C 版 NetCDF ユーザマニュアル

自己記述的アクセスインターフェース , ポータブルデータ

Version 3

May 1999

Russ Rew, Glenn Davis, Steve Emmerson, and Harvey Davies
Unidata Program Center

Unidata Program Center は University Corporation for Atmospheric Research によって運営され、National Science Foundation による補助を受けています。この作品中に示されている見解・発見・結論・推奨等は著者のものであり、必ずしも National Science Foundation の見解・発見・結論・推奨等を反映しているとは限りません。

序文

Unidata（http://www.unidata.ucar.edu）は National Science Foundation がスポンサーしている計画で、全米の大学にコンピューター及びネットワークの革新的な使用方法を提供することによって、大気及び大気関連のデータを最大限に利用し教育・研究に活かすための強力な武器を与えています。そのようなデータを解析・表示するにあたって Unidata Program Center は University of Wisconsin, Purdue University, NASA, and the National Weather Service 等を含む他団体が開発したソフトウェアパッケージを大学側に提供しています。これらのソフトウェアに共通していることはデータをリアルタイムで取得し管理する Unidata が開発したシステムを使用していることです。このことによって Unidata の主張でもある、各地域に必要とされている各大学によるデータベースの独自取得・自己管理を実現できました。重要なのは Unidata 計画がデータセンターを有しないことです。データ管理は「分担」される任務であるべきなのです。

このマニュアル中で紹介されている Network Common Data Form (NetCDF) ソフトウェアは本来、数ある Unidata のアプリケーション用に共通のデータアクセス方法を提供する目的で開発されました。これらは定点観測・時系列・等間隔格子・衛星やレーダー観測等の様々なデータの種類を網羅しています。

NetCDF ソフトウェアは I/O ライブラリとして機能し、 C・FORTRAN・C++・ Perl 等の NetCDF が存在する全ての言語から呼び出し可能です。このライブラリは自己記述的マシン独立型のデータベースにデータを格納・引出します。個々の NetCDF ファイルは多次元の定義された変数（整数・実数・文字・バイト等の複数の種類を含む）を含むことが可能で、さらに各々の変数に従属的なデータ（単位・説明文など）を付随させることが出来ます。このインターフェースは既存の NetCDF ファイルに既定された方法でデータを追加することが出来、機能的には（固定長の）記録方式と類似しているところもあります。しかしながら、NetCDF ライブラリでは変数名・インデックスによってのみデータの直接アクセス格納や引出が可能であり、ディスク（もしくはメモリ）保存型のファイルにのみ適応することが出来ます。

Unidata のソフトウェアの半分ほどは既に NetCDF アクセス可能になっており、以後、残りの Unidata のアプリケーションについても同様の共有性を持たせる予定です。それによって次のことが可能になります。

- 異なるアプリケーションによる同一ファイルの共有。
- 異なるコンピュータ間で透過的な（変換されていない）ステータスでのファイルの転送、もしくは共有。
- フォーマットの違いに対応するために必要とされるプログラミングの時間の短縮。
- データ、または従属的なデータの誤った解釈を防ぐ。
- あるアプリケーションからの出力データを別のアプリケーションの入力データとして簡単に使用できる。
- Unidata システムに新たなソフトウェアを導入する作業を容易にする標準を設ける。

NetCDF は既にいくらか成功を収めています。現在では NetCDF は CRAY からパーソナルコンピュータ、そしてほとんどの UNIX ワークステーションを含むコンピュータのプラットフォームとして幅広く使用されています。NetCDF を使ってあるコンピュータ上

で（例えばFORTRANで）複雑なファイルを作成し、その同じ自己記述的なファイルを他のコンピュータ上（例えばC）で 一切の変換なしで引き出すことが出来ます。NetCDFのファイルはネットワーク経由で転送したり、適切なネットワークファイルシステムを使用することによりリモート・アクセスすることも可能です。

Unidataソフトウェア以外のソフトウェアにおいてNetCDFアクセスを可能にすることはUnidataの支持層の利益に繋がると信じ、NetCDFライブラリをライセンスや重大な規制無く配布し、最新のバージョンをanonymous FTP経由で手に入れられるようにしてあります。このように自由に使用することを許可することによりUnidataの情報を解析・表示する手段のオプションが豊富になることと思われます。Unidataのソフトウェアは大気科学コミュニティー以外でも幅広く受け入れられているようで、現在では数多くのパブリックドメインや商業用データ解析システムがNetCDFファイルを読みこむことが出来ます。

いくつかの組織ではNetCDFはデータ・アクセス法の標準として採用されており、NCSA (National Center for Supercomputer Applications; University of Illinois at Urbana-Champaignと提携している）ではHDFファイル形式（NCSAで使用されているツールがNetCDFプログラミング・インターフェースを支持する動きもあります。我々はこれらの動きを支持し、協力してきました。

NetCDFのソフトウェアがどれほどサポートされているのかという疑問が時々寄せられます。Unidataの正式な立場はNetCDFライブラリに添付されている著作権に関する事項にも述べられておりますが、ソフトウェアは全て‘as is(無保証)’のステータスで提供されているというものです。実際には、ソフトウェアは随時アップデートされていくものなので、Unidataは当面、ソフトウェアを改良しつづける予定であります。Unidataの目的は米国の地球科学者をサポートすることであるので、それらの学会・団体より寄せられた問題点が最優先されることをご了承下さい。

ユーザーの皆様がこのソフトウェアを重宝して下さり、活用法に関するフィードバックや改良点に関する提案を返してくだされば光栄に存じます。

David Fulker

Unidata Program Center Director

University Corporation for Atmospheric Research

概要

Network Common Data Form (NetCDF) インターフェースのの 目的は 配列指向型のデータを自己記述的でかつポータブルなフォマーットで作成・アクセス・共有することにあります。「自己記述的」とはそのファイルが自身に含まれるデータに関する情報を内包しているという意味です。「ポータブル」とはファイル内のデータが整数・文字・浮動小数点の格納方式が異なるコンピュータ間でやり取りできるということです。NetCDF インターフェースを使用して作った新しいファイルは、即、「ポータブル」になります。データアクセス・管理・解析・表示するソフトウェアに NetCDF インターフェースを使用することにより、より有用なソフトウェアを作ることが出来ます。

NetCDF のソフトウェアには NetCDF データアクセス用に C と FORTRAN のインターフェースを搭載しています。共通のプラットフォーム用にこのようなライブラリは用意されております。

NetCDF データアクセス用の C++ と Perl のインターフェース も Unidata により提供されています。NetCDF ユーザのご助力によりそのほかのプラットフォームや他のプログラム言語用のソフトウェア・ポート もあります。配列指向型のデータやソフトウェアを共有し、より価値のあるファイルを作成することを目的に、NetCDF のソフトウェア・ライブラリのソースコードは無料で配布されています。

このユーザー・ガイドは NetCDF データモデルの紹介ですが、C のインターフェースのみで表示できます。他の言語のインターフェースリンクについては NetCDF World Wide Web Site `http://www.unidata.ucar.edu/packages/NetCDF/` をご参照下さい。C, FORTRAN, C++ and Perl 用の表示文書がオンラインであります。同じサイトに UNIX システム用の参照文書も C と FORTRAN のインターフェース用に UNIX ‘man’ ページの形式であります。NetCDF World Wide Web Site には他にも NetCDF に関する膨大な情報と NetCDF データを使用できるソフトウェアへのポインタも掲載されています。

1　イントロダクション

1.1　NetCDF インターフェース

Network Common Data Form、すなわち NetCDF、は配列形式のデータを格納・引き出すためのデータアクセス関数ライブラリへのインターフェースです。*配列*とは n 次元（n は 0,1,2,…）の矩形構造を持ち、その要素が全て同じデータ型（(例) 8 ビット文字、32 ビット整数）のものを指します。*スカラー*（単純な一つの値）は 0 次元の配列です。

NetCDF はデータとは自己記述的でポータブルなオブジェクトの集合体であり、簡単なインターフェースを通じて引出し可能であるべきであるという見方を支持する抽象概念です。配列値はデータの格納方式に関する事前の知識無しに直接アクセスできます。データに関する補助的な情報（例えば単位等）はデータと伴に格納できます。NetCDF のデータベースは一般的なユーティリティやアプリケーションプログラムを使用してアクセスでき、データの特定フィールドを変換・統合・解析・表示することが可能です。そのようなアプリケーションの開発はデータの有用性を向上させ、又、配列指向型のデータの管理・解析・表示を行うソフトウェアの再利用性の向上に繋がるでしょう。

NetCDF ソフトウェアは*抽象的データ型*を利用します。これは NetCDF ファイル内のデータにアクセス・操作する命令は全てインターフェースによって提供されている関数のみを使わなければならないということです。データの表現はインターフェースを使うアプリケーションからは隠されており、データの格納方式は既存のプログラムに影響を及ぼすことなく変更できます。NetCDF データの物理的な表現方法はデータが作成されたコンピュータから独立しているように設計されています。

Unidata は C・FORTRAN・C++・Perl・色々な UNIX OS のための NetCDF インターフェースを サポートしています。このソフトウェアは各メジャーリリース前に、他数種類の OS 用にこれらの OS のユーザーの皆様のご助力で移植テストをされています。 Unidata の NetCDF ソフトウェアは幅広い利用を促進するために FTP を通じて無料で配布されております。

1.2　NetCDF はデータベース管理システムではありません

何故、配列指向型のデータ格納に関して既存のデータベース管理システムより NetCDF の方が優れているのでしょうか？それはリレーショナルデータベースソフトウェアが NetCDF インターフェースがサポートするデータアクセス法に適していないからです。

まず、既存の関係モデルをサポートするデータベースシステムは データアクセスの基本単位として多次元のオブジェクト（配列）をサポートしていません。配列を関係として表示することは便利なデータアクセス法を不便にし、又、多次元データや座標系の抽象化に対してはほとんど何のサポートもしていません。配列指向型データを引出・修正・数学的に扱い・表示するためにはまったく異なるデータモデルが必要なのです。

これに関連し、汎用的なデータベースシステムに関する 2 番目に大きな問題があります。大きな配列に対するパフォーマンスの悪さです。衛星写真・科学的モデルの結果・長期的な全地球気象観測のデータなどを集積を効率的に引出せるように系統立て索引をつけることは既存のデータベースシステムの能力を超えています。

最後に、汎用的なデータベースシステムは資源面でもアクセスパフォーマンス面でも多大な犠牲の元に、配列指向型のデータを解析・管理・表示するためには不必要な機能を提供しています。例えば、精巧なアップデート機能・履歴検査・報告書のフォーマット・業務処理用の機能など科学的な操作には不必要なものばかりです。

1.3 File Format

ネットワーク透過性（マシン独立性）を達成するために、NetCDF はデータの表現・コード化のための標準プロトコルである XDR（eXternal Data Representation；ftp://ds.internic.net/rfc/rfc1832.txt 参照）に似た外部表現機能を利用します。この表現機能はデータをマシン独立型のビット列へとコード化します。これは 8 ビットのバイトのみが一貫してコード化されるという前提のみにて、多種類のコンピューター上で既に実装されています。IEEE 754 浮動小数点標準プロトコルが浮動小数点のデータを表現するのに使用されています。

NetCDF ファイルののおおまかな構造の説明は 9 章 「NetCDF ファイルの構造と性能」(p.101) にあります。

ファイル形式の詳細については Appendix A 「ファイルフォーマット仕様」 p.121 を参照してください。ただし、ファイル形式を指定した形で NetCDF ファイルを読み取り・作成する独自の低レベルソフトウェアを開発することは好ましくありません。後に、フォーマットが更新された時に互換性に問題が生じる危険性があります。

1.4 パフォーマンスは？

NetCDF の目的の一つは大きなファイルの部分集合へのアクセスを効率的に行うことであります。この目的のために、NetCDF は順次アクセスではなく直接アクセスを行います。その方がデータが作成された順番と異なる順序で読み取られる場合や異なるアプリケーションによって読み取られる順番が異なる場合に有効です。

ポータブルな外部表現機能（XDR）に必要なオーバーヘッドの量は多くの要素に左右されます。例えばデータの種類・コンピュータの種類・データアクセスの粒度・コンピュータに実装された XDR がのチューニング等の要素に依存します。通常の場合、オーバヘッドはアプリケーションが使用する全リソース量に比べると小さいため、いずれの場合にも、XDR レイヤーにかかるオーバヘッドはデータのポータブルアクセスの利便性を考えるとたいした犠牲ではありません。

NetCDF を設計・実装するにあたってデータアクセスの効率は重大な要素でした。しか

しながら、NetCDF インターフェースを非効率的に利用することは不可能ではありません。例えば、各記録から一つの値を要求するようなデータ摘出を行う場合などがそれにあたります。効率的にインターフェースを利用する方法に付いては 9 章 「NetCDF ファイルの構造と性能」（p.101）をご参照下さい。

1.5 NetCDF は良いアーカイブフォーマットですか？

NetCDF は配列を格納するための汎用的なアーカイブフォーマットとして使用できます。NetCDF におけるデータ圧縮は（低解像度の浮動小数点数を 32 ビットの配列で表す代わりに 8 ビットもしくは 16 ビットの整数配列を使用することにより）可能です。しかし、NetCDF の現行版はデータ圧縮率を最適にする設計にはなっていません。それ故、NetCDF は特定のデータベースのある特徴を生かした特殊目的用アーカイブフォーマットよりも多くのスペースを必要とするかもしれません。

1.6 規約に従った自己記述的データの作成法

NetCDF を使うことが、即、人間とマシンにとって意味のある「自己記述的」データを作成することと等価ではありません。変数や次元の名前は意味のあるものを用い、存在する規約に従った形を取るべきです。次元に関しては（意味があると思われる場合には）対応する座標変数も与えるべきです。

属性は従属的な情報を供給する上で大変重要です。関連する規約に従い、対応する標準属性を使用することが大切です。8.1 節 「属性の規約」（p.86）に一般的なアプリケーションソフトウェアのための NetCDF ライブラリ専用の属性やその規約が記述されています。

いくつかの団体は NetCDF データ用に独自のコンベンション（付加的規約）やスタイルを定義しています。これらの規約やそれらの利用法については NetCDF Conventions site, `http://www.unidata.ucar.edu/packages/NetCDF/conventions.html` を参照してください。

上記の規約は都合の良い場合には使用すべきです。ローカルな使用のためにはしばしば付加的な規約が必要とされます。このような規約を敷く場合には、関連分野のユーザのためにも上記の NetCDF conventions site に掲載しておくことが望ましいです。

1.7 NetCDF の背景と発展

NetCDF の 開発は Unidata の必要に迫られたしごく控えめな目標に向かって始められました。その目標とは Unidata のアプリケーションとリアルタイムの気象データとの間に共通のインターフェースを 提供することです。元々 Unidata のソフトウェアは複数のハードウェアプラットフォーム上で実行され、C と FORTRAN の両方からアクセスされることが前提にあったので、Unidata の目標を達成することはより広く応用できるパッケージを提供する可能性をも秘めていました。これらのパッケージを広く提供し、かつ同じような需要のある団体と協力することによって、我々は 科学的なデータを取得す

るために作られたあるソフトウェアが他の分野ばかりではなく同じ分野の中でさえも利用されない現状を打破しようと試みました (Fulker, 1988)。

NetCDF ソフトウェアの重要な コンセプトは NASA Goddard National Space Science Data Center (NSSDC) で開発されたデータアクセスソフトウェアの解説である論文、Treinish and Gough (1987) に記述されています。このソフトウェアによって提供されているインターフェースは Common Data Format (CDF) と呼ばれ、 NASA CDF は元は配列を格納するための抽象化をサポートするプラットフォーム特定型の FORTRAN ライブラリとして開発されました。

NASA CDF パッケージは様々な種類のデータと幅広いアプリケーションに応用されてきました。NASA CDF は単純さ (サブルーチンは 13 個のみ)・格納フォーマットからの独立性・汎用性・データの論理的な見方をサポートする能力・一般的なアプリケーションに対するサポートという利点を備えていました。

1987 年の 8 月に Unidata はコロラド州ボルダーで ワークショップが開催されました。NASA と協力し、NASA の既存のインターフェースと互換性を持たせながら CDF FORTRAN インターフェースを拡張・C インターフェースを定義・一つのセルによるデータ集合体のアクセス許可をする可能性が追求されました。

それとは独自に New Mexico Institute of Mining and Technology の Dave Raymond は UNIX 用にある C ソフトウェアのパッケージを開発していました。それは自己記述的データへの順次アクセスを可能にし、データの解析・分析・表示に対して「パイプとフィルター (又はデータフロー)」的なアプローチをサポートするものでした。このパッケージもまた,「Common Data Format」という名を冠しており、後に C-Based Analysis and Display System (CANDIS) へと改められました。Unidata は Raymond の成果を知り (Raymond, 1988)、名前付き次元、及び同一データオブジェクト内に形の異なる変数を使用するなどといった、彼の着眼点のいくつかを Unidata NetCDF インターフェースに起用しました。

1988 年の初頭に Unidata の Glenn Davis が C で書かれ XDR の上に被さった NetCDF パッケージの試作品を完成させました。この試作品は次の 2 点を証明しました。ひとつは単一ファイルの XDR 上に実装された CDF インターフェースの開発費用が許容内であること。そして 2 点目はそのようなプログラムが UNIX と VMS との両方に実装可能であることでした。同時にそれは、小さく、ポータブルで NASA CDF と互換性のある FORTRAN インターフェースが望まれている汎用性を持ち得ないことも証明しました。 NASA CDF と Unidata's NetCDF とはその後独自の発展を遂げましたが、NASA CDF の最新版は NetCDF と似たような特徴を持っています。

1988 年の初頭に、1987 年の Unidata CDF ワークショップにも参加した SeaSpace, Inc.(カリフォルニア州サンディエゴにある商用ソフトウェア開発会) の Joe Fahle が独自に NASA CDF インターフェースをいくつか重要な点で拡張した CDF パッケージを C で開発しました (Fahle, 1989)。Raymond のパッケージと同様に、SeaSpace CDF ソフトウェアは関連の無い形の変数を同一データオブジェクト内に含むことを許容し、多次元の配列に対する一般的なアクセス方法を可能にしました。Fahle の成果は SeaSpace 社では、画像処理システムに おける中間的な段階での格納形態として使われていました。このインターフェースとフォーマットは後に Terascan データフォーマットへと発展し

ていきます。

Fahle のインターフェースは NASA のインターフェースを我々の目的に応じる形に拡張しようとした際に直面した問題の大部分を解決していました。1988 年 8 月に Unidata NetCDF 用インターフェースの形式を決定し、残された問題を解決するために小規模のワークショップが開催されました。参加者は SeaSpace 社の Joe Fahle 、Apple 社の Michael Gough (NASA CDF ソフトウェアの開発者の一人)、Miami 大学の Angel Li (VMS に NetCDF ソフトウェアの試作品を実装し、ユーザー候補である人)、それに Unidata のシステム開発部のスタッフ達でした。いくつか簡略できる点が指摘された後にワークショップとしての合意が得られました。Glenn Davis と Russ Rew がソフトウェアの最初のバージョンを完成させる前に、ワークショップの成果を含んだ Unidata NetCDF インターフェースの仕様に関する文書が意見交換を促すために広く配布されました。他のデータアクセスインターフェースとの比較や NetCDF を使用した感想に付いては Rew and Davis (1990a)、Rew and Davis (1990b)、Jenter and Signell (1992)、and Brown, Folk, Goucher, and Rew (1993) で議論されています。

1991 年 10 月に NetCDF ソフトウェア 2.0 版の配布開始を発表しました。C インターフェースに小さな修正を加えた（次元の長さを `int` ではなく `long` で宣言した）ことによって MS-DOS コンピューター等の安価なプラットフォーム上での NetCDF の利便性を向上させました。さらに他のプラットフォーム上での再コンパイル作業を必要としないという利点もありました。このインターフェースへの変更は関連するファイルフォーマットの変更が必要となることもありませんでした。

1993 年 6 月に NetCDF 2.3 版がリリースされました。このバージョンではファイルフォーマットに変更はなされませんでしたが、記録への単一呼び出しアクセス・不連続なデータに関する断面へのアクセスの最適化・'stride' を使用した指定断面への部分サンプリング・'mapped array sections'（マップされた配列断面）を使用した不連続データへのアクセス・ ncdump と ncgen ユーティリティの改良・試験的な C++ インターフェース等が追加されました。

1996 年 2 月にリリースされた 2.4 版では新たなプラットフォームや C++ インターフェースへのサポートが加えられ、又、スーパーコンピューターのアーキテクチャに関しては重要な最適化がなされました。

1996 年 5 月に NetCDF データに高レベルなインターフェースを提供するソフトウェアの FAN (File Array Notation) の配布が開始された。FAN のユーティリティーには NetCDF のデータセットから配列指向データを抽出し操作する・NetCDF 配列から特定のデータを印刷する・ASCII データを NetCDF データにコピーする・NetCDF 配列上で様々な統計操作 (sum, mean, max, min, product,…) を行う等が含まれました。FAN に関する詳細は FAN Utilities document, `http://www.unidata.ucar.edu/packages/NetCDF/fan_utils.html` にあります。

1.8 過去のリリースから何が新しくなったか？

このガイドは 1997 年 1 月にリリースされた netCDF 3 の説明文です。NetCDF 3 版は過

去のバージョンと同じファイルフォーマットを使用しますが、2.4 版に比べていくつかの大きな変更がなされています。

- NetCDF ライブラリの ANSI C での完全な再記述
- 新しい type-safe C と FORTRAN のインターフェース
- 自動型変換機能
- 内部アーキテクチャの重大な変更による新しいプラットフォーム上での高パーフォマンス化と容易な最適化
- NetCDF 2 関数インターフェース・グローバル変数・後方互換性の全てに対するサポート
- 文書の修正、及び報告されたバグに対する修正

1.9 NetCDF の制限

NetCDF データモデルは名前付き属性を持つ名前付き配列変数の集合として系統だてられるデータに関しては広く応用が利きます。しかしながらこのモデルとソフトウェアの実装にはいくつかの重要な制限があります。この制限の一部は NetCDF が包含する要求の中で相反するものに対するトレードオフに内在するものであります。他の制限に関しては次のバージョンにて対応していく予定です。

現在 NetCDF で使用できる外部数値データ種は 8-、16-、32- ビットの整数、32- もしくは 64- ビットの浮動小数点数 に限られています。これらの限られたサイズはビットフィールドにデータを格納することに比べるとファイルスペースを無駄に使用する可能性があります。例えば、9- ビットの数値の配列は 16- ビットの短い整数として格納しなければなりません。1-、2- ビット長の数値を 8- ビット長の値として格納するのは更に無駄が多くなります。

現行の NetCDF ファイルフォーマットでは一つの NetCDF ファイルに格納できるデータは 2 ギガバイトです。この制約はファイル内の配置格納のために 32 ビットオフセットを使用しているために生じています。

現行のモデルの制約の一つに各 NetCDF ファイルに対して無制限の（可変の）次元が一つしか使用できないことです。無制限の次元においては複数の変数を共有することが可能ですが、それらの変数は同時に発展しなければなりません。これによって NetCDF モデルでは同一ファイル内において複数の無制限次元を持つ変数を扱ったり異なる変数に複数の無制限次元を持たせることができません。つまり、NetCDF モデルは矩型でない変数（例えば不揃いな配列）の表現には不向きということになります。

データの完全な自己表現性にも限界があります。実際にデータを共有したり格納したりする際には必ずと言って良いほど既存の決まり事が存在します。NetCDF では変数・次元・属性に意味のある名前、計算する際に使用可能な形態の単位、ファイル全体に関する属性値のテキスト文字列、簡単な座標系に関する情報を格納出来ます。しかし、より複雑なメタデータ（例えば一般的ではないグリッド上に正確に地球座標系のデータを投影したり衛星からの映像を正確に表現するために必要な情報等）に対応するためには規約を敷く必要がでてきます。

NetCDF データモデルに適切な修正を加えることによりこれらの規約が不必要になったり、メタデータの幾つかの種類を統一的かつコンパクトな方法で表現できるようになるかもしれません。例えば、NetCDF データモデルに明確な 地球座標系を与えることによって複雑な地球座標系の規約を簡易化することは可能ですが、データモデルが複雑になるという弊害があります。ここで問題となるのはモデルの豊かさと汎用性（多種多様なデータを扱える能力）との間に適切なトレードオフ地点を見つけることです。ある特定の分野の研究者同士が共有する概念を表すためだけに作られたデータモデルは複数の分野でデータを共有したり統合したりすることには不向きかもしれません。

NetCDF データモデルはツリー・ネスト配列・ 循環的なデータ等のネスト型配列構造をサポートしていません。その最たる理由は現行の FORTRAN インターフェースによって任意の NetCDF ファイルを書き込み読み取れなければならないからです。複雑な表現方法や規約によって、幾つかのネスト型構造を表現することは可能ですが、その結果、NetCDF の目標である自己記述的データではなくなってしまう可能性があります。

最後に、現行の実装では NetCDF ファイルへの同時アクセスは制限されています。一つのファイルは同時に複数の人が読み取ることが出来ますが、書き込める人は一人に限られており、複数人による同時書き込みはサポートされていません。

1.10 NetCDF の将来計画

現時点における計画では 透過的なデータパッキングの追加、同時アクセスのサポートの向上、2 ギガバイト以上のファイルへのアクセス機能です。他にも実現可能であれば加えられる可能性のある拡張機能としてキーもしくは座標値によるデータアクセス、効率的な構造変更（例えば、新しい変数や属性の追加・変更等）、別のファイルのデータ断面へのポインタ機能、ネスト型配列（不調和配列・ツリー配列・循環型配列の表現の実現）への対応、複数の無制限次元の導入などです。

References

1. Brown, S. A, M. Folk, G. Goucher, and R. Rew, “Software for Portable Scientific Data Management,” *Computers in Physics*, American Institute of Physics, Vol. 7, No. 3, May/June 1993.
2. Davies, H. L., “FAN - An array-oriented query language,” Second Workshop on Database Issues for Data Visualization (Visualization 1995), Atlanta, Georgia, IEEE, October 1995.
3. Fahle, J., *TeraScan Applications Programming Interface*, SeaSpace, San Diego, California, 1989.
4. Fulker, D. W., “The NetCDF: Self-Describing, Portable Files---a Basis for ‘Plug-Compatible’ Software Modules Connectable by Networks,” *ICSU Workshop on Geophysical Informatics*, Moscow, USSR, August 1988.
5. Fulker, D. W., “Unidata Strawman for Storing Earth-Referencing Data,” *Seventh International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, New Orleans, La., American Meteorology Society, January 1991.
6. Gough, M. L., *NSSDC CDF Implementer’s Guide (DEC VAX/VMS) Version 1.1*, National Space Science Data Center, 88-17, NASA/Goddard Space Flight Center, 1988.

7. Jenter, H. L. and R. P. Signell, “NetCDF: A Freely-Available Software-Solution to Data-Access Problems for Numerical Modelers,” *Proceedings of the American Society of Civil Engineers Conference on Estuarine and Coastal Modeling*, Tampa, Florida, 1992.
8. Raymond, D. J., “A C Language-Based Modular System for Analyzing and Displaying Gridded Numerical Data,” *Journal of Atmospheric and Oceanic Technology*, **5**, 501-511, 1988.
9. Rew, R. K. and G. P. Davis, “The Unidata NetCDF: Software for Scientific Data Access,” *Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, Anaheim, California, American Meteorology Society, February 1990.
10. Rew, R. K. and G. P. Davis, “NetCDF: An Interface for Scientific Data Access,” *Computer Graphics and Applications*, IEEE, pp. 76-82, July 1990.
11. Rew, R. K. and G. P. Davis, “Unidata’s NetCDF Interface for Data Access: Status and Plans,” *Thirteenth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, Anaheim, California, American Meteorology Society, February 1997.
12. Treinish, L. A. and M. L. Gough, “A Software Package for the Data Independent Management of Multi-Dimensional Data,” *EOS Transactions*, American Geophysical Union, **68**, 633-635, 1987.

2　NetCDF ファイルの構成

2.1 NetCDF データモデル

NetCDF ファイルは *dimensions*(*次元*)、*variables*(*変数*)、*attributes*（*属性*）等の情報を含み、全てに固有の名と ID 番号が割り振られています。データの意味や配列指向のデータフィールド間の関係をを把握するためにこれらの構成成分を同時に使用することが出来ます。NetCDF ライブラリでは通常のファイル名のみでなく ID 番号によっても指定される複数の NetCDF ファイルに同時にアクセス可能です。

NetCDF ファイルには記号テーブルが存在し、変数の名・データ型・ランク（次元数)・次元・開始ディスクアドレス等の情報が記載されています。個々の要素はその ID を表す配列インデックス（subscript、添字）の一次関数であるディスクアドレスに記憶されています。つまり、これらの索引を別々に保存する必要が無く（この点で関係データベースと異なる)、素早くコンパクトな記憶法である。

2.1.1 規約の命名

次元、変数、属性の名はローマ字もしくはアンダースコアで始まる任意のローマ字と数字で構成されている文字列（アンダースコア'_'、ハイフン'-'を含む）で表されます。(ただし、アンダースコアで始まる名はシステム用にのみ使用します。)

2.1.2 Network Common Data Form Language (CDL)

ここで簡単な NetCDF の例を使い、NetCDF データモデルの原理を説明します。このデータには次元も変数も属性も含まれています。この簡単な NetCDF オブジェクトの表記は CDL (network Common Data form Language) と呼ばれ、NetCDF ファイルを表記するのに大変適しています。NetCDF システムにはバイナリの NetCDF ファイルから人間指向の CDL テキストファイルを作成する、及び逆の操作を行うためのユーティリティが含まれています。

```
NetCDF example_1 {  // example of CDL notation for a NetCDF dataset

dimensions:         // dimension names and lengths are declared first
        lat = 5, lon = 10, level = 4, time = unlimited;

variables:          // variable types, names, shapes, attributes
        float   temp(time,level,lat,lon);
                    temp:long_name     = "temperature";
                    temp:units         = "celsius";
        float   rh(time,lat,lon);
                    rh:long_name = "relative humidity";
                    rh:valid_range = 0.0, 1.0;      // min and max
        int     lat(lat), lon(lon), level(level);
                    lat:units       = "degrees_north";
                    lon:units       = "degrees_east";
```

```
                level:units     = "millibars";
        short   time(time);
                time:units      = "hours since 1996-1-1";
        // global attributes
                :source = "Fictional Model Output";

data:                // optional data assignments
        level   = 1000, 850, 700, 500;
        lat     = 20, 30, 40, 50, 60;
        lon     = -160,-140,-118,-96,-84,-52,-45,-35,-25,-15;
        time    = 12;
        rh      =.5,.2,.4,.2,.3,.2,.4,.5,.6,.7,
                 .1,.3,.1,.1,.1,.1,.5,.7,.8,.8,
                 .1,.2,.2,.2,.2,.5,.7,.8,.9,.9,
                 .1,.2,.3,.3,.3,.3,.7,.8,.9,.9,
                  0,.1,.2,.4,.4,.4,.4,.7,.9,.9;
}
```

NetCDF ファイル用の CDL 表記は後述（10.5 節 「ncdump」（p. 111）を参照）のユーティリティプログラム ncdump を使って簡単に自動作成できます。別の NetCDF ユーティリティである ncgen は NetCDF ファイル（もしくは随意に NetCDF ファイルを作成するために必要な呼び出しを含む C 及び FORTRAN のソースコード）を CDL インプットから作成します。（10.4 節 「ncgen」（p. 110））

CDL 表記法は単純で大部分が自明です。 NetCDF ファイルの構成要素を説明してゆくに従い CDL 表記法のより詳細な部分を明らかにしていきます。この時点では、CDL 文がセミコロンで終わることに注意してください。スペース・タブ・改行は自由に使って文を読みやすくしてください。CDL のコメントはどの行においても '//' に続きます。NetCDF ファイルは CDL では以下のように記述されます。

```
NetCDF name {
  dimensions: …
  variables: …
  data: …
}
```

ここで *name*（名）は ncgen ユーティリティー を使ってファイル名を作成する際に単にデフォルトとして使用されます。CDL 記述には３つのオプションがあり、dimensions・ variables・data のキーワードによって開始されます。NetCDF 次元の宣言は dimensions の後に記述されます。NetCDF 変数及び属性は variables の後に定義され、変数データの割り当ては data の後に続きます。

2.2 次元

次元は実際の物理的な次元（例えば、時刻・緯度・経度・高度等）を表すために使用します。また、次元は他の数量のインデックス（例えば、ステーションやモデル現行番号）としても使用できます。

NetCDF 次元は*名前 name* と *長さ length* を持っています。次元長とは任意の正の整数ですが、NetCDF ファイル中の一つの次元は `UNLIMITED` の長さを持つことが出来ます。

そのような次元は*無制限次元 unlimited dimension* もしくは*記録次元 record dimension* と呼ばれます。無制限次元を持つ変数はその次元に沿って無制限に延びることが出来ます。無制限次元インデックスは従来の記録指向型ファイルにおける記録番号のようなものです。一つの NetCDF ファイルは最大で一つの無制限次元しか持てませんが、無制限次元を持たなくても構いません。もし、変数が無制限次元を持つとしたら、その次元は最も重要な（最も遅く変化する）ものでなくてはなりません。従って、無制限次元は必ず CDL 形式の最初の次元でなければならず、C 配列宣言においては最初 の次元でなくてはなりません。

CDL 次元宣言は CDL キーワードの次元 `dimensions` に続く行（複数行でも可）に書かれます。同一行における複数の次元宣言は コンマで区切ります。宣言は各々*名前＝長さ name = length* で表されます。

上記の例では 4 つの次元 `lat`、`lon`、`level`、そして `time` があります。最初の 3 つの次元は固定長です。 `Time` は無制限長 `UNLIMITED` を与えられており、これは `time` が*無制限次元 unlimited* であることを意味します。

NetCDF ファイル中で名前のあるデータの基本単位は*変数 variable* です。変数はその*形 shape* が次元のリストとして定義されます。それらの次元は既に存在していなければなりません。次元の個数は*ランク rank*（またはディメンショナリティ *dimensionality*）と呼ばれます。スカラー変数はランク 0 となり、ベクトルはランク 1、行列はランク 2 ということになります。

