbf_new_category
2.3.8 cbf_force_new_category
2.3.9 cbf_new_column
2.3.10 cbf_new_row
2.3.11 cbf_insert_row
2.3.12 cbf_set_datablockname, cbf_set_saveframename
2.3.17 cbf_remove_datablock, cbf_remove_saveframe
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.7 cbf_new_category

PROTOTYPE

#include "cbf.h"

int cbf_new_category (cbf_handle *handle*, const char **categoryname*);

DESCRIPTION

cbf_new_category creates a new category in the current data block with name *categoryname* and makes it the current category.

If a category with this name already exists, the existing category becomes the current category.

ARGUMENTS

handle CBF handle.
categoryname The name of the new category.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.5 cbf_new_datablock, cbf_new_saveframe
2.3.6 cbf_force_new_datablock, cbf_force_new_saveframe
2.3.8 cbf_force_new_category
2.3.9 cbf_new_column
2.3.10 cbf_new_row
2.3.11 cbf_insert_row
2.3.18 cbf_remove_category
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.8 cbf_force_new_category

PROTOTYPE

#include "cbf.h"

int cbf_force_new_category (cbf_handle *handle*, const char **categoryname*);

DESCRIPTION

cbf_force_new_category creates a new category in the current data block with name *categoryname* and makes it the current category. Duplicate category names are allowed.

Even if a category with this name already exists, a new category of the same name is created and becomes the current category. The allows for the creation of unlooped tag/value lists drawn from the same category.

ARGUMENTS

handle CBF handle.
categoryname The name of the new category.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.5 cbf_new_datablock, cbf_new_saveframe
2.3.6 cbf_force_new_datablock, cbf_force_new_saveframe
2.3.7 cbf_new_category
2.3.9 cbf_new_column
2.3.10 cbf_new_row

2.3.11 cbf_insert_row
2.3.18 cbf_remove_category
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.9 cbf_new_column

PROTOTYPE

#include "cbf.h"

int cbf_new_column (cbf_handle *handle*, const char **columnname*);

DESCRIPTION

cbf_new_column creates a new column in the current category with name *columnname* and makes it the current column.

If a column with this name already exists, the existing column becomes the current category.

ARGUMENTS
handle CBF handle.
columnname The name of the new column.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.5 cbf_new_datablock, cbf_new_saveframe
2.3.6 cbf_force_new_datablock, cbf_force_new_saveframe
2.3.7 cbf_new_category
2.3.8 cbf_force_new_category
2.3.10 cbf_new_row
2.3.11 cbf_insert_row
2.3.19 cbf_remove_column
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.10 cbf_new_row

PROTOTYPE

#include "cbf.h"

int cbf_new_row (cbf_handle *handle*);

DESCRIPTION

cbf_new_row adds a new row to the current category and makes it the current row.

ARGUMENTS

handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.5 cbf_new_datablock, cbf_new_saveframe
2.3.6 cbf_force_new_datablock, cbf_force_new_saveframe
2.3.7 cbf_new_category
2.3.8 cbf_force_new_category
2.3.9 cbf_new_column
2.3.11 cbf_insert_row
2.3.12 cbf_delete_row
2.3.20 cbf_remove_row
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.11 cbf_insert_row

PROTOTYPE

#include "cbf.h"

int cbf_insert_row (cbf_handle *handle*, unsigned int *rownumber*);

DESCRIPTION

cbf_insert_row adds a new row to the current category. The new row is inserted as row *rownumber* and existing rows starting from *rownumber* are moved up by 1. The new row becomes the current row.

If the category has fewer than *rownumber* rows, the function returns CBF_NOTFOUND.

The row numbers start from 0.

ARGUMENTS

handle CBF handle.
rownumber The row number of the new row.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.5 cbf_new_datablock, cbf_new_saveframe
2.3.6 cbf_force_new_datablock, cbf_force_new_saveframe

2.3.7 cbf_new_category
2.3.8 cbf_force_new_category
2.3.9 cbf_new_column
2.3.10 cbf_new_row
2.3.12 cbf_delete_row
2.3.20 cbf_remove_row
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.12 cbf_delete_row

PROTOTYPE

#include "cbf.h"

int cbf_delete_row (cbf_handle *handle*, unsigned int *rownumber*);

DESCRIPTION

cbf_delete_row deletes a row from the current category. Rows starting from *rownumber* +1 are moved down by 1. If the current row was higher than *rownumber*, or if the current row is the last row, it will also move down by 1.

The row numbers start from 0.

ARGUMENTS
handle CBF handle.
rownumber The number of the row to delete.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.10 cbf_new_row
2.3.11 cbf_insert_row
2.3.17 cbf_remove_datablock, cbf_remove_saveframe
2.3.18 cbf_remove_category
2.3.19 cbf_remove_column
2.3.20 cbf_remove_row
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.13 cbf_set_datablockname, cbf_set_saveframename

PROTOTYPE

#include "cbf.h"

int cbf_set_datablockname (cbf_handle *handle*, const char **datablockname*);

int cbf_set_saveframename (cbf_handle *handle*. const char **saveframename*);

DESCRIPTION

cbf_set_datablockname changes the name of the current data block to *datablockname*. cbf_set_saveframename changes the name of the current save frame to *saveframename*.

If a data block or save frame with this name already exists (comparison is case-insensitive), the function returns CBF_IDENTICAL.

ARGUMENTS
handle CBF handle.
datablockname The new data block name.
datablockname The new save frame name.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.5 cbf_new_datablock, cbf_new_saveframe
2.3.14 cbf_reset_datablocks
2.3.15 cbf_reset_datablock, cbf_reset_saveframe
2.3.17 cbf_remove_datablock, cbf_remove_saveframe
2.3.42 cbf_datablock_name

2.3.14 cbf_reset_datablocks

PROTOTYPE

#include "cbf.h"

int cbf_reset_datablocks (cbf_handle *handle*);

DESCRIPTION

cbf_reset_datablocks deletes all categories from all data blocks.

The current data block does not change.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.15 cbf_reset_datablock, cbf_reset_saveframe

2.3.18 cbf_remove_category

2.3.15 cbf_reset_datablock, cbf_reset_datablock

PROTOTYPE

#include "cbf.h"

int cbf_reset_datablock (cbf_handle *handle*);
int cbf_reset_saveframe (cbf_handle *handle*);

DESCRIPTION

cbf_reset_datablock deletes all categories from the current data block. cbf_reset_saveframe deletes all categories from the current save frame.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.14 cbf_reset_datablocks
2.3.18 cbf_remove_category

2.3.16 cbf_reset_category

PROTOTYPE

#include "cbf.h"

int cbf_reset_category (cbf_handle *handle*);

DESCRIPTION

cbf_reset_category deletes all columns and rows from current category.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.16 cbf_reset_category
2.3.19 cbf_remove_column

2.3.20 cbf_remove_row

2.3.17 cbf_remove_datablock, cbf_remove_saveframe

PROTOTYPE

#include "cbf.h"

int cbf_remove_datablock (cbf_handle *handle*);
int cbf_remove_saveframe (cbf_handle *handle*);

DESCRIPTION

cbf_remove_datablock deletes the current data block. cbf_remove_saveframe deletes the current save frame.

The current data block becomes undefined.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.5 cbf_new_datablock, cbf_new_saveframe
2.3.6 cbf_force_new_datablock, cbf_force_new_saveframe
2.3.18 cbf_remove_category
2.3.19 cbf_remove_column
2.3.20 cbf_remove_row
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.18 cbf_remove_category

PROTOTYPE

#include "cbf.h"

int cbf_remove_category (cbf_handle *handle*);

DESCRIPTION

cbf_remove_category deletes the current category.

The current category becomes undefined.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.7 cbf_new_category
2.3.8 cbf_force_new_category
2.3.17 cbf_remove_datablock, cbf_remove_saveframe
2.3.19 cbf_remove_column
2.3.20 cbf_remove_row
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.19 cbf_remove_column

PROTOTYPE

#include "cbf.h"

int cbf_remove_column (cbf_handle *handle*);

DESCRIPTION

cbf_remove_column deletes the current column.

The current column becomes undefined.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.9 cbf_new_column
2.3.17 cbf_remove_datablock, cbf_remove_saveframe
2.3.18 cbf_remove_category
2.3.20 cbf_remove_row
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.20 cbf_remove_row

PROTOTYPE

#include "cbf.h"

int cbf_remove_row (cbf_handle *handle*);

DESCRIPTION

cbf_remove_row deletes the current row in the current category.

If the current row was the last row, it will move down by 1, otherwise, it will remain the same.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.10 cbf_new_row
2.3.11 cbf_insert_row
2.3.17 cbf_remove_datablock, cbf_remove_saveframe
2.3.18 cbf_remove_category
2.3.19 cbf_remove_column
2.3.12 cbf_delete_row
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.21 cbf_rewind_datablock

PROTOTYPE

#include "cbf.h"

int cbf_rewind_datablock (cbf_handle *handle*);

DESCRIPTION

cbf_rewind_datablock makes the first data block the current data block.

If there are no data blocks, the function returns CBF_NOTFOUND.

The current category becomes undefined.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.22 cbf_rewind_category, cbf_rewind_saveframe, cbf_rewind_blockitem
2.3.19 cbf_rewind_column

2.3.24 cbf_rewind_row
2.3.25 cbf_next_datablock

2.3.22 cbf_rewind_category, cbf_rewind_saveframe, cbf_rewind_blockitem

PROTOTYPE

#include "cbf.h"

int cbf_rewind_category (cbf_handle *handle*);
int cbf_rewind_saveframe (cbf_handle *handle*);
int cbf_rewind_blockitem (cbf_handle *handle*);

DESCRIPTION

cbf_rewind_category makes the first category in the current data block the current category. cbf_rewind_saveframe makes the first saveframe in the current data block the current saveframe. cbf_rewind_blockitem makes the first blockitem (category or saveframe) in the current data block the current blockitem.

If there are no categories, saveframes or blockitems the function returns CBF_NOTFOUND.

The current column and row become undefined.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.21 cbf_rewind_datablock
2.3.19 cbf_rewind_column
2.3.24 cbf_rewind_row
2.3.26 cbf_next_category, cbf_next_saveframe, cbf_next_blockitem

2.3.23 cbf_rewind_column

PROTOTYPE

#include "cbf.h"

int cbf_rewind_column (cbf_handle *handle*);

DESCRIPTION

cbf_rewind_column makes the first column in the current category the current column.

If there are no columns, the function returns CBF_NOTFOUND.

The current row is not affected.

ARGUMENTS

handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.21 cbf_rewind_datablock
2.3.22 cbf_rewind_category, cbf_rewind_saveframe, cbf_rewind_blockitem
2.3.24 cbf_rewind_row
2.3.27 cbf_next_column

2.3.24 cbf_rewind_row

PROTOTYPE

#include "cbf.h"

int cbf_rewind_row (cbf_handle *handle*);

DESCRIPTION

cbf_rewind_row makes the first row in the current category the current row.

If there are no rows, the function returns CBF_NOTFOUND.

The current column is not affected.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.21 cbf_rewind_datablock
2.3.22 cbf_rewind_category, cbf_rewind_saveframe, cbf_rewind_blockitem
2.3.19 cbf_rewind_column
2.3.28 cbf_next_row

2.3.25 cbf_next_datablock

PROTOTYPE

#include "cbf.h"

int cbf_next_datablock (cbf_handle *handle*);

DESCRIPTION

cbf_next_datablock makes the data block following the current data block the current data block.

If there are no more data blocks, the function returns CBF_NOTFOUND.

The current category becomes undefined.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.21 cbf_rewind_datablock
2.3.26 cbf_next_category, cbf_next_saveframe, cbf_next_blockitem
2.3.27 cbf_next_column
2.3.28 cbf_next_row

2.3.26 cbf_next_category

PROTOTYPE

#include "cbf.h"

int cbf_next_category (cbf_handle *handle*);

DESCRIPTION

cbf_next_category makes the category following the current category in the current data block the current category.

If there are no more categories, the function returns CBF_NOTFOUND.

The current column and row become undefined.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.22 cbf_rewind_category, cbf_rewind_saveframe, cbf_rewind_blockitem
2.3.25 cbf_next_datablock
2.3.27 cbf_next_column
2.3.27 cbf_next_row

2.3.27 cbf_next_column

PROTOTYPE

#include "cbf.h"

int cbf_next_column (cbf_handle *handle*);

DESCRIPTION

cbf_next_column makes the column following the current column in the current category the current column.

If there are no more columns, the function returns CBF_NOTFOUND.

The current row is not affected.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.19 cbf_rewind_column
2.3.25 cbf_next_datablock
2.3.26 cbf_next_category, cbf_next_saveframe, cbf_next_blockitem
2.3.28 cbf_next_row

2.3.28 cbf_next_row

PROTOTYPE

#include "cbf.h"

int cbf_next_row (cbf_handle *handle*);

DESCRIPTION

cbf_next_row makes the row following the current row in the current category the current row.

If there are no more rows, the function returns CBF_NOTFOUND.

The current column is not affected.

ARGUMENTS
handle CBF handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.24 cbf_rewind_row
2.3.25 cbf_next_datablock
2.3.26 cbf_next_category, cbf_next_saveframe, cbf_next_blockitem
2.3.27 cbf_next_column

2.3.29 cbf_find_datablock

PROTOTYPE

#include "cbf.h"

int cbf_find_datablock (cbf_handle *handle*, const char **datablockname*);

DESCRIPTION

cbf_find_datablock makes the data block with name *datablockname* the current data block.

The comparison is case-insensitive.

If the data block does not exist, the function returns CBF_NOTFOUND.

The current category becomes undefined.

ARGUMENTS
handle CBF handle.
datablockname The name of the data block to find.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.21 cbf_rewind_datablock
2.3.25 cbf_next_datablock
2.3.30 cbf_find_category, cbf_find_saveframe, cbf_find_blockitem
2.3.31 cbf_find_column
2.3.32 cbf_find_row
2.3.42 cbf_datablock_name
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.30 cbf_find_category

PROTOTYPE

#include "cbf.h"

int cbf_find_category (cbf_handle *handle*, const char **categoryname*);

DESCRIPTION

cbf_find_category makes the category in the current data block with name *categoryname* the current category.

The comparison is case-insensitive.

If the category does not exist, the function returns CBF_NOTFOUND.

The current column and row become undefined.