変数の形を定義するのに同じ次元を複数回使用しても構いません。(以前のバージョンの NetCDF ではこれは不可能でした。) 例えば、`correlation(instrument, instrument)` と定義して、異なる機器で測定された値の相関を示す行列を表すことが出来ます。しかし、物理的な空間 / 時間に相当する次元を持つデータは、たとえその次元の幾つかが同じ値を取る場合においても、異なる次元で構成される形を取るべきです。

2.3 変数

変数 は大部分のデータを NetCDF のファイルとして格納するのに使用されます。変数とは同一型の値の配列を指します。スカラー値は 0 次元の配列として扱われます。変数は名前・データ型・変数が定義されたときに与えられた次元のリストによる形を持ちます。また、変数は関連する属性を持つことも出来ます。この属性は後に加え・削除し・変更することが出来ます。

変数外部データ型とは NetCDF の型 *types* の小さな集合の一つであり、次のような名前を持ちます。C インターフェースで `NC_BYTE, NC_CHAR,NC_SHORT, NC_INT,NC_FLOAT,` 及び `NC_DOUBLE`。 `NC_LONG` は C インターフェースにおける `NC_INT` の deprecated 同義語です。

CDL 表記ではではこれらはより単純な `byte` ・ `char` ・`short`・ `int`・ `float`・及び

double 等の名前を与えられています。real は CDL 表記において float の同義語として使用できます。long は int に対する同義語です。各変数の厳密な意味については「NetCDF 外部データ型」3.1 節 「NetCDF 外部データ型」（p.19）をご参照下さい。

CDL 変数宣言は CDL 単位中のキーワード variable に続きます。それらの形式は次元付きの変数については

type variable_name (*dim_name_1, dim_name_2, …*);

また、スカラー変数については

type variable_name;

という形を取ります。

t 前述の CDL の例では変数が 6 つあります。次に述べるように、その内 4 つは座標変数です。残りの 2 つの変数 temp と rh は*主変数 primary variables* とも呼ばれ、一般的にデータと見なされるもので構成されています。各々の変数は time という無制限の次元を第一次元として含み、よって*記録変数 record variables* と呼ばれます。記録変数ではない変数は固定長（データ値の個数）を持ち、次元長の積に相当します。記録変数の長さはもその次元長の積ですが、この場合には無制限次元の長さが一定ではないためにその積は変数であり、当然変化します。無制限次元の長さは記録数に該当します。

2.3.1 座標変数

NetCDF においては変数が次元と同一の名前を持つことが許されています。それらの変数は NetCDF ライブラリにとっては特別な意味を持ちません。しかしながら、そのライブラリを使用するソフトウェアに特別な意味を持つ変数として扱われるという規約があります。

次元と同じ名前を持つ 変数は*座標変数 coordinate variable* と称されます。通常はその次元に対応する物理的な座標を定義するために使われます。前出の CDL の例には次のように定義される座標変数 lat、lon、level 及び time が含まれています。

```
        int     lat(lat), lon(lon), level(level);
        short   time(time);
…
data:
        level   = 1000, 850, 700, 500;
        lat     = 20, 30, 40, 50, 60;
        lon     = -160,-140,-118,-96,-84,-52,-45,-35,-25,-15;
        time    = 12;
```

これらはこの次元に沿った地点における緯度・経度・気圧・時刻を定義しています。つまり、ここでは高度 1000、850、700、及び 500 mbar に相当する高度と北緯 20、30、40、50、60 度におけるデータが存在するということです。各座標変数はベクトルであり同一の名前を持つ次元のみで構成されている形を持つことに注意してください。

次元に沿った位置は*インデックス index* を使用することによって指定できます。インデックスは整数であり、最小値は C では 0　になります。前出の例では 700 mbar レベルにおけるインデックスは 2 となります。

次元に対応する座標変数が存在する場合には、その次元に沿った位置を指定するための代替的的で通常はより便利な方法があります。現行の座標変数を使用するアプリケーションパッケージでは、それらの値が数値ベクトルであり、狭義の意味で単調である（全ての値は異なり、一方的に増加もしくは減少する）という仮定をしています。

2.4　属性

NetCDF の *属性 attributes* はデータに関するデータ（*補助的データ ancillary data*・*メタデータ metadata*）を格納するために使用されます。その手法は従来のデータベースシステムのデータ辞書や図表を格納するのに使用されている手法と多くの類似点があります。大半の属性は特定の変数に関する情報を含んでいます。その変数の名前（もしくは ID）と属性の名前と併せて識別されます。

幾つかの 属性はファイル全体の情報を与えており、グローバル属性（*global* attributes）と呼ばれます。これらは属性の名前と CDL の場合には空白の変数名、C 及び FORTRAN の場合には特別な null グローバル変数 ID によって識別されます。

属性には関連する変数（グローバル属性の場合には null グローバル変数）、名前、データ型、データ長、そして値があります。現行版においては全ての属性をベクトルとして扱っています。スカラー値は単一要素ベクトルとして扱われます。

可能な場合には従来の 属性名を使用する方が好ましいでしょう。新しく名前をつける場合には出来る限り意味のあるものを付けましょう。

属性の 外部型は定義される際に指定されます。属性に使用できる型は変数の場合の NetCDF の外部データ型と同じです。異なる変数に同一の名前の属性がある場合には異なる型の場合があります。例えば、変数型 `int` の有効データ値の最大値を特定する属性 `valid_max` は `int` 型であるべきです。それに対して変数型 `double` に対する属性 `valid_max` は `double` 型であるべきです。

属性は変数や次元よりも ダイナミックです。属性は削除可能で、作成後にも型・長さ・値を変更することが可能です。それに対して、NetCDF インターフェースでは変数を削除したり、変数の型や形を変更することは出来ません。

属性を定義するための CDL 表記法では変数属性は

variable_name:attribute_name = *list_of_values*;

であり、グローバル属性は

:attribute_name = *list_of_values*;

となります。CDL においては各属性の 型や長さは明確には宣言されません。それらは

属性に割り振られた値によって決定されます。単一の属性に所属する値は全て同一型でなければなりません。色々なNetCDF型の定数に使用される表記法については後述する。(10.3節 「データ 定数のCDL表記」 (p.108))

NetCDF の例 (2.1.2節 「Network Common Data Form Language (CDL)」 (p.12)) では units は変数 lat に対する属性で13文字列 'degrees_north' の配列値を持ちます。そして valid_range とは長さ２、値 '0.0' と '1.0' を持つ変数 rh の属性です。

NetCDFファイルの例では一つの グローバル属性 ---source--- が定義されています。実際の NetCDFファイルではファイル全体の起源・歴史・規約・特徴などを記述するためにより多くのグローバル属性を持つかもしれません。

NetCDFファイルを処理する 一般的なアプリケーションの多くは 標準的な属性の規約規約に従っており、特に理由がない場合には規約に従うことをお薦めします。 Units, long_name、valid_min、valid_max、valid_range、scale_factor、add_offset、_FillValue、及び他の規約的な属性については8.1節 「属性の規約」 (p.86) を参照してください。

任意のNetCDFファイルが最初に作成されてから時がたっていても属性を定義することは可能です。ですから、ファイルの作成当初に使用される可能性のある属性を全て網羅するようと悩む必要はありません。しかし、既存のファイルに新しい属性を加えることはファイルをコピーするのと同じ作業量が必要となる場合があります。より詳しい議論は9章 「NetCDFファイルの構造と性能」 (p.101) にあります。

2.5 属性と変数との違い

データの塊を処理するために使われる変数に 対し、属性は補助的なデータやデータに関する情報のために使用されます。NetCDFのオブジェクトに関連し、属性に格納された補助的なデータの総量は通常、メモリ上に十分保存できます。それに対し、変数は全体をメモリ上に保管するにはしばしば大きすぎ、処理するために分割する必要が出てきます。

属性と変数の異なる点はまだあります。それは変数は多次元であることができることです。属性は全てスカラー (単一数値) もしくはベクトル (一方向に既定された次元) です。

変数はデータ値を割り当てられる前に名前・型・形を定義されます。ですから値の無い変数が存在することもあります。属性の値は作成時に指定する必要があるので、値の無い属性は存在しません。

変数は属性を持ち得ますが、属性は属性を持つことが出来ません。変数に割り当てられた属性は変数と同じ単位を持つことが出来ます (例えば valid_range)。単位の無い属性というのも可能です (例えば scale_factor)。関連する変数と異なる単位を使用するデータを格納したい場合には属性よりも変数を使うことをお薦めします。より一般的には、データが説明のための補助的データを必要としたり、多次元であったり、データの値のインデックスとして定義されたNetCDF次元を必要としたり、格納量が多大であ

る場合には場合には 、データは属性よりも変数として表現されるべきでしょう。

3　データ

この章では６つの基本的な NetCDF の外部データ型、及び NetCDF インターフェースによってサポートされているデータアクセスの種類を紹介し、さらに配列型以外のデータ構造が NetCDF ファイルによって実装可能であるかを紹介します。

3.1　NetCDF 外部データ型

NetCDF インターフェースによってサポートされている外部データ型は以下の通りです。

`char`	テキストを表現するための 8 ビット文字
`byte`	符号付、又は符号無しの 8 ビット整数（下記参照）
`short`	符号付 16 ビット整数
`int`	符号付 32 ビット整数
`float or real`	32 ビットの浮動小数点数
`double`	64 ビットの浮動小数点数

これらはデータの精度と個々の値に必要なビット数のトレードオフの幅を広げるために設定されました。これらの外部データ型は任意のマシンや言語の組み合わせによってサポートされている内部データ型から完全に独立しています。

これらのデータ型が「外部」と呼ばれるのは NetCDF データのポータブル外部表記に対応するからです。あるプログラムがデータを内部変数として読み込む際に、必要であれば指定された内部変数型に変換されます。同様に、内部データ型が NetCDF 変数の外部データ型と異なる場合には、内部データを NetCDF 変数として書き込む際に、異なる外部データ型に変換されてしまう可能性があります。

外部型と内部型を分離し、自動的に外部－内部型変換をすることにはいくつかの利点があります。数値変数の外部データ型を知らなくても自動的にどのような数値型にも変換できるからです。この特性を利用して、十分に幅広い範囲の内部データ型を使用することによって外部データ型から独立した 形にコードを単純化したりすることも可能です。即ち、数種類の異なる外部データ型を持つ数値 NetCDF データに関しては倍精度になります。ある変数の外部データ型が変更されてもプログラムを書き換える必要は無いのです。

外部数値型から、もしくは外部数値型へ変換をする場合にはライブラリに任せます。このように外部データ表記と内部データ型間の変換を自動化し、両者を切り離すことは NetCDF の将来のバージョンにとってはより一層重要な意味を持ちます。圧縮データに新たな外部データ型が加えられ、それに自然に対応する内部データが存在しないケースも出てくるかもしれません。(例えば 11 ビット値の圧縮配列等)

ある数値型から別の型に変換する場合に、変換された値を表現しきれない型に変換するとエラーが生じます。例えば、内部の短い整数型では外部で整数として格納されているデータを表しきれないでしょう。数値配列にアクセスする際に、表現可能な領域から一つ以上の値がはみ出してしまった場合にはレンジエラーが返されてきます。領域内に収まる他の数値については正常な変換が行われます。

ここで注意して頂きたいのはデータ型の変換に伴う単なる精度の悪化ではエラーが返されないということです。つまり、倍精度の数値を単精度の浮動小数点数に変換した場合には、倍精度の値が変換先のプラットフォームで表現可能な単精度の浮動小数点数の範囲から逸脱しない限り、エラーは返されません。同様に、浮動小数点数の仮数の有効桁数では表しきれない程の大きな整数値を読み込んだ場合にも、この操作によって失われた精度に対するエラーは返されません。このような精度のロスを避けるにはアクセスする前に外部データの変数型をチェックし、十分な精度を持つ内部データ型に変換するようにしてください。

基本外部データ型の名前（`byte`, `char`, `short`, `int`, `float` 又は `real`, 及び `double`）はCDLにおいては予約語です。ですから、変数・次元・属性の名前はこれらを使用してはいけません。

バイトデータ型は符号付整数値（-128 ～ 127）としても符号なし整数値（0 ～ 255）としても扱うことができます。しかし、バイトデータ型を他の数値表現型に変換する場合には符号付数値として認識されます。

NetCDF外部データ型と任意の言語のデータ型との互換性については2.3節 「変数」(p.14) を参照してください。

3.2 データアクセス

NetCDFデータにアクセスする（読み込む・書き込む）場合には，オープンされたNetCDFファイル、NetCDF変数、及び変数の要素を特定する情報（例：番号）を指定します。アクセス機能の名前は内部データ型の名前に対応します。内部データ型と外部変数型の表現が異なる場合にはデータが読み書きされる際に内部型と外部型との間の変換が行われます。

データへは *direct*（直接）アクセスします。これによって大きなファイルから小さな部分集合を効率的にアクセスすることができます。その部分集合の前にあるデータを先にアクセスしないからです。データを、ファイル中の位置ではなく、変数を指定することによって読み書きすることは 、データアクセスをそのファイルの中に他に幾つ変数が存在するかとは無関係になります。これによってデータに新たな変数が加わるデータフォーマットの変更に対してプログラムの書き換えは不必要になります。

CとFORTRANインターフェースでは、データアクセスをする度にファイルを名前で特定せずに、ファイルが初めて作成・オープンされた時に割り当てられるファイルIDと呼ばれる小さな整数によって識別されます。

同様に、任意の変数はデータアクセスの度に名前で識別されません。その代わり、変数IDと呼ばれる、NetCDF中の各変数を識別するのに使用される小さな整数 によって識別

されます。

3.2.1 データアクセスの形式

NetCDF インターフェースにはオープンな NetCDF ファイル中のデータ値に直接アクセスする方法が幾つか用意されています。これらのアクセス形式を汎用性の小さいほうから順に説明します：

- 全ての要素へのアクセス形式
- *index vector*（インデックスベクトル）によって識別された個々の要素へのアクセス形式
- *index vector*（インデックスベクトル）と *count vector*（カウントベクトル）によって識別された配列断面へのアクセス形式
- *index vector*（インデックスベクトル）、*count vector* 又は *stride vector*（ストライドベクトル）によって識別された部分サンプルされた配列断面へのアクセス形式
- *index vector*、*count vector*、*stride vector*、及び *index mapping vector*（インデックスマッピングベクトル）によって識別されたマップされた配列断面へのアクセス形式

4 種類のベクトル（*index vector*・ *count vector*・ *stride vector* ・ *index mapping vector*）は変数の各次元に対応する要素を一つずつ持っています。ですから、n 次元の変数（rank = n）については n 個のベクトルが必要となります。変数がスカラー量（無次元）の場合には、これらのベクトルは無視されます。

Array section（配列断面）とは２つのベクトルによって指定される連続的な直方体、もしくは「板切れ」のようなものです。*Index vector* が原点に最も近い角の要素の座標を表します。 *Count vector* は各変数の次元に沿った板切れの縁の長さを順番に表します。アクセスされた値の個数はこれらの縁の長さの積です。

Subsampled array section（部分サンプルされた配列断面）は *array section* に似ていますが、さらにに *stride vector* というベクトルを使用してサンプリングを識別するために使用されます。このベクトルは各次元ごとに要素があり、その次元に沿って取るべきストライドの長さを表しています。例えば、ストライドが４であるなら、その次元に沿って４つ置きの値をとるという意味になります。この場合にも、アクセスされた値の総数は *count vector*（カウントベクトル）の各要素の積になります。.

Mapped array section（マップされた配列断面）は *subsampled array section* に似ていますが、さらに *index mapping vector*（インデックスマッピング）が加わり、NetCDF 変数に関連するデータのメモリー中の配置を指定することができます。各値の参照値からのオフセットは 各インデックスと対応する index mapping vector の要素を掛け合わせたものの和になります。(マッピングがされていない場合には仮想的な内部配列のインデックスが使用されます。)アクセスされた値の個数は *subsampled array section* の場合と同じになります。

マップされた配列断面の応用については後により詳細に述べます。その前に、より一般

的な配列断面へのアクセスの例を見ましょう。

3.2.2 配列断面のアクセス例

先に扱った NetCDF ファイルの例 (2.1.2 節 「Network Common Data Form Language (CDL)」 (p.12)) において、あるレベル（例えば２段目）の temp 変数の全データの断面を読み取りたいとし、そしてその NetCDF ファイルには記録が３つ （time 値）あるとます。次元は

```
lat = 5, lon = 10, level = 4, time = unlimited;
```

と定義されます。そして、変数 temp は CDL 表記においては

```
float   temp(time, level, lat, lon);
```

と宣言されます。

対応する C 変数で、ある一つのレベルのみのデータを保持しているものは、データを１次元配列に留めておくために、

```
#define LATS  5
#define LONS 10
#define LEVELS 1
#define TIMES 3                 /* 現在 */
   …
float   temp[TIMES*LEVELS*LATS*LONS];
```

と宣言されるかもしれません。又は、多次元配列を使って

```
   …
float   temp[TIMES][LEVELS][LATS][LONS];
```

と宣言されます。

第２レベルにのみある全時刻・全緯度・全経度のデータブロックを識別するためには、始点インデックスと縁の長さを与えなければなりません。始点インデックスは C では (0, 1, 0, 0) であるので、time・ lon・lat 次元については最初から開始したいのですが、level 次元については２番目の値から開始したいわけです。Time 値については 3 個全て、level 値については 1 個のみ、lat 値については 5 個全て、そして lon 値については 10 個全てを取得したいので、縁の長さは C では (3, 1, 5, 10) （になります。この操作によって合計 150 個 (3*1*5*1) の浮動小数点数が返されるので、これだけの数を収容するのに十分な配列スペースを 確保しなければなりません。このデータが返される順番は最も早く変化する 最後の次元 lon, になります：

```
temp[0][1][0][0]
temp[0][1][0][1]
temp[0][1][0][2]
temp[0][1][0][3]
```

…

```
temp[2][1][4][7]
temp[2][1][4][8]
temp[2][1][4][9]
```

Cや FORTRANなどの異なる言語インターフェースにおける 次元の順番が異なるのはディスク上で保存されている順番が異なるからではなく、単に各言語に対する手続きインターフェースのによってサポートされいる順番が異なっているからです。一般的に、NetCDF ファイルが C や FORTRAN、又は他の言語インターフェースで作成されていても何も変わりません。NetCDF をサポートする言語で作成された NetCDF ファイルは他の言語で書かれたプログラムを使って読み取ることができます。

3.2.3 一般的な部分アクセスに関する追記

Mapped array sections を使用することによって変数要素のディスクアドレスとメモリ上で格納されているアドレスの間に自明ではない関係を確立することができます。例えば、メモリ上の行列はディスク上の行列を移項したもので、要素が全く異なる順番で格納されるかもしれません。通常の *array section* においてはディスク上とメモリアドレス の関係は自明です：メモリ内での値の構造（次元サイズと順番）は *array section* のものと一致しています。しかしながら、*mapped array section* においては NetCDF の変数要素の指数とそれらのメモリ上のアドレスとのマッピングを定義するのに *index mapping vector* が使用されます。

マップされた配列へのアクセスによって，メモリに常駐する配列の原点とある任意の点間とのオフセット量（配列の要素の数）は index mapping vector とその点の *coordinate offset vector* の *inner product*[1] (*内積*) で表される。 ある任意の 点の *coordinate offset vector* は各次元の内包される配列の原点からその点までのオフセット量を与えます。C 言語ではある点の coordinate offset vector は元の coordinate vector と同一です。

通常の配列部分の index mapping vector は‘最も早く変化する次元から最も遅く変化する次元の順番に’常に定数１を持つはずです。なぜならば、その値と最も早く変化する次元の一辺の長さの積を取り、その値と次に早く変化する次元の一辺の長さとの積を取る、という操作を繰る返すからです。 しかし、mapped array においては，NetCDF 変数のディスク上での位置とメモリ上での位置との相関は異なることもあります。

下記の C 言語での定義の場合を例に取りましょう。

```
struct vel {
    int flags;
    float u;
    float v;
```

1. ベクトル [x0, x1, …, xn] と [y0, y1, …, yn] の内積は単に x0*y0 + x1*y1 + … + xn*yn となります。

```
} vel[NX][NY];
ptrdiff_t imap[2] = {
    sizeof(struct vel),
    sizeof(struct vel)*NY
};
```

ここで、imap は index mapping vector であり、メモリ上に常駐している NetCDF 変数 vel(NY,NX) にアクセスするのに使用できます。例え次元が移項されていて、データが２次元の浮動小数点数ではなく２次元の配列構造を取っていても大丈夫です。

マップされた配列へのアクセスに関する詳しい例はマップされた配列へのアクセスに関するインターフェースの説明文にあります。p.66 「マップされた配列の値を書き込む： nc_put_varm_type」を参照してください。

NetCDF 抽象化によって部分サンプルされた配列断面やマップされた配列断面によるアクセスを可能ですが、これらを使用する必要はありません。これらのより汎用的なアクセス法が不必要な場合には、これらの機能を無視して単一値によるアクセスや通常の配列断面アクセス方法を使用してください。

3.3 型変換

NetCDF 変数には各々、最初に定義された時に指定される外部型を所有しています。この外部型によってデータがテキストや数値として扱われるか判別されます。数値として扱われる場合には、その範囲と精度も指定されます。

NetCDF の変数の外部型が char の場合、テキスト配列である文字データのみが変数として書き込み、読み取ることが可能です。テキストデータを異なる型のデータに自動変換する機能はサポートされていません。

ただし、数値データである場合には変数を異なるデータ型としてアクセスし、メモリに格納されている数値データと NetCDF 変数との間で自動的に型変換する機能を NetCDF ライブラリは保有しています。例えば、全ての数値データを倍精度の浮動小数点数として扱うプログラムを作成した場合には、NetCDF 変数の外部型がどんな型であるかを気にせずに、NetCDF データを倍精度配列に読み取ることができます。NetCDF データを読み取る際、様々な大きさの整数 や単精度の浮動小数点数は、倍精度の数値用のデータアクセスインターフェースを使ってアクセスすれば、自動的に全て倍精度数値になります。もちろん、このように自動的に数値が変換されることを望まない場合には、その数値の型が存在すれば、各々の Net CDF 変数の外部データ型に対応したインターフェースを使用すれば避けられます。

NetCDF が行う数値変換は大変わかりやすいものばかりです。それは、数値変換が任意の型のデータを別のデータ型の変数を取るよう指定する操作であるからです。例えば、浮動小数点 NetCDF データを整数として読む場合には、結果は零に打ち切られます。浮動小数点数を整数の変数に割り付ける場合と同様です。このような打ち切りは数値変換に伴う精度悪化の例といえます。

ある数値型から他の異なる数値型に変換する場合にも、変換先の型が変換された数値を

表すことのできない型の場合にもエラーが生じます。例えば、整数では外部型として IEEE 浮動小数点数として格納されているデータを表せません。数値の配列をアクセスする際には、一つ以上の数値が表せる範囲外である場合にはレンジエラーが返ってきます。その他の範囲内にある数値については正しく変換されます。

注意すべき点は、型変換による精度のロスのみではエラーを引き起こさないということです。例えば、倍精度の数値を整数として読む場合には、倍精度の数値の大きさが読込先のプラットフォームで表し切れる整数の範囲外でなければエラーと判定されません。同様に、大きな整数を浮動小数点数に変換する際に、浮動小数点数の仮数部にその整数の全てのビットを表し切れなくて精度にロスが生じたとしてもエラーにはなりません。このような精度のロスを避けたい場合にはアクセスする変数の外部型をチェックして、使用する内部型と整合性があることを確認してください。

表し切れる範囲の境界に近い大きな浮動小数点数を書き込む場合にレンジエラーが生じるかどうかはプラットフォーム次第です。NetCDF 浮動小数点変数に書き込める最大の浮動小数点数は、使用しているシステムで表せる 最大の浮動小数点数であり、2 の 128 乗よりは小さい値です。倍精度変数に書き込める最大の倍精度数値は、使用しているシステムで表せる最大の倍精度数値であり、2 の 104 乗より小さくなります。

この自動変換と外部データ表示を内部データ型から切り離すことは、NetCDF の将来のバージョンにおいてより重要性を増してくるでしょう。それは、将来、対応する内部型が存在しない詰めありデータ用（例えば 11 ビット数値の配列等）に新たな外部データ型が導入されることが考えられるからです。

3.4 データ構造

NetCDF 抽象化が直接的にサポートする唯一の データ構造は ベクトル属性つきの名前のついている配列の集合のみです。NetCDF はリンクされたリストや、ツリー、粗い配列、不均一な配列等、ポインタを必要とする種類のデータ構造を表現するのには適していません。

ある配列のデータを他の配列のデータへのポインタとして使用することに関する様々な規約を採用することにより、配列の集合により他のデータ構造を構築することは可能です。そのようなデータ構造を構築する際に、NetCDF ライブラリは役にも立ちませんが阻害もしません。その代わり、そのような規約を設計する手段を提供します。

次の例は属性 `row_index` を使用して不均質な配列 `ragged_mat` を格納し、それによって各列の始点となるインデックスを指定することにより関連するインデックス変数の名前を与えています。この例では、最初の列は 12 個（12-0）の要素からなり、2 列目は 7 個（19 -12)、という具合に続きます。

```
        float   ragged_mat(max_elements);
                ragged_mat:row_index = "row_start";
        int     row_start(max_rows);
data:
        row_start   = 0, 12, 19, …
```

もう一つの例として、NetCDF 変数は任意の NetCDFG ファイルの中でグループ化することが挙げられます。各グループの変数の名前を伝統的な区切り文字であるスペースやコンマによってリストにしている属性を定義することによってこれは可能になります。このようなグループ化のために属性名の規約付けをすることによって幾つもの名前のある変数グループを作ることを可能にします。ある特定の規約に従った属性を各々の変数に与えることによって変数がどのグループに属しているかのリストが作れます。他の属性や変数を指定する属性や変数の導入により、NetCDF ファイルにおける幾種の複雑な構造を表すための柔軟な手段が与えられます。

4　NetCDF ライブラリの使用

NetCDF ライブラリ 使用するために NetCDF インターフェースの事を全て知っている必要はありません。NetCDF ファイル作るのであれば片手で足りるほどのルーチンさえ知っていれば必要な次元・変数・属性を定義し、NetCDF ファイルにデータを書き込むことができます。(ncgen ユーティリティを使用して予めファイルを作成しておいてから、NetCDF ライブラリのデータ書き込みコールを活用したプログラムを走らせたならば、使用するルーチンの数はより少なくなります。同様に、ある NetCDF オブジェクトに格納されたデータにアクセスするソフトウェアを作成する際には、NetCDF ファイルを開き、データにアクセスするためには NetCDF ライブラリのほんの一部の NetCDF ライブラリしか使用しません。もちろん、任意の NetCDF ファイルにアクセスする包括的なアプリケーションを作る場合には、NetCDF ライブラリにより精通している必要があります。

この章では通常の使用に必要な一般的な NetCDF のコールのシークエンスのテンプレートを幾つか紹介します。明確さのため、ここではルーチンの名前のみを挙げています。宣言やエラーチェックについては触れていません。また、型に限定される変数や属性のルーチン名のサフィクスについても省略してあります。複数回使用される宣言文は字下げをしてあります。また、… を使用して他の宣言文の任意のシークエンスを表しています。全パラメーターのリストは後の章で説明します。

4.1　NetCDF ファイルを作成する

これは新しい NetCDF ファイルを生成するために使用する 一般的な NetCDF コールの配列です：

```
nc_create           /* NetCDF ファイルを作成： 定義モードに入る */
    …
  nc_def_dim        /* 次元の定義： 名前とサイズから */
    …
  nc_def_var        /* 変数の定義： 名前、型,... から */
    …
  nc_put_att        /* 属性を置く： 属性値を割り当てる */
    …
nc_enddef           /* 定義終了： 定義モードから抜ける */
    …
  nc_put_var        /* 変数に値を与える */
    …
nc_close            /* 閉じる： 新しい NetCDF ファイルを保存する */
```

コール一つで NetCDF ファイルを作成できます。その時点では、二つある NetCDF モードの最初の*モード*に入っています。開かれた NetCDF ファイルにアクセスする際でしたら、*定義モード*もしくは*データモード*に 入るはずです。定義モードでは次元・変数・新しい属性などを作れますが、変数データを読んだり書き込んだりすることは出来ません。データモードではデータにアクセスし、既存の属性を変更することは出来ますが、次元・変数・属性を 新たに作ることは出来ません。

新たに作られた次元には各々 nc_def_dim へのコールが一つ必要となります。同様に全ての変数には nc_def_var へのコールが一つ必要です。さらに、定義され、値を割り振られた属性には nc_put_att ファミリーのメンバーへのコールが必要となります。定義モードから出て、データモードに入るには nc_enddef とコールしてください。

一度データモードに入ると、変数に新たなデータを加えたり、古い値を変更したり、既存の属性値を変更することが出来ます（ただし、属性については格納スペースが増加しないことが条件です。）NetCDF 変数に単一の値を書き込むためには、書き込むデータ種によっては nc_put_var1 ファミリーのメンバーが必要となります。 nc_put_var ファミリーのメンバーを使用して変数の取るべき値を全て一度に書き込むにことも出来ます。変数の配列や配列断面は nc_put_vara ファミリーを使って書き込めます。部分サンプルされた配列断面も nc_put_vars ファミリーのメンバーを使うことによって書き込めます。マップされた配列断面も nc_put_varm ファミリーのメンバーを使うことによって書き込めます。（部分サンプルやマップされたアクセスは通常のデータアクセス法の一種であり、後に説明します。

最後に、書き込むために開いた NetCDF ファイルは nc_close を使って必ず閉じてください。ファイルシステムへのアクセスはデフォルトで NetCDF ライブラリによってバッファーされています。データを書き込める開かれた異常なステータスでプログラムが終了された場合には、その回に加えた変更が全て無効になる可能性があります。このデフォルトでバッファーしてしまう機能は、ファイルを開く際に、NC_SHARE フラグを立てることによって避けられます。フラグが立っていても、定義モードで行なわれた属性値の変更や定義モードで変更された事項は nc_sync 又は nc_close がコールされない限り、実行されません。

4.2 既知の名前の NetCDF ファイルを読む

ここでは NetCDF ファイルの名前ばかりでなく、それに含まれている次元・変数・属性の名前も既知である場合を取り上げます。(そうでない場合には "inquire" コールをする必要があります。NetCDF ファイルの中の変数のデータを読むためのごく一般的な C でのコールの順序は：

```
nc_open                 /* 既存の NetCDF ファイルを開く */
    …
  nc_inq_dimid          /* 次元 ID を取得 */
    …
  nc_inq_varid          /* 変数 ID を取得 */
    …
  nc_get_att            /* 属性値を取得 */
    …
  nc_get_var            /* 変数の値を取得 */
    …
nc_close                /*  NetCDF ファイルを閉じる */
```

まず、ファイルの名前を与えることにより、最初のコールが NetCDF ファイルを開きます。そして、その後、開かれたファイルを参照するために必要な NetCDF ID を返しま

す。

次に、 nc_inq_dimid へのコールでアクセスする次元ごとに 次元名に由来した次元 ID が割り振られます。同様にして、必要な変数 ID も変数名に由来する名前が nc_inq_varid へのコールで決定されます。一旦、変数 ID を手に入れれば、NetCDF ID、変数 ID、そして必要な属性名を使うことで、nc_get_att ファミリーのメンバーとして入力として、変数の属性値も読み取れます（通常、各々の属性に対して nc_get_att_text もしくは、nc_get_att_double)。変数データの値はNetCDF ファイルから、直接アクセスすることが出来ます。単一の値の場合には、nc_get_var1 ファミリーのメンバーへのコールのよって、そして変数全体の場合には nc_get_var ファミリーへ、又は配列・部分サンプル・マップされたアクセスの場合には nc_get_vara, nc_get_vars, もしくは nc_get_varm ファミリーへのコールを使います。

最後に、NetCDF ファイルは nc_close によって閉じられます。読み取るだけのためにファイルを開いた場合には閉じる必要はありません。

4.3 名前が未知の NetCDF ファイルを読む場合

変数の名前を前もって知らなくてもその全ての変数を処理するようなプログラム（例えば総括的なソフトウェア）を作成することは可能です。同様に、次元や属性名も明らかではない場合もあります。

NetCDF のオブジェクトに関するほかの情報も”inquire”機能を使用して NetCDF ファイルから得られます。この機能は全 NetCDF ファイル・次元・変数・属性等の情報を返します。下記のテンプレートはそれらの使用法を示しています：

```
nc_open                    /* 既存の NetCDF ファイルを開く */
  …
nc_inq                     /* 内容を調べる */
     …
   nc_inq_dim              /* 次元名と次元長を得る */
     …
   nc_inq_var              /* 変数名・型・形状を得る */
        …
      nc_inq_attname       /* 属性名を得る */
        …
      nc_inq_att           /* 属性型と属性長を得る */
        …
      nc_get_att           /* 属性値を得る */
        …
   nc_get_var              /* 変数の値を得る */
     …
nc_close                   /* NetCDF ファイルを閉じる */
```

上記の例のようにコール一つで既存の NetCDF ファイルが開き、NetCDF ID を返します。この NetCDF ID は nc_inq ルーチンに送られ、その操作によって次元数・変数の数・グローバル属性の数・そして存在すれば無制限次元の ID が返されます

この inquire 機能は手頃で、I/O を必要としません。それは、最初に NetCDF ファイルを開いた時に、提供する情報がメモリ内に格納されるからです。

次元 ID は 0 で始まる連続な整数を取り、一旦割り当てられると消去することは出来ません。また、次元も定義されたら消去することは出来ません。ですから、NetCDF ファイル中の次元 ID の数を知るということは全ての次元 ID を知ることと道義になります。それらは 0, 1, 2, … 等の整数で次元の数だけ存在します。各次元 ID に対しては、inquire 機能への nc_inq_dim で次元名と次元長が返されます。

変数 ID もまた連続した整数 0, 1, 2, … で表され、変数の数だけ存在します。変数 ID は nc_inq_var コールを使用して各変数に割り当てられた名前、型、形状、と属性数を知ることが出来ます。

一旦、ある変数の 属性値が 既知になると、nc_inq_attname コールによって任意の変数に割り当てられた NetCDF ID・変数 ID・属性数を知ることが出来ます。属性名が分かると，nc_inq_att コールで属性型と属性長が分かります。型と長さから、属性値を格納するために十分なスペースを確保しておくことが出来ます。 次に、 nc_get_att ファミリーの一員へコールすることにより変数値が返されます。

一度 NetCDF 変数の ID と形状が既知になると、データの値は単一の値の場合は nc_get_var1 ファミリーへの一員へのコール、そして複数の場合には、 nc_get_var, nc_get_vara, nc_get_vars, 又は様々な種類の配列アクセス法に関しては nc_get_varm へのコールすることになります。

4.4 新たに次元・変数・属性を加える

既存の NetCDF ファイルはかなりの変更を加えることが出来ます。すでに存在している次元・変数・属性などに新たに加えたり、名前を変更することも可能ですし、既存の属性は抹消することが出来ます。次のコードのテンプレートは既存のファイルに新しい要素を加えるためのごく一般的な例です。

```
nc_open             /* 既存の NetCDF ファイルを開く */
  …
nc_redef            /* 定義モードに入る */
    …
  nc_def_dim        /* （あれば）新しい次元を定義し、加える。*/
    …
  nc_def_var        /* （あれば）新しい変数を定義し、加える。*/
    …
  nc_put_att        /* （あれば）新しい属性を定義し、加える。*/
    …
nc_enddef           /* 定義をチェックし、定義モードから出る。*/
    …
  nc_put_var        /* 新しい変数に値を与える。*/
    …
nc_close            /* NetCDF ファイルを閉じる。*/
```

NetCDF ファイルは、まず、 nc_open コールによって開きます。このコールによって、開かれたファイルはデータモードに入ります。このモードでは既存のデータ値にアクセスしたり変更を加えたりすることが出来ます。また、属性値も（大きくならない限りにおいては）変更できます。ただし、このモードでは何もたすことは出来ません。新しい NetCDF 次元・変数・属性を加えるには nc_redef. コールによって定義モードに入らなければなりません。定義モードでは、新しい次元を定義するためには nc_def_dim コールを、新しい変数を加えるには nc_def_var コールを、そして古い変数や増大してしまった古い属性に新しい属性を与えるには nc_put_att ファミリーへコールします。

定義モードから出て、再びデータモードに入ることも出来ます。そこで、新しい定義に矛盾が無いか等をチェックし、ディスクに保存するには nc_enddef コールをしてください。データモードに戻りたくなければ、単に nc_close コールをしてください。これは、最初に nc_enddef コールをしたことと同義になります。

NcN_enddef コールがなされる前であれば、nc_abort. コールによって、定義モードで行なった全ての再定義を無効にして NetCDF ライブラリを元のステータスに戻せます。また、この nc_abort コールを使って、nc_enddef コールが失敗した場合に NetCDF ファイルを矛盾の無いステータスまで復帰させることが出来ます。定義モードから nc_close コールをしたら自動的に追従する nc_enddef へのコールが失敗した際には、nc_abort コールが自動的に呼び出され、NetCDF ライブラリは閉じられ、元の矛盾の無いステータス（定義モードに入る前のステータス）に戻ります。

一つのプロセスは書き込み用に一時に最大一個の NetCDF ファイルを開いていなければなりません。ライブラリは、統制の取れた nc_sync 関数の利用と NC_SHARE フラグを立てることによって同時に複数の読者に扱われることに大してのすポートに制限を設けています。もし、書き込むほうが定義モードに変更を加えれば（例：新しい変数、次元、属性)、そのライブラリに対して読者が同時にアクセスすることを防ぐ制約を外部から加える必要があり、また、読者に対して次回のアクセスの前に nc_sync を呼び出すように注意を促す必要が出てきます。

4.5 エラー処理

NetCDF ライブラリはエラー処理を柔軟に行なうのに必要な機能を揃えています。個々の NetCDF 関数は整数のステータス値を返します。もし、返されたステータス値によっエラーが発見されると、その処理方法をどのようにするかは自由です。関連するエラーメッセージを表示することから、エラー表示を無視して続行することも（後者は推奨しかねますが）可能です。簡単な例として、このガイド中の例はエラーステータスを調べ、エラーを処理するために別個の関数を呼び出すようになっています。

返された整数のエラーステータスをエラーメッセージ文字列に変換するために nc_strerror 関数が準備されています。

たまに、low levell I/O エラーが NetCDF ライブラリより下層で起こる可能性があります。 例えば、ある書き込みオペレーションにより割り当てられたディスク容量を越えてしまった、既に存在しないデバイスに書き込もうとした場合に、NetCDF ライブラリ

より下層からエラーメッセージが表示されることがあります。しかしながら、結果として書き込みエラーは返されたステータス値に反映されます。

4.6 NetCDF ライブラリへのコンパイルとリンク

オペレーティングシステム、使用するコンパイラー、NetCDF ライブラリやインクルードファイルの格納先などの要因で、NetDCF の C や FORTRAN インターフェースを使用するプログラムのコンパイルと NetCDF ライブラリとリンクさせる方法 は各々の条件により異なります。それでもここでは敢えて、UNIX プラットフォーム上で NetCDF ライブラリを使用する プログラムをコンパイルしリンクする例を挙げます。各自、使用する状況に応じてこれらの例を応用してください。

NetCDF 関数又は定数を参照する C のファイルは最初にそれらが参照される前に適切な `#include` 文を含んでいなければなりません：

```
#include <netcdf.h>
```

C コンパイラが必ず参照する標準的なディレクトリに `netcdf.h` ファイルがインストールされていない限り、コンパイラ を呼び出す際には `-I` オプションを使用し、`netcdf.h` がインストールされているディレクトリを指定する必要があります。例えば、

```
cc -c -I/usr/local/NetCDF/include myprogram.c
```

別の手段として、`#include` 文に絶対パス名を指定することもできます。しかし、この方法でプログラムを作成すると、NetCDF が異なる場所にインストールされているプラットフォーム上ではコンパイルできなくなります。

tNetCDF ライブラリが必ずリンクが参照する標準的なディレクトリにインストールされていない限り、`-L` と `-l` オプションを利用して NetCDF ライブラリを使用するオブジェクトファイルをリンクしなければなりません。例えば：

```
cc -o myprogram myprogram.o -L/usr/local/NetCDF/lib -lNetCDF
```

別の手段として、ライブラリに絶対パスを指定することも出来ます。

```
cc -o myprogram myprogram.o -l/usr/local/NetCDF/lib/libNetCDF.a
```

5　ファイル

この章では、単独の NetCDF ファイルもしくは NetCDF ライブラリ全体を扱う NetCDF 関数のインターフェースに付いて解説します。

開かれていない NetCDF ファイルを参照する場合にはそのファイル名でのみ参照することが可能です。一度 NetCDF ファイルが開かれた後には、*NetCDF ID* によって参照されます。NetCDF ID とはファイルを生成又は開いた時に返される小さな非負の整数である。NetCDF ID は C におけるファイル記述子もしくは FORTRAN における論理装置番号によく似ています。単一のプログラムにおいては、 開かれた NetCDF ファイルの NetCDF ID はファイルごとに個別の値をとります。ある NetCDF ファイルが複数回開かれた場合には複数の異なる NetCDF ID を持つことになります。しかし、書き込み可能な NetCDF ファイルは開かれたファイルのある一つの ID のファイルに限定されます。開かれていた NetCDF ファイルが閉じられると、割り当てられていた NetCDF ID とそのファイル間の関連付けは断たれます。

NetCDF ライブラリを操作する関数には以下のものがあります：

・ ライブラリのバージョンを取得する。
・ 返されたエラーコードに呼応するエラーメッセージを取得する。

単一のオブジェクトとして NetCDF ファイルで サポートされている操作は以下の通りです。

・ ファイル名と上書き可能かどうかを指定されたらファイルを生成する。
・ ファイル名と読み／書き込みを指定し、アクセスのためにファイルを開く。
・ 次元・変数・属性を加えるために定義モードにする。
・ 追加された内容の一貫性をチェックして定義モードを出る。
・ ファイルを閉じ、必要な場合にはディスクに書き込む。
・ 次元の数・変数の数・グローバル属性数・存在するならば無制限次元の ID を取得する。
・ 最新のステータスであるかディスクと同期して確証する。
・ 最適な連続書き込みのために *nofill* モードをセット／解除する。

この章では、NetCDF のインターフェースを表現するために使用される規約のまとめの後に、これらの操作のためのインターフェースについて詳細に記述します。

5.1　NetCDF ライブラリインターフェースについての記述

この章、及びこれに続く章の中で、各々のインターフェースでの NetCDF 関数についての解説には以下の項目が含まれます：

・ 関数の説明と目的
・ C におけるその関数のプロト型の関数の正式なパラメーターの型・順序を示す。
・ C インターフェースにおける各正式パラメーターの解説

・ 起こりうるエラーステータスと原因
・ 解説される NetCDF 関数（時に他の関数も）を呼び出す C プログラムの例

この例はエラーハンドリングに関する単純な規約に沿って、各 NetCDF 関数に対する呼び出しで返されたステータスをもらさずチェックし、エラーが発見されると handle_error 関数を呼び出します。そのような　関数の例は 5.2 節 「エラーステータスに対応したエラーメッセージを得る： nc_strerror」（p.34）にあります。

5.2　エラーステータスに対応したエラーメッセージを得る： nc_strerror

関数 nc_strerror は、他の NetCDF 関数を呼び出したときに返されであろう、整数 NetCDF エラーステータス又はシステムエラー番号に対応するエラーメッセージ文字列に対し、静的なリファレンスを返します。NetCDF のエラーステータスのリストは各言語バインディング中の対応する内部ファイルにあります。

用法

```
const char * nc_strerror(int ncerr);
```

ncerr	以前の NetCDF 関数への呼び出しに対して返されたかもしれないエラーステータス

エラー

どの NetCDF エラーメッセージ、又は、（システム strerror 関数によって理解されるところの）システムエラーメッセージのどれにも対応しない、無効な整数エラーステータスを入力すると、nc_strerror はそのようなエラーステータスが存在しない旨の文字列を出力します。

例

これは簡単なエラー取り扱い 関数 の例で、nc_strerror を使用し、任意の NetCDF 関数呼び出しによって返された NetCDF エラーステータスに対応するエラーメッセージを出力した後 exit します。

```
#include <netcdf.h>
   …
void handle_error(int status) {
if (status != NC_NOERR) {
   fprintf(stderr, "%s\n", nc_strerror(status));
   exit(-1);
   }
}
```

5.3 NetCDF ライブラリバージョンを取得： nc_inq_libvers

関数 nc_inq_libvers は NetCDF ライブラリのバージョンと、いつ作成されたかを示す文字列を返します。

用法

```
const char * nc_inq_libvers(void);
```

エラー

この関数は引数を取らないのでこれの呼び出しによってエラーは発生し得ない。

例

この 例では nc_inq_libvers を使ってプログラムとリンクしている NetCDF ライブラリのバージョンを表示する。

```
#include <netcdf.h>
   …
   printf("%s\n", nc_inq_libvers());
```

5.4 NetCDF ファイルの生成： nc_create

この関数によって新規の NetCDF ファイルが生成されます。これによって返された NetCDF ID は他の NetCDF 関数呼び出しにおいてこのファイルファイルを参照するために使用できます。書き込みアクセス用に開かれ、定義モードになっている NetCDF ファイルファイルに、新しい次元・変数・属性などを加えることが出来ます。

生成モードのフラグによって、既存の同一名のファイルを上書きするか、及びファイルへのアクセスが共有されるかなどを指定できます。

用法

```
int nc_create (const char* path, int cmode, int *ncidp);
```

path	新しい NetCDF ファイルの名前

cmode	生成モード。零値（又は NC_CLOBBER）はデフォルトステータスを指定します：既存の同一名のファイルは上書き、及び効率のためにアクセスをバッファリングおよびキャッシュする。 それ以外の場合は、生成モードは NC_NOCLOBBER、NC_SHARE、又は NC_NOCLOBBER|NC_SHARE にあります。NC_NOCLOBBER フラグを立てることによって 既存のファイルを上書きしないことを表明できます。既に指定されたファイルが存在する場合には、エラー （NC_EEXIST）が返されます。 NC_SHARE フラグはファイルに書き込む処理とファイルを読み取る処理が一つもしくは複数行なわれている場合に適切です。これによって、ファイルへのアクセスはバッファーされず、キャッシュも制限されます。バッファー機構は連続アクセスに対して最適化されているので、データを連続的にアクセスしないプログラムにおいては NC_SHARE フラグを設定することによりパフォーマンスの向上が望めます。
ncidp	出力 NetCF ID が格納される場所へのポインタ

エラー

エラーが発生していない場合には、nc_create は NC_NOERR の値を返します。エラーの原因として下記が挙げられます。

- 存在しないディレクトリを含むファイルを渡している
- 既存のファイル名もしくはファイルを指定しながら、NC_NOCLOBBER も同時に指定している。
- 生成モードにとって無意味な値を与えている。
- ファイルの作成が許可されていないディレクトリに新しい NetCDF ファイルを生成しようとしている。

例

この例では、foo.nc と言う名前の NetCDF ファイルを生成します。現行のディレクトリに同一名のファイルが存在しない場合に限り、新しいファイルを生成します。

```
#include <netcdf.h>
   …
int status;
int ncid;
   …
status = nc_create("foo.nc", NC_NOCLOBBER, &ncid);
if (status != NC_NOERR) handle_error(status);
```

5.5 アクセスするために NetCDF ファイルを開く： nc_open

関数 nc_open は既存の NetCDF ファイルとアクセスするために開きます。

用法

```
int nc_open (const char *path, int omode, int *ncidp);
```

path	開く NetCDF ファイルのファイル名
omode	零値は（又は NC_NOWRITE）はデフォルトステータスを示します。ファイルは読取専用に開き、効率のためにバッファリング及びキャッシュする。 それ以外の場合には、生成モードは NC_WRITE、NC_SHARE、又は NC_WRITE|NC_SHARE です。 NC_WRITE フラグを設定することによりファイルを読取－書き込み両用に開きます。(”書き込み”とはファイルに加え得る全ての変更を指し、データの付加又は変更、次元・変数・属性の付加又は名前の変更、属性の削除等の操作を含みます。） NC_SHARE フラグはファイルに書き込む処理とファイルを読み取る処理が一つもしくは複数行なわれている場合に適切です。これによって、ファイルへのアクセスはバッファーされず、キャッシュも制限されます。バッファー機構は連続アクセスに対して最適化されているので、データを連続的にアクセスしないプログラムにおいては NC_SHARE フラグを設定することによりパフォーマンスの向上が望めます。
ncidp	出力 NetCF ID が格納される場所へのポインタ

エラー

エラーが発生していなければ、nc_open は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因として下記が挙げられます。

- 指定された NetCDF ファイルが存在しない。
- 意味の無いモードが指定された。

例

この 例は nc_open を使って、既存の foo.nc という NetCDF ファイルを読取専用、非共有アクセス用に開きます。

```
#include <netcdf.h>
   …
int status;
int ncid;
   …
status = nc_open("foo.nc", 0, &ncid);
if (status != NC_NOERR) hendle_error(status);
```

5.6 開かれたNetCDFファイルを定義モードにする：nc_redef

関数 nc_redef は開かれたNetCDFファイルを定義モードにし、次元・変数・属性などを付加又はそれらの名前を変更し、さらに属性を削除できるようにする。

用法

```
int nc_redef(int ncid);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返されたNetCDF ID。。

エラー

エラーが発生していなければ、nc_redef は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因として下記が挙げられます。

- 指定されたNetCDFファイルが既に定義モードにある。
- 指定されたNetCDFファイルは読取専用に開かれている。
- 指定されたNetCDF IDが開かれたNetCDFファイルを参照していない。

例

この 例では nc_redef を使って、既存の foo.nc というNetCDFファイルを開き、それを定義モードにする。

```
#include <netcdf.h>
   …
int status;
int ncid;
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);  /* ファイルを開く */
if (status != NC_NOERR) handle_error(status);
   …
status = nc_redef(ncid);                      /* 定義モードに入る */
if (status != NC_NOERR) handle_error(status);
```

5.7 定義モードから抜ける：nc_enddef

関数 nc_enddef は開かれたNetCDFファイルを定義モードから抜きます。定義モード中にNetCDFファイルに加えられた変更はチェックされ、問題なければディスクに書き込まれます。この時、非記録変数を〝フィル値”に初期化することも可能です。(5.12節 「書き込みのフィルモードを設定する：nc_set_fill」 (p.44) を参照。) NetCDFファイルはデータモードになり、変数データを読み取り・書き込みが可能になる。

この呼び出しは、場合によってはデータをコピーする作業が含まれる。これに関しての詳細は 9章 「NetCDFファイルの構造と性能」 (p.101) にあります。

用法

```
int nc_enddef(int ncid);
```

ncid　　以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。

エラー

エラーが発生していなければ、nc_enddef は　NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因として下記が挙げられます。

・ 指定された NetCDF ファイルが定義モードに無い。
・ 指定された NetCDF ID が開いている NetCDF ファイルを参照していない。

例

この 例は nc_enddef　を使って foo.nc という NetCDF ファイルの定義モードを終了し、データモードにします。

```
#include <netcdf.h>
   …
int status;
int ncid;
   …
status = nc_create("foo.nc", NC_NOCLOBBER, &ncid);
if (status != NC_NOERR) handle_error(status);

   …        /* 次元・変数・属性を生成 */

status = nc_enddef(ncid);  /* 定義モードを抜ける */
if (status != NC_NOERR) handle_error(status);
```

5.8　開かれた NetCDF ファイルを閉じる：nc_close

関数 nc_close は開いている NetCDF ファイルを閉じます。ファイルが定義モードにある場合には、閉じる前に nc_enddef が呼び出されます。(この場合には、もし nc_enddef がエラーを返せば、nc_abort が自動的に呼び出され、最後に定義モードに入った時の矛盾の無いステータスに復旧します。) 開かれた NetCDF ファイルが閉じられた後は、その NetCDF ID は次に開かれる又は生成される NetCDF ファイルに割り当てることが出来ます。

用法

```
int nc_close(int ncid);
```

ncid　　以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。

エラー

エラーが発生していなければ、nc_close は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因として下記が挙げられます。

- 定義モードに入り、nc_enddef への自動呼出しが失敗した。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。

例

この 例では、nc_close を使って、新しい foo.nc という NetCDF ファイルの定義モードを終了し、その NetCDF ID を開放する。

```
#include <netcdf.h>
   …
int status;
int ncid;
   …
status = nc_create("foo.nc", NC_NOCLOBBER, &ncid);
if (status != NC_NOERR) handle_error(status);

   …       /* 次元・変数・属性を生成 */

status = nc_close(ncid);       /* NetCDF ファイルを閉じる */
if (status != NC_NOERR) handle_error(status);
```

5.9 開かれた NetCDF ファイルに ついて問い合わせる： nc_inq ファミリー

関数 nc_inq のファミリーは NetCDF ID を与えられた開かれた NetCDF ファイルに関する情報を返します。ファイル問い合わせ関数は定義モードとデータモードのどちらからでも呼び出すことが出来ます。最初の関数 nc_inq は次元数・変数の数・グローバル属性の数・無制限長で定義された次元があればその次元 ID を返します。このファミリーの他の関数はこれらのうちどれか一つの情報を返します。

C では､これに属する関数には nc_inq、nc_inq_ndims、nc_inq_nvars、nc_inq_natts、nc_inq_unlimdim があります。

これらの関数が呼び出されても、必要な情報は開かれた個々の NetCDF ファイルについてメモリ上にあるので、I/O は行われません。

用法

```
int nc_inq          (int ncid, int *ndimsp, int *nvarsp, int *ngattsp,
                     int *unlimdimidp);

int nc_inq_ndims    (int ncid, int *ndimsp);
```

```
int nc_inq_nvars    (int ncid, int *nvarsp);

int nc_inq_natts    (int ncid, int *ngattsp);

int nc_inq_unlimdim (int ncid, int *unlimdimidp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
ndimsp	この NetCDF ファイルについて定義されている出力された次元数の位置へのポインタ
nvarsp	この NetCDF ファイルについて定義されている出力された変数の数の位置へのポインタ
ngattsp	この NetCDF ファイルについて定義されている出力されたグローバル属性の数の 位置へのポインタ
unlimdimidp	この NetCDF ファイルについて定義されている出力された無制限次元（存在すれば）の ID の位置へのポインタ 。無制限次元が定義されていない場合には、-1 の値が返される。

エラー

エラーが発生していなければ、nc_inq のファミリーは全て NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因としては下記が挙げられます。

・ 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。

例

この 例では nc_inq を使って、foo.nc という NetCDF ファイルに関する情報を得ます。

```
#include <netcdf.h>
   …
int status, ncid, ndims, nvars, ngatts, unlimdimid;
   …
status = nc_open("foo.nc", NC_NOWRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq(ncid, &ndims, &nvars, &ngatts, &unlimdimid);
if (status != NC_NOERR) handle_error(status);
```

5.10 開かれた NetCDF ファイルをディスクに同期させる： nc_sync

関数 nc_sync はメモリ内バッファと NetCDF ファイルのディスク上コピーとを同期させる方法を提供します。書き込み後に、同期させたい理由としては 2 つ挙げられます。

・ 異常終了の場合のデータ損失を最低限に抑える。

・ 書き込まれた直後から 他の処理においてデータを アクセス可能にする。しかしながら、既にファイルを開いている処理に関しては、書き込み処理が nc_sync を呼び出すときに記録数が増加していることを感知できないことに注意してください。その処理が記録数の増加を感知するためには、読み取り処理が nc_sync を呼び出さなくてはなりません。

この関数は NetCDF ライブラリ以前のバージョンと後方互換性があります。その目的は、一つの NetCDF ファイルを複数の読者と一人の作成者の間で共有可能にすることにあります。作成者は書き込み後に nc_sync を呼び出し、読者は読み取る前に毎回 nc_sync を呼び出します。作成者側では、この操作によってバッファされているものが全てディスク上に移動します。読者側では、この操作によって次に読み取られる記録が以前にキャッシュされたバッファからではなく、ディスクからの読み取りであることが保証されます。これによって、読者はファイルを閉じて新たに開くことなく、書込み操作によって加えられた変更を見ることが出来ます（例えば書き込まれた記録数)。わずかなデータ量をアクセスする場合には、書き込み後にいちいちディスクと同期させることは、バッファすることの有用性を手放すことになり、コンピュータ資源のコストを上げてしまいます。

共有を簡単にするために（そして推奨される方法は)、作成者、読者共にファイルを NC_SHARE フラグを立てて開くことです。そうすれば nc_sync を呼び出す必要は 全くなくなります。しかし、異なる処理間において少数の NetCDF アクセスのみを同期させる場合には、nc_sync 関数は NC_SHARE フラグよりも より細かい粒度を持ちます。

従属的なデータ（属性値など）に加えられた変更にも注意する必要があります。これらは NC_SHARE フラグによっては自動的に伝達されません。このためには nc_sync 関数を使わなければなりません。

作成者がデータの設計を変えるために定義モードに入った時にファイルを共有する場合は特に注意しなくてはなりません。以前のバージョンでは、作成者が定義モードを抜けると、変更は新ファイルに加えられたために、読者は旧ファイルを参照したままでした。読者が変更を見るためにはファイルを一度閉じて開きなおさなければなりませんでした。それによって、変更されたファイルが手元にあっても、読者側の内部テーブルが新しいファイルの設計と一致していないことには読者には伝わりません。再定義後にも NetCDF ファイルが共有されるためには、再定義中に読者がデータにアクセスするのを防ぎ、次にアクセスする前に読者に nc_sync を呼び出させる 、何らかの NetCDF ライブラリ外のメカニズムが必要になります。

nc_sync を呼び出すとき、NetCDF ファイルは データモードになくてはなりません。定義モードの NetCDF ファイルは nc_enddef が呼び出されたときにのみディスクと同期します。他の処理によって書き込まれている NetCDF ファイルを読み取る処理は nc_sync を呼び出すことによって 、ファイルを閉じて再度開くことなく、書き込み処理によって変えられた最新の変更（例えば、書かれた記録数）に関する情報を得られます。

NetCDF ファイルを閉じた時、又は定義モードから抜けるたびに、データは 自動的にディスクと同期します。

用法

```
int nc_sync(int ncid);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。

エラー

エラーが発生していなければ、nc_sync は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因として下記が挙げられます。

- NetCDF ファイルが定義モードにある。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。

例

この 例では nc_sync を使って、foo.nc という NetCDF ファイルのディスク書き込みを同期させます。

```
#include <netcdf.h>
   …
int status;
int ncid;
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);  /* 書き込み用に開く */
if (status != NC_NOERR) handle_error(status);

   …            /* データを書き込む、又は属性を変更する */

status = nc_sync(ncid);      /* ディスクに同期させる */
if (status != NC_NOERR) handle_error(status);
```

5.11 最新の定義を撤回する： nc_abort

この関数はもう、呼び出す必要がありません。ファイルが定義モード中に、不具合が生じ変更を決定できない場合には、nc_close によって自動的に呼び出されます。関数 nc_abort は 定義モードに無い場合には、単に NetCDF ファイルを閉じます。もし、ファイルが生成されている最中で、まだ定義モードにある場合には、ファイルは削除されます。nc_redef への呼び出しによって定義モードに入った場合には、NetCDF ファイルは定義モードに入る以前のステータスに復旧され、ファイルは閉じられます。

用法

```
int nc_abort(int ncid);
```

ncid　　以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。

エラー

エラーが発生していなければ、nc_abort は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因として下記が挙げられます。

- NetCDF ファイル生成中に定義モードから呼ばれた際、ファイルの削除が失敗した。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。

例

この例では nc_abort を使って、foo.nc というファイルの再定義から撤退する。

```
#include <netcdf.h>
   …
int ncid, status, latid;
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);/* 書き込み用に開く */
if (status != NC_NOERR) handle_error(status);
   …
status = nc_redef(ncid);                    /* 定義モードに入る */
if (status != NC_NOERR) handle_error(status);
   …
status = nc_def_dim(ncid, "lat", 18L, &latid);
if (status != NC_NOERR) {
   handle_error(status);
   status = nc_abort(ncid);                 /* 定義失敗、中止 */
   if (status != NC_NOERR) handle_error(status);
}
```

5.12 書き込みのフィルモードを設定する： nc_set_fill

この関数は以下に述べる状況下での書き込みを最適化するための高度な使用を目的としています。関数 nc_set_fill は書き込み用に開かれた NetCDF ファイルの *フィルモード* を設定し、戻り値として現行のモードを返します。フィルモードは NC_FILL 又は NC_NOFIL のどちらででも設定でき、NC_FILL に対応したデフォルトステータスはデータがフィル値によって既に埋められているというものです。即ち、非記録型変数を生成する際、もしくは未だに書き込まれていないデータを超えた値を記入する際に、フィル値が記入されます。これによって書き込まれる前にデータを読み取ってしまうことを感知できます。フィル値の使用法の 詳細については、p. 83 「フィル値」を参照してください。。独自のフィル値の定義の仕方については p. 86 「属性の規約」を参照してください。。

NC_NOFILL に対応する動作は、データをフィル値で満たそうとするデフォルト動作を

無効にします。これによって、NetCDF ライブラリがフィル値を書き込み、さらにそれらの値が後にデータによって上書きされるという二重の操作を避けることができ、パフォーマンスが向上します。

戻り値によって NetCDF ファイルがどのモードにあったかということが分かります。この値を利用して、開かれた NetCDF ファイルのフィルモードを一時的に変更し、後で元のモードに復旧させることが出来ます。

開かれた NetCDF ファイルを NC_NOFILL モードにした後は、後で読み取られる全ての位置に有効なデータが書き込まれていることを確認してください。nofill モードは書き込み用に開かれた NetCDF ファイルの一時的な性質でしかない点に注意してください。ファイルを一旦閉じて再度開いたときには、デフォルト動作に戻ります。又、フィルモードを陽に NC_FILL に設定するために、再び nc_set_fill を呼び出すことによってデフォルト動作に戻ることが出来ます。

nofill モードを設定することが有益な場合が 3 つあります。

1. NetCDF ファイルを生成・初期化するとき。この場合には nc_enddef を呼び出す前に nofill モードを設定しましょう。その後に、非記録型変数と初期化したい記録変数の初期値を*完全に*書き込んで下さい。
2. 既存の記録指向の NetCDF ファイルを拡張する時。書き込み用にファイルを開いた後に、nofill モードを設定し、追加する記録を漏れなく付加する。間に書き込まれていない記録が存在してはいけない。
3. 既存の NetCDF ファイルに初期化する予定の新しい変数を追加するとき。nc_enddef を呼び出す前に nofill モードを設定し、新しい変数を完全に書き込む。

もし、NetCDF ファイルが無制限次元を持ち、最後の記録が nofill モードにおいて書き込まれた場合には、nofill モードが設定されていない場合に比べてファイルが短い可能性がある。しかし、これは NetCDF インターフェースを通してのみデータアクセスすれば完全に透過性は保たれる。

将来のリリースでは、この機能はなくなっている（又は不必要）であるかもしれない。プログラマーの方はこの機能に必要以上に頼らないことが望ましい。

用法

```
int nc_set_fill (int ncid, int fillmode, int *old_modep];
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
fillmode	ファイルの取るべきフィルモード。NC_NOFILL 又は NC_FILL
old_modep	この呼び出し以前の返された現行のフィルモードの位置を示すポインタ。NC_NOFILL 又は NC_FILL

エラー

エラーが発生していなければ、nc_set_fill は NC_NOERR の値を返します。それ以外

の場合には、返されたステータスがエラーを示します。エラーの原因として下記が挙げられます。

- 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。
- 指定された NetCDF ID が読み取り専用に開かれたファイルを参照している。
- フィルモード引数が NC_NOFILL 又は NC_FILL のどちらでもない。

例

この 例では nc_set_fill を使って、foo.nc という NetCDF ファイルの連続書き込みの nofill モードを設定します。

```
#include <netcdf.h>
   …
int ncid, status, old_fill_mode;
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);  /* 書き込み用に開く */
if (status != NC_NOERR) handle_error(status);

   …            /* デフォルト prefill 設定でデータを書き込む */

status = nc_set_fill(ncid, NC_NOFILL, &old_fill_mode); /* nofill 設定 */
if (status != NC_NOERR) handle_error(status);

   …            /* prefill 無しでデータを書き込む */
```

6　次元

NetCDF ファイルの次元はファイルが作成されると同時に定義され、NetCDF ファイルが定義モード中に行われる。後に次元数を増やしたい場合などには定義モードに再び入れば良い。NetCDF 次元には次元名と次元長が存在する。一つの NetCDF ファイルは最大で一つの `unlimited` （無制限）次元を持つことが出来る。この次元を使用する変数はこの次元沿いには無制限に成長できる。

一つの NetCDF ファイルには定義できる次元数に上限（100）が提唱されている。この上限値は 事前に定義されたマクロ `NC_MAX_DIMS.` である。上限値を規定する目的は一般的なアプリケーションをより簡潔に作成できる点にある。それらのアプリケーションでは `NC_MAX_DIMS` 配列を用意することのよって任意の NetCDF ファイルを扱うことができる。NetCDF ライブラリの実装においてはこの勧告された上限値 は強制項目ではない。それ故、必要に応じてそれ以上の次元数を使用することも可能である。しかし、勧告された上限値を仮定した NetCDF ユーティリティは、その結果生じる NetCDF ファイルを扱えなくなる可能性もある

通常、次元名と次元長は次元がはじめて定義されるときに固定される。後に次元名を変更することは可能ではあるが、次元長（無制限次元を除いて）は既存のファイルの内容を新たに次元長を固定した新しい NetCDF ファイルに コピーする以外に次元長を増やす手段はない。

C インターフェース上の次元長は `int` 型ではなく `size_t` 型である。それによって 16 ビット `int` データ型しか扱わないプラットフォーム（例えば MS-DOS）上の NetCDF ファイル中のデータに全てアクセスすることが可能になる。もし逆に、次元長が `int` 型ならば、16 ビット `int` が許容できる以上のの次元長を持つ変数からデータアクセスが出来なくなる。

開かれた NetCDF ファイル中の NetCDF 次元は *dimension ID* と呼ばれる小さい整数によって参照されている。C インターフェースでは､次元 ID は定義された順に 0, 1, 2, …, となる。

次元に対してサポートされている操作は以下の通りである。

- 次元名と次元長を与えて次元を作成する。
- 次元名から次元 ID を取得する。
- 次元の ID から次元名と次元長を取得する。
- 次元を名前を変更する。

6.1　次元を生成する： `nc_def_dim`

関数 `nc_def_dim` 等は定義モード中であれば、新しい次元を開かれた NetCDF ファイルに加えることが出来る。NetCDF の ID、次元名、次元長を与えると、(引数として）次元 ID を返す。最大で一つの無制限次元（記録次元）が NetCDF ファイルごとに定義できる

用法

```
int nc_def_dim (int ncid, const char *name, size_t len, int *dimidp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
name	次元名。アルファベットの文字で始まり、次に零もしくはアンダースコア（'_'）を含む英数字が続く。大文字小文字は区別される。
len	次元長。この次元をインデックスとして使用する変数に対して、この次元が持ちうる値の数量。正の整数（size_t 型）もしくは事前に定義された定数 NC_UNLIMITED である。
dimidp	返された次元 ID の位置を示すポインタ。

エラー

エラーが発生していなければ、nc_def_dim は NC_NOERR の値を返します。それ以外の場合には返されたステータスがエラーの発生を示します。エラーの原因としては：

- NetCDF ファイルが定義モードにない。
- 指定された次元名は別の既存の次元名である。
- 指定された次元長が零より大きくない。
- 指定された次元長は無制限であるが、その NetCDF ファイル内に既に無制限の次元長を持つ次元が定義されている。
- 指定された NetCDF ID が開かれている NetCDF ファイルを参照しない。

例

これは nc_def_dim 機能を使用して次元名 lat 次元長 18、そして次元名 rec 次元長無制限の二つの次元を持つ新しい foo.nc という NetCDF ファイルを生成する例です：