ARGUMENTS

handle CBF handle.
categoryname The name of the category to find.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.22 cbf_rewind_category, cbf_rewind_saveframe, cbf_rewind_blockitem
2.3.26 cbf_next_category, cbf_next_saveframe, cbf_next_blockitem
2.3.29 cbf_find_datablock
2.3.31 cbf_find_column
2.3.32 cbf_find_row
2.3.43 cbf_category_name
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.31 cbf_find_column

PROTOTYPE

#include "cbf.h"

int cbf_find_column (cbf_handle *handle*, const char **columnname*);

DESCRIPTION

cbf_find_column makes the columns in the current category with name *columnname* the current column.

The comparison is case-insensitive.

If the column does not exist, the function returns CBF_NOTFOUND.

The current row is not affected.

ARGUMENTS

handle CBF handle.
columnname The name of column to find.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.19 cbf_rewind_column
2.3.27 cbf_next_column
2.3.29 cbf_find_datablock
2.3.30 cbf_find_category, cbf_find_saveframe, cbf_find_blockitem
2.3.32 cbf_find_row
2.3.44 cbf_column_name
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.32 cbf_find_row

PROTOTYPE

#include "cbf.h"

int cbf_find_row (cbf_handle *handle*, const char **value*);

DESCRIPTION

cbf_find_row makes the first row in the current column with value *value* the current row.

The comparison is case-sensitive.

If a matching row does not exist, the function returns CBF_NOTFOUND.

The current column is not affected.

ARGUMENTS
handle CBF handle.
value The value of the row to find.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.24 cbf_rewind_row
2.3.28 cbf_next_row
2.3.29 cbf_find_datablock
2.3.30 cbf_find_category, cbf_find_saveframe, cbf_find_blockitem
2.3.31 cbf_find_column
2.3.33 cbf_find_nextrow
2.3.46 cbf_get_value, cbf_require_value
2.3.48 cbf_get_typeofvalue

2.3.33 cbf_find_nextrow

PROTOTYPE

#include "cbf.h"

int cbf_find_nextrow (cbf_handle *handle*, const char **value*);

DESCRIPTION

cbf_find_nextrow makes the makes the next row in the current column with value *value* the current row. The search starts from the row following the last row found with cbf_find_row or cbf_find_nextrow, or from the current row if the current row was defined using any other function.

The comparison is case-sensitive.

If no more matching rows exist, the function returns CBF_NOTFOUND.

The current column is not affected.

ARGUMENTS
handle CBF handle.
value the value to search for.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.24 cbf_rewind_row
2.3.28 cbf_next_row
2.3.29 cbf_find_datablock
2.3.30 cbf_find_category, cbf_find_saveframe, cbf_find_blockitem
2.3.31 cbf_find_column
2.3.32 cbf_find_row
2.3.46 cbf_get_value, cbf_require_value
2.3.48 cbf_get_typeofvalue

2.3.34 cbf_count_datablocks

PROTOTYPE

#include "cbf.h"

int cbf_count_datablocks (cbf_handle *handle*, unsigned int **datablocks*);

DESCRIPTION

cbf_count_datablocks puts the number of data blocks in **datablocks* .

ARGUMENTS
handle CBF handle.
datablocks Pointer to the destination data block count.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.35 cbf_count_categories, cbf_count_saveframes, cbf_count_blockitems
2.3.36 cbf_count_columns
2.3.37 cbf_count_rows
2.3.38 cbf_select_datablock

2.3.35 cbf_count_categories

PROTOTYPE

#include "cbf.h"

int cbf_count_categories (cbf_handle *handle*, unsigned int **categories*);

DESCRIPTION

cbf_count_categories puts the number of categories in the current data block in **categories*.

ARGUMENTS
handle CBF handle.
categories Pointer to the destination category count.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.34 cbf_count_datablocks
2.3.36 cbf_count_columns
2.3.37 cbf_count_rows
2.3.39 cbf_select_category, cbf_select_saveframe, cbf_select_blockitem

2.3.36 cbf_count_columns

PROTOTYPE

#include "cbf.h"

int cbf_count_columns (cbf_handle *handle*, unsigned int **columns*);

DESCRIPTION

cbf_count_columns puts the number of columns in the current category in **columns*.

ARGUMENTS
handle CBF handle.
columns Pointer to the destination column count.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.34 cbf_count_datablocks
2.3.35 cbf_count_categories, cbf_count_saveframes, cbf_count_blockitems
2.3.37 cbf_count_rows
2.3.40 cbf_select_column

2.3.37 cbf_count_rows

PROTOTYPE

#include "cbf.h"

int cbf_count_rows (cbf_handle *handle*, unsigned int **rows*);

DESCRIPTION

cbf_count_rows puts the number of rows in the current category in **rows* .

ARGUMENTS
handle CBF handle.
rows Pointer to the destination row count.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.34 cbf_count_datablocks
2.3.35 cbf_count_categories, cbf_count_saveframes, cbf_count_blockitems
2.3.36 cbf_count_columns
2.3.41 cbf_select_row

2.3.38 cbf_select_datablock

PROTOTYPE

#include "cbf.h"

int cbf_select_datablock (cbf_handle *handle*, unsigned int *datablock*);

DESCRIPTION

cbf_select_datablock selects data block number *datablock* as the current data block.

The first data block is number 0.

If the data block does not exist, the function returns CBF_NOTFOUND.

ARGUMENTS

handle CBF handle.
datablock Number of the data block to select.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.34 cbf_count_datablocks
2.3.39 cbf_select_category, cbf_select_saveframe, cbf_select_blockitem
2.3.40 cbf_select_column
2.3.41 cbf_select_row

2.3.39 cbf_select_category

PROTOTYPE

#include "cbf.h"

int cbf_select_category (cbf_handle *handle*, unsigned int *category*);

DESCRIPTION

cbf_select_category selects category number *category* in the current data block as the current category.

The first category is number 0.

The current column and row become undefined.

If the category does not exist, the function returns CBF_NOTFOUND.

ARGUMENTS

handle CBF handle.
category Number of the category to select.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.35 cbf_count_categories, cbf_count_saveframes, cbf_count_blockitems
2.3.38 cbf_select_datablock
2.3.40 cbf_select_column
2.3.41 cbf_select_row

2.3.40 cbf_select_column

PROTOTYPE

#include "cbf.h"

int cbf_select_column (cbf_handle *handle*, unsigned int *column*);

DESCRIPTION

cbf_select_column selects column number *column* in the current category as the current column.

The first column is number 0.

The current row is not affected

If the column does not exist, the function returns CBF_NOTFOUND.

ARGUMENTS
handle CBF handle.
column Number of the column to select.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.36 cbf_count_columns
2.3.38 cbf_select_datablock
2.3.39 cbf_select_category, cbf_select_saveframe, cbf_select_blockitem
2.3.41 cbf_select_row

2.3.41 cbf_select_row

PROTOTYPE

#include "cbf.h"

int cbf_select_row (cbf_handle *handle*, unsigned int *row*);

DESCRIPTION

cbf_select_row selects row number *row* in the current category as the current row.

The first row is number 0.

The current column is not affected

If the row does not exist, the function returns CBF_NOTFOUND.

ARGUMENTS
handle CBF handle.
row Number of the row to select.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.37 cbf_count_rows
2.3.38 cbf_select_datablock
2.3.39 cbf_select_category, cbf_select_saveframe, cbf_select_blockitem
2.3.40 cbf_select_column

2.3.42 cbf_datablock_name

PROTOTYPE

#include "cbf.h"

int cbf_datablock_name (cbf_handle *handle*, const char ***datablockname*);

DESCRIPTION

cbf_datablock_name sets **datablockname* to point to the name of the current data block.

The data block name will be valid as long as the data block exists and has not been renamed.

The name must not be modified by the program in any way.

ARGUMENTS
handle CBF handle.
datablockname Pointer to the destination data block name pointer.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.29 cbf_find_datablock

2.3.43 cbf_category_name

PROTOTYPE

#include "cbf.h"

int cbf_category_name (cbf_handle *handle*, const char ***categoryname*);

DESCRIPTION

cbf_category_name sets **categoryname* to point to the name of the current category of the current data block.

The category name will be valid as long as the category exists.

The name must not be modified by the program in any way.

ARGUMENTS

handle CBF handle.
categoryname Pointer to the destination category name pointer.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.30 cbf_find_category, cbf_find_saveframe, cbf_find_blockitem

2.3.44 cbf_column_name

PROTOTYPE

#include "cbf.h"

int cbf_column_name (cbf_handle *handle*, const char ***columnname*);

DESCRIPTION

cbf_column_name sets **columnname* to point to the name of the current column of the current category.

The column name will be valid as long as the column exists.

The name must not be modified by the program in any way.

ARGUMENTS

handle CBF handle.
columnname Pointer to the destination column name pointer.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.31 cbf_find_column

2.3.45 cbf_row_number

PROTOTYPE

#include "cbf.h"

int cbf_row_number (cbf_handle *handle*, unsigned int **row*);

DESCRIPTION

cbf_row_number sets **row* to the number of the current row of the current category.

ARGUMENTS

handle CBF handle.
row Pointer to the destination row number.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.41 cbf_select_row

2.3.46 cbf_get_value, cbf_require_value

PROTOTYPE

#include "cbf.h"

int cbf_get_value (cbf_handle *handle*, const char ***value*);
int cbf_require_value (cbf_handle *handle*, const char ***value*, const char **defaultvalue*);

DESCRIPTION

cbf_get_value sets **value* to point to the ASCII value of the item at the current column and row. cbf_set_value sets **value* to point to the ASCII value of the item at the current column and row, creating the data item if necessary and initializing it to a copy of *defaultvalue*.

If the value is not ASCII, the function returns CBF_BINARY.

The value will be valid as long as the item exists and has not been set to a new value.

The value must not be modified by the program in any way.

ARGUMENTS
handle CBF handle.
value Pointer to the destination value pointer.
defaultvalue Default value character string.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.47 cbf_set_value
2.3.48 cbf_get_typeofvalue
2.3.49 cbf_set_typeofvalue
2.3.50 cbf_get_integervalue, cbf_require_integervalue
2.3.52 cbf_get_doublevalue, cbf_require_doublevalue
2.3.54 cbf_get_integerarrayparameters, cbf_get_realarrayparameters
2.3.55 cbf_get_integerarray, cbf_get_realarray
2.3.62 cbf_require_column_value
2.3.63 cbf_require_column_integervalue
2.3.64 cbf_require_column_doublevalue

2.3.47 cbf_set_value

PROTOTYPE

#include "cbf.h"

int cbf_set_value (cbf_handle *handle*, const char **value*);

DESCRIPTION

cbf_set_value sets the item at the current column and row to the ASCII value *value*.

ARGUMENTS
handle CBF handle.
value ASCII value.
defaultvalue default ASCII value.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.46 cbf_get_value, cbf_require_value
2.3.48 cbf_get_typeofvalue
2.3.49 cbf_set_typeofvalue
2.3.51 cbf_set_integervalue
2.3.53 cbf_set_doublevalue
2.3.56 cbf_set_integerarray, cbf_set_realarray
2.3.62 cbf_require_column_value
2.3.63 cbf_require_column_integervalue
2.3.64 cbf_require_column_doublevalue

2.3.48 cbf_get_typeofvalue

PROTOTYPE

#include "cbf.h"

int cbf_get_typeofvalue (cbf_handle *handle*, const char ***typeofvalue*);

DESCRIPTION

cbf_get_value sets **typeofvalue* to point an ASCII descriptor of the value of the item at the current column and row. The strings that may be returned are "null" for a null value indicated by a "." or a "?", "bnry" for a binary value, "word" for an unquoted string, "dblq" for a double-quoted string, "sglq" for a single-quoted string, and "text" for a semicolon-quoted text field. A field for which no value has been set sets **typeofvalue* to NULL rather than to the string "null".

The *typeofvalue* must not be modified by the program in any way.

ARGUMENTS
handle CBF handle.
typeofvalue Pointer to the destination type-of-value string pointer.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.46 cbf_get_value, cbf_require_value
2.3.47 cbf_set_value
2.3.49 cbf_set_typeofvalue
2.3.50 cbf_get_integervalue, cbf_require_integervalue
2.3.52 cbf_get_doublevalue, cbf_require_doublevalue
2.3.54 cbf_get_integerarrayparameters, cbf_get_realarrayparameters
2.3.55 cbf_get_integerarray, cbf_get_realarray
2.3.62 cbf_require_column_value
2.3.63 cbf_require_column_integervalue
2.3.64 cbf_require_column_doublevalue

2.3.49 cbf_set_typeofvalue

PROTOTYPE

#include "cbf.h"

int cbf_set_typeofvalue (cbf_handle *handle*, const char **typeofvalue*);

DESCRIPTION

cbf_set_typeofvalue sets the type of the item at the current column and row to the type specified by the ASCII character string given by *typeofvalue*. The strings that may be used are "null" for a null value indicated by a "." or a "?", "word" for an unquoted string, "dblq" for a double-quoted string, "sglq" for a single-quoted string, and "text" for a semicolon-quoted text field. Not all types may be used for all values. No changes may be made to the type of binary values. You may not set the type of a string that contains a single quote followed by a blank or a tab or which contains multiple lines to "sglq". You may not set the type of a string that contains a double quote followed by a blank or a tab or which contains multiple lines to "dblq".

ARGUMENTS

handle CBF handle.
typeofvalue ASCII string for desired type of value.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.46 cbf_get_value, cbf_require_value
2.3.47 cbf_set_value
2.3.48 cbf_get_typeofvalue
2.3.51 cbf_set_integervalue
2.3.53 cbf_set_doublevalue
2.3.56 cbf_set_integerarray, cbf_set_realarray
2.3.62 cbf_require_column_value
2.3.63 cbf_require_column_integervalue
2.3.64 cbf_require_column_doublevalue

2.3.50 cbf_get_integervalue, cbf_require_integervalue

PROTOTYPE

#include "cbf.h"

int cbf_get_integervalue (cbf_handle *handle*, int **number*);
int cbf_require_integervalue (cbf_handle *handle*, int **number*, int *defaultvalue*);

DESCRIPTION

cbf_get_integervalue sets **number* to the value of the ASCII item at the current column and row interpreted as a decimal integer. cbf_require_integervalue sets **number* to the value of the ASCII item at the current column and row interpreted as a decimal integer, setting it to *defaultvalue* if necessary.

If the value is not ASCII, the function returns CBF_BINARY.

ARGUMENTS
handle CBF handle.
number pointer to the number.
defaultvalue default number value.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.46 cbf_get_value, cbf_require_value
2.3.48 cbf_get_typeofvalue
2.3.51 cbf_set_integervalue
2.3.52 cbf_get_doublevalue, cbf_require_doublevalue
2.3.54 cbf_get_integerarrayparameters, cbf_get_realarrayparameters
2.3.55 cbf_get_integerarray, cbf_get_realarray
2.3.62 cbf_require_column_value
2.3.63 cbf_require_column_integervalue
2.3.64 cbf_require_column_doublevalue

2.3.51 cbf_set_integervalue

PROTOTYPE

#include "cbf.h"

int cbf_set_integervalue (cbf_handle *handle*, int *number*);

DESCRIPTION

cbf_set_integervalue sets the item at the current column and row to the integer value *number* written as a decimal ASCII string.

ARGUMENTS
handle CBF handle.
number Integer value.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.46 cbf_get_value, cbf_require_value
2.3.47 cbf_set_value
2.3.48 cbf_get_typeofvalue
2.3.49 cbf_set_typeofvalue
2.3.50 cbf_get_integervalue, cbf_require_integervalue
2.3.51 cbf_set_integervalue
2.3.53 cbf_set_doublevalue
2.3.56 cbf_set_integerarray, cbf_set_realarray
2.3.62 cbf_require_column_value
2.3.63 cbf_require_column_integervalue
2.3.64 cbf_require_column_doublevalue

2.3.52 cbf_get_doublevalue, cbf_require_doublevalue

PROTOTYPE

#include "cbf.h"

int cbf_get_doublevalue (cbf_handle *handle*, double **number*);
int cbf_require_doublevalue (cbf_handle *handle*, double **number*, double *defaultvalue*);

DESCRIPTION

cbf_get_doublevalue sets **number* to the value of the ASCII item at the current column and row interpreted as a decimal floating-point number. cbf_require_doublevalue sets **number* to the value of the ASCII item at the current column and row interpreted as a decimal floating-point number, setting it to *defaultvalue* if necessary.

If the value is not ASCII, the function returns CBF_BINARY.

ARGUMENTS

handle CBF handle.
number Pointer to the destination number.
defaultvalue default number value.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.46 cbf_get_value, cbf_require_value
2.3.48 cbf_get_typeofvalue
2.3.49 cbf_set_typeofvalue
2.3.50 cbf_get_integervalue, cbf_require_integervalue
2.3.53 cbf_set_doublevalue
2.3.54 cbf_get_integerarrayparameters, cbf_get_realarrayparameters
2.3.55 cbf_get_integerarray, cbf_get_realarray
2.3.62 cbf_require_column_value

2.3.63 cbf_require_column_integervalue
2.3.64 cbf_require_column_doublevalue

2.3.53 cbf_set_doublevalue

PROTOTYPE

#include "cbf.h"

int cbf_set_doublevalue (cbf_handle *handle*, const char **format*, double *number*);

DESCRIPTION

cbf_set_doublevalue sets the item at the current column and row to the floating-point value *number* written as an ASCII string with the format specified by *format* as appropriate for the printf function.

ARGUMENTS

handle CBF handle.
format Format for the number.
number Floating-point value.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.46 cbf_get_value, cbf_require_value
2.3.47 cbf_set_value
2.3.48 cbf_get_typeofvalue
2.3.49 cbf_set_typeofvalue
2.3.51 cbf_set_integervalue
2.3.52 cbf_get_doublevalue, cbf_require_doublevalue
2.3.56 cbf_set_integerarray, cbf_set_realarray
2.3.62 cbf_require_column_value
2.3.63 cbf_require_column_integervalue
2.3.64 cbf_require_column_doublevalue

2.3.54 cbf_get_integerarrayparameters, cbf_get_realarrayparameters

PROTOTYPE

#include "cbf.h"

int cbf_get_integerarrayparameters (cbf_handle *handle*, unsigned int **compression*, int **binary_id*, size_t **elsize*, int **elsigned*, int **elunsigned*, size_t **elements*, int **minelement*, int **maxelement*);
int cbf_get_realarrayparameters (cbf_handle *handle*, unsigned int **compression*, int **binary_id*, size_t **elsize*, size_t **elements*);

DESCRIPTION

cbf_get_integerarrayparameters sets **compression*, **binary_id*, **elsize*, **elsigned*, **elunsigned*, **elements*, **minelement* and **maxelement* to values read from the binary value of the item at the current column and row. This provides all the arguments needed for a subsequent call to cbf_set_integerarray, if a copy of the arry is to be made

into another CIF or CBF. cbf_get_realarrayparameters sets **compression*, **binary_id*, **elsize*, **elements* to values read from the binary value of the item at the current column and row. This provides all the arguments needed for a subsequent call to cbf_set_realarray, if a copy of the arry is to be made into another CIF or CBF.

If the value is not binary, the function returns CBF_ASCII.

ARGUMENTS

handle CBF handle.
compression Compression method used.
elsize Size in bytes of each array element.
binary_id Pointer to the destination integer binary identifier.
elsigned Pointer to an integer. Set to 1 if the elements can be read as signed integers.
elunsigned Pointer to an integer. Set to 1 if the elements can be read as unsigned integers.
elements Pointer to the destination number of elements.
minelement Pointer to the destination smallest element.
maxelement Pointer to the destination largest element.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.46 cbf_get_value, cbf_require_value
2.3.48 cbf_get_typeofvalue
2.3.49 cbf_set_typeofvalue
2.3.50 cbf_get_integervalue, cbf_require_integervalue
2.3.52 cbf_get_doublevalue, cbf_require_doublevalue
2.3.55 cbf_get_integerarray, cbf_get_realarray
2.3.56 cbf_set_integerarray, cbf_set_realarray
2.3.62 cbf_require_column_value
2.3.63 cbf_require_column_integervalue
2.3.64 cbf_require_column_doublevalue

2.3.55 cbf_get_integerarray, cbf_get_realarray

PROTOTYPE

#include "cbf.h"

int cbf_get_integerarray (cbf_handle *handle*, int **binary_id*, void **array*, size_t *elsize*, int *elsigned*, size_t *elements*, size_t **elements_read*);
int cbf_get_integerarray (cbf_handle *handle*, int **binary_id*, void **array*, size_t *elsize*, size_t *elements*, size_t **elements_read*);

DESCRIPTION

cbf_get_integerarray reads the binary value of the item at the current column and row into an integer array. The array consists of *elements* elements of *elsize* bytes each, starting at *array*. The elements are signed if *elsigned* is non-0 and unsigned otherwise. **binary_id* is set to the binary section identifier and **elements_read* to the number of elements actually read. cbf_get_realarray reads the binary value of the item at the current column and row into a real array. The array consists of *elements* elements of *elsize* bytes each, starting at *array*. **binary_id* is set to the binary section identifier and **elements_read* to the number of elements actually read.

If any element in the integer binary data cant fit into the destination element, the destination is set the nearest

possible value.

If the value is not binary, the function returns CBF_ASCII.

If the requested number of elements cant be read, the function will read as many as it can and then return CBF_ENDOFDATA.

Currently, the destination array must consist of chars, shorts or ints (signed or unsigned). If *elsize* is not equal to sizeof (char), sizeof (short) or sizeof (int), for cbf_get_integerarray, or sizeof(double) or sizeof(float), for cbf_get_realarray the function returns CBF_ARGUMENT.

An additional restriction in the current version of CBFlib is that values too large to fit in an int are not correctly decompressed. As an example, if the machine with 32-bit ints is reading an array containing a value outside the range 0 .. 2^{32}-1 (unsigned) or -2^{31} .. 2^{31}-1 (signed), the array will not be correctly decompressed. This restriction will be removed in a future release. For cbf_get_realarray, only IEEE format is supported. No conversion to other floating point formats is done at this time.

ARGUMENTS

handle CBF handle.
binary_id Pointer to the destination integer binary identifier.
array Pointer to the destination array.
elsize Size in bytes of each destination array element.
elsigned Set to non-0 if the destination array elements are signed.
elements The number of elements to read.
elements_read Pointer to the destination number of elements actually read.

RETURN VALUE

Returns an error code on failure or 0 for success.
SEE ALSO

2.3.46 cbf_get_value, cbf_require_value
2.3.48 cbf_get_typeofvalue
2.3.49 cbf_set_typeofvalue
2.3.50 cbf_get_integervalue, cbf_require_integervalue
2.3.52 cbf_get_doublevalue, cbf_require_doublevalue
2.3.54 cbf_get_integerarrayparameters, cbf_get_realarrayparameters
2.3.56 cbf_set_integerarray, cbf_set_realarray
2.3.62 cbf_require_column_value
2.3.63 cbf_require_column_integervalue
2.3.64 cbf_require_column_doublevalue

2.3.56 cbf_set_integerarray, cbf_set_realarray

PROTOTYPE

#include "cbf.h"

int cbf_set_integerarray (cbf_handle *handle*, unsigned int *compression*, int *binary_id*, void **array*, size_t *elsize*, int *elsigned*, size_t *elements*);
int cbf_set_realarray (cbf_handle *handle*, unsigned int *compression*, int *binary_id*, void **array*, size_t *elsize*, size_t *elements*);

DESCRIPTION

cbf_set_integerarray sets the binary value of the item at the current column and row to an integer *array*. The array consists of *elements* elements of *elsize* bytes each, starting at *array*. The elements are signed if *elsigned* is non-0 and unsigned otherwise. *binary_id* is the binary section identifier. cbf_set_realarray sets the binary value of the item at the current column and row to an integer *array*. The array consists of *elements* elements of *elsize* bytes each, starting at *array*. *binary_id* is the binary section identifier.

The array will be compressed using the compression scheme specifed by *compression*. Currently, the available schemes are:

CBF_CANONICAL Canonical-code compression (section 3.3.1)
CBF_PACKED CCP4-style packing (section 3.3.2) CBF_NONE No compression. NOTE: This scheme is by far the slowest of the three and uses much more disk space. It is intended for routine use with small arrays only. With large arrays (like images) it should be used only for debugging.

The values compressed are limited to 64 bits. If any element in the array is larger than 64 bits, the value compressed is the nearest 64-bit value.

Currently, the source array must consist of chars, shorts or ints (signed or unsigned), for cbf_set_integerarray, or IEEE doubles or floats for cbf_set_realarray. If *elsize* is not equal to sizeof (char), sizeof (short) or sizeof (int), the function returns CBF_ARGUMENT.

ARGUMENTS

handle CBF handle.
compression Compression method to use.
binary_id Integer binary identifier.
array Pointer to the source array.
elsize Size in bytes of each source array element.
elsigned Set to non-0 if the source array elements are signed.
elements The number of elements in the array.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.47 cbf_set_value
2.3.48 cbf_get_typeofvalue
2.3.49 cbf_set_typeofvalue
2.3.51 cbf_set_integervalue
2.3.53 cbf_set_doublevalue
2.3.54 cbf_get_integerarrayparameters, cbf_get_realarrayparameters
2.3.55 cbf_get_integerarray, cbf_get_realarray
2.3.62 cbf_require_column_value
2.3.63 cbf_require_column_integervalue
2.3.64 cbf_require_column_doublevalue

2.3.57 cbf_failnez

DEFINITION

#include "cbf.h"

#define cbf_failnez(f) {int err; err = (f); if (err) return err; }

DESCRIPTION

cbf_failnez is a macro used for error propagation throughout CBFlib. cbf_failnez executes the function *f* and saves the returned error value. If the error value is non-0, cbf_failnez executes a return with the error value as argument. If CBFDEBUG is defined, then a report of the error is also printed to the standard error stream, stderr, in the form

CBFlib error *f* in "*symbol*"

where *f* is the decimal value of the error and *symbol* is the symbolic form.

ARGUMENTS
f Integer error value.

SEE ALSO

2.3.58 cbf_onfailnez

2.3.58 cbf_onfailnez

DEFINITION

#include "cbf.h"

#define cbf_onfailnez(f,c) {int err; err = (f); if (err) {{c; }return err; }}

DESCRIPTION

cbf_onfailnez is a macro used for error propagation throughout CBFlib. cbf_onfailnez executes the function *f* and saves the returned error value. If the error value is non-0, cbf_failnez executes first the statement *c* and then a return with the error value as argument. If CBFDEBUG is defined, then a report of the error is also printed to the standard error stream, stderr, in the form

CBFlib error *f* in "*symbol*"

where *f* is the decimal value of the error and *symbol* is the symbolic form.

ARGUMENTS
f integer function to execute.
c statement to execute on failure.

SEE ALSO
2.3.57 cbf_failnez

2.3.59 cbf_require_datablock

PROTOTYPE

#include "cbf.h"

int cbf_require_datablock (cbf_handle *handle*, const char **datablockname*);

DESCRIPTION

cbf_require_datablock makes the data block with name *datablockname* the current data block, if it exists, or creates it if it does not.

The comparison is case-insensitive.

The current category becomes undefined.

ARGUMENTS

handle CBF handle.
datablockname The name of the data block to find or create.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.21 cbf_rewind_datablock
2.3.25 cbf_next_datablock
2.3.29 cbf_find_datablock
2.3.30 cbf_find_category, cbf_find_saveframe, cbf_find_blockitem
2.3.31 cbf_find_column
2.3.32 cbf_find_row
2.3.42 cbf_datablock_name
2.3.60 cbf_require_category
2.3.61 cbf_require_column

2.3.60 cbf_require_category

PROTOTYPE

#include "cbf.h"

int cbf_require_category (cbf_handle *handle*, const char **categoryname*);

DESCRIPTION

cbf_rewuire_category makes the category in the current data block with name *categoryname* the current category, if it exists, or creates the catagory if it does not exist.

The comparison is case-insensitive.

The current column and row become undefined.

ARGUMENTS

handle CBF handle.
categoryname The name of the category to find.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.22 cbf_rewind_category, cbf_rewind_saveframe, cbf_rewind_blockitem
2.3.26 cbf_next_category, cbf_next_saveframe, cbf_next_blockitem
2.3.29 cbf_find_datablock
2.3.31 cbf_find_column
2.3.32 cbf_find_row
2.3.43 cbf_category_name
2.3.59 cbf_require_datablock
2.3.61 cbf_require_column

2.3.61 cbf_require_column

PROTOTYPE

#include "cbf.h"

int cbf_require_column (cbf_handle *handle*, const char **columnname*);

DESCRIPTION

cbf_require_column makes the columns in the current category with name *columnname* the current column, if it exists, or creates it if it does not.

The comparison is case-insensitive.

The current row is not affected.

ARGUMENTS
handle CBF handle.
columnname The name of column to find.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.19 cbf_rewind_column
2.3.27 cbf_next_column
2.3.29 cbf_find_datablock
2.3.30 cbf_find_category, cbf_find_saveframe, cbf_find_blockitem
2.3.32 cbf_find_row
2.3.44 cbf_column_name
2.3.59 cbf_require_datablock
2.3.60 cbf_require_category

2.3.62 cbf_require_column_value

PROTOTYPE

#include "cbf.h"

int cbf_require_column_value (cbf_handle *handle*, const char **columnname*, const char ***value*, const char *defaultvalue*);

DESCRIPTION

cbf_require_column_doublevalue sets **value* to the ASCII item at the current row for the column given with the name given by **columnname*, or to the string given by *defaultvalue* if the item cannot be found.