```
#include <netcdf.h>
   …
int status, ncid, latid, recid;
   …
status = nc_create("foo.nc", NC_NOCLOBBER, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_def_dim(ncid, "lat", 18L, &latid);
if (status != NC_NOERR) handle_error(status);
status = nc_def_dim(ncid, "rec", NC_UNLIMITED, &recid);
if (status != NC_NOERR) handle_error(status);
```

6.2 次元名から次元 ID を取得する：nc_inq_dimid

関数 nc_inq_dimid は次元名を与えると、(引数として) NetCDF の次元 ID ををを返します。仮に ndims ある NetCDF ファイルに定義された次元数だとすると、各々の次元の ID は 0 と ndims-1. の間の値を取ります。

用法

```
int nc_inq_dimid (int ncid, const char *name, int *dimidp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
name	次元名。アルファベットの文字で始まり、次に零もしくはアンダースコア（'_'）を含む英数字列が続く。大文字小文字は次元名において区別される。
dimidp	返された次元 ID の位置を示すポインタ。

エラー

エラーが発生していなければ,nc_inq_dimid は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーの発生を示します。エラーの原因としては：

- 指定された次元名に対応する次元名が NetCDF ファイル内に存在しない。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。

例

これは nc_inq_dimid を使用して次元名 lat の次元 ID を取得する例です。この lat は以前に foo.nc という NetCDF ファイル内で定義されているという仮定です：

```
#include <netcdf.h>
   …
int status, ncid, latid;
   …
status = nc_open("foo.nc", NC_NOWRITE, &ncid);  /* 読み取り用に開く */
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_dimid(ncid, "lat", &latid);
if (status != NC_NOERR) handle_error(status);
```

6.3 次元について問い合わせる：nc_inq_dim ファミリー

この関数のファミリーは NetCDF 次元についての情報を返します。次元に関するの情報には次元名と次元長があります。無制限長の次元の長さは、存在していれば、その段階までに書かれた記録の数です。

このファミリーに属する関数は nc_inq_dim, nc_inq_dimname, そして nc_inq_dimlen があります。関数 nc_inq_dim はその次元についての全ての情報を返します。他の機能はその次元についてある一つの情報を返します。

用法

```
int nc_inq_dim     (int ncid, int dimid, char* name, size_t* lengthp);
```

```
int nc_inq_dimname (int ncid, int dimid, char *name);

int nc_inq_dimlen  (int ncid, int dimid, size_t *lengthp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
dimid	以前の nc_inq_dimid もしくは nc_def_dim 等への呼び出しからの次元 ID。
name	返された 次元名。呼び出すには予めスペースを割り当てておく必要がある。次元名の文字数の最大長は、事前に定義した定数 NC_MAX_NAME によって決まる。
lengthp	返された次元長の位置を示すポインタ。無制限次元においては、これはその時点までに書き込まれた記録の数である。

エラー

これらの関数はエラーが発生していない場合には NC_NOERR 値を返します。それ以外の表示が出た場合は、返されたステータスがエラーが発生したことを示します。エラーの原因としては：

- 指定された NetCDF ファイルに対して次元 ＩＤ が無効である。
- 指定された NetCDF ID が開かれている NetCDF ファイルを参照していない。

例

この 例では nc_inq_dim を使用して既存の NetCDF ファイル foo.nc の lat と名づけられた次元の長さと無限長次元の現在の長さをを求めます：

```
#include <netcdf.h>
   …
int status, ncid, latid, recid;
size_t latlength, recs;
char recname[NC_MAX_NAME];
   …
status = nc_open("foo.nc", NC_NOWRITE, &ncid);  /* 読み取り用に開く */
if (status != NC_NOERR) handle_error(status);
status = nc_inq_unlimdim(ncid, &recid); /* 無制限次元の ID 取得 */
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_dimid(ncid, "lat", &latid);  /* lat 次元の ID 取得 */
if (status != NC_NOERR) handle_error(status);
status = nc_inq_dimlen(ncid, latid, &latlength); /* lat の長さ取得 */
if (status != NC_NOERR) handle_error(status);
/* 無制限次元の名前と現在の長さ取得 */
status = nc_inq_dim(ncid, recid, recname, &recs);
if (status != NC_NOERR) handle_error(status);
```

6.4 次元の名前を変更する：nc_rename_dim

関数 nc_rename_dim は開かれた書きこみステータスにある NetCDF ファイル中の次元の名前を変更します。新しい名前が古い名前よりも長い場合には NetCDF ファイルは定義モードになければなりません。他に同名の次元がある場合にはその名前に変更することはできません。

用法

```
int nc_rename_dim(int ncid, int dimid, const char* name);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
dimid	以前の nc_inq_dimid 又は nc_def_dim 呼び出しによって返された次元 ID
name	新規の次元名。

エラー

エラーが発生していない場合には関数 nc_rename_dim は NC_NOERR 値を返します。それ以外の場合には返されたステータスがエラーが発生したことを示します。エラーの原因としては：

- 新規の次元名がすでに他の次元名に使用されている。
- 指定された NetCDF ファイルに対して次元 ＩＤ が無効である。
- 指定された NetCDF ID が開かれている NetCDF ファイルを参照していない。
- 新規の次元名が旧次元名よりも長く、さらに NetCDF ファイルが定義モードに入っていない。

例

この 例では nc_rename_dim を使用して既存の NetCDF ファイル foo.nc 中の次元 lat を latitude に変更します：

```
#include <netcdf.h>
   …
int status, ncid, latid;
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);  /* 書き込み用に開く */
if (status != NC_NOERR) handle_error(status);
   …
status = nc_redef(ncid);  /* 次元の名前を変更するために定義モードに入る */
if (status != NC_NOERR) handle_error(status);
status = nc_inq_dimid(ncid, "lat", &latid);
if (status != NC_NOERR) handle_error(status);
status = nc_rename_dim(ncid, latid, "latitude");
if (status != NC_NOERR) handle_error(status);
status = nc_enddef(ncid); /* 定義モードを抜ける */
```

```
if (status != NC_NOERR) handle_error(status);
```

7　変数

NetCDF ファイルにおける 変数は NetCDF ファイルが生成され、定義モードに あるときに定義されます。再度、定義モードに入ることによって変数を足すことができます。NetCDF 変数には名前、型、及び形があり、変数が定義されるときに指定されます。変数は値を持つこともでき、後にデータモードにある時に確立されます。

通常、変数が最初に定義された段階で 名前、型、及び形は固定されますが、名前は後から変更できます。しかし、変数の型と形は変更できません。無限長次元を使用して定義された変数はその次元に沿っては無限に成長できます。

開かれた NetCDF ファイル中の NetCDF 変数 は *variable ID 変数 ID* という小さな整数によって参照されます。

変数 ID は NetCDF ファイル中に定義された順番になっています。よって、変数 ID は 0, 1, 2,…, という値を取ります。変数 ID から変数名を取得 、またその逆をもできる関数が備わっています。

単位などの性質を指定するために変数に属性（8 章 「属性」（p. 86））を関連付けることもできます。

変数に対してサポートされている機能は：

- 名前・データ型・形を与えて変数を生成する。
- 名前から変数 ID を取得する。
- 変数 ID から変数の名前・データ型・形・属性の数を取得する。
- 変数 ID・番号・値を与えて変数に値を入れる。
- 変数 ID・角の座標・縁の長さ・値のかたまりを与えて変数に値の配列を入れる。
- 変数 ID・角の座標・縁の長さ・ ストライドベクトル・インデックスマッピングベクトル・値のかたまりを与えて部分サンプルされた又はマッピングされた配列断面を変数に入れる。
- 変数 ID と番号を与えて変数からデータの値を取得する。
- 変数 ID・角の座標・縁の長さを与えて変数から値の配列を取得する。
- 変数 ID・角の座標・縁の長さ・ ストライドベクトル・ インデックスマッピングベクトルを与えて部分サンプルされた又はマップされた配列断面を取得する。
- 変数の名前を変更する。

7.1　NetCDF 外部データ型に対応した言語の型

下の表には変数を C インターフェースで定義するために必要な NetCDF 外部データ型とそれに対応する型の定数を示してあります。

NetCDF/CDL Data Type	C API Mnemonic	Bits
byte	NC_BYTE	8
char	NC_CHAR	8
short	NC_SHORT	16
int	NC_INT	32
float	NC_FLOAT	32
double	NC_DOUBLE	64

1 列目には NetCDF 外部データ型がリストされていますが、これは CDL データ型と同じです。2 列目は NetCDF 関数で使用する対応する C プリプロセッサマクロ （プリプロセッサマクロは NetCDF C ヘッダーファイル netcdf.h の中で定義されています。）です。最後の列は対応する型の値を外部表記するために使用されるビット数です。

なお、現行の NetCDF ライブラリには 64 ビット整数又は複数バイトの文字に対応する NetCDF 型はありません。

7.2 変数を生成する： nc_def_var

関数 nc_def_var は定義モードにある開かれた NetCDF ファイルに新たに変数を追加します。NetCDFID・変数名・変数型・次元数・次元 ID のリストを与えると、(引数として) 変数 ID を返します。

用法

```
int nc_def_var (int ncid, const char *name, nc_type xtype,
                int ndims, const int dimids[], int *varidp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
name	変数名。アルファベット文字で始まり、アンダースコア（'_'）を含む零もしくは英数字が続かなければならない。大文字小文字は区別されます。
xtype	前もって定義された NetCDF 外部データ型の集合のひとつ。このパラメータの型 nc_type は NetCDF ヘッダーファイルで定義されています。有効な NetCDF 外部データ型は NC_BYTE, NC_CHAR, NC_SHORT, NC_INT, NC_FLOAT, and NC_DOUBLE です。

ndims	変数の次元の数。例えば、2 は行列、1 はベクトル、0 はをの変数が次元のないスカラーであることを表します。この値は負であったり、前もって定義された 定数 NC_MAX_VAR_DIMS より大きくてもいけない。
dimids	変数次元に対応する次元 ID ndims のベクトル。無制限長次元の ID を含む場合には冒頭にしなければならない。この引数は ndims が零の場合には無視される。
varidp	返された変数 ID の位置を示すポインタ。

エラー

エラーが発生していなければ nc_def_var 機能は NC_NOERR の値を返します。それ以外の場合は、返されたステータスがエラーが発生したことを示します。エラーの原因としては：

- NetCDF ファイルが定義モードになっていない。
- 指定された変数名は既存の別な変数の名前である。
- 指定された型が有効な NetCDF 型ではない。
- 指定された次元の数が負、もしくは NetCDF 変数に許された最大の次元の数を表す定数 NC_MAX_VAR_DIMS より大きい。
- 次元のリストの中の 次元 ID のひとつ、もしくはそれ以上が NetCDF ファイル中で有効ではない次元 ID である。
- 変数の数が NetCDF 変数に許された最大の次元の数を表す定数 NC_MAX_VARS より大きい。
- 指定された NetCDF ID が開いている NetCDF ファイルを参照していない。

例

この例では nc_def_var を使用して、新しい foo.nc という名前の NetCDF ファイル中に、time, lat, and lon の 3 つの次元を持つ、変数名 rh の倍精度型の変数を生成します：

```
#include <netcdf.h>
   …
int  status;                        /* エラーステータス */
int  ncid;                          /* NetCDF ID */
int  lat_dim, lon_dim, time_dim;    /* 次元 ID */
int  rh_id;                         /* 変数 ID */
int  rh_dimids[3];                  /* 変数の形 */
   …
status = nc_create("foo.nc", NC_NOCLOBBER, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
                                    /* 次元の定義 */
status = nc_def_dim(ncid, "lat", 5L, &lat_dim);
if (status != NC_NOERR) handle_error(status);
```

```
status = nc_def_dim(ncid, "lon", 10L, &lon_dim);
if (status != NC_NOERR) handle_error(status);
status = nc_def_dim(ncid, "time", NC_UNLIMITED, &time_dim);
if (status != NC_NOERR) handle_error(status);
   …
                                        /* 変数の定義 */
rh_dimids[0] = time_dim;
rh_dimids[1] = lat_dim;
rh_dimids[2] = lon_dim;
status = nc_def_var (ncid, "rh", NC_DOUBLE, 3, rh_dimids, &rh_id);
if (status != NC_NOERR) handle_error(status);
```

7.3 変数名から変数 ID を取得する：nc_inq_varid

関数 nc_inq_varid 変数名を与えると NetCDF 変数の ID を返す。

用法

```
int nc_inq_varid (int ncid, const char *name, int *varidp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
name	取得したい ID の変数名。
varidp	返された変数 ID の位置を示すポインタ。

エラー

関数 nc_inq_varid はエラーが発生していなければ NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーが発生したことを示します。エラーの原因としては：

- 指定された変数名が指定された NetCDF ファイル内で有効な変数名ではない。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。

例

この 例では nc_inq_varid を使用して rh という名の変数の ID を既存の NetCDF ファイル foo.nc 内で探します：

```
#include <netcdf.h>
   …
int  status, ncid, rh_id;
   …
status = nc_open("foo.nc", NC_NOWRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
```

7.4 ID から変数の情報を取得する：nc_inq_var ファミリー

この 関数のファミリーは変数の ID を与えるとその NetCDF 変数に関する情報を返します。 変数にに関する情報にはその名前・型・次元の数・変数の形を表す変数 ID のリスト・変数に割り当てられている変数属性の数等です。

関数 nc_inq_var はある変数の ID を与えると NetCDF 関数に関する情報をすべて返します。その他の関数はある変数に関する一つの情報を返します。

このほかの関数とは nc_inq_varname, nc_inq_vartype, nc_inq_varndims, nc_inq_vardimid, nc_inq_varnatts. 等です。

用法

```
int nc_inq_var      (int ncid, int varid, char *name, nc_type *xtypep,
                         int *ndimsp, int dimids[], int *nattsp);

int nc_inq_varname  (int ncid, int varid, char *name);

int nc_inq_vartype  (int ncid, int varid, nc_type *xtypep);

int nc_inq_varndims (int ncid, int varid, int *ndimsp);

int nc_inq_vardimid (int ncid, int varid, int dimids[]);

int nc_inq_varnatts (int ncid, int varid, int *nattsp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	変数 ID.
name	返された 変数名。呼び出すためには返された名前用のスペースを確保しておく必要がある。変数名の文字数の最大値は 予め定義された定数 NC_MAX_NAME である。
xtypep	返された変数型（予め定義された NetCDF の外部データ型）の位置を示すポインタ。このパラメーター nc_type の型は NetCDF ヘッダーファイルにより定義されています。有効な NetCDF データ型は NC_BYTE, NC_CHAR, NC_SHORT, NC_INT, NC_FLOAT, と NC_DOUBLE. です。
ndimsp	変数が使用していると定義された次元の数の位置を示すポインタ。例えば、２は行列、１はベクトル、ゼロはその変数が無次元のスカラーであることを示す。
dimids	返された 変数の次元に対応する次元 ID*ndimsp のベクトル 。最低でも返されてくる *ndimsp の整数のベクトルのためのスペースを予め確保しておく必要がある。変数が持ち得る次元の最大数は予め定義された定数 NC_MAX_VAR_DIMS である。

nattsp	返されたこの変数に割り当てられている変数属性の数の位置を示すポインタ。

エラー

これらの関数はエラーが発生していない場合には NC_NOERR 値を返します。それ以外の場合は、返されたステータスがエラーの発生を示します。エラーの原因としては：

- 変数 ID が指定された NetCDF ファイルに対して有効ではない。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。

例

これは nc_inq_var を使用して NetCDF ファイル foo.nc の中の rh という変数に関しての情報を探す例です。

```
#include <netcdf.h>
   …
int  status                         /* エラーステータス */
int  ncid;                          /* NetCDF ID */
int  rh_id;                         /* 変数 ID */
nc_type rh_type;                    /* 変数型 */
int rh_ndims;                       /* 次元の数 */
int  rh_dims[NC_MAX_VAR_DIMS];      /* 変数の形 */
int rh_natts                        /* 属性の数 */
   …
status = nc_open ("foo.nc", NC_NOWRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
/* we don't need name, since we already know it */
status = nc_inq_var (ncid, rh_id, 0, &rh_type, &rh_ndims, rh_dims,
                     &rh_natts);
if (status != NC_NOERR) handle_error(status);
```

7.5 単一のデータ値を書きこむ：nc_put_var1_*type*

関数 nc_put_var1_*type* は指定された*型*（*type*）の単一のデータ値を開かれたステータスでデータモードにある NetCDF ファイルの変数に書きこみます。入力は NetCDF ID・変数 ID・書き加え又は変更するインデックス・データ値です。必要な場合には、その値は変数の外部データ型に変換されます。

用法

```
int nc_put_var1_text  (int ncid, int varid, const size_t index[],
                       const char *tp);
```

```
int nc_put_var1_uchar (int ncid, int varid, const size_t index[],
                       const unsigned char *up);

int nc_put_var1_schar (int ncid, int varid, const size_t index[],
                       const signed char *cp);

int nc_put_var1_short (int ncid, int varid, const size_t index[],
                       const short *sp);

int nc_put_var1_int   (int ncid, int varid, const size_t index[],
                       const int *ip);

int nc_put_var1_long  (int ncid, int varid, const size_t index[],
                       const long *lp);

int nc_put_var1_float (int ncid, int varid, const size_t index[],
                       const float *fp);

int nc_put_var1_double(int ncid, int varid, const size_t index[],
                       const double *dp);
```

ncid	以前の nc_open 又は nc_create 呼び出しからの NetCDF ID。
varid	変数 ID。
index[]	書きこむデータ値のインデックス。インデックスは零に相対的なものであり、例えば 2 次元の変数の最初のデータ値のインデックスは (0,0) になります。インデックスの要素は変数の次元に対応しなければなりません。よって、変数が無制限次元を使用していれば、最初のインデックスはその無制限次元に対応しています。
tp, up, cp, sp, ip, lp, fp, or dp	書きこまれるデータ値へのポインタ。データ値の型が NetCDF 変数型と異なる場合には型の変換が行われます。詳細は 3.3 節 「型変換」 (p.24) にあります。

エラー

関数 nc_put_var1_*type* はエラーが発生していない場合には NC_NOERR 値を返します。それ以外の場合は返されたステータスがエラーが発生したことを示します。エラーの原因としては：

- 変数 ID が指定された NetCDF ファイルでは有効ではない。
- 指定されたインデックスが指定された変数のランクの範囲外である。例えば、負のインデックスや対応する次元長より大きなインデックスを与えるとエラーを発生させる。
- 指定された値が変数の外部データ型で表現できる範囲外である。。
- 指定された NetCDF ファイルがデータモードではなく定義モードにある。
- 指定された NetCDF ID は開かれた NetCDF ファイルを参照していない。

例

この 例は nc_put_var1_double を使用して、既存の NetCDF ファイル foo.nc の変数 rh の (1,2,3) 要素 を 0.5 にします。簡潔にするためにこの例の中では変数 rh の次元が time, lat, lon であることを既知とします。よって、変数 rh に書きこむ値は 2 番目の time 値・3 番目の lat 値・4 番目の lon 値に対応します。

```
#include <netcdf.h>
   …
int  status;                            /* エラーステータス */
int  ncid;                              /* NetCDF ID */
int  rh_id;                             /* 変数 ID */
static size_t rh_index[] = {1, 2, 3}; /* 値の格納場所 */
static double rh_val = 0.5;             /* 格納される値 */
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_put_var1_double(ncid, rh_id, rh_index, &rh_val);
if (status != NC_NOERR) handle_error(status);
```

7.6 全ての値を変数に書きこむ： nc_put_var_*type*

関数 nc_put_var_*type* のファミリーは開かれた NetCDF ファイルの NetCDF 変数にすべての値を書きこみます。これはスカラー変数に値を書き込んだり多次元変数の値が一度にすべて書き込める場合に使用できるもっとも単純なインターフェースです。書きこまれる値は、C インターフェースにおいて最も早く変化する NetCDF 変数が最後 の次元であるという仮定の下に NetCDF 変数と関連付けられます。必要に応じて、値は外部データ型に変換されます。

用法

```
int nc_put_var_text  (int ncid, int varid, const char *tp);

int nc_put_var_uchar (int ncid, int varid, const unsigned char *up);

int nc_put_var_schar (int ncid, int varid, const signed char *cp);

int nc_put_var_short (int ncid, int varid, const short *sp);

int nc_put_var_int   (int ncid, int varid, const int *ip);

int nc_put_var_long  (int ncid, int varid, const long *lp);

int nc_put_var_float (int ncid, int varid, const float *fp);
```

```
int nc_put_var_double(int ncid, int varid, const double *dp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	変数 ID。
tp, up, cp, sp, ip, lp, fp, or dp	書きこまれるデータ値の塊へのポインタ。指定された変数の最後の次元が最も早く変化する順番でデータ値が NetCDF 変数に書き込まれます。データ値の型が NetCDF 変数型と異なる場合には型変換が行われます。詳細については 3.3 節 「型変換」 (p.24) を参照してください。

エラー

関数 nc_put_var_*type* ファミリーに属する関数は、エラーが発生していない場合には NC_NOERR の値を返します。その他の場合には、返されたステータスがエラーが発生したことを示します。エラーの原因としては：

- 変数 ID が指定された NetCDF ファイルでは有効ではない。
- 指定されたデータ値の一つ、もしくはそれ以上が変数の外部型として表現できる値の範囲外である。
- 指定された NetCDF ファイルがデータモードではなく定義モードになっている。
- 指定された NetCDF ID が開いている NetCDF ファイルを参照していない。

例

この 例は nc_put_var_double を使用して、既存の NetCDF ファイル foo.nc の変数 rh の値の全てを 0.5 にします。簡潔にするために、変数 rh の次元が time, lat, と lon であり、 time 値は 3 個、lat 値は 5 個、lon 値は 10 個あることが既知の事とする。

```
#include <netcdf.h>
   …
#define TIMES 3
#define LATS  5
#define LONS  10
int  status;                        /* エラーステータス */
int  ncid;                          /* NetCDF ID */
int  rh_id;                         /* 変数 ID */
double rh_vals[TIMES*LATS*LONS];    /* 値を保持する配列 */
int i;
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
   …
for (i = 0; i < TIMES*LATS*LONS; i++)
    rh_vals[i] = 0.5;
```

```
/* NetCDF 変数に値を書き込む */
status = nc_put_var_double(ncid, rh_id, rh_vals);
if (status != NC_NOERR) handle_error(status);
```

7.7 値の配列を書きこむ： nc_put_vara_*type*

関数 nc_put_vara_*type* は開かれた NetCDF ファイルの NetCDF 変数のに値を書き込みます。書き込む NetCDF 変数の部分は変数の部分配列の隅と縁の長さを与えることによって指定されます。書き込まれる値は NetCDF 変数の 最後 の次元が C インターフェースにおいて最も早く変化するという仮定の下に NetCDF 変数に関連付けられます。NetCDF ファイルはデータモードになっていなければなりません。

用法

```
int nc_put_vara_type  (int ncid, int varid, const size_t start[],
                       const size_t count[], const type *valuesp);

int nc_put_vara_text  (int ncid, int varid, const size_t start[],
                       const size_t count[], const char *tp);

int nc_put_vara_uchar (int ncid, int varid, const size_t start[],
                       const size_t count[], const unsigned char *up);

int nc_put_vara_schar (int ncid, int varid, const size_t start[],
                       const size_t count[], const signed char *cp);

int nc_put_vara_short (int ncid, int varid, const size_t start[],
                       const size_t count[], const short *sp);

int nc_put_vara_int   (int ncid, int varid, const size_t start[],
                       const size_t count[], const int *ip);

int nc_put_vara_long  (int ncid, int varid, const size_t start[],
                       const size_t count[], const long *lp);

int nc_put_vara_float (int ncid, int varid, const size_t start[],
                       const size_t count[], const float *fp);

int nc_put_vara_double(int ncid, int varid, const size_t start[],
                       const size_t count[], const double *dp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	変数 ID。

start	最初のデータ値が書き込まれる変数の中のインデックスを指定する size_t 型整数のベクトル。インデックスは零に相対的なものなので、変数の最初のデータ値のインデックスは (0, 0, … , 0) になります。start のサイズは指定された変数の次元数と同じでなければなりません。又、start の要素は変数の次元に順番に対応していなければならない。従って、記録変数の場合には、最初のインデックスはデータ値を書き込むための開始記録番号に対応する。h
count	書き込むデータ値の塊の各次元の縁の長さを指定する size_t 型整数のベクトル。例えば単一のデータ値を書き込むためには、count を (1, 1, … , 1) と指定します。count の長さが指定された変数の次元数になります。count の要素は変数の次元に対応します。従って、記録変数の場合には count の最初の要素は書きこむ記録数の count に対応します。
tp, up, cp, sp, ip, lp, fp, or dp	書き込むデータ値の塊へのポインタ。指定された変数の最後の次元が最も早く変化するような順番でデータは NetCDF 変数に書きこまれます。データ値の型が NetCDF 変数型と異なる場合には型変換が行われます。詳細については 3.3 節 「型変換」 (p. 24) を参照のこと。

エラー

関数 nc_put_vara_*type* はエラーが発生していない場合には NC_NOERR 値を返します。それ以外の場合には、返されたステータスがエラーの発生を示します。エラーの原因としては：

- 変数 ID が指定された NetCDF ファイルでは無効である。
- 指定された隅のインデックスが指定された変数のランクの範囲外である。例えば、負のインデックス、又は対応する次元長より大きなインデックスはエラーを引き起こします。
- 指定された隅に指定された縁の長さを加えると、参照するデータが指定された変数のランクの範囲外になってしまう。例えば、対応する次元長から隅のインデックスを引いたものより縁の長さが大きい場合にはエラーが発生します。
- 指定された値の一つ、もしくはそれ以上が変数の外部型として表現可能な値の範囲外にある。
- 指定された NetCDF ファイルがデータモードではなく定義モードになっている。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。

例

この 例は nc_put_vara_double を使用して既存の NetCDF ファイル中 foo.nc の変数 rh を 0.5 を加えるか 0.5 にします。簡潔にするためにこの例では、変数 rh の次元は time, lat, と lon であり、time 値は 3 個、lat 値は 5 個、そして lon 値は 10 個ある事が既知だとします。

```
#include <netcdf.h>
   …
```

```
#define TIMES 3
#define LATS  5
#define LONS  10
int  status;                        /* エラーステータス */
int  ncid;                          /* NetCDF ID */
int  rh_id;                         /* 変数 ID */
static size_t start[] = {0, 0, 0}; /* 最初の値から始める */
static size_t count[] = {TIMES, LATS, LONS};
double rh_vals[TIMES*LATS*LONS];   /* 値を保持する配列 */
int i;
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
   …
for (i = 0; i < TIMES*LATS*LONS; i++)
    rh_vals[i] = 0.5;
/* NetCDF 変数に値を書き込む */
status = nc_put_vara_double(ncid, rh_id, start, count, rh_vals);
if (status != NC_NOERR) handle_error(status);
```

7.8 部分サンプルされた配列の値を書き込む： **nc_put_vars_*type***

関数 nc_put_vars_*type* のファミリーに属するものは各々部分サンプルされた（ストライドされた）配列断面を開かれた NetCDF ファイルの変数に書き込みます。部分サンプルされた配列断面は隅、カウントのベクトル、そして ストライドベクトルを与えることによって指定します。NetCDF ファイルはデータモードになっていなければなりません。

用法

```
int nc_put_vars_text  (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const char *tp);

int nc_put_vars_uchar (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const unsigned char *up);

int nc_put_vars_schar (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const signed char *cp);

int nc_put_vars_short (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const short *sp);

int nc_put_vars_int   (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
```

```
                        const int *ip);

int nc_put_vars_long  (int ncid, int varid, const size_t start[],
                        const size_t count[], const ptrdiff_t stride[],
                        const long *lp);

int nc_put_vars_float (int ncid, int varid, const size_t start[],
                        const size_t count[], const ptrdiff_t stride[],
                        const float *fp);

int nc_put_vars_double(int ncid, int varid, const size_t start[],
                        const size_t count[], const ptrdiff_t stride[],
                        const double *dp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	変数 ID。
start	全ての書き込まれるデータの中で先頭のデータ値が書き込まれる変数を指定する、size_t の整数のベクトル。インデックスは零に相対的なので、変数の最初のデータ値のインデックスは (0,0 、... 、0) になります。 start の要素は変数の次元と順番に対応していなければならない。よって、記録変数の場合には、最初のインデックスはデータ値を書き込む開始記録番号に相当する。
count	各次元に沿って選ばれたインデックスの数を指定する size_t の整数のベクトル。例えば、単一の値を書き込む場合には、count を (1, 1, … , 1) と指定する。 count の要素は変数の次元に順番に対応する。よって、記録変数の場合には、count の最初の要素は書き込まれる記録数にのカウントに相当する。
stride	NetCDF 変数の各次元に沿ってのサンプリング間隔を指定する ptrdiff_t 整数のベクトル。ストライドベクトルの要素は NetCDF 変数の次元に順番に対応する。(stride[0] は NetCDF 変数の次元中で最も遅く変化する次元のサンプリング間隔を与える。) サンプリングの間隔は要素の型独立の単位で指定される。(値が 1 は対応する次元に沿って隣接する NetCDF 変数を選定する。値が 2 ならば NetCDF 変数の次元に沿って一つおきの値にアクセスする。) ストライドが NULL (0) の場合には各次元に沿って (1, 1, … , 1)、つまり隣接した値にアクセスしていくとデフォルトで定義されている。
tp, up, cp, sp, ip, lp, fp, or dp	データ値の塊を指し示すポインタ。NetCDF 変数にデータが書き込まれていく順番は指定された最後の変数の次元が最も早く変化していくような順番である。もしデータの型が NetCDF 変数型と異なる場合には型変換が行われる。詳細については 3.3 節 「型変換」 (p.24) を参照のこと。

エラー

エラーが発生していない場合には nc_put_vars_*type* は NC_NOERR の値を返します。その他の場合には、返されたステータスがエラーの発生を示します。エラーの原因としては：

- 変数 ID が指定された NetCDF ファイルに対して有効ではない。
- 指定された start・count・stride が領域外のインデックスを生成してしまう。
- 指定された値のうち、少なくとも一つが変数の外部データ型で表現可能な値の範囲外である。
- 指定された NetCDF ファイルがデータモードではなく定義モードになっている。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照しない。

例

この例は nc_put_vars_float を使用して、内部配列から、rh と言う名の NetCDF 変数を一つおきに書き込んでいく例です。変数 rh は C 宣言文 float rh[4][6] において定義されています。(次元の大きさに注目してください。)

```
#include <netcdf.h>
   …
#define NDIM 2                  /* NetCDF 変数のランク */
int ncid;                       /* NetCDF ID */
int status;                     /* エラーステータス */
int rhid;                       /* 変数 ID */
static size_t start[NDIM]       /* NetCDF 変数のスタート地点： */
                 = {0, 0};      /* 最初の要素 */
static size_t count[NDIM]       /* 内部配列のサイズ：全体 */
                   = {2, 3};    /* （部分サンプルされた）NetCDF 変数 ( */
static ptrdiff_t stride[NDIM] /* 変数の部分サンプル間隔： */
                 = {2, 2};      /* NetCDF 要素に一つおきにアクセス */
float rh[2][3];                 /* 部分サンプルのサイズを記録 */
                                /* NetCDF 変数の次元 */
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid(ncid, "rh", &rhid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_put_vars_float(ncid, rhid, start, count, stride, rh);
if (status != NC_NOERR) handle_error(status);
```

7.9 マップされた配列の値を書き込む：nc_put_varm_*type*

関数 nc_put_varm_*type* のファミリーはマップされた配列断面の値を開かれた NetCDF ファイルの変数に書き込んでいきます。マップされた配列断面は隅の位置・カウントのベクトル・ストライドベクトル・インデックスマッピングベクトルを与えることに

よって指定されます。インデックスマッピングベクトルとは整数のベクトルで、NetCDF 変数の次元と内部データ配列のメモリ内構造間のマッピングを指定するベクトルです。データ配列に関する次元の順番や長さに関する仮定は一切なされません。NetCDF ファイルはデータモードになっていなければなりません。

用法

```
int nc_put_varm_text  (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], const char *tp);

int nc_put_varm_uchar (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], const unsigned char *up);

int nc_put_varm_schar (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], const signed char *cp);

int nc_put_varm_short (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], const short *sp);

int nc_put_varm_int   (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], const int *ip);

int nc_put_varm_long  (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], const long *lp);

int nc_put_varm_float (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], const float *fp);

int nc_put_varm_double(int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], const double *dp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	変数 ID。
start	全ての書き込まれるデータの中で先頭のデータ値が書き込まれる変数を指定する、size_t の整数のベクトル。インデックスは零に相対的なので、変数の最初のデータ値のインデックスは (0,0 、... 、0) になります。 start の要素は変数の次元と順番に対応していなければならない。よって、記録変数の場合には、最初のインデックスはデータ値を書き込む開始記録番号に相当する。

`count` 各次元に沿って選ばれたインデックスの数を指定する size_t の整数のベクトル。例えば、単一の値を書き込む場合には、`count` を `(1, 1, … , 1)` と指定する。 `count` の要素は変数の次元に順番に対応する。よって、記録変数の場合には、`count` の最初の要素は書き込まれる記録数にのカウントに相当する。

`stride` NetCDF 変数の各次元に沿ってのサンプリング間隔を指定する ptrdiff_t 型整数のベクトル。ストライドベクトルの要素は NetCDF 変数の次元に順番に対応する。(stride[0] は NetCDF 変数の次元中で最も遅く変化する次元のサンプリング間隔を与える。) サンプリングの間隔は要素の型独立の単位で指定される。(値が 1 は対応する次元に沿って隣接する NetCDF 変数をせん定する。値が 2 ならば NetCDF 変数の次元に沿って一つおきの値にアクセスする。) ストライドが `NULL` (0) の場合には各次元に沿って `(1, 1, … , 1)`、つまり隣接した値にアクセスしていくとデフォルトで定義されている。

`imap` NetCDF 変数と内部データ配列のメモリ内構造間のマッピングを指定する ptrdiff_t 型の整数ベクトル。インデックスマッピングベクトルの要素は NetCDF 変数の次元と順番に対応します。(imap[0] は NetCDF 変数の次元のうち、最も遅く変化する次元に対応する内部配列の要素間の距離を与えます。) 要素間の距離は型独立な要素の単位で示されます。(メモリ内で隣接している位置にある内部要素間の距離は 1 であり、NetCDF 2 の場合のように要素のバイト長ではありません。) 引数 `NULL` はメモリ内の値が関連付けられる NetCDF 変数と同じ構造を持っていることを示します。

`tp, up, cp, sp, ip, lp, fp, or dp` データ値の位置を計算するために使用される位置を示すポインタ。データ値は呼び出された関数に適当な型でなくてはならない。データが NetCDF 変数型と異なる場合には型変換が行なわれます。詳細については 3.3 節 「型変換」 (p.24) を参照してください。

エラー

エラーが発生していない場合には関数 `nc_put_varm_`*type* は `NC_NOERR` の値を返します。それ以外の場合には、返されたステータスがエラーが発生したことを示します。エラーの原因としては：

- 変数 ID が指定された NetCDF ファイルに対して有効ではない。
- 指定された `start`, `count`, 及び `stride` では範囲外のインデックスを生じてしまう。`imap` ベクトルにおいてはエラーチェックが出来ないことに注意してください。
- 指定された値のうち、少なくとも一つが変数の外部データ型で表現可能な値の範囲外である。
- 指定された NetCDF ファイルがデータモードではなく定義モードになっている。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照しない。

例

以下の imap ベクトルは 4x3x2 NetCDF 変数と同じ形の内部配列を簡潔な方法でマップします。

```
float a[4][3][2];       /* NetCDF 変数と同じ形 */
int   imap[3] = {6, 2, 1};
                        /* NetCDF 次元             要素間距離 */
                        /* ----------------       ---------------------- */
                        /* 最も早く変化             1                    */
                        /* 中間                     2 (=imap[2]*2)       */
                        /* 最も遅く変化             6 (=imap[1]*3)       */
```

上記の例で imap ベクトルと併せて nc_put_varm_float 使用すると、単に nc_put_var_float を使用した場合と同じ結果が得られます。

この例では nc_put_varm_float を使用して、転置された内部配列から、NetCDF 変数 rh を書きます。変数 rh は C 宣言文 float rh[6][4] で定義されています。(次元の大きさに注意してください。)

```
#include <netcdf.h>
   …
#define NDIM 2              /* NetCDF 変数のランク */
int ncid;                   /* NetCDF ID */
int status;                 /* エラーステータス */
int rhid;                   /* 変数 ID */
static size_t start[NDIM]   /* NetCDF 変数スタート地点 : */
                 = {0, 0};  /* 最初の要素 */
static size_t count[NDIM]   /* 内部配列のサイズ : NetCDF 全体 */
                 = {6, 4};  /* 変数 ; 順番は NetCDF 変数に対応  */
                            /*  -- 内部配列の順番ではない */
static ptrdiff_t stride[NDIM]/* 変数の部分サンプル間隔 : */
                 = {1, 1};  /* 全ての NetCDF 要素をサンプル */
static ptrdiff_t imap[NDIM] /* 内部配列の要素間距離 ; */
                 = {1, 6};  /* 置換されなければ {4, 1} */
float rh[4][6];             /* NetCDF 変数次元の置き換えに注意 */
                            /* 次元 */
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid(ncid, "rh", &rhid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_put_varm_float(ncid, rhid, start, count, stride, imap, rh);
if (status != NC_NOERR) handle_error(status);
```

この例では nc_put_varm_float を使用して転置された内部配列から同じ NetCDF 変数からなる部分サンプル配列を、NetCDF 変数を一つおきのポイントに書き込むことによっ

て作成します。

```
#include <netcdf.h>
   …
#define NDIM 2                  /* NetCDF 変数のランク */
int ncid;                       /* NetCDF ID */
int status;                     /* エラーステータス */
int rhid;                       /* 変数 ID */
static size_t start[NDIM]       /* NetCDF 変数のスタート地点： */
                 = {0, 0};      /* 最初の要素 */
static size_t count[NDIM]       /* 内部配列のサイズ： 全体 */
                  = {3, 2};     /*（部分サンプルされた）NetCDF 変数； 次元の順番は */
                                /* NetCDF 変数の順番に対応 */
                                /* -- 内部配列の順番ではない */
static ptrdiff_t stride[NDIM] /* 変数部分サンプル間隔： */
                 = {2, 2};      /* NetCDF 要素を一つおきにサンプル */
static ptrdiff_t imap[NDIM]   /* 内部配列の要素間距離； */
                 = {1, 3};      /* 置換しなければ {2, 1} */
float rh[2][3];                 /* （部分サンプルされた）NetCDF 変数の */
                                /* 次元が置換されていることに注意 */
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid(ncid, "rh", &rhid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_put_varm_float(ncid, rhid, start, count, stride, imap, rh);
if (status != NC_NOERR) handle_error(status);
```

7.10 単一のデータ値を読み取る： **nc_get_var1_*type***

関数 nc_get_var1_*type* は開かれたデータモードにある NetCDF ファイルの変数から単一のデータ値を取得します。入力は NetCDF ID・変数 ID・取得する値を指定する多次元のインデックス・データ値が読み込まれる位置のアドレスです。この値は必要に応じて変数の外部データ型から変換されます。

用法

```
int nc_get_var1_text  (int ncid, int varid, const size_t index[],
                       char *tp);

int nc_get_var1_uchar (int ncid, int varid, const size_t index[],
                       unsigned char *up);

int nc_get_var1_schar (int ncid, int varid, const size_t index[],
                       signed char *cp);

int nc_get_var1_short (int ncid, int varid, const size_t index[],
```

```
                        short *sp);

int nc_get_var1_int   (int ncid, int varid, const size_t index[],
                        int *ip);

int nc_get_var1_long  (int ncid, int varid, const size_t index[],
                        long *lp);

int nc_get_var1_float (int ncid, int varid, const size_t index[],
                        float *fp);

int nc_get_var1_double(int ncid, int varid, const size_t index[],
                        double *dp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	変数 ID。
index[]	読み込まれるデータ値のインデックス。インデックスは零に相対的であるので、２次元変数の最初のデータ値のインデックスは (0,0) になります。 index の要素は変数の次元に対応していなければなりません。よって、記録変数の場合には、最初のインデックスは記録番号になります。
tp, up, cp, sp, ip, lp, fp, or, dp	データ値が読み込まれる位置へのポインタ。もしデータ型が NetCDF 変数型と異なる場合には、型変換が行なわれます。詳細については 3.3 節 「型変換」 (p. 24) を参照してください。

エラー

エラーが発生していない場合には関数 nc_get_var1_*type* は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーが発生したことを示します。エラーの原因としては：

- 変数 ID が指定された NetCDF ファイルに対して有効ではない。
- 指定されたインデックスが指定された変数のランクの範囲外であった。例えば、負のインデックスや、対応する次元の長さより大きなインデックスを与えるとエラーを引き起こす。
- 値が望まれるデータ型で表現可能な値の範囲外であった。
- 指定された NetCDF ファイルがデータモードではなく定義モードにあった。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照しない。

例

この例では nc_get_var1_double を使用して、既存の NetCDF ファイル foo.nc から変数 rh の (1,2,3) 要素を取得します。簡潔にするために、この例では rh の次元が time, lat, と lon であることが既知であるとし、よって、取得したいのは２番目の time 値、３番目の lat 値、そして４番目の lon 値となります。

```
#include <netcdf.h>
   …
int  status;                              /* エラーステータス */
int ncid;                                 /* NetCDF ID */
int rh_id;                                /* 変数 ID */
static size_t rh_index[] = {1, 2, 3};     /* 値を取得する場所 */
double rh_val;                            /* 値を置く場所 */
   …
status = nc_open("foo.nc", NC_NOWRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_get_var1_double(ncid, rh_id, rh_index, &rh_val);
if (status != NC_NOERR) handle_error(status);
```

7.11 全変数を読み取る： nc_get_var_*type*

関数 nc_get_var_*type* のファミリーは開かれた NetCDF ファイルの変数の値を全て読みます。これは、スカラー変数や多次元変数の値を全て一度で読むためには最も簡単なインターフェースです。変数は隣接する位置に 最後の 次元が最も早く変化するように次々と書き込まれていきます。NetCDF ファイルはデータモードになければなりません。

用法

```
int nc_get_var_text  (int ncid, int varid, char *tp);

int nc_get_var_uchar (int ncid, int varid, unsigned char *up);

int nc_get_var_schar (int ncid, int varid, signed char *cp);

int nc_get_var_short (int ncid, int varid, short *sp);

int nc_get_var_int   (int ncid, int varid, int *ip);

int nc_get_var_long  (int ncid, int varid, long *lp);

int nc_get_var_float (int ncid, int varid, float *fp);

int nc_get_var_double(int ncid, int varid, double *dp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	変数 ID。
tp, up, cp, sp, ip, lp, fp, or dp	データ値が読み込まれる位置のポインタ。もし、データ型が NetCDF 変数型と異なる場合には型変換が行なわれます。詳細については 3.3 節 「型変換」 (p.24) を参照のこと。

エラー

エラーが発生していなければ、nc_get_var_*type* は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因としては下記が挙げられます。

- 変数 ID が指定された NetCDF ファイルにおいて無効である。
- 一つ、もしくは複数の値が、指定された型によって表わせる値の範囲を超えている。
- 指定された NetCDF ファイルがデータモードではなく定義モードにある。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照しない。

例

この例では nc_get_var_double を使用して既存の NetCDF ファイル foo.nc の変数 rh の値を全て読み取ります。簡潔にするためにこの例では、変数 rh の次元は time, lat, 及び lon であり、 time 値は３個、 lat 値は５個､そして， lon 値は 10 個あることを既知とします。

```
#include <netcdf.h>
   …
#define TIMES 3
#define LATS 5
#define LONS 10
int  status;                        /* エラーステータス */
int ncid;                           /* NetCDF ID */
int rh_id;                          /* 変数 ID */
double rh_vals[TIMES*LATS*LONS];    /* 値を保持する配列 */
   …
status = nc_open("foo.nc", NC_NOWRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
   …
/* NetCDF 変数から値を読み取る */
status = nc_get_var_double(ncid, rh_id, rh_vals);
if (status != NC_NOERR) handle_error(status);
```

7.12 値の配列を読む： nc_get_vara_*type*

関数 nc_get_vara_*type* のファミリーは開かれた NetCDF ファイルの変数から値の配列を読み取ります。配列は隅の位置と各辺の長さを表わすベクトルを与えて指定します。値は 最後の 次元が最も早く変化するように、次々と読み込まれます。

用法

```
int nc_get_vara_text  (int ncid, int varid, const size_t start[],
                       const size_t count[] char *tp);

int nc_get_vara_uchar (int ncid, int varid, const size_t start[],
                       const size_t count[] unsigned char *up);

int nc_get_vara_schar (int ncid, int varid, const size_t start[],
                       const size_t count[] signed char *cp);

int nc_get_vara_short (int ncid, int varid, const size_t start[],
                       const size_t count[] short *sp);

int nc_get_vara_int   (int ncid, int varid, const size_t start[],
                       const size_t count[] int *ip);

int nc_get_vara_long  (int ncid, int varid, const size_t start[],
                       const size_t count[] long *lp);

int nc_get_vara_float (int ncid, int varid, const size_t start[],
                       const size_t count[] float *fp);

int nc_get_vara_double(int ncid, int varid, const size_t start[],
                       const size_t count[] double *dp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	変数 ID。
start	全ての書き込まれるデータの中で先頭のデータ値が読み込まれる変数を指定する、size_t の整数のベクトル。インデックスは零に相対的なので、変数の最初のデータ値のインデックスは (0,0 、... 、0) になります。start の長さは指定された変数の次元数と一致していなければなりません。 start の要素は指定された変数の次元と順番に対応しています。よって、記録変数の場合には、最初のインデックスはデータ値を読み取る開始記録番号に相当する。
count	読み取るデータ値の塊の各次元の辺の長さを指定する size_t の整数ベクトル。例えば単一の値を読み取る場合には、 count を (1, 1, … , 1) と指定すればよい。 count の長さは指定された変数の次元の数に相当する。count の要素は変数の次元に順番に対応する。よって、記録変数の場合には count の最初の要素が読み取る記録数の総計に対応する。
tp, up, cp, sp, ip, lp, fp, or, dp	データ値が読み込まれる位置へのポインタ。データ型が NetCDF 変数型と異なる場合には型変換が行なわれます。詳細は 3.3 節 「型変換」 (p.24) を参照のこと。

エラー

エラーが発生していなければ、関数 nc_get_vara_*type* は NC_NOERR の値を返します。それ以外の場合は、返されたステータスがエラーが発生したことを示します。エラーの原因としては：

- 変数 ID が指定された NetCDF ファイルに対して有効ではない。
- 指定された隅のインデックスが指定された変数のランクの範囲外であった。例えば、負のインデックス、もしくは対応する次元の長さよりも大きいインデックスなどを与えるとエラーが発生する。
- 指定された縁の長さを指定された隅に加えると、参照すべきデータ値が指定された変数のランクの範囲外になってしまう。例えば、指定された次元長よりも大きい縁の長さから隅のインデックスを引くとエラーを生じる。
- 値の一つもしくはそれ以上が望まれる型で表現できる値の範囲外になってしまう。
- 指定された NetCDF ファイルがデータモードではなく定義モードになっている。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照しない。

例

この例では nc_get_vara_double を使用して 既存の NetCDF ファイル foo.nc の変数 rh から値の配列を読み込みます。簡潔にするためにこの例では変数 rh の次元は time, lat, と lon であり、time 値は3個、lat 値は5個、そして lon 値は10個あることを既知とします。

```
#include <netcdf.h>
   …
#define TIMES 3
 #define LATS 5
#define LONS 10
int  status;                        /* エラーステータス */
int ncid;                           /* NetCDF ID */
int rh_id;                          /* 変数 ID */
static size_t start[] = {0, 0, 0}; /* 最初の値から開始 */
static size_t count[] = {TIMES, LATS, LONS};
double rh_vals[TIMES*LATS*LONS];   /* 値を保持する配列 */
   …
status = nc_open("foo.nc", NC_NOWRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
   …
/* NetCDF 変数から値を読み取る */
status = nc_get_vara_double(ncid, rh_id, start, count, rh_vals);
if (status != NC_NOERR) handle_error(status);
```

7.13 部分サンプルされた配列の値を読み取る： **nc_get_vars_*type***

関数 nc_get_vars_*type* のファミリーは開かれた NetCDF ファイルから部分サンプルされた （ストライドした）NetCDF 変数の配列断面 の値を読みます。部分サンプルされた配列断面は隅・縁の長さを示すベクトル・ ストライドベクトルを与えることによって指定されます。値は NetCDF 変数の中で最初の 次元が最も早く変化するように読まれます。NetCDF ファイルはデータモードに無くてはなりません。

用法

```
int nc_get_vars_text  (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       char *tp);

int nc_get_vars_uchar (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       unsigned char *up);

int nc_get_vars_schar (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       signed char *cp);

int nc_get_vars_short (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       short *sp);

int nc_get_vars_int   (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       int *ip);

int nc_get_vars_long  (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       long *lp);

int nc_get_vars_float (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       float *fp);

int nc_get_vars_double(int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       double *dp)
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	変数 ID。

start	先頭のデータ値が読み込まれる変数を指定する、size_t の整数のベクトル。インデックスは零に相対的なので、変数の最初のデータ値のインデックスは（0,0 、... 、0）になります。start の要素は指定された変数の次元と順番に対応しています。よって、記録変数の場合には、最初のインデックスはデータ値を読み取る開始記録番号に相当する。
count	各次元に沿って幾つのインデックスが選定されるかを指定する size_t の整数ベクトル。例えば単一の値を読み取る場合には、count を (1, 1, … , 1)と指定すればよい。 count の要素は変数の次元に順番に対応する。よって、記録変数の場合には count の最初の要素が読み取る記録数の総計に対応する。
stride	各次元ごとに選定されたインデックスの間隔を指定するサイズ ptrdiff_t の整数ベクトル 。ストライドベクトルの要素は変数の次元に順番に対応する。値が１ならば、対応する次元の NetCDF 変数で隣接する値をアクセスする。値が２の場合には対応する NetCDF 変数の値を一つおきにアクセスする。 NULL ストライド引数は (1, 1, … , 1) として扱われる。
tp, up, cp, sp, ip, lp, fp, or, dp	データ値が読み込まれる位置へのポインタ。データ型が NetCDF 変数型と異なる場合には型変換が行なわれます。詳細は 3.3 節 「型変換」（p.24）を参照のこと。

エラー

エラーが発生していない場合には、関数 nc_get_vars_*type* は NC_NOERR の値を返します。それ以外の場合には返されたステータスがエラーの発生を示します。絵 r- の原因としては：

- 変数 ID が指定された NetCDF ファイルに対して有効ではない。
- 指定された start・count・stride では範囲外のインデックスを生成してしまう。
- 一つもしくはそれ以上の値が希望の型で表わせる値の範囲外である。
- 指定された NetCDF ファイルがデータモードではなく定義モードになっている。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照しない。

例

この例では 関数 nc_get_vars_double を使用して、既存の NetCDF ファイル foo.nc の変数 rh の各次元から値を一つおきに読みます。簡潔にするために、この例では rh の次元が time, lat, と lon であり、 time 値は３個、lat 値は５個、そして values, and ten lon 値が 10 個あることが既知のこととします。

```
#include <netcdf.h>
   …
#define TIMES 3
#define LATS  5
```

```
#define LONS 10
int  status;                            /* エラーステータス */
int ncid;                               /* NetCDF ID */
int rh_id;                              /* 変数 ID */
static size_t start[] = {0, 0, 0};      /* 最初の値から開始 */
static size_t count[] = {TIMES, LATS, LONS};
static ptrdiff_t stride[] = {2, 2, 2};/* 一つおきの値 */
double data[TIMES][LATS][LONS];         /* 値を保持する配列 */
 …
status = nc_open("foo.nc", NC_NOWRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
 …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
 …
/* NetCDF 変数の部分サンプルの値を配列に読み込む */
status = nc_get_vars_double(ncid, rh_id, start, count, stride,
                            &data[0][0][0]);
if (status != NC_NOERR) handle_error(status);
 …
```

7.14 マップされた配列の値を読む： nc_get_varm_*type*

関数 nc_get_varm_*type* のファミリーは開かれた NetCDF ファイルの NetCDF 変数からマップされた配列断面を読みます。マップされた配列断面は隅・縁の長さ・ ストライドベクトル・インデックスマッピングベクトル を与えることによって指定されます。インデックスマッピングベクトルとは NetCDF 変数と内部データ配列のメモリ内構造との間のマッピングを指定する整数ベクトルです。データ配列に関しては順番や長さなどについていかなる仮定もされません。

用法

```
int nc_get_varm_text  (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], char *tp);

int nc_get_varm_uchar (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], unsigned char *up);

int nc_get_varm_schar (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], signed char *cp);

int nc_get_varm_short (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], short *sp);

int nc_get_varm_int   (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
```

```
                      const ptrdiff_t imap[], int *ip);

int nc_get_varm_long  (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], long *lp);

int nc_get_varm_float (int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], float *fp);

int nc_get_varm_double(int ncid, int varid, const size_t start[],
                       const size_t count[], const ptrdiff_t stride[],
                       const ptrdiff_t imap[], double *dp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	変数 ID。
start	先頭のデータ値が読み込まれる変数を指定する、size_t の整数のベクトル。インデックスは零に相対的なので、変数の最初のデータ値のインデックスは（0,0 、... 、0）になります。 start の要素は指定された変数の次元と順番に対応しています。よって､記録変数の場合には、最初のインデックスはデータ値を読み取る開始記録番号に相当する。
count	各次元に沿って幾つのインデックスが選定されるかを指定する size_t の整数ベクトル。例えば単一の値を読み取る場合には、count を (1, 1, … , 1)と指定すればよい。 count の要素は変数の次元に順番に対応する。よって、記録変数の場合には count の最初の要素が読み取る記録数の総計に対応する。
stride	各次元ごとに選定されたインデックスの間隔を指定するサイズ ptrdiff_t の整数ベクトル 。ストライドベクトルの要素は変数の次元に順番に対応する。値が１ならば、対応する次元の NetCDF 変数で隣接する値をアクセスする。値が２の場合には対応する NetCDF 変数の値を一つおきにアクセスする。 NULL ストライド引数は (1, 1, … , 1) として扱われる。
imap	NetCDF 変数の次元と内部データ配列とのメモリ内構造間のマッピングを指定する整数ベクトル。imap[0] は最も遅く変化する NetCDF 変数の次元に対応する内部配列の要素と要素間の距離を示します。imap[n-1] （n は NetCDF 変数のランク ）最も早く変化する NetCDF 変数の次元に対応する内部配列の要素と要素の間の距離を示します。この二つの要素の間にある他の要素が他の NetCDF 変数の次元に対応することは自明です。要素間の距離は要素の型独立の単位で指定されます。(隣接する位置にある内部メモリ間の距離は１であり、NetCDF2 のように要素のバイト長ではありません。)

`tp, up, cp, sp, ip, lp, fp, or, dp`

データ値が読み込まれる位置へのポインタ。データ型は呼び出された関数に適した型でなければなりません。データ型が NetCDF 変数型と異なる場合には型変換が行なわれます。詳細は 3.3 節 「型変換」(p.24) を参照のこと。

エラー

エラーが発生していなければ、関数 `nc_get_varm_`*type* は `NC_NOERR` の値を返します。その他の場合は返されたステータスがエラーの発生を示します。エラーの原因としては：

- 変数 ID が指定された NetCDF ファイルでは有効ではない。
- 指定された `start`, `count`, と `stride` では範囲外のインデックスを生成してしまう。`imap` ベクトルではエラーチェックが出来ないことに注意してください。
- 一つもしくはそれ以上の値が希望された型で表現し得る範囲外である。
- 指定された NetCDF がデータモードではなく定義モードになっている。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照しない。

例

次の `imap` ベクトルは 4x3x2 の NetCDF 変数と同じ形の内部配列を自明な形でマップします。

```
float a[4][3][2];       /* NetCDF 変数と同じ形 */
size_t imap[3] = {6, 2, 1};
                        /* NetCDF 次元             要素間距離 */
                        /* ----------------       ---------------------- */
                        /* 最も早く変化する          1                   */
                        /* 中間                     2 (=imap[2]*2)      */
                        /* 最も遅く変化する         6 (=imap[1]*3)      */
```

上記の `imap` ベクトルと `nc_get_varm_float` とを使用した場合と、単に `nc_get_var_float` を使用した場合とでは同じ結果が得られます。

この例では `nc_get_varm_float` を使用して C 宣言文 `float rh[6][4]` （次元のサイズと順番に注目）で表わされた NetCDF 変数 `rh` を移項します。

```
#include <netcdf.h>
   …
#define NDIM 2                  /* rank of NetCDF 変数のランク */
int ncid;                       /* NetCDF ID */
int status;                     /* エラーステータス */
int rhid;                       /* 変数 ID */
static size_t start[NDIM]       /* NetCDF 変数のスタート地点 : */
                 = {0, 0};      /* 最初の要素 */
static size_t count[NDIM]       /* 内部配列のサイズ : NetCDF 変数全体 ; */
                 = {6, 4};      /* 順番は NetCDF 変数の順番に対応 */
```

```
                                /*  -- 内部配列の順番ではない */
static ptrdiff_t stride[NDIM] /* 変数の部分サンプル間隔： */
                  = {1, 1};    /* NetCDF 要素を一つおきにサンプル */
static ptrdiff_t imap[NDIM]   /* 内部配列の要素間距離； */
                  = {1, 6};    /* 置換しなければ {4, 1}  */
float rh[4][6];               /* NetCDF 変数の次元が */
                              /* 置換されている点に注意 */
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid(ncid, "rh", &rhid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_get_varm_float(ncid, rhid, start, count, stride, imap, rh);
if (status != NC_NOERR) handle_error(status);
```

関数 nc_get_varm_float を使用したこの別の例では、NetCDF 変数の点を一つおきにアクセスして同じ NetCDF 変数を移項すると同時に部分サンプルします。

```
#include <netcdf.h>
   …
#define NDIM 2                /* NetCDF 変数のランク */
int ncid;                     /* NetCDF ID */
int status;                   /* エラーステータス */
int rhid;                     /* 変数 ID */
static size_t start[NDIM]     /* NetCDF 変数のスタート地点： */
                  = {0, 0};   /* 最初の要素 */
static size_t count[NDIM]     /* 内部配列のサイズ： 全ての */
                    = {3, 2}; /* （部分サンプルされた）NetCDF 変数；次元の */
                              /* 順番は NetCDF 変数に対応 */
                              /* -- 内部配列ではない */
static ptrdiff_t stride[NDIM]/* 変数の部分サンプル間隔： */
                  = {2, 2};   /* NetCDF 要素を一つおきにサンプル */
static ptrdiff_t imap[NDIM]  /* 内部配列の要素間距離； */
                  = {1, 3};   /* 置換しなければ {2, 1}  */
float rh[2][3];               /* （部分サンプルされた）NetCDF 変数次元が */
                              /* 置換されていることに注意 */
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid(ncid, "rh", &rhid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_get_varm_float(ncid, rhid, start, count, stride, imap, rh);
if (status != NC_NOERR) handle_error(status);
```

7.15 文字列値を読み書きする

文字 列 は 基本的な NetCDF 外部データ型ではありません。なぜならば、FORTRAN では可変長の文字列の抽象化をサポートしていないからです。(FORTRAN の `LEN` 関数は文字列の動的な長さではなく、静的な長さを返します。) その結果、NetCDF インターフェースでは文字列は単一のオブジェクトとして読み書きすることが出来ません。文字列は文字の配列として扱わなければならないのです。それ故、NetCDF ファイルの変数データとして文字列を読み書きするためには配列アクセスをしなければなりません。さらに、NetCDF インターフェースでは可変長の文字列は規約による場合を除いてはサポートされていません。例えば、零バイトを文字列を終了するものとして扱うことは可能ですが、NetCDF 変数に読み書きされる文字列の長さを明示しなければなりません。

文字列を属性値として扱えば使用しやすくなる。それは文字列がアクセスする際に一つの単位として扱われるからである。しかしながら、文字列の属性値の値はやはり固有の長さを持つ文字の配列であり、その長さは属性が定義されるときに指定される必要がある。

文字列値を持つ変数を定義する際には，*文字列位置次元 character-position dimension* を最も早く変化する次元として使用しなければならない。(C の変数において最後の 次元) 文字列次元の長さは文字列変数に格納されるあらゆる文字列の最大長である。最大長の列を格納するスペースは、使用するか否かにかかわらず、文字列変数のディスク表現の中に割り当てられる。仮に、２個以上の変数の最大長が同じである場合には、変数の形を定義するにあたって同じ文字位置次元を使用しても良い。

文字列変数に文字列の値を書き込むには、全変数アクセスもしくは配列アクセスを使用します。後者を使用する場合には隅と縁の長さのベクトルの両方を指定する必要があります。文字位置次元の隅は C. において 0 です。もし書き込む列の長さが `n` と仮定すると、縁の長さのベクトルは文字位置次元に `n` を指定し、他の次元には全て１を指定します：`(1, 1, … , 1, n)`。

C においては、固定長の列は null 文字無しに NetCDF ファイルに書き込むことが出来ます。可変長の列は null 文字が必要となります。それによって、後で読み取られる際に目的の列の長さが明確になるからです。

この例では、記録変数 `tx` を文字列用に定義して、３番目の記録に `nc_put_vara_text` を使用して文字列値を格納します。ここで、この文字列変数とその値は `time` という無制限記録次元を持つ既存の NetCDF ファイル `foo.nc` に書き加えると仮定します。

```
#include <netcdf.h>
   …
int  ncid;            /* NetCDF ID */
int  chid;            /* 文字の位置の次元 ID */
int  timeid;          /* 記録次元の次元 ID */
int  tx_id;           /* 変数 ID */
#define TDIMS 2       /* tx 変数のランク */
int tx_dims[TDIMS];   /* 変数の形 */
size_t tx_start[TDIMS];
```

```
size_t tx_count[TDIMS];
static char tx_val[] =
        "example string"; /* 置かれる文字列 */
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
status = nc_redef(ncid);       /*  定義モードに入る */
if (status != NC_NOERR) handle_error(status);
   …
/* 最大長が 40 文字の文字列における文字の位置の次元を定義する */
status = nc_def_dim(ncid, "chid", 40L, &chid);
if (status != NC_NOERR) handle_error(status);
   …
/* 文字列変数を定義する */
tx_dims[0] = timeid;
tx_dims[1] = chid;    /* 最後の文字の位置の次元 */
status = nc_def_var (ncid, "tx", NC_CHAR, TDIMS, tx_dims, &tx_id);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_enddef(ncid);       /* 定義モードを抜ける */
if (status != NC_NOERR) handle_error(status);
   …
/* 記録 3 において tx_val を tx NetCDF 変数に書き込む */
tx_start[0] = 3;      /* 書き込む記録の数 */
tx_start[1] = 0;      /* 変数の先頭から開始 */
tx_count[0] = 1;      /* 一つの記録のみ書き込む */
tx_count[1] = strlen(tx_val) + 1;  /* 書き込む文字数 */
status = nc_put_vara_text(ncid, tx_id, tx_start, tx_count, tx_val);
if (status != NC_NOERR) handle_error(status);
```

7.16 フィル値

開かれた NetCDF ファイルに書き込まれたことのない値を読み取ろうとしたならば何が起こるでしょう？必ずエラーが発生し、エラーメッセージもしくはエラーステータスが返されると思われがちです。確かに、開かれていない NetCDF ファイルからデータを読もうとした場合、指定された NetCDF ファイルにおいてその変数 ID が有効でない場合、または指定されたインデックスが指定された変数の次元長で定義された領域外にある場合にはエラーが*発生*します。しかし、それ以外の場合には、書き込まれていない値を読もうとすると、初めに NetCDF 変数が書かれたときに未定義の全ての値を埋めるための使用される特別な*フィル値 fill value* が返されます。

このフィル値を無視して NetCDF 外部データ型の全領域を使うことも出来ますが、その場合には読む前に全てのデータ値を書き込んだことを確認しなければなりません。もし、読む前に全てのデータ値を書き込むことが確かであれば、書き込む前に `nc_set_fill` を呼び出すことによってフィル値を持っている変数が前もって埋められてしまわないと確信できます。これによって NetCDF の書き込み効率が著しく向上することもあります。

変数属性 _FillValue はある変数のフィル値を指定するためにも使えます。各型ごとにデフォルトのフィル値があり、インクルードファイル netcdf.h の中で定義されています。 netcdf.h: NC_FILL_CHAR, NC_FILL_BYTE, NC_FILL_SHORT, NC_FILL_INT, NC_FILL_FLOAT, および NC_FILL_DOUBLE.