ARGUMENTS

handle CBF handle.
columnname Name of the column containing the number.
number pointer to the location to receive the integer value.
defaultvalue Value to use if the requested column and value cannot be found.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.46 cbf_get_value, cbf_require_value
2.3.47 cbf_set_value
2.3.48 cbf_get_typeofvalue
2.3.49 cbf_set_typeofvalue
2.3.51 cbf_set_integervalue
2.3.52 cbf_get_doublevalue, cbf_require_doublevalue
2.3.56 cbf_set_integerarray, cbf_set_realarray
2.3.63 cbf_require_column_integervalue
2.3.64 cbf_require_column_doublevalue

2.3.63 cbf_require_column_integervalue

PROTOTYPE

#include "cbf.h"

int cbf_require_column_integervalue (cbf_handle *handle*, const char **columnname*, int **number*, const int *defaultvalue*);

DESCRIPTION

cbf_require_column_doublevalue sets **number* to the value of the ASCII item at the current row for the column given with the name given by **columnname*, with the value interpreted as an integer number, or to the number given by *defaultvalue* if the item cannot be found.

ARGUMENTS

handle CBF handle.
columnname Name of the column containing the number.

number pointer to the location to receive the integer value.
defaultvalue Value to use if the requested column and value cannot be found.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.46 cbf_get_value, cbf_require_value
2.3.47 cbf_set_value
2.3.48 cbf_get_typeofvalue
2.3.49 cbf_set_typeofvalue
2.3.51 cbf_set_integervalue
2.3.52 cbf_get_doublevalue, cbf_require_doublevalue
2.3.56 cbf_set_integerarray, cbf_set_realarray
2.3.62 cbf_require_column_value
2.3.64 cbf_require_column_doublevalue

2.3.64 cbf_require_column_doublevalue

PROTOTYPE

#include "cbf.h"

int cbf_require_column_doublevalue (cbf_handle *handle*, const char **columnname*, double **number*, const double *defaultvalue*);

DESCRIPTION

cbf_require_column_doublevalue sets **number* to the value of the ASCII item at the current row for the column given with the name given by **columnname*, with the value interpreted as a decimal floating-point number, or to the number given by *defaultvalue* if the item cannot be found.

ARGUMENTS
handle CBF handle.
columnname Name of the column containing the number.
number pointer to the location to receive the floating-point value.
defaultvalue Value to use if the requested column and value cannot be found.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.3.46 cbf_get_value, cbf_require_value
2.3.47 cbf_set_value
2.3.48 cbf_get_typeofvalue
2.3.49 cbf_set_typeofvalue
2.3.51 cbf_set_integervalue
2.3.52 cbf_get_doublevalue, cbf_require_doublevalue
2.3.56 cbf_set_integerarray, cbf_set_realarray

2.3.62 cbf_require_column_value
2.3.63 cbf_require_column_integervalue

2.3.65 cbf_get_local_integer_byte_order, cbf_get_local_real_byte_order, cbf_get_local_real_format

PROTOTYPE

#include "cbf.h"

int cbf_get_local_integer_byte_order (char ** *byte_order*);
int cbf_get_local_real_byte_order (char ** *byte_order*);
int cbf_get_local_real_format (char ** *real_format*);

DESCRIPTION

cbf_get_local_integer_byte_order returns the byte order of integers on the machine on which the API is being run in the form of a character string returned as the value pointed to by *byte_order*. cbf_get_local_real_byte_order returns the byte order of reals on the machine on which the API is being run in the form of a character string returned as the value pointed to by *byte_order*. cbf_get_local_real_format returns the format of floats on the machine on which the API is being run in the form of a character string returned as the value pointed to by *real_format*. The strings returned must not be modified in any way.

The values returned in *byte_order* may be the strings "little_endian" or "big-endian". The values returned in *real_format* may be the strings "ieee 754-1985" or "other". Additional values may be returned by future versions of the API.

ARGUMENTS
byte_order pointer to the returned string
real_format pointer to the returned string

RETURN VALUE

Returns an error code on failure or 0 for success.

2.3.66 cbf_get_dictionary, cbf_set_dictionary, cbf_require_dictionary

PROTOTYPE

#include "cbf.h"

int cbf_get_dictionary (cbf_handle *handle*, cbf_handle * *dictionary*);
int cbf_set_dictionary (cbf_handle *handle*, cbf_handle *dictionary_in*);
int cbf_require_dictionary (cbf_handle *handle*, cbf_handle * *dictionary*)

DESCRIPTION

cbf_get_dictionary sets **dictionary* to the handle of a CBF which has been associated with the CBF *handle* by cbf_set_dictionary. cbf_set_dictionary associates the CBF handle *dictionary_in* with *handle* as its dictionary. cbf_require_dictionary sets **dictionary* to the handle of a CBF which has been associated with the CBF *handle* by cbf_set_dictionary or creates a new empty CBF and associates it with *handle*, returning the new handle in **dictionary*.

ARGUMENTS
handle CBF handle.

dictionary Pointer to CBF handle of dictionary.
dictionary_in CBF handle of dcitionary.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.3.67 cbf_convert_dictionary

PROTOTYPE

#include "cbf.h"

int cbf_convert_dictionary (cbf_handle *handle*, cbf_handle *dictionary*)

DESCRIPTION

cbf_convert_dictionary converts *dictionary* as a DDL1 or DDL2 dictionary to a CBF dictionary of category and item properties for *handle*, creating a new dictionary if none exists or layering the definitions in *dictionary* onto the existing dictionary of *handle* if one exists.

If a CBF is read into *handle* after calling cbf_convert_dictionary, then the dictionary will be used for validation of the CBF as it is read.

ARGUMENTS
handle CBF handle.
dictionary CBF handle of dictionary.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.3.68 cbf_find_tag, cbf_find_local_tag

PROTOTYPE

#include "cbf.h"

int cbf_find_tag (cbf_handle *handle*, const char **tag*)
int cbf_find_local_tag (cbf_handle *handle*, const char **tag*)

DESCRIPTION

cbf_find_tag searches all of the CBF *handle* for the CIF tag given by the string *tag* and makes it the current tag. The search does not include the dictionary, but does include save frames as well as categories.

The string *tag* is the complete tag in either DDL1 or DDL2 format, starting with the leading underscore, not just a category or column.

ARGUMENTS
handle CBF handle.
tag CIF tag.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.3.69 cbf_find_category_root, cbf_set_category_root, cbf_require_category_root

PROTOTYPE

#include "cbf.h"

int cbf_find_category_root (cbf_handle *handle*, const char* *categoryname*, const char** *categoryroot*);
int cbf_set_category_root (cbf_handle *handle*, const char* *categoryname_in*, const char**categoryroot*);
int cbf_require_category_root (cbf_handle handle, const char* *categoryname*, const char** *categoryroot*);

DESCRIPTION

cbf_find_category_root sets **categoryroot* to the root category of which *categoryname* is an alias.
cbf_set_category_root sets *categoryname_in* as an alias of *categoryroot* in the dictionary associated with *handle*, creating the dictionary if necessary. cbf_require_category_root sets **categoryroot* to the root category of which *categoryname* is an alias, if there is one, or to the value of *categoryname*, if *categoryname* is not an alias.

A returned *categoryroot* string must not be modified in any way.

ARGUMENTS

handle CBF handle.
categoryname category name which may be an alias.
categoryroot pointer to a returned category root name.
categoryroot_in input category root name.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.3.70 cbf_find_tag_root, cbf_set_tag_root, cbf_require_tag_root

PROTOTYPE

#include "cbf.h"

int cbf_find_tag_root (cbf_handle *handle*, const char* *tagname*, const char** *tagroot*);
int cbf_set_tag_root (cbf_handle *handle*, const char* *tagname*, const char**tagroot_in*);
int cbf_require_tag_root (cbf_handle handle, const char* *tagname*, const char** *tagroot*);

DESCRIPTION

cbf_find_tag_root sets **tagroot* to the root tag of which *tagname* is an alias. cbf_set_tag_root sets *tagname* as an alias of *tagroot_in* in the dictionary associated with *handle*, creating the dictionary if necessary.
cbf_require_tag_root sets **tagroot* to the root tag of which *tagname* is an alias, if there is one, or to the value of *tagname*, if *tagname* is not an alias.

A returned *tagroot* string must not be modified in any way.

ARGUMENTS

handle CBF handle.
tagname tag name which may be an alias.
tagroot pointer to a returned tag root name.
tagroot_in input tag root name.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.3.71 cbf_find_tag_category, cbf_set_tag_category

PROTOTYPE

#include "cbf.h"

int cbf_find_tag_category (cbf_handle *handle*, const char* *tagname*, const char** *categoryname*);
int cbf_set_tag_category (cbf_handle *handle*, const char* tagname, const char* *categoryname_in*);

DESCRIPTION

cbf_find_tag_category sets *categoryname* to the category associated with *tagname* in the dictionary associated with *handle*. cbf_set_tag_category upddates the dictionary associated with *handle* to indicated that *tagname* is in category *categoryname_in*.

ARGUMENTS

handle CBF handle.
tagname tag name.
categoryname pointer to a returned category name.
categoryname_in input category name.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4 High-level function prototypes

2.4.1 cbf_read_template

PROTOTYPE

#include "cbf_simple.h"

int cbf_read_template (cbf_handle *handle*, FILE **file*);

DESCRIPTION

cbf_read_template reads the CBF or CIF file *file* into the CBF object specified by *handle* and selects the first datablock as the current datablock.

ARGUMENTS

handle Pointer to a CBF handle.
file Pointer to a file descriptor.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.2 cbf_get_diffrn_id, cbf_require_diffrn_id

PROTOTYPE

```
#include "cbf_simple.h"

int cbf_get_diffrn_id (cbf_handle handle, const char **diffrn_id);
int cbf_require_diffrn_id (cbf_handle handle, const char **diffrn_id, const char *default_id)
```

DESCRIPTION

cbf_get_diffrn_id sets **diffrn_id* to point to the ASCII value of the "diffrn.id" entry. cbf_require_diffrn_id also sets **diffrn_id* to point to the ASCII value of the "diffrn.id" entry, but, if the "diffrn.id" entry does not exist, it sets the value in the CBF and in**diffrn_id* to the character string given by *default_id*, creating the category and column is necessary.

The *diffrn_id* will be valid as long as the item exists and has not been set to a new value.

The *diffrn_id* must not be modified by the program in any way.

ARGUMENTS

handle CBF handle.
diffrn_id Pointer to the destination value pointer.
default_id Character string default value.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.3 cbf_set_diffrn_id

PROTOTYPE

```
#include "cbf_simple.h"

int cbf_set_diffrn_id (cbf_handle handle, const char *diffrn_id);
```

DESCRIPTION

cbf_set_diffrn_id sets the "diffrn.id" entry of the current datablock to the ASCII value *diffrn_id*.

This function also changes corresponding "diffrn_id" entries in the "diffrn_source", "diffrn_radiation", "diffrn_detector" and "diffrn_measurement" categories.

ARGUMENTS
handle CBF handle.
diffrn_id ASCII value.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.4 cbf_get_crystal_id

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_crystal_id (cbf_handle *handle*, const char **_crystal_id_);

DESCRIPTION

cbf_get_crystal_id sets *_crystal_id_ to point to the ASCII value of the "diffrn.crystal_id" entry.

If the value is not ASCII, the function returns CBF_BINARY.

The value will be valid as long as the item exists and has not been set to a new value.

The value must not be modified by the program in any way.

ARGUMENTS
handle CBF handle.
crystal_id Pointer to the destination value pointer.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.5 cbf_set_crystal_id

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_crystal_id (cbf_handle *handle*, const char *_crystal_id_);

DESCRIPTION

cbf_set_crystal_id sets the "diffrn.crystal_id" entry to the ASCII value *crystal_id*.

ARGUMENTS
handle CBF handle.
crystal_id ASCII value.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.6 cbf_get_wavelength

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_wavelength (cbf_handle *handle*, double **wavelength*);

DESCRIPTION

cbf_get_wavelength sets **wavelength* to the current wavelength in Å.

ARGUMENTS
handle CBF handle.
wavelength Pointer to the destination.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.7 cbf_set_wavelength

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_wavelength (cbf_handle *handle*, double *wavelength*);

DESCRIPTION

cbf_set_wavelength sets the current wavelength in Å to *wavelength*.

ARGUMENTS
handle CBF handle.
wavelength Wavelength in Å.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.8 cbf_get_polarization

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_polarization (cbf_handle *handle*, double **polarizn_source_ratio*, double **polarizn_source_norm*);

DESCRIPTION

cbf_get_polarization sets **polarizn_source_ratio* and **polarizn_source_norm* to the corresponding source polarization parameters.

Either destination pointer may be NULL.

ARGUMENTS

handle CBF handle.
polarizn_source_ratio Pointer to the destination polarizn_source_ratio.
polarizn_source_norm Pointer to the destination polarizn_source_norm.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.9 cbf_set_polarization

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_polarization (cbf_handle *handle*, double *polarizn_source_ratio*, double *polarizn_source_norm*);

DESCRIPTION

cbf_set_polarization sets the source polarization to the values specified by *polarizn_source_ratio* and *polarizn_source_norm*.

ARGUMENTS

handle CBF handle.
polarizn_source_ratio New value of polarizn_source_ratio.
polarizn_source_norm New value of polarizn_source_norm.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.10 cbf_get_divergence

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_divergence (cbf_handle *handle*, double **div_x_source*, double **div_y_source*, double **div_x_y_source*);

DESCRIPTION

cbf_get_divergence sets **div_x_source*, **div_y_source* and **div_x_y_source* to the corresponding source divergence parameters.

Any of the destination pointers may be NULL.

ARGUMENTS

handle CBF handle.
div_x_source Pointer to the destination div_x_source.
div_y_source Pointer to the destination div_y_source.
div_x_y_source Pointer to the destination div_x_y_source.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.11 cbf_ set_divergence

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_divergence (cbf_handle *handle*, double *div_x_source*, double *div_y_source*, double *div_x_y_source*);

DESCRIPTION

cbf_set_divergence sets the source divergence parameters to the values specified by *div_x_source*, *div_y_source* and *div_x_y_source*.