NetCDF バイトと文字型は異なるデフォルトのフィル値を持ちます。文字用のデフォルトのフィル値は 零バイトであり、可変長の C 文字列の終わりを判別するのに役立ちます。バイト変数にフィル値が必要なときには、適した _FillValue 属性を定義することをお勧めします。それは、ncdump 等の一般的なユーティリティではバイト変数に関してはデフォルトのフィル値を仮定しないからです。

フィル値の型変換は他の値の型変換と全く同様です。ある値をその値を表現できない別の型に変換しようとするとレンジエラーが生じます。そのようなエラーは、大きな型（例えば倍精度型）から小さな型（例えば単精度型）へと読み書きする際に、大きい方のフィル値が小さい方の型では表現できない時に生じることがあります。

7.17 変数の名前を変更する： nc_rename_var

関数 nc_rename_var は開かれた NetCDF ファイルの NetCDF 変数の名前を変更します。もし新しい名前が以前の名前よりも長い場合には NetCDF ファイルは定義モードになっていなければなりません。既に存在している変数名にすることは出来ません。

用法

```
int nc_rename_var(int ncid, int varid, const char* name);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	変数 ID。
name	指定された変数の新しい名前。

エラー

エラーが発生していなければ、関数 nc_rename_var は NC_NOERR 値を返します。それ以外の場合には、返されたステータスがエラーの発生を示しています。エラーの原因としては：

- 新しい名前が他の変数の名前として既に使用されている。
- 変数 ID が指定された NetCDF ファイルで有効ではない。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照しない。

例

この例では nc_rename_var を使用して、既存の NetCDF ファイル foo.nc 内の変数 rh の名前を rel_hum に変更します。

```
#include <netcdf.h>
   …
int  status;              /* エラーステータス */
int  ncid;                /* NetCDF ID */
int  rh_id;               /* 変数 ID */
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_redef(ncid);  /* 変数名を変更するため定義モードに入る */
if (status != NC_NOERR) handle_error(status);
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
status = nc_rename_var (ncid, rh_id, "rel_hum");
if (status != NC_NOERR) handle_error(status);
status = nc_enddef(ncid); /* 定義モードを抜ける */
if (status != NC_NOERR) handle_error(status);
```

8 属性

NetCDF 変数の性質である単位・特別な値・有効な値の最大値と最小値・スケーリングファクター・オフセット等を指定するために、各変数には属性が伴う。NetCDF ファイルの属性はファイル生成時に、NetCDF ファイルが定義モードにある時に定義される。NetCDF ファイルを再度定義モードにすることによって、属性を追加することが可能です。NetCDF 属性はその属性が割り当てられている NetCDF 変数・名前・ 型・長さ・一つ又は複数の値のシーケンスを持っています。属性はその変数 ID と名前で示されます。属性名が不明の場合には、その変数 ID と数を使い、関数 `nc_inq_attname` で名前を知ることができます。

変数に伴う属性は、通常、変数が生成された直後に、NetCDF ファイルがまだ定義モードにあるうちに定義されます。データ型・長さ・属性値はファイルがデータモードにあっても変更できます。ただしこれは、元々属性が定義された際に使用した以上のスペースが必要とされない場合に限ります。

どの変数とも関連していない属性を定義することも可能です。これらは *グローバル属性* と呼ばれ、関数 `NC_GLOBAL` を変数の擬似 ID として使います。グローバル属性は通常 NetCDF ファイル全体に関係し、NetCDF ファイルのタイトルや作業記録を付けるために使われます。

以下の操作が属性によってサポートされています。

- 変数 ID・名前・データ型・長さ・値を与えて属性を作成する。
- 変数 ID と名前から属性のデータ型と長さを得る。
- 変数 ID と名前から属性値を得る。
- ある NetCDF 変数から別の変数に属性をコピーする。
- 属性番号から属性名を得る。
- 属性名を変更する。
- 属性を削除する。

8.1 属性の規約

アンダースコア（'_'）で始まる名前は NetCDF ライブラリ専用です。NetCDF ファイルを処理する一般的なアプリケーションは標準的な 属性の規約を仮定しており、よほどの理由が無い限り、これらの規約に従いましょう。以下に、有用であることが証明済みの、推奨される標準的な属性の名前や意味が表記されています。これらの中には数値データを仮定しているものもあり（例えば、`units`, `valid_range`, `scale_factor`）、文字データ用には使うべきではない属性が幾つかあることに注意されたい。

units
: 変数データの単位を指定する文字列。Unidata は自由に取得できるルーチンライブラリを開発・提供しています。これを使えば文字列と単位指定のバイナリ形式との間の変換やバイナリ形式で様々な有用な操作を行なうことができる。このライブラリは幾つかの NetCDF アプリケーションで使われている。推奨される単位構文を使用すれば、整合単位で表現されたデータを、算術演算用に一般的な単位に自動的に変換することが可能です。詳しくは、12「単位」 p.119 参照。

long_name
: 長い記述的な名前。プロットのラベルなどに使える。変数に long_name 属性が割り当てられていなければ、変数名をデフォルトとして使用しましょう。

valid_min
: この変数の有効値の最小値を示すスカラー。

valid_max
: この変数の有効値の最大値を示すスカラー。

valid_range
: この変数の有効値の最小値と最大値を示す２つの数値のベクトル。valid_min と valid_max 属性の値を指定することと 等価である。これらの変数はどれも *有効範囲*を定義している。 valid_min 又は valid_max のどちらか一方でも定義されていたら、valid_range 属性を定義してはいけません。

  一般的なアプリケーションは*有効範囲*外の値は 欠損として扱うのが望ましい。各 valid_range、valid_min そして valid_max 属性の型はその変数の型と一致してしてなければなりません。(ただし、byte データ型は除く：これらは意図する範囲を符号付整数型によって指定できる。)

  valid_min、valid_max 、valid_range のいずれも定義されていない場合には、一般的なアプリケーションは有効範囲を次の方法で定義するのが良い。データがバイト型で _FillValue が明示されていない場合、有効範囲は全ての可能な値を含む。それ以外の場合には、有効範囲から（明示された、もしくはデフォルト指定の）_FillValue を下記の要領で除外する。_FillValue が正の値の場合には、それが有効な最大値とし、正で無い場合には有効な最小値として定義する。整数型については、_FillValue とこの有効な最大値又は最小値の差を 1 とする。浮動小数点型については、丸め誤差を念頭に置き、この差を表現可能な最小値（最も下位のビットで１）の 2 倍に設定する。

scale_factor
: ある変数についてこの属性が与えられていれば、データにアクセスするアプリケーションによってデータが読み込まれた後に、データはこの係数と掛け合わせる。

add_offset

ある変数についてこの属性が与えられている場合、データはそれにアクセスするアプリケーションによって読み込まれた後にこの数が加えられる。もし、scale_factor と add_offset の両方の属性が与えられている場合には、データはまずスケールされ、その後でオフセットが加えられる。scale_factor と add_offset を同時に使うことによって、簡単なデータ圧縮を行なうことができ、これによって、NetCDF ファイル内に低解像度の浮動小数点データを小さい整数として格納することが出来ます。スケールされたデータが書き込まれた場合、アプリケーションはます、オフセットを差し引き、その後にスケールファクターで割ればよい。

scale_factor と add_offset が 圧縮に使われる際には、関連する変数（圧縮データを格納している）の型は通常、byte 型か short 型である。一方で、解凍されたデータは float 型や double 型となるようにされている。scale_factor と add_offset 属性は両方とも解凍されたデータの持つ型（float 型や double 型）でなければなりません。

_FillValue

_FillValue 属性は、変数に割り当てられているディスクスペースを予め埋めるために使用される *フィル値* を指定します。このように予めスペースを埋める動作は、nc_set_fill を使って*ノーフィル モード*が設定されていない限り行なわれます。詳細については、p. 44 「書き込みのフィルモードを設定する：nc_set_fill」を参照してください。書き込まれた事のない値を読み取った時に *フィル値*が返されます。_FillValue が定義されていれば、それはスカラーで、変数と同じ型を取ります。デフォルトの*フィル値*が変数の型に合致していれば、変数について _FillValue 属性をいちいち定義する必要はありません。しかし、byte 型データにデフォルトのフィル値を使用することはお勧めできません。この変数の属性の値を変更する際には、その値がそれ以降の書き込みに対してのみ有効である点に注意してください。それ以前にフィル値が書き込まれたデータは変更されません。

一般のアプリケーションはしばしば、未定義の値又は 欠損値を表現するために値を書き込む必要があります。*フィル値*はこれに対して適切な値を提供します。それは、フィル値が通常、有効範囲外の値を取るために、一般のアプリケーションでは欠損値として扱われるからです。*フィル値*を有効範囲内に設定することは出来ますが、薦められません。

より詳しい説明については、7.16 節 「フィル値」（p. 83）を参照してください。

missing_value　この属性はライブラリや規約に従った一般のアプリケーションによって特別扱いされるわけではありませんが、しばしば有用な文書であるので特定のアプリケーションで使われることがあります。missing_value 属性はスカラーでもベクトルでも良く、欠損データを示す値を含んでいます。一般のアプリケーションが、これらの値を欠損値として取り扱えるように、これらの値は有効範囲外にあるべきです。

signedness　使用価値の無くなった属性です。元は byte 値が 符号付か符号無しのどちらで扱われるべきか指定するために作られました。現在ではこの目的のために、 valid_min と valid_max の属性を使用できます。例えば、byte 変数に非負の値のみ格納したい場合には、valid_min = 0 と valid_max = 255 とを使えます。NetCDF ライブラリはこの属性を無視します。

C_format　この変数の値を プリントする C アプリケーションが使用するべきフォーマットを与える文字配列です。例えば、ある変数が有効数字 3 桁の精度しかないことが明らかであれば、C_format 属性を "%.3g" と定義するのが適当です。ncdump ユーティリティプログラムは、これが定義されている変数に対してはこの属性を使います。このフォーマットは scale_factor 及び add_offset 等のスケーリングする属性のある無しに関わらず、スケールされた（内部）型と値に適用されます。

title　グローバル属性で、データセットの中身を簡潔に説明する文字配列。

history　検査履歴のためにグローバル属性。これは文字配列で、ファイルを修正したプログラムの各呼び出しに対して一行割り振られています。性質の良い 一般の NetCDF アプリケーションは、アクセスする際に日付・時刻・ユーザー名・プログラム名・コマンドの引数を含む一行を追加します。

Conventions　存在する場合には、‘Conventions’ はグローバル属性であり、データセットが従う規約の名前を示す文字配列です。ある分野に固有な規約の集合体を記述した文書の貯蔵場所のディレクトリの相対的なディレクトリ名として解釈される文字列の形式を取ります。これによって、規約の階層構造が可能になり、規約の記述や例を、それを定義した機関やグループが保持する場所を与えている。規約のディレクトリ名は現在ではホストマシン ftp.unidata.ucar.edu. 上の pub/NetCDF/Conventions/ ディレクトリから相対的に解釈される。代わりに、規約を記述した文書が維持されている WWW サイトを指定するために、完全な URL 指定子を使用しても良い。

例えば、NUWG というグループが、ある 分野に固有のデータ構造の次元名・変数名・必要な属性・NetCDF 表現に対する規約について合意したとする。NUWG は合意された規約を記述した文書を Conventions ディレクトリのサブディレクトリ NUWG/ に 保管しておくことができる。これらの規約に従ったデータセットは "NUWG" という値を持ったグローバル Conventions 属性を含むこととなる。

後にこのグループが、NUWG データの特定の部分集合（例えば時系列等）について新たに規約を追加することに決めた場合、その追加される規約の記述は NUWG/Time_series/ サブディレクトリに保管されます。これらの追加された規約に従ったデータセットは "NUWG/Time_series" の値を持つグローバル Conventions 属性を使い、NUWG 規約と追加された NUWG 時系列規約にも従ったことを示します。

8.2 属性を生成する： nc_put_att_*type*

関数 nc_put_att_*type* は、開かれた NetCDF ファイルの変数属性又はグローバル属性を追加・変更する。新規の属性、又は属性を格納するために必要なスペースが前より大きくなる場合には、NetCDF ファイルは 定義モードでなくてはなりません。

用法

どんな型の属性も生成可能ですが、ほとんどの用途にはテキストや倍精度属性で十分です。

```
int nc_put_att_text   (int ncid, int varid, const char *name,
                                  size_t len, const char *tp);

int nc_put_att_uchar  (int ncid, int varid, const char *name,
                       nc_type xtype, size_t len, const unsigned char *up);

int nc_put_att_schar  (int ncid, int varid, const char *name,
                       nc_type xtype, size_t len, const signed char *cp);

int nc_put_att_short  (int ncid, int varid, const char *name,
                       nc_type xtype, size_t len, const short *sp);

int nc_put_att_int    (int ncid, int varid, const char *name,
                       nc_type xtype, size_t len, const int *ip);

int nc_put_att_long   (int ncid, int varid, const char *name,
                       nc_type xtype, size_t len, const long *lp);

int nc_put_att_float  (int ncid, int varid, const char *name,
                       nc_type xtype, size_t len, const float *fp);
```

```
int nc_put_att_double (int ncid, int varid, const char *name,
                       nc_type xtype, size_t len, const double *dp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	属性が割り当てられる変数の変数 ID、又はグローバル属性の場合には NC_GLOBAL 。
name	属性名。アルファベット文字で始り、アンダースコア（'_'）を含む零又は英数字が続きます。大文字小文字は区別されます。属性名の規約は幾つかの NetCDF の一般的なアプリケーション で仮定されています。例えば、units は NetCDF 変数に単位を与える文字列属性の名前です。属性の規約の例については、8.1 節 「属性の規約」（p.86）を参照してください。
xtype	前もって定義されている NetCDF 外部データ型のひとつ。このパラメーターの型 nc_type は NetCDF ヘッダーファイルにおいて定義されています。有効な NetCDF 外部データ型は NC_BYTE、NC_CHAR、NC_SHORT、NC_INT、NC_FLOAT、NC_DOUBLE 等です。どんな型の属性も生成できますが、ほとんどの用途には NC_CHAR と NC_DOUBLE の属性で十分です。
len	属性に与えられた 値の数。
tp, up, cp, sp, ip, lp, fp, or dp	一つ又は複数の値へのポインタ。値の型が xtype と指定された NetCDF 属性の型と異なる場合には、型変換が行なわれます。詳細については、3.3 節 「型変換」（p.24）を参照してください。

エラー

エラーが発生していなければ、nc_put_att_*type* は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因として次のようなものが考えられます。

- 変数 ID が指定された NetCDF ファイルで無効である。
- 指定された NetCDF 型が無効である。
- 指定された長さが負の値である。
- 指定された開かれた NetCDF ファイルはデータモードにあり、指定された属性が大きくなっている。
- 指定された開かれた NetCDF ファイルはデータモードにあり、指定された属性がまだ存在していない。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。
- この変数の 属性の数が NC_MAX_ATTRS を超過している。

例

この例では、nc_put_att_double を使って、既存の foo.nc という名前の NetCDF ファイルにおいて、rh という名前の NetCDF 変数に対して valid_range という属性、及び、

title という名前のグローバル属性を追加している。

```
   #include <netcdf.h>
      …
   int  status;                              /* エラーステータス */
   int  ncid;                                /* NetCDF ID */
   int  rh_id;                               /* 変数 ID */
   static double rh_range[] = {0.0, 100.0};/* 属性値  */
   static char title[] = "example NetCDF dataset";
      …
   status = nc_open("foo.nc", NC_WRITE, &ncid);
   if (status != NC_NOERR) handle_error(status);
      …
   status = nc_redef(ncid);                  /* 定義モードに入る */
   if (status != NC_NOERR) handle_error(status);
   status = nc_inq_varid (ncid, "rh", &rh_id);
   if (status != NC_NOERR) handle_error(status);
      …
   status = nc_put_att_double (ncid, rh_id, "valid_range",
                               NC_DOUBLE, 2, rh_range);
   if (status != NC_NOERR) handle_error(status);
   status = nc_put_att_text (ncid, NC_GLOBAL, "title",
                             NC_CHAR, strlen(title), title)
   if (status != NC_NOERR) handle_error(status);
      …
   status = nc_enddef(ncid);                 /* 定義モードを出る */
   if (status != NC_NOERR) handle_error(status);
```

8.3 属性に関する情報を取得する： **nc_inq_att** のファミリー

この関数のファミリーは NetCDF 属性に関する 情報を返します。これらの関数は一つを除いて全て変数 ID と属性名を必要とします。例外は nc_inq_attname 関数です。属性に関する情報には型・長さ・名前・番号などが含まれます。属性値を取得する方法については nc_get_att の節を参照してください。

関数 nc_inq_attname は変数 ID と番号を与えると、属性の名前を返します。この関数は、他の全ての属性関数において属性は番号ではなく名前によってアクセスされるために、変数に関連した属性の名前を全て必要とする 一般的なアプリケーションにおいて役に立ちます。属性の番号は名前よりも 揮発性があり、同じ変数の属性が 削除された時に変わることがあります。このため、属性の番号は 属性 ID とは呼ばれません。

関数 nc_inq_att は属性の型と長さを返します。他の関数は各々、属性の情報を一つだけ返します。

用法

```
int nc_inq_att    (int ncid, int varid, const char *name,
                   nc_type *xtypep, size_t *lenp);
```

```
int nc_inq_atttype(int ncid, int varid, const char *name,
                   nc_type *xtypep);

int nc_inq_attlen  (int ncid, int varid, const char *name, size_t *lenp);

int nc_inq_attname(int ncid, int varid, int attnum, char *name);

int nc_inq_attid   (int ncid, int varid, const char *name, int *attnump);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	属性の変数の変数 ID 、又はグローバル属性の場合には NC_GLOBAL 。
name	属性名。 nc_inq_attname の場合、これは返された属性名の位置を示すポインタです。
xtypep	返された属性型の位置を示すポインタ。前もって定義された NetCDF 外部データ型の集合の一つ。このパラメーターの型 nc_type は NetCDF ヘッダーファイル内で定義されています。有効な NetCDF 外部データ型は NC_BYTE、NC_CHAR、NC_SHORT、NC_INT、NC_FLOAT、NC_DOUBLE です。このパラメータが ‘0’ (null ポインタ) で与えられていると、型は返されないので、その型を保持する変数を宣言する必要がありません。
lenp	現在属性に格納されている値の数が返された位置を示すポインタ。NC_CHAR 型の属性では、 これが null 文字を含むと仮定しないこと。属性が元々 null 文字無しに格納されている場合には含まれません。FORTRAN プログラムなどの場合がこれに相当します。C の記号列としてこの値を使用する前に、終端が null コードであること (null-terminated) を確認してください。このパラメータが ‘0’ (null ポインタ) で与えられている場合には長さは返されないので、この情報を保持する変数を宣言する必要がありません。
attnum	nc_inq_attname に対する属性番号。各変数の 属性は 0 (最初の属性) から natts-1 までの番号が振られています。(natts はその変数の属性の数で、nc_inq_varnatts への呼び出しで返されます。)
attnump	nc_inq_attid に対する返された属性番号の位置を示すポインタ。この変数 (又はグローバル属性) のどの属性であるかを示す。属性の情報を取得するためには属性の名前が必要であるので、属性の名前が既知であれば、この番号はあまり役に立ちません。

エラー

各関数は、エラーが発生していなければ NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因として次のようなものが考えられます。

・ 変数 ID が指定された NetCDF ファイルで無効である。

・ 指定された属性が存在しない。
・ 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。
・ nc_inq_attname に対して、指定された属性番号が負であるか、もしくは指定された変数に定義されている属性の数よりも多い。

例

この例では、nc_inq_att を使って、既存の foo.nc という名前の NetCDF ファイルにおいて、rh という名前の NetCDF 変数の属性 valid_range の型と長さと、title という名前のグローバル属性とを調べる。

```
#include <netcdf.h>
   …
int  status;               /* エラーステータス */
int  ncid;                 /* NetCDF ID */
int  rh_id;                /* 変数 ID */
nc_type vr_type, t_type;   /* 属性型 */
int  vr_len, t_len;        /* 属性長 */

   …
status = nc_open("foo.nc", NC_NOWRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_att (ncid, rh_id, "valid_range", &vr_type, &vr_len);
if (status != NC_NOERR) handle_error(status);
status = nc_inq_att (ncid, NC_GLOBAL, "title", &t_type, &t_len);
if (status != NC_NOERR) handle_error(status);
```

8.4 属性値を取得する：nc_get_att_*type*

nc_get_att_*type* のファミリーの関数は、変数 ID と名前を与えると NetCDF 属性の値を 返す。

用法

```
int nc_get_att_text   (int ncid, int varid, const char *name,
                       char *tp);

int nc_get_att_uchar  (int ncid, int varid, const char *name,
                       unsigned char *up);

int nc_get_att_schar  (int ncid, int varid, const char *name,
                       signed char *cp);

int nc_get_att_short  (int ncid, int varid, const char *name,
                       short *sp);
```

```
int nc_get_att_int    (int ncid, int varid, const char *name,
                       int *ip);

int nc_get_att_long   (int ncid, int varid, const char *name,
                       long *lp);

int nc_get_att_float  (int ncid, int varid, const char *name,
                       float *fp);

int nc_get_att_double (int ncid, int varid, const char *name,
                       double *dp);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	属性の変数の変数 ID 、又はグローバル属性の場合には NC_GLOBAL 。
name	属性名
tp, up, cp, sp, ip, lp, fp, or dp	返された 属性の値の位置を示すポインタ。属性値のベクトルの要素が全て返されるために、それらを格納するために十分なスペースを確保する必要があります。NC_CHAR 型の属性では、 これが null 文字を含むと仮定しないこと。属性が元々 null 文字無しに格納されている場合には含まれません。FORTRAN プログラムなどの場合がこれに相当します。C の記号列としてこの値を使用する前に、終端が零であること（null-terminated）を確認してください。どれだけのスペースを確保しておかなければならないか分からない時には、まず nc_inq_attlen を呼び出して属性の 長さを調べましょう。

エラー

エラーが発生していなければ、nc_get_att_*type* は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因として次のようなものが考えられます。

- 変数 ID が指定された NetCDF ファイルで無効である。
- 指定された属性が存在しない。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。
- 属性値の一つ又はそれ以上が希望される型で表現し得る値の範囲から外れている。

例

この例では nc_get_att_double を使って、既存の foo.nc という NetCDF ファイルにおいて 、rh という名前の NetCDF 変数の属性 valid_range の属性値と、title という名前のグローバル属性の値を調べます。この例では、幾つの値が返されるかは不明だが、属性の型は既知であることを前提としています。従って、返された値を格納するスペースを十分に取るために、ます始めに属性の長さについて問い合わせます。.

```
#include <netcdf.h>
   …
```

```
int  status;                /* エラーステータス */
int  ncid;                  /* NetCDF ID */
int  rh_id;                 /* 変数 ID */
int  vr_len, t_len;         /* 属性長 */
double *vr_val;             /* 属性値へ ptr  */
char *title;                /* 属性値へ ptr  */
extern char *malloc();      /* メモリ配置 */

   …
status = nc_open("foo.nc", NC_NOWRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
   …
/* 属性値に必要なスペース調べる */
status = nc_inq_attlen (ncid, rh_id, "valid_range", &vr_len);
if (status != NC_NOERR) handle_error(status);
status = nc_inq_attlen (ncid, NC_GLOBAL, "title", &t_len);
if (status != NC_NOERR) handle_error(status);

/* 値を取得する前に必要なスペースを配置 */
vr_val = (double *) malloc(vr_len * sizeof(double));
title = (char *) malloc(t_len + 1);  /* + 1 trailing null用 */

/* 属性値を取得 */
status = nc_get_att_double(ncid, rh_id, "valid_range", vr_val);
if (status != NC_NOERR) handle_error(status);
status = nc_get_att_text(ncid, NC_GLOBAL, "title", title);
if (status != NC_NOERR) handle_error(status);
title[t_len] = '\0';        /* null terminate */
   …
```

8.5 一つの NetCDF から他へ属性をコピーする： nc_copy_att

関数 nc_copy_att は開かれた NetCDF ファイルから他のファイルへ属性をコピーします。また同じ NetCDF 内で、ある変数の属性を別の変数にコピーするときにも使えます。

用法

```
int nc_copy_att (int ncid_in, int varid_in, const char *name,
                 int ncid_out, int varid_out);
```

ncid_in	以前の nc_open 又は nc_create 呼び出しで返された、属性のコピー元となる入力 NetCDF ファイルの NetCDF ID。
varid_in	属性のコピー元となる、入力 NetCDF ファイルの変数 ID 、又はグローバル属性の 場合には NC_GLOBAL 。

name	コピーされる入力 NetCDF ファイルの属性名。
ncid_out	属性のコピー先となる、出力 NetCDF ファイルの NetCDF ID 。以前の nc_open 又は nc_create 呼び出しから。入力と出力 NetCDF ID が同じでも構わない。コピーされる属性が出力 NetCDF ファイル内にまだ存在しない場合、又は既存のコピー先の属性が大きくなる場合には、出力 NetCDF ファイルは定義モードにしておく必要がある。
varid_out	属性のコピー先の出力 NetCDF ファイルの変数 ID 、又はグローバル属性をコピーする場合には NC_GLOBAL 。

エラー

エラーが発生していなければ、nc_copy_att は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因として次のようなものが考えられます。

- 入力又は出力変数 ID が指定された NetCDF ファイルで無効である。
- 指定された属性が存在しない。
- 出力 NetCDF が定義モードになく、コピーされる属性が新しいか、又は存在する属性より大きい。
- 入力又は出力 NetCDF ID が開かれた NetCDF ファイルを参照していない。

例

この例では、nc_copy_att を使って 、既存の foo.nc という NetCDF ファイルにおける変数 rh から変数属性 units をコピーして、他の既存の bar.nc という NetCDF ファイルの変数 avgrh に貼り付ける。変数 avgrh は既に存在するが、属性 units はまだ持っていないと仮定する。

```
#include <netcdf.h>
   …
int  status;               /* エラーステータス */
int  ncid1, ncid2;         /* NetCDF ID */
int  rh_id, avgrh_id;      /*  変数 IDs */
   …
status = nc_open("foo.nc", NC_NOWRITE, ncid1);
if (status != NC_NOERR) handle_error(status);
status = nc_open("bar.nc", NC_WRITE, ncid2);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid1, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
status = nc_inq_varid (ncid2, "avgrh", &avgrh_id);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_redef(ncid2);  /* 定義モードに入る */
if (status != NC_NOERR) handle_error(status);
/* 変数属性を "rh" からコピーして "avgrh" に貼り付ける */
```

```
status = nc_copy_att(ncid1, rh_id, "units", ncid2, avgrh_id);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_enddef(ncid2); /* 定義モードを抜ける */
if (status != NC_NOERR) handle_error(status);
```

8.6 属性名を変更する：nc_rename_att

関数 nc_rename_att は属性の名前を変更します。新しい名前が元の名前より長い場合には、NetCDF ファイルは定義モードになっている必要があります。同じ変数の他の属性名と同じ名前になってしまうような属性名の変更はできない。

用法

```
int nc_rename_att (int ncid, int varid, const char* name,
                   const char* newname);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	属性の変数の ID、又はグローバル属性の NC_GLOBAL 。
name	現行の属性名。
newname	指定された属性に割り当てられる新しい名前。新しい名前が現行の名前よりも長い場合には、NetCDF ファイルは定義モードになっていなければならない。

エラー

エラーが発生していなければ、nc_rename_att は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因として次のようなものが考えられます。

- 変数 ID が無効である。
- 新しい属性名は指定された変数の他の属性が既に使用している。
- 指定された NetCDF ファイルはデータモードになっていて、新しい名前は元の名前より長い。
- 指定された属性が存在しない。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。

例

この例では、nc_rename_att を使って、既存の foo.nc という NetCDF ファイルにおける変数 rh の変数属性の名前を units から Units に変更する。

```
#include <netcdf.h>
   …
int  status;      /* エラーステータス */
```

```
int  ncid;        /* NetCDF ID */
int  rh_id;       /* 変数 ID */
   …
status = nc_open("foo.nc", NC_NOWRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
   …
/* 属性名を変更 */
status = nc_rename_att(ncid, rh_id, "units", "Units");
if (status != NC_NOERR) handle_error(status);
```

8.7 属性を削除する： nc_del_att

関数 nc_del_att は開かれた NetCDF ファイルから NetCDF 属性を削除します。NetCDF ファイルは定義モードになっている必要があります。

用法

```
int nc_del_att (int ncid, int varid, const char* name);
```

ncid	以前の nc_open 又は nc_create 呼び出しで返された NetCDF ID。
varid	属性の変数の ID、又はグローバル属性の NC_GLOBAL 。
name	削除される属性の名前。

エラー

エラーが発生していなければ、nc_del_att は NC_NOERR の値を返します。それ以外の場合には、返されたステータスがエラーを示します。エラーの原因として次のようなものが考えられます。

- 変数 ID が無効である。
- 指定された NetCDF ファイルがデータモードになっている。
- 指定された属性が存在しない。
- 指定された NetCDF ID が開かれた NetCDF ファイルを参照していない。

例

この例では、nc_del_att を使って、既存の foo.nc という NetCDF ファイルから変数 rh の変数属性 Units を削除します。

```
#include <netcdf.h>
   …
int  status;      /* エラーステータス */
int  ncid;        /* NetCDF ID */
```

```
int  rh_id;       /* 変数 ID */
   …
status = nc_open("foo.nc", NC_WRITE, &ncid);
if (status != NC_NOERR) handle_error(status);
   …
status = nc_inq_varid (ncid, "rh", &rh_id);
if (status != NC_NOERR) handle_error(status);
   …
/* 属性を削除 */
status = nc_redef(ncid);        /* 定義モードに入る */
if (status != NC_NOERR) handle_error(status);
status = nc_del_att(ncid, rh_id, "Units");
if (status != NC_NOERR) handle_error(status);
status = nc_enddef(ncid);       /* 定義モードを抜ける */
if (status != NC_NOERR) handle_error(status);
```

9　NetCDF ファイルの構造と性能

この章では、NetCDF の性能について理解するのに必要な NetCDF のファイル 構造の詳細を説明します。

NetCDF は配列指向のデータアクセスの為のデータ抽象化であり、その抽象化をサポートするインターフェースの具体的な実装を与えるソフトウェアライブラリです。この実装によって配列を表現するための機種独立型のフォーマットを提供されます。NetCDF ファイルフォーマットはインターフェースの表面からは見えないが、現行の実装と関連するファイル構造を幾らか理解していれば、どの NetCDF 操作が他よりコストがかかるか明らかになるでしょう。

NetCF フォーマットの詳細に関しては、Appendix A 「ファイルフォーマット仕様」 p.121 を参照してください。フォーマットの知識が無くても、NetCDF データの読み書きや 効率に関する問題点のほとんどを理解することは可能です。文書化されたインターフェースのみを使い、フォーマットに関しては何の仮定もしていないプログラムは、将来 NetCDF フォーマットが変更されても機能しつづけます。それは、フォーマットの変更は全て文書化されたインターフェースの下層で行なわれ、かつ、以前のバージョンの NetCDF ファイルフォーマットはサポートされるからです。

9.1　NetCDF ファイルの構成要素

NetCDF ファイルは 2 つの部分からなる一つのファイルとして格納されています。

- ヘッダー*部分*は変数のデータ以外の次元・属性・変数の情報を全て含みます。
- *データ* 部分は無制限次元を持たない変数のデータを含む *固定サイズデータ*、そして無制限次元を有する変数のデータを含む *変動サイズデータ*からなります。

ヘッダー部分とデータ部分は両方とも機種独立型で表現されています。この形式は、配列や非バイトデータの効率的な格納をサポートするために拡張された XDR（eXternal Data Representation）と非常に似ています。

ファイルの先頭にあるヘッダー部分はファイルに含まれる次元・変数・属性についての名前・型・その他の性質に関する情報を含みます。各変数の情報には固定サイズデータについては変数データの先頭の オフセットや、記録中の他の変数の相対オフセット 等がある。ヘッダーは、又、次元長や各変数の複数次元のインデックスを適切なオフセットにマップするのに必要な情報を含みます。

このヘッダーの使用可能なスペースに余分はありません。NetCDF ファイル中の次元・変数・属性（属性値を全て含む）に必要な最低限の大きさしかありません。これによって、NetCDF ファイルは コンパクトであり、データを自己記述的にするための従属的なデータを格納するのにほとんど オーバーヘッドを必要としないという利点があります。この構造の欠点は、NetCDF ファイルのヘッダーを増大（又は可能性としては低いが縮小）させるようなどの操作も、データをコピーすることによって移動させるということです。例えば、新しい次元や変数を追加したりする場合がそうです。このコストは

nc_redef への呼び出しの後に nc_enddef が呼び出されたときにかかります。データを書き込む前に必要な次元・変数・属性を生成して、ファイルのヘッダー部分により多くのスペースを必要とする生成後の NetCDF 要素の追加や名前の変更を避けることによって、その後のヘッダー部分の変更に伴うコストを回避することが出来ます。

ヘッダーのサイズが変更されると、ファイル中のデータは移動され、ファイル内におけるデータ値の位置が変更されます。再定義中に他のプログラムがこのファイルを読み取っている場合には、そのファイルを間違っている可能性のある旧インデックスを使用して参照することとなります。NetCDF ファイルが再定義を超えて共有されるためには、再定義中の読み取りアクセスを防ぎ、次回のアクセスの前に読み取る側に nc_sync を呼び出させるような、NetCDF ライブラリ外の機構が必要となります。

ヘッダーに続く固定サイズデータ部分は無制限次元を有さない変数の変数データを全て含みます。各変数のデータはこのファイル部分に連続的に格納されています。無制限次元が無い場合には、これが NetCDF ファイルの末尾の部分となります。

固定サイズデータ部分に続く記録データ部分は、各々記録データの情報を全て含む固定サイズ記録の変数番号からなります。各変数の記録データは各記録中に連続的に格納されています。

各データ部分における変数データの順番は変数が定義された順番と同じで、NetCDF 変数 ID の昇順になります。これを知っていると、現行ではデータを連続的に読み書きすることが最適なデータアクセス法なので、場合によってはデータアクセス性能を向上させることが出来ます。

9.2 拡張 XDR 階層

XDR はデータ記述とコード化の為の標準であり、外部データ表現の為のライブラリでもあります。これによって、プログラマーは機種独立な手法によってデータ構造をコード化することができます。NetCDF はヘッダー部分とデータ部分で情報を表現するために拡張された XDR 形式を採用しています。この拡張 XDR はライブラリが実装されているどのマシン上でも読み取れるポータブルなデータを書くのに使用されています。

データ表現のために規範的な外部データ表現を使用するコストはデータの型によって異なり、又、その外部データ型がマシンの本来の形式であるかにもよります。

ある機種のデータ型では、外部データ形式からデータを変換したり、外部形式へと変換するのに膨大な時間を費やすかもしれません。最悪の場合は、IEEE 浮動小数点が本来の表現法では無いマシン上で 浮動小数点データの大きな配列を読み書きすることでしょう。

9.3 I/O 階層

I/O 階層の実装は、NetCDF ファイルのポータブルデータを読み書きするのための C standard I/O (stdio) ライブラリの実装とよく似ています。よって、標準的な I/O ラ

イブラリを理解すれば、 同時にデータをアクセスする複数の処理や、I/O バッファの使われ方、NetCDF ファイルの開け閉めのコストに関する様々な疑問が解決されます。特に、NetCDF ファイルに対して一つの書き込み処理が行なわれている間に 、別の複数の読み取り処理が進行することも可能です。データの読み書きには、stdio `fread()` や `fwrite()` への呼び出しより下位のものは使用していない。`nc_sync` 呼び出しは全て C standard I/O ライブラリの `fflush` 呼び出しと類似しており、他の処理が読み取れるように未記入の バッファデータを書き込んでいきます。`nc_sync` は又、ヘッダーの変更(例えば、属性値への変更)を最新のものにします。`NC_SHARE` は `the _IONBF` フラグを `setvbuf` ステータスにした、バッファされていない stdio stream を設定することと同義です。

stdio ライブラリの場合と同様に、ファイルの異なる部分への〝探索〟が生じると、flush が実行される。従って、書き込み操作の順番は I/O 性能に著しい影響を与えます。各記録中にデータが書き込まれたのと同じ順番でデータを読み取ることによってバッファ flush を最小限に留めることができます。

NetCDF データアクセスは、同一のファイルに同時に複数の書き込み処理が行えるようにはなっていません。

I/O 階層を別のプラットフォーム固有の I/O 階層に置き換えることにより、NetCDF の実装をあるプラットフォームに合わせて調整できます。これによって NetCDF と標準的な I/O との相似点、つまりデータ共有の性質・バッファ動作・I/O 操作のコストなどが変わる可能性があります。

配布された NetCDF 実装は ポータブルであることを目標にしています。場合によっては、よりよい I/O 性能のために実装を最適化するような プラットフォーム固有のポート の方が実用的でしょう。

9.4 UNICOS 最適化

前出のように、I/O 効率を向上させるために I/O 階層を置き換えることは可能です。Cray Y-MP と類似した、Cray コンピューターの OS である UNICOS に関しては、これは既になされています。

更に、`NetCDF_FFIOSPEC` 環境変数を正しく設定することにより、ユーザーは一層、I/O 効率を上げることが出来ます。この変数は、UNICOS OS 下で実行中に、NetCF I/O の Flexible File I/OT バッファを指定します。(この変数は他の OS では無視されます。)適切な設定を選択すれば NetCDF I/O の効率を飛躍的に向上させることが出来ます―デフォルトの FORTRAN bianry I/O を超えることもも可能です。下記のような指定が可能です。

`bufa:336:2`	2、非同期、各 336 ブロックの I/O バッファ (つまりダブルバッファ)。これはデフォルト指定で、連続的な I/O 向け。
`cache:256:8`	8、同期、256 ブロックバッファ。大きいランダムアクセス向け。

`cachea:256:8:2`	8、非同期、2 ブロック先行読み取り / 後方書き込みファクタ付き 256 ブロックバッファ 。より大きなランダムアクセス向け。
`cachea:8:256:0`	256、非同期、先行読み取り / 後方書き込み無しの 8 ブロックバッファ。これは小さなページ向けで、ランダムアクセス向きに先行読み取り機能がありません。NetCDF 配列をスライスなどが例として挙げられます。
`cache:8:256,cachea.sds:1024:4:1`	これは２階層キャッシュです。最初の（同期）階層は メモリ内の 256 個の 8 ブロックバッファからなり、２番目の（非同期）階層は SSD 上の 4 個の 1024 ブロックバッファからなります。この機構はデータセット内を 2x1024 ブロックの単位で 波状にランダムアクセスする場合に向いています。

CRI の FFIO ライブラリによってサポートされているオプション / 構成はこの機構を通じて利用できます。FFIO の機能を最大限利用するために CRI の I/O 最適化ガイドを参照することをお勧めします。この機構は、又、CRI の EIE I/O ライブラリとも互換性があります。

Tuning the `NetCDF_FFIOSPEC` 変数をプログラムの I/O パターンに調整すれば、性能が飛躍的に向上します。何百倍というスピードが得られた例もあります。

10　NetCDF ユーティリティ

配列を扱うアプリケーションが NetCDF インターフェースを使用する主な理由の一つに、高位レベルの NetCDF ユーティリティと NetCDF データの一般的なアプリケーションを利用することがある。現在では、NetCDF ソフトウェア配布版の一部として 2 つの NetCDF ユーティリティが用意されている。

- `ncdump` は NetCDF ファイルを読み、データ内の情報をテキスト表記で出力する。
- `ncgen` は NetCDF ファイルのテキスト表記を読み、対応するバイナリの NetCDF ファイルもしくはその NetCDF ファイルを作成する C 又は FORTRAN のプログラムを作成する。

より汎用の NetCDF ユーティリティが２つ FAN （File Array Notation）パッケージのに含まれている。

- `ncmeta` は一つ又は複数の NetCDF ファイルから選択されたメタデータを出力する。
- `ncrob` はテキストファイル /NetCDF 変数又は属性の選択範囲からから読み込まれ、そこに出力又は書き込まれたデータに対して様々な操作（コピー、合計、平均、最大値、最小値）を行なう。

FAN に関しての詳細は see `http://www.unidata.ucar.edu/packages/NetCDF/fan_utils.html`.

他の NetCDF ユーティリティにはユーザーからの寄与があり、NetCDF データをアクセスする様々な視覚化や解析パッケージが存在する。無償・有償両方のの NetCDF データをアクセスし扱えるソフトウェアの最新情報については、NetCDF Software リスト が次のサイトにあります。`http://www.unidata.ucar.edu/packages/NetCDF/software.html`.

この章には `ncgen` と `ncdump` ユーティリティの説明があります。これら２つのツールはバイナリの NetCDF ファイルと NetCDF ファイルのテキスト表記間の変換を行ないます。 `ncdump` の出力と `ncgen` の入力は CDL（network Common data form Description Language）として知られる簡単な言語によってテキスト表記されたものです。

10.1 CDL 構文

以下の CDL の例では、幾つかの名前付き次元（`lat, lon, time`）、変数（`z, t, p, rh, lat, lon, time`）、変数属性（`units, _FillValue, valid_range`）とデータからなる NetCDF ファイルを記述しています。