ARGUMENTS
handle CBF handle.
div_x_source New value of div_x_source.
div_y_source New value of div_y_source.
div_x_y_source New value of div_x_y_source.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.12 cbf_count_elements

PROTOTYPE

#include "cbf_simple.h"

int cbf_count_elements (cbf_handle *handle*, unsigned int **elements*);

DESCRIPTION

cbf_count_elements sets **elements* to the number of detector elements.

ARGUMENTS
handle CBF handle.
elements Pointer to the destination count.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.13 cbf_get_element_id

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_element_id (cbf_handle *handle*, unsigned int *element_number*, const char ***element_id*);

DESCRIPTION

cbf_get_element_id sets **element_id* to point to the ASCII value of the *element_number*'th "diffrn_data_frame.detector_element_id" entry, counting from 0.

If the detector element does not exist, the function returns CBF_NOTFOUND.

The *element_id* will be valid as long as the item exists and has not been set to a new value.

The *element_id* must not be modified by the program in any way.

ARGUMENTS

handle CBF handle.
element_number The number of the detector element counting from 0 by order of appearance in the "diffrn_data_frame" category.
element_id Pointer to the destination.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.14 cbf_get_gain

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_gain (cbf_handle *handle*, unsigned int *element_number*, double **gain*, double **gain_esd*);

DESCRIPTION

cbf_get_gain sets **gain* and **gain_esd* to the corresponding gain parameters for element number *element_number*.

Either of the destination pointers may be NULL.

ARGUMENTS

handle CBF handle.
element_number The number of the detector element counting from 0 by order of appearance in the "diffrn_data_frame" category.
gain Pointer to the destination gain.
gain_esd Pointer to the destination gain_esd.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.15 cbf_ set_gain

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_gain (cbf_handle *handle*, unsigned int *element_number*, double *gain*, double *gain_esd*);

DESCRIPTION

cbf_set_gain sets the gain of element number *element_number* to the values specified by *gain* and *gain_esd*.

ARGUMENTS

handle CBF handle.
element_number The number of the detector element counting from 0 by order of appearance in the "diffrn_data_frame" category.
gain New gain value.
gain_esd New gain_esd value.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.16 cbf_get_overload

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_overload (cbf_handle *handle*, unsigned int *element_number*, double **overload*);

DESCRIPTION

cbf_get_overload sets **overload* to the overload value for element number *element_number*.

ARGUMENTS

handle CBF handle.
element_number The number of the detector element counting from 0 by order of appearance in the "diffrn_data_frame" category.
overload Pointer to the destination overload.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.17 cbf_ set_overload

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_overload (cbf_handle *handle*, unsigned int *element_number*, double *overload*);

DESCRIPTION

cbf_set_overload sets the overload value of element number *element_number* to *overload*.

ARGUMENTS

handle CBF handle.
element_number The number of the detector element counting from 0 by order of appearance in the "diffrn_data_frame" category.
overload New overload value.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.18 cbf_get_integration_time

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_integration_time (cbf_handle *handle*, unsigned int *reserved*, double **time*);

DESCRIPTION

cbf_get_integration_time sets **time* to the integration time in seconds. The parameter *reserved* is presently unused and should be set to 0.

ARGUMENTS
handle CBF handle.
reserved Unused. Any value other than 0 is invalid.
time Pointer to the destination time.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.19 cbf_set_integration_time

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_integration_time (cbf_handle handle, unsigned int *reserved*, double *time*);

DESCRIPTION

cbf_set_integration_time sets the integration time in seconds to the value specified by *time*. The parameter *reserved* is presently unused and should be set to 0.

ARGUMENTS
handle CBF handle.
reserved Unused. Any value other than 0 is invalid.
time Integration time in seconds.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.20 cbf_get_timestamp

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_timestamp (cbf_handle *handle*, unsigned int *reserved*, double **time*, int **timezone*);

DESCRIPTION

cbf_get_timestamp sets **time* to the collection timestamp in seconds since January 1 1970. **timezone* is set to timezone difference from UTC in minutes. The parameter *reserved* is presently unused and should be set to 0.

Either of the destination pointers may be NULL.

ARGUMENTS

handle CBF handle.
reserved Unused. Any value other than 0 is invalid.
time Pointer to the destination collection timestamp.
timezone Pointer to the destination timezone difference.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.21 cbf_set_timestamp

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_timestamp (cbf_handle *handle*, unsigned int *reserved*, double *time*, int *timezone*, double *precision*);

DESCRIPTION

cbf_set_timestamp sets the collection timestamp in seconds since January 1 1970 to the value specified by *time*. The timezone difference from UTC in minutes is set to *timezone*. If no timezone is desired, *timezone* should be CBF_NOTIM EZONE. The parameter *reserved* is presently unused and should be set to 0.

The precision of the new timestamp is specified by the value *precision* in seconds. If *precision* is 0, the saved timestamp is assumed accurate to 1 second.

ARGUMENTS

handle CBF handle.
reserved Unused. Any value other than 0 is invalid.
time Timestamp in seconds since January 1 1970.
timezone Timezone difference from UTC in minutes or CBF_NOTIMEZONE.
precision Timestamp precision in seconds.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.22 cbf_get_datestamp

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_datestamp (cbf_handle *handle*, unsigned int *reserved*, int **year*, int **month*, int **day*, int **hour*, int **minute*, double **second*, int **timezone*);

DESCRIPTION

cbf_get_datestamp sets **year*, **month*, **day*, **hour*, **minute* and **second* to the corresponding values of the collection timestamp. **timezone* is set to timezone difference from UTC in minutes. The parameter *reserved* is presently unused and should be set to 0.

Any of the destination pointers may be NULL.

ARGUMENTS

handle CBF handle.
reserved Unused. Any value other than 0 is invalid.
year Pointer to the destination timestamp year.
month Pointer to the destination timestamp month (1-12).
day Pointer to the destination timestamp day (1-31).
hour Pointer to the destination timestamp hour (0-23).
minute Pointer to the destination timestamp minute (0-59).
second Pointer to the destination timestamp second (0-60.0).
timezone Pointer to the destination timezone difference from UTC in minutes.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.23 cbf_set_datestamp

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_datestamp (cbf_handle *handle*, unsigned int *reserved*, int *year*, int *month*, int *day*, int *hour*, int *minute*, double *second*, int *timezone*, double *precision*);

DESCRIPTION

cbf_set_datestamp sets the collection timestamp in seconds since January 1 1970 to the value specified by *time*. The timezone difference from UTC in minutes is set to *timezone*. If no timezone is desired, *timezone* should be CBF_NOTIM EZONE. The parameter *reserved* is presently unused and should be set to 0.

The precision of the new timestamp is specified by the value *precision* in seconds. If *precision* is 0, the saved timestamp is assumed accurate to 1 second.

ARGUMENTS

*handle*CBF handle.
*reserved*Unused. Any value other than 0 is invalid.
*time*Timestamp in seconds since January 1 1970.
*timezone*Timezone difference from UTC in minutes or CBF_NOTIMEZONE.
*precision*Timestamp precision in seconds.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.24 cbf_set_current_timestamp

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_current_timestamp (cbf_handle *handle*, unsigned int *reserved*, int *timezone*)

DESCRIPTION

cbf_set_current_timestamp sets the collection timestamp to the current time. The timezone difference from UTC in minutes is set to *timezone*. If no timezone is desired, *timezone* should be CBF_NOTIMEZONE. If no timezone is used, the timest amp will be UTC. The parameter *reserved* is presently unused and should be set to 0.

The new timestamp will have a precision of 1 second.

ARGUMENTS

handle CBF handle.
reserved Unused.
Any value other than 0 is invalid.
timezone Timezone difference from UTC in minutes or CBF_NOTIMEZONE.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.25 cbf_get_image_size

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_image_size (cbf_handle *handle*, unsigned int *reserved*, unsigned int *element_number*, size_t **ndim1*, size_t **ndim2*);

DESCRIPTION

cbf_get_image_size sets **ndim1* and **ndim2* to the slow and fast dimensions of the image array for element number *element_number*. If the array is 1-dimensional, **ndim1* will be set to the array size and **ndim2* will be se t to 1.

Either of the destination pointers may be NULL.

The parameter *reserved* is presently unused and should be set to 0.

ARGUMENTS

handle CBF handle.
reserved Unused. Any value other than 0 is invalid.
element_number The number of the detector element counting from 0 by order of appearance in the "diffrn_data_frame" category.
ndim1 Pointer to the destination slow dimension.
ndim2 Pointer to the destination fast dimension.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.26 cbf_get_image, cbf_get_real_image

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_image (cbf_handle *handle*, unsigned int *reserved*, unsigned int *element_number*, void **array*, size_t *elsize*, int *elsign*, size_t *ndim1*, size_t *ndim2*);
int cbf_get_real_image (cbf_handle *handle*, unsigned int *reserved*, unsigned int *element_number*, void **array*, size_t *elsize*, size_t *ndim1*, size_t *ndim2*);

DESCRIPTION

cbf_get_image reads the image array for element number *element_number* into an *array*. The array consists of *ndim1*°*ndim2* elements of *elsize* bytes each, starting at *array*. The elements are signed if *elsign* is non-0 and unsigned otherwise. cbf_get_real image reads the image array of IEEE doubles or floats for element number *element_number* into an *array*. A real array is always signed.

If the array is 1-dimensional, *ndim1* should be the array size and *ndim2* should be set to 1.

If any element in the binary data can't fit into the destination element, the destination is set the nearest possible value.

If the value is not binary, the function returns CBF_ASCII.

If the requested number of elements can't be read, the function will read as many as it can and then return CBF_ENDOFDATA.

Currently, the destination *array* must consist of chars, shorts or ints (signed or unsigned) for cbf_get_image, or IEEE doubles or floats for cbf_get_real_image. If *elsize* is not equal to sizeof (char), sizeof (short), sizeof (int), sizeof(double) or sizeof(float), the function returns CBF_ARGUMENT.

The parameter *reserved* is presently unused and should be set to 0. **ARGUMENTS**
handle CBF handle.
reserved Unused. Any value other than 0 is invalid.
element_number The number of the detector element counting from 0 by order of appearance in the "diffrn_data_frame" category.
array Pointer to the destination array.
elsize Size in bytes of each destination array element.
elsigned Set to non-0 if the destination array elements are signed.
ndim1 Slow array dimension.
ndim2 Fast array dimension.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.27 cbf_set_image, cbf_set_real_image

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_image (cbf_handle *handle*, unsigned int *reserved*, unsigned int *element_number*, unsigned int compression, void **array*, size_t *elsize*, int *elsign*, size_t *ndim1*, size_t *ndim2*);
int cbf_set_real_image (cbf_handle *handle*, unsigned int *reserved*, unsigned int *element_number*, unsigned int compression, void **array*,size_t *elsize*, size_t *ndim1*, size_t *ndim2*);

DESCRIPTION

cbf_set_image writes the image array for element number *element_number*. The *array* consists of *ndim1*°*ndim2* elements of *elsize* bytes each, starting at *array*. The elements are signed if *elsign* is non-0 and unsigned otherwise. cbf_set_real_image writes the image array for element number *element_number*. The *array* consists of *ndim1*°*ndim2* IEEE double or float elements of *elsize* bytes each, starting at *array*.

If the array is 1-dimensional, *ndim1* should be the array size and *ndim2* should be set to 1.

The array will be compressed using the compression scheme specifed by compression. Currently, the available schemes are:

CBF_CANONICALCanonical-code compression (section 3.3.1) CBF_PACKEDCCP4-style packing (section 3.3.2) CBF_NONENo compression.

The values compressed are limited to 64 bits. If any element in the array is larger than 64 bits, the value compressed is the nearest 64-bit value.

Currently, the source *array* must consist of chars, shorts or ints (signed or unsigned)for cbf_set_image, or IEEE doubles or floats for cbf_set_real_image. If *elsize* is not equal to sizeof (short), sizeof (int), sizeof(double) or sizeof(float), the function returns CBF_ARGUMENT.

The parameter *reserved* is presently unused and should be set to 0.

ARGUMENTS

handle CBF handle.
reserved Unused. Any value other than 0 is invalid.
element_number The number of the detector element counting from 0 by order of appearance in the "diffrn_data_frame" category.
compression Compression type.
array Pointer to the image array.
elsize Size in bytes of each image array element.
elsigned Set to non-0 if the image array elements are signed.
ndim1 Slow array dimension.
ndim2 Fast array dimension.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.28 cbf_get_axis_setting

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_axis_setting (cbf_handle *handle*, unsigned int *reserved*, const char **axis_id*, double **start*, double **increment*);

DESCRIPTION

cbf_get_axis_setting sets **start* and **increment* to the corresponding values of the axis *axis_id*.

Either of the destination pointers may be NULL.

The parameter *reserved* is presently unused and should be set to 0.

ARGUMENTS

handle CBF handle.
reserved Unused. Any value other than 0 is invalid.
axis_id Axis id.
start Pointer to the destination start value.
increment Pointer to the destination increment value.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.29 cbf_set_axis_setting

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_axis_setting (cbf_handle *handle*, unsigned int *reserved*, const char **axis_id*, double *start*, double *increment*);

DESCRIPTION

cbf_set_axis_setting sets the starting and increment values of the axis *axis_id* to *start* and *increment*.

The parameter *reserved* is presently unused and should be set to 0.

ARGUMENTS

handle CBF handle.
reserved Unused. Any value other than 0 is invalid.
axis_id Axis id.
start Start value.
increment Increment value.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.30 cbf_construct_goniometer

PROTOTYPE

#include "cbf_simple.h"

int cbf_construct_goniometer (cbf_handle *handle*, cbf_goniometer **goniometer*);

DESCRIPTION

cbf_construct_goniometer constructs a goniometer object using the description in the CBF object handle and initialises the goniometer handle **goniometer*.

ARGUMENTS
handle CBF handle.
goniometer Pointer to the destination goniometer handle.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.31 cbf_free_goniometer

PROTOTYPE

#include "cbf_simple.h"

int cbf_free_goniometer (cbf_goniometer *goniometer*);

DESCRIPTION

cbf_free_goniometer destroys the goniometer object specified by *goniometer* and frees all associated memory.

ARGUMENTS
goniometer Goniometer handle to free.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.32 cbf_get_rotation_axis

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_rotation_axis (cbf_goniometer *goniometer*, unsigned int *reserved*, double **vector1*, double **vector2*, double *vector3*);

DESCRIPTION

cbf_get_rotation_axis sets **vector1*, **vector2*, and **vector3* to the 3 components of the goniometer rotation axis used for the exposure.

Any of the destination pointers may be NULL.

The parameter *reserved* is presently unused and should be set to 0.