```
NetCDF foo {    // CDL による NetCDF 指定の例

dimensions:
lat = 10, lon = 5, time = unlimited;

variables:
  int     lat(lat), lon(lon), time(time);
```

```
    float   z(time,lat,lon), t(time,lat,lon);
    double  p(time,lat,lon);
    int     rh(time,lat,lon);

    lat:units = "degrees_north";
    lon:units = "degrees_east";
    time:units = "seconds";
    z:units = "meters";
    z:valid_range = 0., 5000.;
    p:_FillValue = -9999.;
    rh:_FillValue = -1;

  data:
    lat   = 0, 10, 20, 30, 40, 50, 60, 70, 80, 90;
    lon   = -140, -118, -96, -84, -52;
  }
```

全ての CDL 宣言文はセミコロンで終わります。スペース・タブ・改行は可読性のために自由に使えます。コメントは ダブルスラッシュ // に続き、どの行にも配置可能です。

CDL 記述は次元・変数・属性の 3 つのオプション部分から構成されます。変数部は変数宣言文や属性割り当てを含むことが出来ます。

次元は CDL 記述で記述される多次元変数の形を定義するために使われます。次元には名前と長さがあります。CDL 記述の次元の内、一つの次元まで無制限長を持つことが出来、それはこの次元を使う変数が任意の長さになり得る（ファイル中の記録番号のように）ことを意味します。

変数は同じ型の値の多次元配列を表現します。変数は名前・データ型・そして次元のリストによって記述された形を持ちます。各変数はデータ値のほかに関連する属性（下記参照）も持ちえます。名前・データ型・変数の形は CDL 記述中の変数部分における宣言文によって指定されます。変数は次元と同じ名前を持つことが出来ます。規約として、そのような変数は 次元の座標値を名前に含んでいます。

属性は変数や NetCDF ファイル全体についての情報を含んでいます。属性は単位・特別な値・有効な値の最大値と最小値・圧縮パラメーターのような特性を指定するのに使われます。属性情報は単一の値や値の配列によって表現されます。例えば、units は celsius 等の文字列によって表現される属性です。属性には関連する変数・名前・データ型・長さがあります。データ用の変数とは対照的に、属性は従属的なデータ（データに関するデータ）のためにあります。

CDL では、属性は変数と属性名とをコロン（':'）で区切ったもので指定される。変数名を省略し、属性名をコロン（':'）ではじめることによって、NetCDF ファイル全体にグローバル属性を割り当てることもできます。CDL の属性のデータ型はそれに割り当てられている値の型で決まります。属性の長さはデータ値の数又はそれに割り当てられた文字列中の文字の数になります。文字でない属性に複数の値を割り当てる場合には、値をコンマ（','）で区切れば可能です。属性に割り当てられた値は全て同じ型でなくてはいけません。

変数・属性・次元に対する CDL 名には、英数字と‘_’及び‘-’の任意の組み合わせが許可されているが、‘_’で始まる名前はライブラリ専用です。CDL 名では大文字小文字は区別されます。NetCDF ライブラリは NetCDF 名に制約を加えていないので、有効な CDL 名ではない名前を使って変数を定義することも可能ですが薦められません。基本的なデータ型の名前は CDL では予約語であるので、変数・次元・属性の名前は型の名前は取れません。

CDL 記述のオプションのデータ部分では、NetCDF 変数が 初期化されます。初期化のお構文は単純です。

variable = *value_1, value_2,* …;

コンマで区切られた定数のリストは、空白・タブ・改行によって分けることができます。多次元配列では、最後の次元が最も早く変わります。よって、行列には行順ではなく列順が使われます。変数を満たすのに不十分な値が与えられた場合には、フィル値によって埋められます。定数の型は変数に宣言された型と一致していなくても良く、例えば、整数を浮動小数点数に強制的に変換するといった操作が行なわれます。意味のある型変換は全てサポートされています。

フィル値用の特別な記述がサポートされています。‘_’文字は変換のためのフィル値を指します。

10.2 CDL データ型

CDL データ型には次のものがあります。

char	文字
byte	8 ビット整数.
short	16 ビット符号付整数
int	32 ビット符号付整数
long	(使用されない傾向にある。現在は int と同義。)
float	IEEE 単精度浮動小数点数 (32 ビット)
real	(float と同義)
double	IEEE 倍精度浮動小数点数 (64 ビット)

byte データ型が追加されていることと、unsigned 修飾子が無いことを除けば、CDL は C と同様の基本的データ型をサポートしています。宣言文では、型名の指定は大文字でも小文字でも 構いません。

byte 型は 8 ビットデータ用である点が char 型と異なります。そして、零バイトは文字データにおけるような特別な意味を持ちません。 ncgen ユーティリティは byte 宣

言文を、出力C コードにおいては char 宣言文に、そして出力FORTRAN コードにおいては BYTE, INTEGER*1 もしくは同類のプラットフォーム固有の宣言文に変換します。

short 型は -32768 と 32767 の間の値を保持します。ncgen ユーティリティは short 宣言文を、出力C コードにおいては short 宣言文に、そして出力FORTRAN コードにおいては INTEGER*2 宣言文に変換します。

int 型は -2147483648 と 2147483647 の間の値を保持します。ncgen ユーティリティは int 宣言文を、出力C コードにおいては int 宣言文に 、そして出力FORTRAN コードにおいては INTEGER 宣言文に変換します。CDL 宣言文では integer と long は int の同義語として認識されています。

float 型は -3.4+38 と 3.4+38 との間の値を保持でき、外部表現には32 ビットのIEEE 規格化された単精度浮動小数点数が使われます。 ncgen ユーティリティは float 宣言文を、出力C コードにおいては float 宣言文に、そして出力FORTRAN コードにおいては REAL 宣言文に変換します。CDL 宣言文では real は float の同義語として認識されています。

double 型は -1.7+308 と 1.7+308 の間の値を保持し、外部表現には64 ビットの IEEE 規格化された倍精度浮動小数点数が使われます。ncgen ユーティリティは double 宣言文を、出力C コードにおいては double 宣言文に、そして出力 FORTRAN コードにおいては DOUBLE PRECISION 宣言文に変換します。

10.3 データ 定数の CDL 表記

この節は定数の CDL 表記についての説明です。

属性は CDL 記述の variables 部において、属性の型と 長さを決める定数のリストを与えることによって、初期化されます。(NetCDF ライブラリへの C と FORTRAN の手続きインターフェースにおいては、属性の型と名前は定義されるときに明記されなければならない。) CDL は、異なる NetCDF 型で区別がつくように、定数値の構文を記述している。CDL 定数の構文は C 構文と似ているが、int と double から区別するために、型の接尾子が short と float に添えてある。

バイト定数は単一の文字、又は、シングルクオートで囲んだ複数文字のエスケープ列で表現されます。例えば、

```
'a'     // ASCII a
'\0'    // null 文字
'\n'    // ASCII 改行文字
'\33'   // ASCII エスケープ文字 (8 進数で 33)
'\x2b'  // ASCII プラス (16 進数で 2b)
'\376'  // 8 進数で 377 = 10 進数で -127 ( 又は 254)
```

文字定数はダブルクオートで囲まれています。文字配列はダブルクオートで囲んだ文

字列として表現できます。複数の文字列は単一の文字配列にと連結されます。これによって、長い文字配列を複数の行に 書くことが出来ます。複数の可変長の文字列値をサポートするためには、‘,’のような規約的な区切り文字を使用することができるが、このような文字列区切りのための規約はNetCDF ライブラリ層の上のソフトウェアに実装されていなければなりません。通常のC 文字列のエスケープ規約はそのまま使用できます。例えば、

```
"a"             // ASCII ‘a’
"Two\nlines\n" // ２つの改行文字を埋め込んだ 10 文字の文字列
"a bell:\007"  // ASCII ベルを含む文字列
"ab","cde"     // "abcde" と同じ
```

short 定数の形式は ‘s’ 又は ‘S’ を付加した整定数である。 short 定数が ‘0’ で始まれば、8 進数であると解釈されます。‘0x’ で始まれば、16 進数の整数として解釈されます。例えば、

```
2s      // short 型の 2
0123s   // 8 進数
0x7ffs  // 16 進数
```

int 定数の形式は普通の整定数です。int 定数が ‘0’ で始まれば、それは 8 進数であると解釈されます。‘0x’ で始まれば、16 進数として解釈されます。有効な int 定数の例を幾つか挙げます。

```
-2
0123            // 8 進数
0x7ff           // 16 進数
1234567890L     // 現在では使用されない。古い long 接尾子を使用している。
```

float 型は有効数字 7 桁の精度を持つデータを表現するのに適しています。 float 定数の形式は C 浮動小数点定数に ‘f’ 又は ‘F’ を付加したものと同じです。CDL float では整数と区別するために小数点が必要です。次に挙げる例は全て、妥当な float 定数です。

```
-2.0f
3.14159265358979f       // 低精度に丸められる
1.f
.1f
```

double 型は有効数字 16 桁の精度を持つデータを表現するのに適しています。double 定数の形式はC 浮動小数点定数と同じです。オプションとして ‘d’ 又は ‘D’ を付加しても構いません。integer と区別するために、CDL double には小数点が必要です。次に挙げる例は全て妥当な double 定数です。

```
-2.0
3.141592653589793
1.0e-20
1.d
```

10.4 ncgen

ncgen ツールは NetCDF ファイル、又は、NetCDF ファイルを生成する C 又は FORTRAN のプログラムを生成します。ncgen, を呼び出す際にオプションを指定しなければ、そのプログラムは単に CDL 入力の構文をチェックし、CDL 構文に合致しないものがあればエラーメッセージを出すだけです。

ncgen を呼び出す UNIX 構文

```
ncgen [-b] [-o NetCDF-file] [-c] [-f] [-n] [input-file]
```

ここで、

-b	(バイナリの) NetCDF ファイルを生成する。'-o' オプションが設定されていなければ、NetCDF 名に拡張子 '.nc' を付加してデフォルトのファイル名が付けられます。(入力の際には NetCDF キーワードの後に指定されています。) **警告：指定されたファイル名と同じ名前のファイルが既に存在している場合、上書きされてしまいます。**
-o NetCDF-file	生成された NetCDF ファイルの名前。このオプションが選択されている場合、'-b' オプションが暗黙のうちに了承されます。(このオプションは、NetCDF ファイルが探索呼び出しによって生成される直接参照ファイルであり、それ故、標準出力に書き出すことが出来ないために必要となります。)
-c	NetCDF 指定に合った新しい NetCDF ファイルを生成する C ソースコードを作成する。 C ソースコードは標準出力に書かれる。これは、生成されたプログラム中の変数を初期化した際に全てのデータが含まれるので、比較的小さな CDL ファイルでのみ有用です。
-f	NetCDF 指定に合った新しい NetCDF ファイルを生成する FORTRAN ソースコードを作成する。FORTRAN ソースコードは標準の出力に書かれる。これは、生成されたプログラム中の変数を初期化した際に全てのデータが含まれるので、比較的小さな CDL ファイルでのみ有用です。
-n	現在では使用されない。出力ファイル名が '-o' オプションによって指定されていない場合に、'-b' オプションと同様に、NetCDF ファイル名を作成しますが、拡張子は '.nc' ではなく '.cdf' になります。このオプションは後方互換性の場合のみにサポートされています。

例

CDL ファイル foo.cdl の構文をチェックする。

```
ncgen foo.cdl
```

CDL ファイル foo.cdl より、bar.nc という名の等価な NetCDF バイナリファイルを生成する。

```
ncgen -o bar.nc foo.cdl
```

CDL ファイル foo.cdl より、等価な NetCDF バイナリファイルを生成するために必要な NetCDF 関数呼び出しを含む C プログラムを作成する。

```
ncgen -c foo.cdl > foo.c
```

10.5 ncdump

ncdump ツールは標準出力に NetCDF ファイルの CDL テキスト表現を出力する。オプションによって、入力されたデータの変数データの一部又は全てを除外することも出来ます。ncdump からの出力は ncgen への入力として使用できるようになっています。よって、ncdump と ncgen はバイナリ表現とテキスト表現との間でデータ表現を変換するための正逆変換として使用できます。

ncdump は又、NetCDF ファイル用の簡単なブラウザとしても使えます。これによって、NetCDF ファイル内の、次元名と次元長・変数名と型と形・属性名と値・オプションとして全てまたは選択された変数の値などを見ることが出来ます。

ncdump は NetCDF の変数データの各型について使用されているデフォルトのフォーマットを定義しています。しかし、これは NetCDF 変数に C_format 属性が定義されていればこちらのほうが優先されます。この場合には、ncdump は C_format 属性を使ってその変数の値をフォーマットします。例えば、浮動小数点数である NetCDF 変数 z の有効数字が 3 桁しかないことが分かっている場合などに、この変数属性を使うと良いでしょう。

```
Z:C_format = "%.3g"
```

ncdump は '_' を使って _FillValue 属性（これはまだ書かれていないデータを表現するためにあります）と等しい値を持つデータ値を表現します。もし、変数が _FillValue 属性を有していなければ、変数がバイト型で無い限り変数型のデフォルトフィル値が使用されます。

ncdump を呼び出す UNIX 構文

```
ncdump  [ -c | -h]  [-v var1,…]  [-b lang]  [-f lang]
[-l len]  [ -p fdig[,ddig]]  [ -n name]  [input-file]
```

ここで、

-c　全ての次元・変数・属性値の宣言文と、*座標* 変数（次元でもある変数）の値を示す。座標変数でない変数のデータ値は出力に含まれない。任意の NetCDF ファイルの構造と内容をざっと見るのに最も適したオプションです。

-h　出力で ヘッダー 情報のみ示す。つまり、入力 NetCDF ファイルの次元・変数・属性 の宣言文のみを出力し、変数のデータ値は一切出力しない。出力は ‘-c’ オプションを使用した場合とほぼ同じですが、座標変数の値も出力に含まれません。(‘-c’ 又は ‘-h’ オプションのどちらか一つのみ指定できます。)

-v var1,…　出力は、全ての次元・変数・属性の宣言文と、指定された変数のデータ値を含みます。このオプションの後にあるカンマで区切られた表中に、一つまたは複数の変数を名前で指定なければなりません。この表はこのコマンドへの唯一の引数でなければならないので、空白や他の空白文字を含むことは出来ない。名前付き変数は入力ファイル中で有効な NetCDF 変数でなければなりません。このオプションが選択されていなく、さらに ‘-c’ 又は ‘-h’ オプションも選択されていない場合のデフォルトでは、 *全ての*変数の値が出力されます。

-b lang　出力のデータ部分において、データの各 ‘ 列 ’ に CDL コメント形式の（‘//’ で始まるテキスト）簡潔な注釈が含まれるようになります。これによって多次元変数のデータ値の確認が容易になります。*lang* が ‘C’ 又は ‘c’ で始まれば、C 言語の規約（零基準のインデックス、最終次元が最も早く変わる）が使用される。*lang* が ‘F’ 又は ‘f’ で始まれば、FORTRAN 言語の規約（1 を基準としたインデックス、最初の次元が最も早く変わる）が使用されます。どちらの場合にも、データは同じ順番で表示され、注釈のみが異なります。このオプションは大量の多次元データを一覧する時に便利です。

-f lang　各データ値（文字配列中の個々の文字は除く）についての注釈が、連なる CDL コメント形式（ ‘//’ で始まるテキスト）でデータ部分に含まれる。*lang* が ‘C’ 又は ‘c’ で始まれば C 言語の規約（零基準のインデックス、最終次元が最も早く変わる）が使用される。*lang* が ‘F’ 又は ‘f’ で始まれば、FORTRAN 言語の規約（1 を基準としたインデックス、最初の次元が最も早く変わる）が使用されます。どちらの場合にも、データは同じ順番で表示され、注釈のみが異なります。このオプションでは、各データ値が完全に識別された形で、別の行に表示されるので、データをほかのフィルタを通してパイプする際に便利です。(‘-b’ 又は ‘-f’ のオプションのどちらか一方のみしか指定できません。)

-l len　文字でないデータ値のリストをフォーマットする際に使われる、一行の最大長のデフォルト値（80）を変える。

`-p float_digits[,double_digits]`

属性や変数の浮動小数点数又は倍精度のデータ値のデフォルトの精度（有効数字の桁数）を指定するのに使われる。指定された場合には、変数の `c_format` 属性の値より優先される。浮動小数点データは有効数字 *float_digits* 桁で表示される。*double_digits* も指定されている場合には、倍精度値も同じ桁数で表示される。'-p' 指定がなされていない場合には、浮動小数点と倍精度のデータはそれぞれ有効数字 7 桁と 15 桁になる。精度を下げれば、CDL ファイルを小さくすることができる。浮動小数点と倍精度の両方が指定する場合には、2 つの値をカンマ（空白無し）で区切り、このコマンドに対して単一の引数として与えなければならない。

`-n name`

CDL で は、'ncgen -b' がデフォルトで NetCDF ファイル名を付ける際に NetCDF ファイルの名前を必要とする。デフォルトでは、`ncdump` は入力 NetCDF ファイル名から拡張子を取り払った後､残されたファイルメ名の最後の要素でもって名前を付ける。違う名前を指定する場合には '-n' オプションを使いましょう。'ncgen -b' で使われる出力ファイル名を指定することは可能ですが、`ncdump` を使用し生成された CDL ファイルを編集し、その編集された CDL ファイルから 'ncgen -b' によって新しい NetCDF ファイルを生成する際に、貴重な NetCDF ファイルをうっかり上書きしてしまわないように、`ncdump` にデフォルト名を変えさせることをお勧めします。

例

`foo.nc` という NetCDF ファイルのデータ構造を見ましょう。

```
ncdump -c foo.nc
```

注釈に C スタイルのインデックスを使い、NetCDF dataset `foo.nc` の構造とデータの CDL 版を注釈付きで生成する。

```
ncdump -b c foo.nc > foo.cdl
```

NetCDF dataset `foo.nc` の変数 `uwind` と `vwind` のみのデータを出力し、浮動小数点データを有効数字 3 桁で表示する。

```
ncdump -v uwind,vwind -p 3 foo.nc
```

インデックスに FORTRAN の規約を使用し、変数 `omega` のデータの完全注釈付き（一行につき１データ）リストを作成し、更に生成される CDL ファイル中の NetCDF ファイル名を `omega:` に変更する。

```
ncdump -v omega -f fortran -n omega foo.nc > Z.cdl
```

11　良くある質問（FAQ）への回答

この章では、NetCDF に関して最も良くある質問に対して答えます。より包括的で最新の FAQ 文書は　`http://www.unidata.ucar.edu/packages/NetCDF/faq.html` にあります。

NetCDF とは？

NetCDF（network Common Data Form）は配列指向のデータアクセスのためにインターフェースで、そのインターフェースの実装を与える C・FORTRAN・C++・Perl 用のソフトウェアライブラリを集めたものです。NetCDF ソフトウェアは Colorado 州の Boulder にある Unidata Program Center の Glenn Davis、Russ Rew、と Steve Emmerson によって開発され、他の NetCDF ユーザからの貢献にによって増強されました。NetCDF ライブラリは配列を表現するための機種独立のフォーマットを定義しています。インターフェース・ライブラリ・フォーマットが合わさって配列指向データの生成、アクセス、そして共有をサポートしています。

NetCDF data は：

- 自己記述的である。NetCDF ファイルはその中身のデータに関する情報を含んでいる。
- ネットワーク透過性がある。NetCDF ファイルは、整数・文字・浮動小数点数を異なる形式で格納するコンピューターからでもアクセスできる。
- 直接アクセスできる。大きなデータセットの小さな部分集合に、全てのデータを最初に読み込む必要無しに、効率的にアクセスできる。.
- 追加ができる。一つの次元に沿って複数の変数に対し、そのデータをコピーしたり構造を再構築する必要無しに、NetCDF ファイルにデータを追加することができる。NetCDF ファイルの構造も変更できるが、場合によってはデータをコピーすることになってしまう。
- 共有できる。一つの書き込みと複数の読み込みが同時に同一の NetCDF ファイルにアクセスすることができる。

NetCDF ソフトウェアパッケージの取得法は？

ソースの 配布は下記のディレクトリから anonymous FTP 経由で手に入れることができる。

`ftp://ftp.unidata.ucar.edu/pub/NetCDF/`

このディレクトリには以下のファイルがある。

`NetCDF.tar.Z`	一般向けの最新版のソースコードの tar ファイルを圧縮したもの。
`NetCDF-beta.tar.Z`	現行のベータテスト版。

幾つかのプラットフォーム用のバイナリ配布版は以下のディレクトリから取得できる。

ftp://ftp.unidata.ucar.edu/pub/binary/

Perl インターフェース用のソースは 別のパッケージとして以下のディレクトリから anonymous FTP 経由で取得できる。

ftp://ftp.unidata.ucar.edu/pub/NetCDF-perl/.

World Wide Web 上で NetCDF の情報にアクセスできるか？

はい、できます。この FAQ 文書の最新版、NetCDF User's Guide のハイパーテキスト版、及びその他の情報は以下にあります。

http://www.unidata.ucar.edu/packages/NetCDF.

以前のバージョンから 何が変わりましたか？

3 版は同じフォーマットを保持していますが、C と Fortran 用に、自動型変換に加え型変換の安全性を向上させた新しいインターフェースを導入しています。詳細については下記を参照してください。

http://www.unidata.ucar.edu/packages/NetCDF/release-notes.html.

NetCDF についての議論や質問のための メーリングリストはありますか？

はい、あります。メーリングリストに関する情報や参加 / 脱会方法についての質問は majordomo@unidata.ucar.edu まで、subject 無しで本文に次のように記入したメールを送ってください。

```
info NetCDFgroup
```

他に誰が NetCDF を 使っていますか？

NetCDF メーリングリストは 15 カ国に渡り 500 程の登録者がいます。(このうちの幾つかはより多くのアドレスへのエイリアスです。) 幾つかのグループは NetCDF を配列指向データを表現する標準的な方法として採用しています。それらには、大気科学、水理学、海洋学、環境モデリング、地球物理学、クロマトグラフィ、質量分析学、ニューロイメージング等が含まれます。

NetCDF を使ったプロジェクトやグループの幾つかについての情報は下記にあります。

http://www.unidata.ucar.edu/packages/NetCDF/.html.

NetCDF ファイルの 物理的なフォーマットはどのようなものですか？

異なるデータ構成の性能の含みを明らかにするのに十分なレベルの NetCDF データの物理的構造の説明に関しては、9 章 「NetCDF ファイルの構造と性能」 (p.101) を参照してください。又、ファイルフォーマットについての詳しい仕様については、Appendix A 「ファイルフォーマット仕様」 p.121 を参照してください。

NetCDF データをアクセスするプログラムは、全てのアクセスを文書化されたインターフェースを通じて行なうべきであり、NetCDF データの物理的フォーマットに依存するべきではない。そのようにしておけば、将来フォーマットが変更されてもプログラムを変更する必要が生じない。なぜならば、そのような変更は旧バージョン・新バージョン両方のフォーマットをサポートするようにライブラリも変更されるからである。

NetCDF はどこで動作しているか？

NetCDF の現行のバージョンは下記のプラットフォーム上でテストされ、成功しています。

- AIX-4.1
- HPUX-9.05
- IRIX-5.3
- IRIX64-6.1
- MSDOS (using gcc, f2c, and GNU make)
- OSF1-3.2
- OpenVMS-6.2
- OS/2 2.1
- SUNOS-4.1.4
- SUNOS-5.5
- ULTRIX-4.5
- UNICOS-8
- Windows NT-3.51

NetCDF データには他にどんなソフトウェアが使えるか？

現行の Unidata からの NetCDF 配布版に含まれるユーティリティは、NetCDF ファイルを可読な ASCII 形式に変換する ncdump、可読な ASCII 形式のファイルからバイナリの NetCDF ファイルに変換し直す又はその NetCDF ファイルを生成する C もしくは FORTRAN のプログラムに戻す ncgen である。

幾つかの商用又は無償の解析及びデータ視覚化 パッケージが NetCDF データアクセスに適応している。これらのパッケージや NetCDF データを処理し表示するために使える他のソフトウェアについては下記を参照のこと。

http://www.unidata.ucar.edu/packages/NetCDF/software.html.

科学的なデータ用には他にどんなフォーマットが存在するか？

Scientific Data Format Information FAQ, が `http://fits.cv.nrao.edu/traffic/scidataformats/faq.html` にあり、CDF やHDF を含む配列指向データ用の他のアクセスインターフェースやフォーマットを分かりやすく紹介しています。

バグ報告はどうすれば良い？

バグを発見したら、その情報をを `support@unidata.ucar.edu` に送ってください。これは `NetCDFgroup` メーリングリスト全体で扱うには適さない質問や議論をするためのアドレスでもあります。

過去の 問題報告はどのようにして検索できるか？

NetCDF ホームページの一番下に検索フォームがあり、サポートの質問及び Unidata サポートスタッフの回答に対して全テキスト検索ができます。

C++ インターフェースは C インターフェースとどう違うのですか？

C++ はC インターフェースによって提供される機能を全て持っている。(ただし、`nc_put_varm_`*type* と `nc_get_varm_`*type* のマップされた配列アクセスを除く。) C++ インターフェースを使用すると (`http://www.unidata.ucar.edu/packages/NetCDF/cxxdoc_toc.html`)NetCDF 要素の ID は不必要になり、属性を生成する際に型の指定が不要になる。さらに、次元を扱う際に、より直接的に扱うことができる。しかし、C++ インターフェースはC に比べて未成熟で、C ほど広く使われていない。さらに、C++ インターフェースの文章はあまり広範ではなく、NetCDF データモデルと C インターフェースに慣れていることを前提としている。

FORTRAN インターフェースと C インターフェースはどう異なるのか？

FORTRAN インターフェースはC インターフェースの全ての機能を提供しています。FORTRAN インターフェースは 配列インデックス、添え字の順番、及び文字列に関して、FORTRAN 規約を使っている。異なる言語インターフェースを使用して書かれたデータのディスク上のフォーマットは同じである。C 言語のプログラムで書かれたデータは FORTRAN プログラムから読むことが出来、又、逆も可能である。

Perl インターフェースと C インターフェースはどう異なるのか？

Perl **インターフェース**は C インターフェース全ての機能を提供しています。Perl インターフェース (`http://www.unidata.ucar.edu/packages/NetCDF-perl/`) は配列や文字列に関して Perl の規約に従っています。異なる言語インターフェースを使用して書かれたデータのディスク上のフォーマットは同じである。C 言語のプログラムで書かれ

たデータはPerl プログラムから読むことが出来、又、逆も可能である。

12　単位

Unidata Program Center が開発した単位ライブラリによって フォーマットされたバイナリ形式の単位間の変換を行い、また、バイナリ形式で単位型代数演算を行なうことが可能です。単位ライブラリそのものは自己完結型であり、NetCDF ライブラリとの間には依存性はありません。それでもこのライブラリは一般的な NetCDF プログラムを書く際には非常に有用ですので、手に入れることをお勧めします。この ライブラリと関連文書は `http://www.unidata.ucar.edu/packages/udunits/` から取得できます。

以下に Unidata 単位ライブラリの関数 `utScan()` によって解釈できる単位文字列の例を挙げてあります。

```
10 kilogram.meters/seconds2
10 kg-m/sec2
10 kg m/s^2
10 kilogram meter second-2
(PI radian)2
degF
100rpm
geopotential meters
33 feet water
milliseconds since 1992-12-31 12:34:0.1 -7:00
```

単位とは単位の任意の整数冪に任意の定数を掛けたものとして指定されます。割り算はスラッシュ `'/'`、掛け算は空白・ピリオド `'.'`・ハイフン `'-'`, のいずれか、冪算は整数の添え字又は冪乗演算子 `'^'` ・ `'**'` で表わされます。括弧を用いて表記をグループ化したり明瞭化することもできます。最後の例のタイムススタンプは特殊なケースとして扱われます。

任意のガリレオ変換（ すなわち、$y = ax + b$）も許されています。特に、温度の変換は正しく扱われています。次の指定：

```
degF @ 32
```

は原点を華氏 32 度（つまり摂氏 0 度）にシフトした華氏での温度表記です。従って、摂氏での表記は次のような単位と等しくなります：

```
1.8 degF @ 32
```

原点シフトの演算が掛け算より優先されることに注意してください。演算の優先順位は（下位から上位に向かって）除算、乗算、原点移動、冪算になります。

関数 `utScan()` は全ての SI 接頭語（つまり、`"mega"` 、`"milli"`）やそれらの短縮形（つまり、`"M"` 、`"m"`）に対応できます。

関数 `utPrint()` は常に単位指定を一意にコード化します。誤った解釈を防ぐために、このコード化のスタイルをデフォルトとして使用することをお勧めします。一般的には、単位は基本単位・因数・ 冪指数によってコード化されます。基本単位は空白に

よって区切られ、冪指数は対応する単位に直接付加されます。上記の例は次のようにコード化されます：

```
10 kilogram meter second-2
9.8696044 radian2
0.555556 kelvin @ 255.372
10.471976 radian second-1
9.80665 meter2 second-2
98636.5 kilogram meter-1 second-2
0.001 seconds since 1992-12-31 19:34:0.1000 UTC
```

(華氏単位が原点 255. 372 kelvin からの分数の偏差として kelvin 単位の分数としてコード化されていることに注意してください。さらに、最後の例では時刻が UTC に変換されていることにも注意してください。)

単位ライブラリのデータベースはフォーマットされたファイルで単位定義を含み、このパッケージを初期化するのに使われます。有効な単位名や記号等はまずここで探して下さい。

この単位ファイルのフォーマットに関しては内部に文書があり、ユーザーは必要に応じてファイルを 修正することが出来ます。特に、単位や定数（さらに既存の単位や定数の異なった綴り）は簡単に付け足していくことが出来ます。

関数 `utScan()` は大文字小文字を区別します。これによって不都合が生じるようでしたら、単位ファイルに適切な項目を追加してください。

デフォルト単位ファイルにある単位の短縮形は直感的ではないかもしれません。特に次に挙げるものについては注意が必要です：

For	Use	Not	Which Instead Means
Celsius	Celsius	C	coulomb
gram	gram	g	〈standard free fall〉
gallon	gallon	gal	〈acceleration〉
radian	radian	rad	〈absorbed dose〉
Newton	newton or N	nt	nit (unit of photometry)

単位ライブラリについての更なる情報については、この配布版に付属のマニュアルページをご参照ください。

Appendix A　ファイルフォーマット仕様

この appendix では NetCDF ファイルフォーマット 1 版の仕様を述べます。このフォーマットは少なくとも NetCDF ライブラリ 3.0 版までは使用される予定です。

このフォーマットはまず最初に BNF 文法表記によって正式に表現されます。この文法では、オプションの要素は括弧（‘[’ と ‘]’）によって囲まれます。注釈は ‘//’ の後に続きます。端末語でないものは小文字で、端末語は大文字で表記されます。０又はそれ以上の項目を並べる場合には ‘[entity …]’ と表記されます。

フォーマット仕様詳細