ARGUMENTS

goniometer Goniometer handle.
reserved Unused. Any value other than 0 is invalid.
vector1 Pointer to the destination x component of the rotation axis.
vector2 Pointer to the destination y component of the rotation axis.
vector3 Pointer to the destination z component of the rotation axis.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.33 cbf_get_rotation_range

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_rotation_range (cbf_goniometer *goniometer*, unsigned int *reserved*, double **start*, double **increment*);

DESCRIPTION

cbf_get_rotation_range sets **start* and **increment* to the corresponding values of the goniometer rotation axis used for the exposure.

Either of the destination pointers may be NULL.

The parameter *reserved* is presently unused and should be set to 0.

ARGUMENTS

goniometer Goniometer handle.
reserved Unused. Any value other than 0 is invalid.
start Pointer to the destination start value.
increment Pointer to the destination increment value.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.34 cbf_rotate_vector

PROTOTYPE

#include "cbf_simple.h"

int cbf_rotate_vector (cbf_goniometer *goniometer*, unsigned int *reserved*, double *ratio*, double *initial1*, double *initial2*, double *initial3*, double **final1*, double **final2*, double **final3*);

DESCRIPTION

cbf_rotate_vector sets **final1*, **final2*, and **final3* to the 3 components of the of the vector (*initial1*, *initial2*, *initial3*) after reorientation by applying the goniometer rotations. The value *ratio* specif ies the goniometer setting and varies from 0.0 at the beginning of the exposure to 1.0 at the end, irrespective of the actual rotation range.

Any of the destination pointers may be NULL.

The parameter *reserved* is presently unused and should be set to 0.

ARGUMENTS

goniometer Goniometer handle.
reserved Unused. Any value other than 0 is invalid.
ratio Goniometer setting. 0 = beginning of exposure, 1 = end.
initial1 x component of the initial vector.
initial2 y component of the initial vector.
initial3 z component of the initial vector.
vector1 Pointer to the destination x component of the final vector.
vector2 Pointer to the destination y component of the final vector.
vector3 Pointer to the destination z component of the final vector.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.35 cbf_get_reciprocal

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_reciprocal (cbf_goniometer *goniometer*, unsigned int *reserved*, double *ratio*, double *wavelength*, double *real1*, double *real2*, double *real3*, double **reciprocal1*, double **reciprocal2*, double **reciprocal3*);

DESCRIPTION

cbf_get_reciprocal sets **reciprocal1*, * *reciprocal2*, and * *reciprocal3* to the 3 components of the of the reciprocal-space vector corresponding to the real-space vector (*real1*, *real2*, *real3*). The reciprocal-space vector is oriented to correspond to the goniometer setting with all axes at 0. The value *wavelength* is the wavlength in Å and the value *ratio* specifies the current goniometer setting and varies from 0.0 at the beginning of the exposur e to 1.0 at the end, irrespective of the actual rotation range.

Any of the destination pointers may be NULL.

The parameter *reserved* is presently unused and should be set to 0.

ARGUMENTS

goniometer Goniometer handle.
reserved Unused. Any value other than 0 is invalid.
ratio Goniometer setting. 0 = beginning of exposure, 1 = end.
wavelength Wavelength in Å.
real1 x component of the real-space vector.
real2 y component of the real-space vector.
real3 z component of the real-space vector.
reciprocal1 Pointer to the destination x component of the reciprocal-space vector.
reciprocal2 Pointer to the destination y component of the reciprocal-space vector.
reciprocal3 Pointer to the destination z component of the reciprocal-space vector.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.36 cbf_construct_detector

PROTOTYPE

#include "cbf_simple.h"

int cbf_construct_detector (cbf_handle *handle*, cbf_detector **detector*, unsigned int *element_number*);

DESCRIPTION

cbf_construct_detector constructs a detector object for detector element number *element_number* using the description in the CBF object handle and initialises the detector handle **detector*.

ARGUMENTS
handle CBF handle.
detector Pointer to the destination detector handle.
element_number The number of the detector element counting from 0 by order of appearance in the "diffrn_data_frame" category.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.37 cbf_free_detector

PROTOTYPE

#include "cbf_simple.h"

int cbf_free_detector (cbf_detector *detector*);

DESCRIPTION

cbf_free_detector destroys the detector object specified by *detector* and frees all associated memory.

ARGUMENTS
detector Detector handle to free.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.38 cbf_get_beam_center, cbf_set_beam_center

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_beam_center (cbf_detector *detector*, double **index1*, double **index2*, double **center1*, double **center2*);
int cbf_set_beam_center (cbf_detector *detector*, double **index1*, double **index2*, double **center1*, double **center2*);

DESCRIPTION

cbf_get_beam_center sets **center1* and **center2* to the displacements in mm along the detector axes from pixel (0, 0) to the point at which the beam intersects the detector and **index1* and **index2* to the corresponding indices. cbf_set_beam_center sets the offsets in the axis category for the detector element axis with precedence 1 to place the beam center at the position given in mm by **center1* and **center2* as the displacements in mm along the detector axes from pixel (0, 0) to the point at which the beam intersects the detector at the indices given **index1* and **index2*.

Any of the destination pointers may be NULL for getting the beam center. For setting the beam axis, either the indices of the center must not be NULL.

The indices are non-negative for beam centers within the detector surface, but the center for an axis with a negative increment will be negative for a beam center within the detector surface.

ARGUMENTS

detector Detector handle.
index1 Pointer to the destination slow index.
index2 Pointer to the destination fast index.
center1 Pointer to the destination displacement along the slow axis.
center2 Pointer to the destination displacement along the fast axis.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.39 cbf_get_detector_distance

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_detector_distance (cbf_detector *detector*, double **distance*);

DESCRIPTION

cbf_get_detector_distance sets **distance* to the nearest distance from the sample position to the detector plane.

ARGUMENTS

detector Detector handle.
distance Pointer to the destination distance.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.40 cbf_get_detector_normal

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_detector_normal (cbf_detector *detector*, double **normal1*, double **normal2*, double **normal3*);

DESCRIPTION

cbf_get_detector_normal sets **normal1*, **normal2*, and **normal3* to the 3 components of the of the normal vector to the detector plane. The vector is normalized.

Any of the destination pointers may be NULL.

ARGUMENTS

detector Detector handle.
normal1 Pointer to the destination x component of the normal vector.
normal2 Pointer to the destination y component of the normal vector.
normal3 Pointer to the destination z component of the normal vector.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.41 cbf_get_pixel_coordinates

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_pixel_coordinates (cbf_detector *detector*, double *index1*, double *index2*, double **coordinate1*, double **coordinate2*, double **coordinate3*);

DESCRIPTION

cbf_get_pixel_coordinates sets **coordinate1*, **coordinate2*, and **coordinate3* to the vector position of pixel (*index1*, *index2*) on the detector surface. If *index1* and *index2* are integers then the coordinates correspond to the center of a pixel.

Any of the destination pointers may be NULL.

ARGUMENTS

detector Detector handle.
index1 Slow index.
index2 Fast index.
coordinate1 Pointer to the destination x component.
oordinate2 Pointer to the destination y component.
coordinate3 Pointer to the destination z component.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.42 cbf_get_pixel_normal

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_pixel_normal (cbf_detector *detector*, double *index1*, double *index2*, double **normal1*, double **normal2*, double **normal3*);

DESCRIPTION

cbf_get_detector_normal sets **normal1*, **normal2*, and **normal3* to the 3 components of the of the normal vector to the pixel at (*index1*, *index2*). The vector is normalized.

Any of the destination pointers may be NULL.

ARGUMENTS

detector Detector handle.
index1 Slow index.
index2 Fast index.
normal1 Pointer to the destination x component of the normal vector.
normal2 Pointer to the destination y component of the normal vector.
normal3 Pointer to the destination z component of the normal vector.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.43 cbf_get_pixel_area

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_pixel_area (cbf_detector *detector*, double *index1*, double *index2*, double **area*, double **projected_area*);

DESCRIPTION

cbf_get_pixel_area sets **area* to the area of the pixel at (*index1*, *index2*) on the detector surface and **projected_area* to the apparent area of the pixel as viewed from the sample position.

Either of the destination pointers may be NULL.

ARGUMENTS

detector Detector handle.
index1 Slow index.
index2 Fast index.
area Pointer to the destination area in mm2.
projected_area Pointer to the destination apparent area in mm2.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.44 cbf_get_pixel_size

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_pixel_size (cbf_handle *handle*, unsigned int *element_number*, unsigned int *axis_number*, double **psize*);

DESCRIPTION

cbf_get_pixel_size sets **psize* to point to the double value in millimeters of the axis *axis_number* of the detector element *element_number*. The *axis_number* is numbered from 1, starting with the fastest axis.

If the pixel size is not given explcitly in the "array_element_size" category, the function returns CBF_NOTFOUND.

ARGUMENTS

handle CBF handle.
element_number The number of the detector element counting from 0 by order of appearance in the "diffrn_data_frame" category.
axis_number The number of the axis, fastest first, starting from 1.
psize Pointer to the destination pixel size.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.45 cbf_set_pixel_size

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_pixel_size (cbf_handle *handle*, unsigned int *element_number*, unsigned int *axis_number*, double *psize*);

DESCRIPTION

cbf_set_pixel_size sets the item in the &quote;size&quote; column of the "array_structure_list" category at the row which matches axis *axis_number* of the detector element *element_number* converting the double pixel size *psize* from meters to millimeters in storing it in the "size" column for the axis *axis_number* of the detector element *element_number*. The *axis_number* is numbered from 1, starting with the fastest axis.

If the "array_structure_list" category does not already exist, it is created.

If the appropriate row in the "array_structure_list" catgeory does not already exist, it is created.

If the pixel size is not given explcitly in the "array_element_size category", the function returns CBF_NOTFOUND.

ARGUMENTS

handle CBF handle.
element_number The number of the detector element counting from 0 by order of appearance in the "diffrn_data_frame" category.
axis_number The number of the axis, fastest first, starting from 1.
psize The pixel size in millimeters.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.46 cbf_get_inferred_pixel_size

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_inferred_pixel_size (cbf_detector *detector*, unsigned int *axis_number*, double **psize*);

DESCRIPTION

cbf_get_inferred_pixel_size sets **psize* to point to the double value in millimeters of the pixel size for the axis *axis_number* value for pixel at (*index1*, *index2*) on the detector surface. The slow index is treated as axis 1 and the fast index is treated as axis 2.

ARGUMENTS

detector Detector handle.
axis_number The number of the axis.
area Pointer to the destination pizel size in mm.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.47 cbf_get_unit_cell

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_unit_cell (cbf_handle *handle*, double *cell*[6], double *cell_esd*[6]);

DESCRIPTION

cbf_get_unit_cell sets *cell*[0:2] to the double values of the cell edge lengths a, b and c in Ångstroms, *cell*[3:5] to the double values of the cell angles α, β and γ in degrees, *cell_esd*[0:2] to the double values of the estimated strandard deviations of the cell edge lengths a, b and c in Ångstroms, *cell_esd*[3:5] to the double values of the estimated standard deviations of the the cell angles α, β and γ in degrees.

The values returned are retrieved from the first row of the "cell" category. The value of "_cell.entry_id" is ignored.

cell or *cell_esd* may be NULL.

If *cell* is NULL, the cell parameters are not retrieved.

If *cell_esd* is NULL, the cell parameter esds are not retrieved.

If the "cell" category is present, but some of the values are missing, zeros are returned for the missing values.

ARGUMENTS

handle CBF handle.
cell Pointer to the destination array of 6 doubles for the cell parameters.

cell_esd Pointer to the destination array of 6 doubles for the cell parameter esds.

RETURN VALUE

Returns an error code on failure or 0 for success. No errors is returned for missing values if the "cell" category exists.

SEE ALSO

2.4.48 cbf_set_unit_cell
2.4.49 cbf_get_reciprocal_cell
2.4.50 cbf_set_reciprocal_cell
2.4.51 cbf_compute_cell_volume
2.4.52 cbf_compute_reciprocal_cell

2.4.48 cbf_set_unit_cell

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_unit_cell (cbf_handle *handle*, double *cell*[6], double *cell_esd*[6]);

DESCRIPTION

cbf_set_unit_cell sets the cell parameters to the double values given in *cell*[0:2] for the cell edge lengths a, b and c in Ångstroms, the double values given in *cell*[3:5] for the cell angles α, β and γ in degrees, the double values given in *cell_esd*[0:2] for the estimated strandard deviations of the cell edge lengths a, b and c in Ångstroms, and the double values given in *cell_esd*[3:5] for the estimated standard deviations of the the cell angles α, β and γ in degrees.

The values are placed in the first row of the "cell" category. If no value has been given for "_cell.entry_id", it is set to the value of the "diffrn.id" entry of the current data block.

cell or *cell_esd* may be NULL.

If *cell* is NULL, the cell parameters are not set.

If *cell_esd* is NULL, the cell parameter esds are not set.

If the "cell" category is not present, it is created. If any of the necessary columns are not present, they are created.

ARGUMENTS
handle CBF handle.
cell Pointer to the array of 6 doubles for the cell parameters.
cell_esd Pointer to the array of 6 doubles for the cell parameter esds.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.4.47 cbf_get_unit_cell
2.4.49 cbf_get_reciprocal_cell

2.4.50 cbf_set_reciprocal_cell
2.4.51 cbf_compute_cell_volume
2.4.52 cbf_compute_reciprocal_cell

2.4.49 cbf_get_reciprocal_cell

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_reciprocal_cell (cbf_handle *handle*, double *cell*[6], double *cell_esd*[6]);

DESCRIPTION

cbf_get_reciprocal_cell sets *cell*[0:2] to the double values of the reciprocal cell edge lengths a^*, b^* and c^* in Ångstroms^{-1}, *cell*[3:5] to the double values of the reciprocal cell angles α*, β* and γ* in degrees, *cell_esd*[0:2] to the double values of the estimated strandard deviations of the reciprocal cell edge lengths a^*, b^* and c^* in Ångstroms^{-1}, *cell_esd*[3:5] to the double values of the estimated standard deviations of the the reciprocal cell angles α*, β* and γ* in degrees.

The values returned are retrieved from the first row of the "cell" category. The value of "_cell.entry_id" is ignored.

cell or *cell_esd* may be NULL.

If *cell* is NULL, the reciprocal cell parameters are not retrieved.

If *cell_esd* is NULL, the reciprocal cell parameter esds are not retrieved.

If the "cell" category is present, but some of the values are missing, zeros are returned for the missing values.

ARGUMENTS

handle CBF handle.
cell Pointer to the destination array of 6 doubles for the reciprocal cell parameters.
cell_esd Pointer to the destination array of 6 doubles for the reciprocal cell parameter esds.

RETURN VALUE

Returns an error code on failure or 0 for success. No errors is returned for missing values if the "cell" category exists.

SEE ALSO

2.4.47 cbf_get_unit_cell
2.4.48 cbf_set_unit_cell
2.4.50 cbf_set_reciprocal_cell
2.4.51 cbf_compute_cell_volume
2.4.52 cbf_compute_reciprocal_cell

2.4.50 cbf_set_reciprocal_cell

PROTOTYPE

#include "cbf_simple.h"

int cbf_set_reciprocal_cell (cbf_handle *handle*, double *cell*[6], double *cell_esd*[6]);

DESCRIPTION

cbf_set_reciprocal_cell sets the reciprocal cell parameters to the double values given in *cell*[0:2] for the reciprocal cell edge lengths a^*, b^* and c^* in Ångstroms^{-1}, the double values given in *cell*[3:5] for the reciprocal cell angles α*, β* and γ* in degrees, the double values given in *cell_esd*[0:2] for the estimated strandard deviations of the reciprocal cell edge lengths a^*, b^* and c^* in Ångstroms, and the double values given in *cell_esd*[3:5] for the estimated standard deviations of the reciprocal cell angles α*, β* and γ* in degrees.

The values are placed in the first row of the "cell" category. If no value has been given for "_cell.entry_id", it is set to the value of the "diffrn.id" entry of the current data block.

cell or *cell_esd* may be NULL.

If *cell* is NULL, the reciprocal cell parameters are not set.

If *cell_esd* is NULL, the reciprocal cell parameter esds are not set.

If the "cell" category is not present, it is created. If any of the necessary columns are not present, they are created.

ARGUMENTS

handle CBF handle.
cell Pointer to the array of 6 doubles for the reciprocal cell parameters.
cell_esd Pointer to the array of 6 doubles for the reciprocal cell parameter esds.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.4.47 cbf_get_unit_cell
2.4.48 cbf_set_unit_cell
2.4.50 cbf_get_reciprocal_cell
2.4.51 cbf_compute_cell_volume
2.4.52 cbf_compute_reciprocal_cell

2.4.51 cbf_compute_cell_volume

PROTOTYPE

#include "cbf_simple.h"

int cbf_compute_cell_volume (double *cell*[6], double **volume*);

DESCRIPTION

cbf_compute_cell_volume sets **volume* to point to the volume of the unit cell computed from the double values in *cell*[0:2] for the cell edge lengths a, b and c in Ångstroms and the double values given in *cell*[3:5] for the cell angles α, β and γ in degrees.

ARGUMENTS

cell Pointer to the array of 6 doubles giving the cell parameters.
volume Pointer to the doubles for cell volume.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.4.46 cbf_get_unit_cell
2.4.47 cbf_set_unit_cell
2.4.50 cbf_get_reciprocal_cell
2.4.50 cbf_set_reciprocal_cell
2.4.52 cbf_compute_reciprocal_cell

2.4.52 cbf_compute_reciprocal_cell

PROTOTYPE

#include "cbf_simple.h"

int cbf_compute_reciprocal_cell (double *cell*[6], double *rcell*[6]);

DESCRIPTION

cbf_compute_reciprocal_cell sets *rcell* to point to the array of reciprocal cell parameters computed from the double values *cell*[0:2] giving the cell edge lengths a, b and c in Ångstroms, and the double values *cell*[3:5] giving the cell angles α, β and γ in degrees. The double values *rcell*[0:2] will be set to the reciprocal cell lengths a^*, b^* and c^* in $Ångstroms^{-1}$ and the double values *rcell*[3:5] will be set to the reciprocal cell angles α*, β* and γ* in degrees.

ARGUMENTS

cell Pointer to the array of 6 doubles giving the cell parameters.
rcell Pointer to the destination array of 6 doubles giving the reciprocal cell parameters.
volume Pointer to the doubles for cell volume.

RETURN VALUE

Returns an error code on failure or 0 for success.

SEE ALSO

2.4.46 cbf_get_unit_cell
2.4.47 cbf_set_unit_cell
2.4.50 cbf_get_reciprocal_cell
2.4.50 cbf_set_reciprocal_cell
2.4.51 cbf_compute_cell_volume

2.4.53 cbf_get_orientation_matrix, cbf_set_orientation_matrix

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_orientation_matrix (cbf_handle *handle*, double *ub_matrix*[9]);
int cbf_set_orientation_matrix (cbf_handle *handle*, double *ub_matrix*[9]);

cbf_get_orientation_matrix sets *ub_matrix* to point to the array of orientation matrix entries in the "diffrn" category in the order of columns:

"UB[1][1]" "UB[1][2]" "UB[1][3]"
"UB[2][1]" "UB[2][2]" "UB[2][3]"
"UB[3][1]" "UB[3][2]" "UB[3][3]"

cbf_set_orientation_matrix sets the values in the "diffrn" category to the values pointed to by *ub_matrix*.

ARGUMENTS

handle CBF handle.
ubmatric Source or destination array of 9 doubles giving the orientation matrix parameters.

RETURN VALUE

Returns an error code on failure or 0 for success.

2.4.54 cbf_get_bin_sizes, cbf_set_bin_sizes

PROTOTYPE

#include "cbf_simple.h"

int cbf_get_bin_sizes(cbf_handle *handle*, unsigned int *element_number*, double * *slowbinsize*, double * *fastbinsize*);
int cbf_set_bin_sizes(cbf_handle *handle*, unsigned int element_number, double *slowbinsize_in*,double *fastbinsize_in*);

cbf_get_bin_sizes sets *slowbinsize* to point to the value of the number of pixels composing one array element in the dimension that changes at the second-fastest rate and *fastbinsize* to point to the value of the number of pixels composing one array element in the dimension that changes at the fastest rate for the dectector element with the ordinal *element_number*. cbf_set_bin_sizes sets the the pixel bin sizes in the "array_intensities" category to the values of *slowbinsize_in* for the number of pixels composing one array element in the dimension that changes at the second-fastest rate and *fastbinsize_in* for the number of pixels composing one array element in the dimension that changes at the fastest rate for the dectector element with the ordinal *element_number*.

In order to allow for software binning involving fractions of pixels, the bin sizes are doubles rather than ints.

ARGUMENTS

handle	CBF handle.
element_number	The number of the detector element counting from 0 by order of appearance in the "diffrn_data_frame" category.
slowbinsize	Pointer to the returned number of pixels composing one array element in the dimension that changes at the second-fastest rate.
fastbinsize	Pointer to the returned number of pixels composing one array element in the dimension that changes at the fastest rate.
slowbinsize_in	The number of pixels composing one array element in the dimension that changes at the second-fastest rate.
fastbinsize_in	The number of pixels composing one array element in the dimension that changes at the fastest rate.

RETURN VALUE

Returns an error code on failure or 0 for success.

3. File format

3.1 General description

With the exception of the binary sections, a CBF file is an mmCIF-format ASCII file, so a CBF file with no binary sections is a CIF file. An imgCIF file has any binary sections encoded as CIF-format ASCII strings and is a CIF file whether or not it contains binary sections. In most cases, CBFlib can also be used to access normal CIF files as well as CBF and imgCIF files.

3.2 Format of the binary sections

Before getting to the binary data itself, there are some preliminaries to allow a smooth transition from the conventions of CIF to those of raw or encoded streams of "octets" (8-bit bytes). The binary data is given as the essential part of a specially formatted semicolon-delimited CIF multi-line text string. This text string is the value associated with the tag "_array_data.data".

The specific format of the binary sections differs between an imgCIF and a CBF file.

3.2.1 Format of imgCIF binary sections

Each binary section is encoded as a ;-delimited string. Within the text string, the conventions developed for transmitting email messages including binary attachments are followed. There is secondary ASCII header information, formatted as Multipurpose Internet Mail Extensions (MIME) headers (see RFCs 2045-49 by Freed, et al.). The boundary marker for the beginning of all this is the special string

--CIF-BINARY-FORMAT-SECTION--

at the beginning of a line. The initial "--" says that this is a MIME boundary. We cannot put "###" in front of it and conform to MIME conventions. Immediately after the boundary marker are MIME headers, describing some useful information we will need to process the binary section. MIME headers can appear in different orders, and can be very confusing (look at the raw contents of a email message with attachments), but there is only one header which is has to be understood to process an imgCIF: "Content-Transfer-Encoding". If the value given on this header is "BINARY", this is a CBF and the data will be presented as raw binary, containing a count (in the header described in 3.2.2 Format of CBF binary sections) so that we'll know when to start looking for more information.

If the value given for "Content-Transfer-Encoding" is one of the real encodings: "BASE64", "QUOTED-PRINTABLE", "X-BASE8", "X-BASE10" or "X-BASE16", the file is an imgCIF, and we'll need some other headers to process the encoded binary data properly. It is a good practice to give headers in all cases. The meanings of various encodings is given in the CBF extensions dictionary.

The "Content-Type" header tells us what sort of data we have (currently always "application/octet-stream" for a miscellaneous stream of binary data) and, optionally, the conversions that were applied to the original data. In this case we have compressed the data with the "CBF-PACKED" algorithm.

The "X-Binary-ID" header should contain the same value as was given for "_array_data.binary_id".

The "X-Binary-Size" header gives the expected size of the binary data. This is the size **after** any compressions, but before any ascii encodings. This is useful in making a simple check for a missing portion of this file. The 8 bytes for

the Compression type (see below) are not counted in this field, so the value of "X-Binary-Size" is 8 less than the quantity in bytes 12-19 of the raw binary data (3.2.2 Format of CBF binary sections).

The optional "Content-MD5" header provides a much more sophisticated check on the integrity of the binary data. Note that this check value is applied to the data occurring after the 8 bytes for the Compression type.

A blank line separator immediately precedes the start of the encoded binary data. Blank spaces may be added prior to the preceding "line separator" if desired (e.g. to force word or block alignment).

Because CBFLIB may jump foreward in the file from the MIME header, the length of encoded data cannot be greater than the value defined by "X-Binary-Size" (except when "X-Binary-Size" is zero, which means that the size is unknown). At exactly the byte following the full binary section as defined by the length value is the end of binary section identifier. This consists of the line-termination sequence followed by:

--CIF-BINARY-FORMAT-SECTION----
;

with each of these lines followed by a line-termination sequence. This brings us back into a normal CIF environment. This identifier is, in a sense, redundant because the binary data length value tells the a program how many bytes to jump over to the end of the binary data. This redundancy has been deliberately added for error checking, and for possible file recovery in the case of a corrupted file and this identifier must be present at the end of every block of binary data.

3.2.2 Format of CBF binary sections

In a CBF file, each binary section is encoded as a ;-delimited string, starting with an arbitrary number of pure-ASCII characters.

Note: For historical reasons, CIFlib has the option of writing simple header and footer sections: "START OF BINARY SECTION" at the start of a binary section and "END OF BINARY SECTION" at the end of a binary section, or writing MIME-type header and footer sections (3.2.1 Format of imgCIF binary sections). If the simple header is used, the actual ASCII text is ignored when the binary section is read. **Use of the simple binary header is deprecated.**

The MIME header is recommended.

Between the ASCII header and the actual CBF binary data is a series of bytes ("octets") to try to stop the listing of the header, bytes which define the binary identifier which should match the "binary_id" defined in the header, and bytes which define the length of the binary section.

Octet	Hex	Decimal	Purpose
1	0C	12	(ctrl-L) End of Page
2	1A	26	(ctrl-Z) Stop listings in MS-DOS
3	04	04	(Ctrl-D) Stop listings in UNIX
4	D5	213	Binary section begins
5..5+n-1			Binary data (n octets)

NOTE: When a MIME header is used, only bytes 5 through 5+n-1 are considered in computing the size and the message digest, and only these bytes are encoded for the equivalent imgCIF file using the indicated Content-Transfer-Encoding.

If no MIME header has been requested (a deprecated use), then bytes 5 through 28 are used for three 8-byte words to hold the binary_id, the size and the compression type:

5..12	Binary Section Identifier (See _array_data.binary_id) 64-bit, little endian	
13..20	The size (n) of the binary section in octets (i.e. the offset from octet 29 to the first byte following the data)	
21..28	Compression type: CBF_NONE CBF_CANONICAL CBF_PACKED CBF_BYTE_OFFSET (112) CBF_PREDICTOR	 0x0040 (64) 0x0050 (80) 0x0060 (96) 0x0070 (112) 0x0080 (128)

The binary data then follows in bytes 29 through 29+n-1.

The binary characters serve specific purposes:

- The Control-L (from-feed) will terminate printing of the current page on most operating systems.
- The Control-Z will stop the listing of the file on MS-DOS type operating systems.
- The Control-D will stop the listing of the file on Unix type operating systems.
- The unsigned byte value 213 (decimal) is binary 11010101. (Octal 325, and hexadecimal D5). This has the eighth bit set so can be used for error checking on 7-bit transmission. It is also asymmetric, but with the first bit also set in the case that the bit order could be reversed (which is not a known concern).
- (The carriage return, line-feed pair before the START_OF_BIN and other lines can also be used to check that the file has not been corrupted e.g. by being sent by ftp in ASCII mode.)

At present three compression schemes are implemented are defined: CBF_NONE (for no compression), CBF_CANONICAL (for and entropy-coding scheme based on the canonical-code algorithm described by Moffat, *et al*. (*International Journal of High Speed Electronics and Systems*, Vol 8, No 1 (1997) 179-231)) and CBF_PACKED for a CCP4-style packing scheme. Other compression schemes will be added to this list in the future.

For historical reasons, CBFlib can read or write a binary string without a MIME header. The structure of a binary string with simple headers is:

Byte value	**ASCII symbol Description**	**Decimal**	
1	;	59	Initial ; delimiter
2	carriage-return	13	
3	line-feed	10	The CBF new-line code is carriage-return, line-feed
4	S	83	
5	T	84	
6	A	65	
7	R	83	
8	T	84	
9		32	
10	O	79	
11	F	70	
12		32	
13	B	66	
14	I	73	
15	N	78	

16	A	65	
17	R	83	
18	Y	89	
19		32	
20	S	83	
21	E	69	
22	C	67	
23	T	84	
24	I	73	
25	O	79	
26	N	78	
27	carriage-return	13	
28	line-feed	10	
29	form-feed	12	
30	substitute	26	Stop the listing of the file in MS-DOS
31	end-of-transmission	4	Stop the listing of the file in unix
32		213	First non-ASCII value
33 .. 40			Binary section identifier (64-bit little-endien)
41 .. 48			Offset from byte 57 to the first ASCII character following the binary data
49 .. 56			Compression type
57 .. 57 + *n*-1			Binary data (*n*bytes)
57 + *n*	carriage-return	13	
58 + *n*	line-feed	10	
59 + *n*	E	69	
60 + *n*	N	78	
61 + *n*	D	68	
62 + *n*		32	
63 + *n*	O	79	
64 + *n*	F	70	
65 + *n*		32	
66 + *n*	B	66	
67 + *n*	I	73	
68 + *n*	N	78	
69 + *n*	A	65	
70 + *n*	R	83	
71 + *n*	Y	89	
72 + *n*		32	
73 + *n*	S	83	
74 + *n*	E	69	
75 + *n*	C	67	
76 + *n*	T	84	
77 + *n*	I	73	
78 + *n*	O	79	
79 + *n*	N	78	
80 + *n*	carriage-return	13	
81 + *n*	line-feed	10	
82 + *n*	;	59	Final ; delimiter

3.3 Compression schemes

Two schemes for lossless compression of integer arrays (such as images) have been implemented in this version of CBFlib:

1. An entropy-encoding scheme using canonical coding
2. A CCP4-style packing scheme.

Both encode the difference (or error) between the current element in the array and the prior element. Parameters required for more sophisticated predictors have been included in the compression functions and will be used in a future version of the library.

3.3.1 Canonical-code compression

The canonical-code compression scheme encodes errors in two ways: directly or indirectly. Errors are coded directly using a symbol corresponding to the error value. Errors are coded indirectly using a symbol for the number of bits in the (signed) error, followed by the error iteslf.

At the start of the compression, CBFlib constructs a table containing a set of symbols, one for each of the 2^n direct codes from $-2^{(n-1)}$ .. $2^{(n-1)}-1$, one for a stop code, and one for each of the *maxbits*-*n* indirect codes, where *n* is chosen at compress time and *maxbits* is the maximum number of bits in an error. CBFlib then assigns to each symbol a bit-code, using a shorter bit code for the more common symbols and a longer bit code for the less common symbols. The bit-code lengths are calculated using a Huffman-type algorithm, and the actual bit-codes are constructed using the canonical-code algorithm described by Moffat, *et al*. (*International Journal of High Speed Electronics and Systems*, Vol 8, No 1 (1997) 179-231).

The structure of the compressed data is:

Byte	Value
1 .. 8	Number of elements (64-bit little-endian number)
9 .. 16	Minimum element
17 .. 24	Maximum element
25 .. 32	(reserved for future use)
33	Number of bits directly coded, *n*
34	Maximum number of bits encoded, *maxbits*
35 .. 35+2^n-1	Number of bits in each direct code
35+2^n	Number of bits in the stop code
35+2^n+1 .. 35+2^n+*maxbits*-*n*	Number of bits in each indirect code
35+2^n+*maxbits*-*n*+1 ..	Coded data

3.3.2 CCP4-style compression

The CCP4-style compression writes the errors in blocks . Each block begins with a 6-bit code. The number of errors in the block is 2^n, where *n* is the value in bits 0 .. 2. Bits 3 .. 5 encode the number of bits in each error:

Value in bits 3 .. 5	Number of bits in each error
0	0
1	4
2	5
3	6
4	7

5	8
6	16
7	65

The structure of the compressed data is:

Byte	Value
1 .. 8	Number of elements (64-bit little-endian number)
9 .. 16	Minumum element (currently unused)
17 .. 24	Maximum element (currently unused)
25 .. 32	(reserved for future use)
33 ..	Coded data

4. Installation

CBFLIB should be built on a disk with at least 200 megabytes of free space. CBFlib_0.7.6.tar.gz is a "gzipped" tar of the code as it now stands. In addition, CBFlib_0.7.6_Data_Files.tar.gz is a "gzipped tar of the data files needed to test the API. Place both gzipped tars in the directory that is intended to contain two new directories, named CBFlib_0.7.6 (the "top-level" directory) and CBFlib_0.7.6_Data_Files. Uncompress both tarballs with gunzip and unpack them with tar:

gunzip CBFlib_0.7.6.tar.gz
tar xvf CBFLIB_0.7.6.tar
gunzip CBFlib_0.7.6_Data_Files.tar.gz
tar xvf CBFLIB_0.7.6_Data_Files.tar

As with prior releases, to run the test programs, you will also need Paul Ellis's sample MAR345 image, example.mar2300, and Chris Nielsen's sample ADSC Quantum 315 image, mb_LP_1_001.img as sample data. Both these files will be extracted by the Makefile from CBFlib_0.7.6_Data_Files. Do not download copies into the top level directory.

Makefile	Makefile for unix

and the subdirectories:

src/	CBFLIB source files
include/	CBFLIB header files
examples/	Example program source files
doc/	Documentation
lib/	Compiled CBFLIB library
bin/	Executable example programs
html_images/	JPEG images used in rendering the HTML files

For instructions on compiling and testing the library, go to the top-level directory and type:

make

The CBFLIB source and header files are in the "src" and "include" subdirectories. The files are:

src/	include/	Description
cbf.c	cbf.h	CBFLIB API functions

cbf_alloc.c	cbf_alloc.h	Memory allocation functions
cbf_ascii.c	cbf_ascii.h	Function for writing ASCII values
cbf_binary.c	cbf_binary.h	Functions for binary values
cbf_byte_offset.c	cbf_byte_offset.h	Byte-offset compression (not implemented)
cbf_canonical.c	cbf_canonical.h	Canonical-code compression
cbf_codes.c	cbf_codes.h	Encoding and message digest functions
cbf_compress.c	cbf_compress.h	General compression routines
cbf_context.c	cbf_context.h	Control of temporary files
cbf_file.c	cbf_file.h	File in/out functions
cbf_lex.c	cbf_lex.h	Lexical analyser
cbf_packed.c	cbf_packed.h	CCP4-style packing compression
cbf_predictor.c	cbf_predictor.h	Predictor-Huffman compression (not implemented)
cbf_read_binary.c	cbf_read_binary.h	Read binary headers
cbf_read_mime.c	cbf_read_mime.h	Read MIME-encoded binary sections
cbf_simple.c	cbf_simple.h	Hidher-level CBFlib functions
cbf_string.c	cbf_string.h	Case-insensitive string comparisons
cbf_stx.c	cbf_stx.h	Parser (generated from cbf.stx.y)
cbf_tree.c	cbf_tree.h	CBF tree-structure functions
cbf_uncompressed.c	cbf_uncompressed.h	Uncompressed binary sections
cbf_write.c	cbf_write.h	Functions for writing
cbf_write_binary.c	cbf_write_binary.h	Write binary sections
cbf.stx.y		bison grammar to define cbf_stx.c (see WARNING)
md5c.c	md5.h	RSA message digest software from mpack
	global.h	

In the "examples" subdirectory, there are 2 additional files used by the example programs (section 5) for reading MAR300, MAR345 or ADSC CCD images:

img.c img.h	Simple image library

and the example programs themselves:

makecbf.c	Make a CBF file from an image
img2cif.c	Make an imgCIF or CBF from an image
cif2cbf.c	Copy a CIF/CBF to a CIF/CBF
convert_image.c	Convert an image file to a cbf using a template file
cif2c.c	Convert a template cbf file into a function to produce the same template in an internal cbf data structure
testcell.C	Exercise the cell functions

as well as three template files: template_adscquantum4_2304x2304.cbf, template_mar345_2300x2300.cbf, and template_adscquantum315_3072x3072.cbf.

The documentation files are in the "doc" subdirectory:

CBFlib.html	This document (HTML)
CBFlib.txt	This document (ASCII)
CBFlib_NOTICES.html	Important NOTICES -- PLEASE READ
CBFlib_NOTICES.txt	Important NOTICES -- PLEASE READ
gpl.txt	GPL -- PLEASE READ
lgpl.txt	LGPL -- PLEASE READ

cbf_definition_rev.txt	Draft CBF/ImgCIF definition (ASCII)
cbf_definition_rev.html	Draft CBF/ImgCIF definition (HTML)
cif_img.html	CBF/ImgCIF extensions dictionary (HTML)
cif_img.dic	CBF/ImgCIF extensions dictionary (ASCII)
ChangeLog,html	Summary of change history (HTML)
ChangeLog	Summary of change history (ASCII)

5. Example programs

The example programs makecbf.c and img2cif.c read an image file from a MAR300, MAR345 or ADSC CCD detector and then uses CBFlib to convert it to CBF format (makecbf) or either imgCIF or CBF format (img2cif). makecbf writes the CBF-format image to disk, reads it in again, and then compares it to the original. img2cif just writes the desired file. makecbf works only from stated files on disk, so that random I/O can be used. img2cif includes code to process files from stdin and to stdout.

makecbf.c is a good example of how many of the CBFlib functions can be used. To compile makecbf and the other example programs use the Makefile in the top-level directory:

make all

This will place the programs in the bin directory.

To run makecbf with the example image, type:

./bin/makecbf example.mar2300 test.cbf

The program img2cif has the following command line interface:

```
img2cif     [-i  input_image]                                \
            [-o  output_cif]                                 \
            [-c  {p[acked]|c[annonical]|[n[one]}]            \
            [-m  {h[eaders]|n[oheaders]}]                    \
            [-d  {d[igest]|n[odigest]}]                      \
            [-e  {b[ase64]|q[uoted-printable]|               \
                  d[ecimal]|h[exadecimal]|o[ctal]|n[one]}]   \
            [-b  {f[orward]|b[ackwards]}]                    \
            [input_image] [output_cif]
```

the options are:

```
-i  input_image (default: stdin)
    the input_image file in MAR300, MAR345 or ADSC CCD detector
    format is given.  If no input_image file is specified or is
    given as "-", an image is copied from stdin to a temporary file.

-o  output_cif (default: stdout)
    the output cif (if base64 or quoted-printable encoding is used)
    or cbf (if no encoding is used).  if no output_cif is specified
    or is given as "-", the output is written to stdout

-c  compression_scheme (packed, canonical or none, default packed)

-m  [no]headers (default headers for cifs, noheaders for cbfs)
    selects MIME (N. Freed, N. Borenstein, RFC 2045, November 1996)
```

```
    headers within binary data value text fields.

-d  [no]digest  (default md5 digest [R. Rivest, RFC 1321, April
    1992 using"RSA Data Security, Inc. MD5 Message-Digest
    Algorithm"] when MIME headers are selected)

-e  encoding (base64, quoted-printable, decimal, hexadecimal,
    octal or none, default: base64) specifies one of the standard
    MIME encodings (base64 or quoted-printable) or a non-standard
    decimal, hexamdecimal or octal encoding for an ascii cif
    or "none" for a binary cbf

-b  direction (forward or backwards, default: backwards)
    specifies the direction of mapping of bytes into words
    for decimal, hexadecimal or octal output, marked by '>' for
    forward or '<' for backwards as the second character of each
    line of output, and in '#' comment lines.
```

The test program **cif2cbf** uses the many of the same command line options as img2cif, but accepts either a CIF or a CBF as input instead of an image file:

```
cif2cbf [-i input_cif] [-o output_cbf]                    \
  [-c {p[acked]|c[annonical]|[n[one]}] \
  [-m {h[eaders]|n[oheaders]}] [-d {d[igest]|n[odigest]}] \
  [-e {b[ase64]|q[uoted-printable]| \
               d[ecimal]|h[exadecimal]|o[ctal]|n[one]}] \
  [-b {f[orward]|b[ackwards]}] \
  [-v dictionary]* [-w] \
  [input_cif] [output_cbf]
```

the options are:

```
-i input_cif (default: stdin)
  the input  file in CIF or CBF  format.  If input_cif is not
  specified or is given as "-", it is copied from stdin to a
  temporary file.

-o output_cbf (default: stdout)
  the output cif (if base64 or quoted-printable encoding is used)
  or cbf (if no encoding is used).  if no output_cif is specified
  or is given as "-", the output is written to stdout
  if the output_cbf is /dev/null, no output is written.
```

The remaining options specify the characteristics of the output cbf. The characteristics of the input cif are derived from context.

```
-c compression_scheme (packed, canonical or none,
  default packed)

-m [no]headers (default headers for cifs, noheaders for cbfs)
  selects MIME (N. Freed, N. Borenstein, RFC 2045, November 1996)
  headers within binary data value text fields.

-d [no]digest  (default md5 digest [R. Rivest, RFC 1321, April
  1992 using"RSA Data Security, Inc. MD5 Message-Digest
```

```
  Algorithm"] when MIME headers are selected)

-e encoding (base64, quoted-printable or none, default base64)
  specifies one of the standard MIME encodings for an ascii cif
  or "none" for a binary cbf

-v dictionary specifies a dictionary to be used to validate
  the input cif and to apply aliases to the output cif.
  This option may be specified multiple times, with dictionaries
  layered in the order given.

-w process wide (2048 character) lines
```

The program **convert_image** requires two arguments: *imagefile* and *cbffile*. Those are the primary input and out. The detector type is extracted from the image file, converted to lower case and used to construct the name of a template cbf file to use for the copy. The template file name is of the form template_*name_columns*x*rows*. The full set of options is:

```
convert_image [-i input_img] [-o output_cbf] [-p template_cbf]  \
  [-d detector name]  -m [x|y|x=y] [-z distance]                 \
  [-c category_alias=category_root]*                            \
  [-t tag_alias=tag_root]*                                      \
  [input_img] [output_cbf]
```

the options are:

```
-i input_img (default: stdin)
  the input file as an image in smv, mar300, or mar345  format.
  If input_img is not specified or is given as "-", it is copied
  from stdin to a temporary file.

-p template_cbf
  the template for the final cbf to be produced.  If template_cbf
  is not specified the name is constructed from the first token
  of the detector name and the image size as
     template__x.cbf

-o output_cbf (default: stdout )
  the output cbf combining the image and the template.  If the
  output_cbf is not specified or is given as "-", it is written
  to stdout.

-d detectorname
  a detector name to be used if none is provided in the image
  header.

-m [x|y|x=y] (default x=y, square arrays only)
  mirror the array in the x-axis (y -> -y)
                  in the y-axis (x -> -x)
               or in x=y ( x -> y, y-> x)

-r n
  rotate the array n times 90 degrees counter clockwise
  x -> y, y -> -x for each rotation, n = 1, 2 or 3
```

```
-z distance
  detector distance along Z-axis

-c category_alias=category_root
-t tag_alias=tagroot
  map the given alias to the given root, so that instead
  of outputting the alias, the root will be presented in the
  output cbf instead.  These options may be repeated as many
  times as needed.
```

Updated 18 July 2006. Contact: yaya@bernstein-plus-sons.com