```
NetCDF_file := header  data

header  := magic  numrecs  dim_array  gatt_array  var_array

magic   := 'C'  'D'  'F'  VERSION_BYTE

VERSION_BYTE := '\001'    // ファイルフォーマットのバージョン番号

numrecs    := NON_NEG

dim_array  :=  ABSENT | NC_DIMENSION  nelems  [dim …]

gatt_array :=  att_array  // グローバル属性

att_array  :=  ABSENT | NC_ATTRIBUTE  nelems  [attr …]

var_array  :=  ABSENT | NC_VARIABLE   nelems  [var …]

ABSENT  := ZERO  ZERO     // 配列が無いことを意味する（ nelems == 0 に同じ）

nelems  := NON_NEG        // 以下のシークエンスの要素数

dim     := name  dim_length

name    := string

dim_length := NON_NEG     // ０であればこれは記録次元。
                          // 記録次元は１つまでである。

attr    := name  nc_type  nelems  [values]

nc_type := NC_BYTE | NC_CHAR | NC_SHORT | NC_INT | NC_FLOAT | NC_DOUBLE

var     := name  nelems  [dimid …]  vatt_array  nc_type  vsize  begin
                          // nelems は変数のランク（次元数）である。
                          // スカラーなら 0 、ベクトルなら 1、行列なら 2、等
```

```
vatt_array :=  att_array  // 変数に特定の属性

dimid   := NON_NEG        // 変数形状のための次元 ID（dim_array へのインデックス）
                          // 最初の次元が記録次元である場合に限って、
                          // これを〝記録変数”と呼ぶ。

vsize    := NON_NEG       // 変数サイズ。記録変数で無い場合には、
                          // 変数データに割り当てらたスペース（単位はバイト）
                          // この数は次元長と次元型のサイズの積であり、
                          // 4 バイト境界に合わせて詰め込まれている。
                          // これが記録変数である場合には
                          // 記録ごとのスペースに対応する。
                          // NetCDF の”記録サイズ”は記録変数の
                          // vsize のを和として計算される。

begin   := NON_NEG        // 変数のスタート位置。この変数のデータの
                          // 先頭のファイル中におけるバイト単位の
                          //  オフセット（インデックス必要）

data    := non_recs  recs

non_recs := [values …]    // 記録変数ではない最初の変数、2 番目、ノのデータ。

recs    := [rec …]        // 最初の記録、2 番目の記録、ノ

rec     := [values …]     // 記録 n に対する最初の記録変数、
                          // 2 番目の記録変数、ノのデータ
                          // 特殊なケースのついては下記の注釈を参照のこと

values  := [bytes] | [chars] | [shorts] | [ints] | [floats] | [doubles]

string  := nelems  [chars]

bytes   := [BYTE …]  padding

chars   := [CHAR …]  padding

shorts  := [SHORT …]  padding

ints    := [INT …]

floats  := [FLOAT …]

doubles := [DOUBLE …]

padding := <次の 4 バイト境界までの 0, 1, 2, または 3 バイト>
                          // ヘッダーでは、詰め込むのは 0 バイトです
                          // データでは、詰め込むのは変数のフィル値である。

NON_NEG := <非負の値を持つ INT>
```

```
ZERO    := < 0の値を持つ INT>

BYTE    := <8 ビット byte>

CHAR    := <8 ビット ACSII/ISO でコード化された character>

SHORT   := <16 ビット符号付整数・ビッグエンディアン・ 2の補数表現 >

INT     := <32 ビット符号付整数・ビッグエンディアン・ 2の補数表現 >

FLOAT   := <32 ビット IEEE 単精度浮動小数点・ビッグエンディアン >

DOUBLE  := <64 ビット IEEE ニ倍精度浮動小数点・ビッグエンディアン >

// タグは 32 ビット整数
NC_BYTE      := 1          // データは8ビット符号付整数の配列
NC_CHAR      := 2          // データは文字配列（テキスト等）
NC_SHORT     := 3          // データは１６ビット符号付整数の配列
NC_INT       := 4          // データは３２ビット符号付整数の配列
NC_FLOAT     := 5          // データは IEEE 単精度浮動小数点の配列
NC_DOUBLE    := 6          //データは IEEE 倍精度浮動小数点の配列
NC_DIMENSION := 10
NC_VARIABLE  := 11
NC_ATTRIBUTE := 12
```

ファイルオフセットの計算

指定されたデータ値の オフセット（ファイル内の位置）を計算するには、指定された変数型 *nc_type* に適切なデータ値の一つの外部サイズ（バイト単位）を *external_sizeof* とする。

```
NC_BYTE         1
NC_CHAR         1
NC_SHORT        2
NC_INT          4
NC_FLOAT        4
NC_DOUBLE       8
```

nc_open（or nc_enddef）呼び出しは前もって *var_array* と示された変数配列内をスキャンし、*recsize* を計算するために〃記録”変数の *vsize* の和を計算する。

変数の次元サイズの積を右から左にとっていき、記録変数の最も左の（記録）次元は飛ばし、各変数についての結果を *product* 配列に格納する。例えば：

```
Non-record variable:

        dimension lengths:      [  5  3  2 7]
        product:                [210 42 14 7]

Record variable:
```

```
dimension lengths:      [0  2  9 4]
product:                [0 72 36 4]
```

この時点では、最も左にある積を次の４の倍数に丸めたものが変数サイズ、すなわち、上の文法においては *vsize* になる。例えば、上記の非記録変数では、*vsize* フィールド値は 212 （210 を次の４の倍数に丸めた値）となる。記録変数に対しては、*vsize* の値はちょうど 72 である。なぜならば、72 は既に４の倍数であるからである。

求めるデータ値の座標の配列を *coord* とし、求める結果を *offset* とする。この時、*offset* は単に、求める変数の最初のデータ値のファイルオフセット（その *begin* フィールド）に *coord* と *product* ベクトルの内積を変数の各データのサイズ（バイト単位）を掛けたものを加えた値となる。最後に、もしその変数が記録変数であれば、記録数 ‘coord[0]’ と記録サイズ *recsize* との積が加算され、最終的な *offset* 値が導かれる。

擬似 C コードにおける *offset* の計算は次のようになる。

```
for (innerProduct = i = 0; i < var.rank; i++)
        innerProduct += product[i] * coord[i]
offset = var.begin;
offset += external_sizeof * innerProduct
if(IS_RECVAR(var))
        offset += coord[0] * recsize;
```

故に、(外部表現法の）データ値を取得するには、次のようになる。

```
lseek(fd, offset, SEEK_SET);
read(fd, buf, external_sizeof);
```

特殊例：記録変数が一つしかない場合には 、各記録が 4 倍と境界に合っていなければならないという制限をはずすので、この場合には記録の詰め込みが行なわれません。

例

上の文法によれば、 最も小さな有効な NetCDF file で次元、変数、属性を持たない、従ってデータを持たないものを導くことができます。空の NetCDF ファイルの CDL 表現は次のようになります：

```
NetCDF empty { }
```

この 空の NetCDF ファイルは 32 バイトの大きさで、CDL 表現から ‘ncgen -b empty.cdl’ を使って空の NetCDF ファイルを生成して確認することが出来ます。この空ファイルはそれが NetCDF 1 版のファイルであることを示す４バイトの”マジックナンバー”である ‘C’, ‘D’, ‘F’, ‘\001’ で始まります。続いて、記録数・次元の空配列・グローバル属性の空配列・変数の空配列を表わす７つの３２ビット０が後にきます。

以下は、次の Unix コマンドを使ってビッグエンディアンマシン上で生成されたファイルの（編集済みの）ダンプです。

```
od -xcs empty.nc
```

ファイルの 16 バイトの各部分は 4 行で表示されています。最初の行はバイトを 16 進数表示し、2 行目はを文字表示しています。3 行目は２バイトごとにグループ化して、それを符号付 16 ビット整数として表示しています。4 行目は（手作業で追加されたものであるが）バイトを NetCDF 要素及び値として解釈したものを表示している。

```
   4344    4601    0000    0000    0000    0000    0000    0000
  C   D   F 001  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
  17220   17921   00000   00000   00000   00000   00000   00000
[magic number ] [  0 records  ] [  0 dimensions   (ABSENT)    ]

   0000    0000    0000    0000    0000    0000    0000    0000
 \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
  00000   00000   00000   00000   00000   00000   00000   00000
[  0 global atts  (ABSENT)    ] [  0 variables    (ABSENT)    ]
```

もう少し意味のある例として、このような CDL を考えてみましょう。

```
NetCDF tiny {
dimensions:
        dim = 5;
variables:
        short vx(dim);
data:
        vx = 3, 1, 4, 1, 5 ;
}
```

これは 92 バイト NetCDF ファイルに対応します。このファイルの変数済みのダンプは下記のようになります。

```
   4344    4601    0000    0000    0000    000a    0000    0001
  C   D   F 001  \0  \0  \0  \0  \0  \0  \n  \0  \0  \0 001
  17220   17921   00000   00000   00000   00010   00000   00001
[magic number ] [  0 records  ] [NC_DIMENSION ] [ 1 dimension ]

   0000    0003    6469    6d00    0000    0005    0000    0000
 \0  \0  \0 003   d   i   m  \0  \0  \0  \0 005  \0  \0  \0  \0
  00000   00003   25705   27904   00000   00005   00000   00000
[  3 char name = "dim"        ] [ size = 5    ] [ 0 global atts

   0000    0000    0000    000b    0000    0001    0000    0002
 \0  \0  \0  \0  \0  \0  \0 013  \0  \0  \0 001  \0  \0  \0 002
  00000   00000   00000   00011   00000   00001   00000   00002
 (ABSENT)     ] [NC_VARIABLE  ] [ 1 variable  ] [ 2 char name =

   7678    0000    0000    0001    0000    0000    0000    0000
  v   x  \0  \0  \0  \0  \0 001  \0  \0  \0  \0  \0  \0  \0  \0
  30328   00000   00000   00001   00000   00000   00000   00000
```

```
 "vx"         ] [1 dimension  ] [ with ID 0   ] [ 0 attributes

   0000    0000    0000    0003    0000    000c    0000    0050
 \0  \0  \0  \0  \0  \0  \0 003  \0  \0  \0  \f  \0  \0  \0   P
  00000   00000   00000   00003   00000   00012   00000   00080
 (ABSENT)     ] [type NC_SHORT] [size 12 bytes] [offset:    80]

   0003    0001    0004    0001    0005    8001
 \0 003  \0 001  \0 004  \0 001  \0 005 200 001
  00003   00001   00004   00001   00005  -32767
[    3] [    1] [    4] [    1] [    5] [fill ]
```

Appendix B Cインターフェースのまとめ

```
const char* nc_inq_libvers (void);
const char* nc_strerror    (int ncerr);

int nc_create         (const char *path, int cmode, int *ncidp);
int nc_open           (const char *path, int mode, int *ncidp);
int nc_set_fill       (int ncid, int fillmode, int *old_modep);
int nc_redef          (int ncid);
int nc_enddef         (int ncid);
int nc_sync           (int ncid);
int nc_abort          (int ncid);
int nc_close          (int ncid);
int nc_inq            (int ncid, int *ndimsp, int *nvarsp,
                       int *ngattsp, int *unlimdimidp);
int nc_inq_ndims      (int ncid, int *ndimsp);
int nc_inq_nvars      (int ncid, int *nvarsp);
int nc_inq_natts      (int ncid, int *ngattsp);
int nc_inq_unlimdim   (int ncid, int *unlimdimidp);

int nc_def_dim        (int ncid, const char *name, size_t len,
                       int *idp);
int nc_inq_dimid      (int ncid, const char *name, int *idp);
int nc_inq_dim        (int ncid, int dimid, char *name, size_t *lenp);
int nc_inq_dimname    (int ncid, int dimid, char *name);
int nc_inq_dimlen     (int ncid, int dimid, size_t *lenp);
int nc_rename_dim     (int ncid, int dimid, const char *name);

int nc_def_var        (int ncid, const char *name, nc_type xtype,
                       int ndims, const int *dimidsp, int *varidp);
int nc_inq_var        (int ncid, int varid, char *name,
                       nc_type *xtypep, int *ndimsp, int *dimidsp,
                       int *nattsp);
int nc_inq_varid      (int ncid, const char *name, int *varidp);
int nc_inq_varname    (int ncid, int varid, char *name);
int nc_inq_vartype    (int ncid, int varid, nc_type *xtypep);
int nc_inq_varndims   (int ncid, int varid, int *ndimsp);
int nc_inq_vardimid   (int ncid, int varid, int *dimidsp);
int nc_inq_varnatts   (int ncid, int varid, int *nattsp);
int nc_rename_var     (int ncid, int varid, const char *name);
int nc_put_var_text   (int ncid, int varid, const char *op);
int nc_get_var_text   (int ncid, int varid,       char *ip);
int nc_put_var_uchar  (int ncid, int varid, const unsigned char *op);
int nc_get_var_uchar  (int ncid, int varid,       unsigned char *ip);
int nc_put_var_schar  (int ncid, int varid, const signed char *op);
int nc_get_var_schar  (int ncid, int varid,       signed char *ip);
int nc_put_var_short  (int ncid, int varid, const short *op);
int nc_get_var_short  (int ncid, int varid,       short *ip);
int nc_put_var_int    (int ncid, int varid, const int *op);
int nc_get_var_int    (int ncid, int varid,       int *ip);
int nc_put_var_long   (int ncid, int varid, const long *op);
int nc_get_var_long   (int ncid, int varid,       long *ip);
```

```
int nc_put_var_float  (int ncid, int varid, const float *op);
int nc_get_var_float  (int ncid, int varid,       float *ip);
int nc_put_var_double (int ncid, int varid, const double *op);
int nc_get_var_double (int ncid, int varid,       double *ip);
int nc_put_var1_text  (int ncid, int varid, const size_t *indexp,
                       const char *op);
int nc_get_var1_text  (int ncid, int varid, const size_t *indexp,
                       char *ip);
int nc_put_var1_uchar (int ncid, int varid, const size_t *indexp,
                       const unsigned char *op);
int nc_get_var1_uchar (int ncid, int varid, const size_t *indexp,
                       unsigned char *ip);
int nc_put_var1_schar (int ncid, int varid, const size_t *indexp,
                       const signed char *op);
int nc_get_var1_schar (int ncid, int varid, const size_t *indexp,
                       signed char *ip);
int nc_put_var1_short (int ncid, int varid, const size_t *indexp,
                       const short *op);
int nc_get_var1_short (int ncid, int varid, const size_t *indexp,
                       short *ip);
int nc_put_var1_int   (int ncid, int varid, const size_t *indexp,
                       const int *op);
int nc_get_var1_int   (int ncid, int varid, const size_t *indexp,
                       int *ip);
int nc_put_var1_long  (int ncid, int varid, const size_t *indexp,
                       const long *op);
int nc_get_var1_long  (int ncid, int varid, const size_t *indexp,
                       long *ip);
int nc_put_var1_float (int ncid, int varid, const size_t *indexp,
                       const float *op);
int nc_get_var1_float (int ncid, int varid, const size_t *indexp,
                       float *ip);
int nc_put_var1_double(int ncid, int varid, const size_t *indexp,
                       const double *op);
int nc_get_var1_double(int ncid, int varid, const size_t *indexp,
                       double *ip);
int nc_put_vara_text  (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const char *op);
int nc_get_vara_text  (int ncid, int varid, const size_t *startp,
                       const size_t *countp, char *ip);
int nc_put_vara_uchar (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const unsigned char *op);
int nc_get_vara_uchar (int ncid, int varid, const size_t *startp,
                       const size_t *countp, unsigned char *ip);
int nc_put_vara_schar (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const signed char *op);
int nc_get_vara_schar (int ncid, int varid, const size_t *startp,
                       const size_t *countp, signed char *ip);
int nc_put_vara_short (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const short *op);
int nc_get_vara_short (int ncid, int varid, const size_t *startp,
                       const size_t *countp, short *ip);
int nc_put_vara_int   (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const int *op);
```

```
int nc_get_vara_int   (int ncid, int varid, const size_t *startp,
                       const size_t *countp, int *ip);
int nc_put_vara_long  (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const long *op);
int nc_get_vara_long  (int ncid, int varid, const size_t *startp,
                       const size_t *countp, long *ip);
int nc_put_vara_float (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const float *op);
int nc_get_vara_float (int ncid, int varid, const size_t *startp,
                       const size_t *countp, float *ip);
int nc_put_vara_double(int ncid, int varid, const size_t *startp,
                       const size_t *countp, const double *op);
int nc_get_vara_double(int ncid, int varid, const size_t *startp,
                       const size_t *countp, double *ip);
int nc_put_vars_text  (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       const char *op);
int nc_get_vars_text  (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       char *ip);
int nc_put_vars_uchar (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       const unsigned char *op);
int nc_get_vars_uchar (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       unsigned char *ip);
int nc_put_vars_schar (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       const signed char *op);
int nc_get_vars_schar (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       signed char *ip);
int nc_put_vars_short (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       const short *op);
int nc_get_vars_short (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       short *ip);
int nc_put_vars_int   (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       const int *op);
int nc_get_vars_int   (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       int *ip);
int nc_put_vars_long  (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       const long *op);
int nc_get_vars_long  (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       long *ip);
int nc_put_vars_float (int ncid, int varid, const size_t *startp,
                       const size_t *countp, const ptrdiff_t *stridep,
                       const float *op);
int nc_get_vars_float (int ncid, int varid, const size_t *startp,
```

```
                      const size_t *countp, const ptrdiff_t *stridep,
                      float *ip);
int nc_put_vars_double(int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const double *op);
int nc_get_vars_double(int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      double *ip);
int nc_put_varm_text  (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, const char *op);
int nc_get_varm_text  (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, char *ip);
int nc_put_varm_uchar (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, const unsigned char *op);
int nc_get_varm_uchar (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, unsigned char *ip);
int nc_put_varm_schar (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, const signed char *op);
int nc_get_varm_schar (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, signed char *ip);
int nc_put_varm_short (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, const short *op);
int nc_get_varm_short (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, short *ip);
int nc_put_varm_int   (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, const int *op);
int nc_get_varm_int   (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, int *ip);
int nc_put_varm_long  (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, const long *op);
int nc_get_varm_long  (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, long *ip);
int nc_put_varm_float (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, const float *op);
int nc_get_varm_float (int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, float *ip);
int nc_put_varm_double(int ncid, int varid, const size_t *startp,
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t *imapp, const double *op);
int nc_get_varm_double(int ncid, int varid, const size_t *startp,
```

```
                      const size_t *countp, const ptrdiff_t *stridep,
                      const ptrdiff_t * imap, double *ip);

int nc_inq_att        (int ncid, int varid, const char *name,
                       nc_type *xtypep, size_t *lenp);
int nc_inq_attid      (int ncid, int varid, const char *name, int *idp);
int nc_inq_atttype    (int ncid, int varid, const char *name,
                       nc_type *xtypep);
int nc_inq_attlen     (int ncid, int varid, const char *name,
                       size_t *lenp);
int nc_inq_attname    (int ncid, int varid, int attnum, char *name);
int nc_copy_att       (int ncid_in, int varid_in, const char *name,
                       int ncid_out, int varid_out);
int nc_rename_att     (int ncid, int varid, const char *name,
                       const char *newname);
int nc_del_att        (int ncid, int varid, const char *name);
int nc_put_att_text   (int ncid, int varid, const char *name, size_t len,
                       const char *op);
int nc_get_att_text   (int ncid, int varid, const char *name, char *ip);
int nc_put_att_uchar  (int ncid, int varid, const char *name,
                       nc_type xtype, size_t len, const unsigned char *op);
int nc_get_att_uchar  (int ncid, int varid, const char *name,
                       unsigned char *ip);
int nc_put_att_schar  (int ncid, int varid, const char *name,
                       nc_type xtype, size_t len, const signed char *op);
int nc_get_att_schar  (int ncid, int varid, const char *name,
                       signed char *ip);
int nc_put_att_short  (int ncid, int varid, const char *name,
                       nc_type xtype, size_t len, const short *op);
int nc_get_att_short  (int ncid, int varid, const char *name, short *ip);
int nc_put_att_int    (int ncid, int varid, const char *name,
                       nc_type xtype,size_t len, const int *op);
int nc_get_att_int    (int ncid, int varid, const char *name, int *ip);
int nc_put_att_long   (int ncid, int varid, const char *name,
                       nc_type xtype, size_t len, const long *op);
int nc_get_att_long   (int ncid, int varid, const char *name, long *ip);
int nc_put_att_float  (int ncid, int varid, const char *name,
                        nc_type xtype, size_t len, const float *op);
int nc_get_att_float  (int ncid, int varid, const char *name, float *ip);
int nc_put_att_double (int ncid, int varid, const char *name,
                       nc_type xtype, size_t len, const double *op);
int nc_get_att_double (int ncid, int varid, const char *name,
                       double *ip);
```

Appendix C　NetCDF 2 C トランジションガイド

C インターフェースの変更のまとめ

NetCDF3 版 では NetCDF ライブラリが完全に書き直されています。このバージョンは以前のより 2 倍は早くなっています。NetCDF ファイルのフォーマットはそのままなので、3 版で書かれたファイルは 2 版で読むことが出来、又、逆も可能です。

ライブラリの芯部は現在、ANSI C で書かれています。例えば、全てを通じてプロト型が使用されており、更に適当な個所では `const` 修飾子が使用されています。このバージョンをコンパイルするには ANSI C コンパイラが必要です。

ライブラリを書き直すことによって、進歩した C や FORTRAN のインターフェースを利用する機会が得られ、かなりの恩恵がありました。

- 一般的な void* ポインタを使用する必要性が無くすことによる型の安全性。
- 言語非依存型である外部 NetCDF 型（NC_BYTE, …, NC_DOUBLE）と言語依存型である内部データ型（char, …, double）との間の不適切な カップリングを排除することによる自動型変換。
- 圧縮データ及びマルチスレッドのサポートを問題なく加えるために障害を取り除くことにより、将来の改正に対しサポートする。
- 各関数の返し値の呼び出しプログラムにエラーステータスを一律に伝達することによりスタンダードな エラー動作 を得る。

2 版の C インターフェースを使用しているプログラムを書き直す必要はありません。なぜならば、NetCDF-3 ライブラリには旧関数・グローバル・動作を全てサポートする後方互換性インターフェースが含まれているからです。 この新しいインターフェースの恩恵が NetCDF のアプリケーション中でそれらを使用するきっかけになることを願います。NetCDF-2 の呼び出しを一つ一つ対応する NetCDF-3 の呼び出しに置き換えていくことで、旧アプリケーションを新しいインターフェースに徐々に変換していくことは可能です。アプリケーション中で NetCDF-3 の呼び出しのみが使用されていることを確認するために、プリプロセッサマクロ（`NO_NetCDF_2`）が用意されています。

NetCDF の実行の変更は、ほとんど全てのプラットフォーム上での携帯性、保全性、及びパフォーマンスの向上に繋がりました。I/O と型層を完全に切り離すことによりプラットフォーム固有の最適化が簡単になりました。新しいライブラリは販売元が提供している XDR ライブラリを使用していないので、NetCDF を使用するプログラム同士をリンクすることが 簡易になり、ほとんどの場合においてデータアクセスのスピードが速くなっています。

新しい C インターフェース

まず最初に NetCDF-2 インターフェースを使った C コードの例です。

```
void *bufferp;
nc_type xtype;
ncvarinq(ncid, varid, ..., &xtype, ...
...
/* 次元と型に応じて bufferp を配置 */
...
if (ncvarget(ncid, varid, start, count, bufferp) == -1) {
    fprintf(stderr, "Can't get data, error code = %d\n",ncerr);
    /* 対応する */
    ...
}
switch(xtype) {
    /* 型別にデータを扱う */
...
case  NC_FLOAT:
    fanalyze((float *)bufferp);
    break;
case NC_DOUBLE:
    danalyze((double *)bufferp);
    break;
}
```

同じことを新しいNetCDF-3の C インターフェースを使って扱うとこのようになります。

```
/*
 * 解析に倍精度を使用したい
 */
double dbuf[NDOUBLES];
int status;

/* よって、データを倍精度として取得する関数を使用する。*/
status = nc_get_vara_double(ncid, varid, start, count, dbuf)
if (status != NC_NOERR) {
   fprintf(stderr, "Can't get data: %s\n", nc_strerror(status));
    /* 対応する */
    ...
}
danalyze(dbuf);
```

上の例は関数の名前、データ型の変換、エラーの取り扱い等における変更を表わしています。詳細については後述してあります。

関数名の規約

NetCDF-3 のライブラリは新しい命名の規約に従っており、NetCDF プログラムをより読

み易くしようと試みています。例えば、変数名を変更する関数の名前は以前の ncvarrename ではなく nc_rename_var となります。

全ての NetCDF-3 C 関数名は nc_ 接頭辞で始まります。関数名の２番目の部分は動詞のようなもので、get、put、inq（問い合わせるの inquire）又は open 等があります。名前の３番目の部分は一般的に動詞の目的語にあたります。例えば、次元、変数、属性を扱う関数では dim、var、及び att となります。様々な変数の I/O 操作を 識別するためには、一文字の修飾子が var に付加されます。

- var 変数全体へのアクセス
- var1 単一の変数へのアクセス
- vara 配列又は配列断面へのアクセス
- vars 値の部分サンプルへのストライドアクセス
- varm メモリ内で隣接していない値へのマップされたアクセス

変数名と属性関数の末尾には最終の引数の型を示す部分があります：text、uchar、schar、short、int、long、float、そして double です。関数名のこの部分はプログラム中で使用しているデータの格納庫の型を指しています：文字列、非符号付記号、符号付記号、等です。

更に、全ての公の C インターフェースでは、マクロ 名は接頭辞 NC_ で始まります。例えば、以前 MAX_NC_NAME であったマクロ は現在、 NC_MAX_NAME であり、以前 FILL_FLOAT であったものは NC_FILL_FLOAT となっています。

既に述べたように、後方互換性を保証するために古い名前は全てサポートされています。

型変換

新しいインターフェースにおいては、どのような数値型へ、又は数値型からの自動変換も提供されているので、ユーザーは数値変数の外部データ型を知っている必要はありません。この特徴を使って、コードを外部データ型に依存しないようにして簡単にすることができます。void* ポインタ の排除することによって、以前のインターフェースでは不可能であった、コンパイル時に型エラーを発見することを可能にしました。変数の外部データ型を扱うに際にプログラムを変更する必要が無いために、新しいインターフェースはプログラムをより強固にすることが出来ます。

外部数値型からの変換が必要な場合はライブラリによって扱われます。この自動変換機能と外部データ表記の内部データ型からの分離は NetCDF4 版においてより重要になります。4 版では、自然に対応する内部データ型が存在しない圧縮データ（例えば 11 ビット値の配列）用の新しい外部データ型が用意される予定です。

ある数値型から他の型に変換する操作は、ターゲットの型が変換された値を表現できない場合にエラーがを引き起こします。(NetCDF-2 においては、そのようなオーバーフローは XDR 階層でのみ起こり得ました。）例えば、float は外部では NC_DOUBLE （IEEE

浮動小数点数）として格納されているデータをもつことが出来ないかもしれません。値の配列をアクセスする際には、表現し得る範囲を超えた値が一つ又はそれ以上ある場合には、NC_ERANGE エラーが返されますが、他の値は正しく変換されます。

型変換において、単に精度のロスが生じただけではエラーが返されないことに注意してください。そのため、例えば int に 二倍精度の値を読み込んだ場合には、その二倍精度の値の大きさがプラットフォーム上の ints で表現できる範囲を超えない限りエラーは生じません。同様に、仮数部分に整数のビットを全て収めることの出来ない float に大きな整数を読み込み、精度が失われてもエラーは生じません。このような精度のロスを防ぐためには、アクセスする変数の外部データ型を確認し、それと互換性のある内部データ型を使用しましょう。

新しいインターフェースはテキスト列を表現する文字配列と小さい整数を表わす 8 ビットバイトの配列とを区別します。このインターフェースはテキスト列のための text、uchar、及び schar、非符号バイト値と符号付バイト値の内部データ型をサポートします。

関数 _uchar と _schar は曖昧さを排除し、非符号付と符号付のバイトデータを両方サポートするために NetCDF-3 で導入されました。NetCDF-2 においては、外部 NC_BYTE 型が非符号付もしくは符号付の値のどちらを表現しているのかを決めるのはユーザーでした。NetCDF-3 においては、NC_BYTE は short、int、long、float、又は double への変換に際し NC_BYTE は符号付の値として扱います。（もちろん、内部データ型が符号付 char の場合には変換は行なわれません。）関数 _uchar では、NC_BYTE をあたかも非符号付のように扱います。よって、NC_BYTE と非符号付 char 間の変換においては NC_ERANGE エラーは発生しようがありません。

エラーの取り扱い

新しいインターフェースのエラーの取り扱い方は NetCDF-2 の方法とは異なる。NetCDF-2 のインターフェースでは、エラーが検知された時のデフォルト動作はエラーメッセージを出力して exit することであった。エラーの取り扱いをコントロールするには、グローバル変数 ncopts にフラグビットを設定しなければならず、エラーの原因を究明するために、別のグローバル変数 ncerr 値をテストしなければならなかった。

新しいインターフェースにおいては、関数が返す整数ステータスは成功 / 失敗のみではなく、エラーの原因をも示す。グローバル変数 ncerr と ncopt は 削除されました。ライブラリは何かを出力したり、exit を呼び出そうとすることはありません。（ただし、NetCDF-2 互換の関数を使用している場合はこの限りではありません。）関数の返されたステータスを確認し、手動で行なわなければなりません。平行した（マルチプロセッサ）実行をきれいにサポートするために、又、NetCDF が使用される環境についての仮定を減らすために、これらのグローバルは削除されました。新しい動作は、独自の GUI インターフェースを持つアプリケーション中で、NetCDF を隠された階層として使用するのにより適したサポートを提供しているはずです。

NC_LONG と NC_INT

NetCDF-2 インターフェースが NC_LONG を使って 32 ビット整数に対応した外部データ型を同定していたのに対し、新しいインターフェースは NC_INT を使います。NC_LONG は後方互換性のために、NC_INT と同じ値を取るように定義されているが、新しいコードでは使用されるべきでない。新しい 64 ビットプラットフォームが 64 ビット整数に long を使用しているので、この名前の衝突によって引き起こされる混乱を少なくしたいのです。未だに 64 ビット整数に対応する NetCDF 外部データ型が存在しないことに注意してください。

何が欠けているか？

新しい C インターフェースは“記録 I/O”関数を 3 つ（ncrecput、ncrecget、そして ncrecinq)、NetCDF-2 インターフェースから除去している。ただし、これらの関数は NetCDF-2 互換のインターフェースではまだサポートされている。

このことは 1 回の記録指向の呼び出しを、各記録変数につき一つの、複数回の型依存の呼び出しで置き換えなければならないことを意味する。例えば、 ncrecput への 1 回の呼び出しは適切な nc_put_var 関数への複数回の（アクセスされた各変数につき一回の）呼び出しによって置き換えることができる。記録指向の関数が削除されたのは、そのようなインターフェースでは型安全性と自動型変換を提供する簡単な方法が無いからである。

2 版のインターフェースの関数 nctypelen に対応する関数は無い。内部データ型と外部データ型を分離することと、新しい型変換インターフェースによって、 nctypelen は不要になる。ユーザーは ネイティブの型でもって読み書きするので、size 演算子さえあれば、完璧にある値に割り当てるスペースを決定することができる。

以前のライブラリでは、NetCDF オブジェクトの名前に使用された記号が CDL の制約に沿っているか判断する方法が無かった。CDL を使用している ncdump と ncgen のユーティリティは、名前に関しては英数字、“_” 、“-” のみの使用をを許可している。この制約は新しい次元・属性・変数を生成する際に、ライブラリによっても強制されることになった。制約の弱い名前を冠する既存の要素はまだ問題なく使える。

その他の変更

NetCDF-2 に対応する関数が存在しない、新しい関数が NetCDF-3 には２つある。nc_inq_libvers と nc_strerror である。現行の NetCDF ライブラリは nc_inq_libvers の文字列として返される。NetCDF 関数の呼び出しによって返されたステータスに対応するエラーメッセージは関数 nc_strerror によって記号列として返される。

新しい NC_SHARE フラグはアクセスのデフォルトバッファを防ぐために、nc_open 又は nc_create 呼び出しで使用できる。 NC_SHARE を使用することによってNetCDFファイルに同時にアクセスすれば、ディスクのアップデートが同期であることを確認するため

に、アクセスが終了するたび毎に nc_sync を呼び出す必要が無い。従属的なデータ（例えば属性値）への変更にも注意しなければならない。なぜならば、これらは NC_SHARE フラグを使用しても自動的に伝達されないからである。このためには、まだ nc_sync 関数が必要である。

2 版のインターフェースの問い合わせ関数は一つしかなく、 ncvarinq によって名前、型、変数の形を得ていた。同様に、次元・属性・NetCDF ファイルに関する情報を得る関数も一つしか無かった。この情報の部分集合が得る場合には、不必要な情報を押さえるために NULL 引数を与えなければならなかった。新しいインターフェースでは、新たな問い合わせ関数ができ、これらの項目を個別に返す。それによって引数の数え損ねによるエラーが起こりにくくなった。

以前の実装では ncvarput と ncvarget 呼び出し中で 0 値のカウント要素が 指定されているとエラーが返された。この制約がはずされたことによって、nc_put_var と nc_get_var のファミリーの関数が 0 値のカウント要素を使って呼び出せることになった。これはデータがアクセスされないことを意味し、一見、無意味のように思われるが、0 値のカウントを特殊なケースとして扱わなくて良いので、プログラムによっては単純になります。

以前の実装では ncvardef 中の変数の形を指定するのに同じ次元を 2 回以上使用するとエラーが返されました。自己相関行列など同じ次元を 2 度使用することに意味のある良い例があるので、この制約は NetCF-3 実装では緩められた。

新しいインターフェースでは、 nc_put_varm と nc_get_varm 族の関数に対する imap 引数の単位は 、望まれる内部データ型のデータ要素の数によって表わされ、NetCDF2 版のマップされたアクセスインターフェースのようにバイトでは表わされない。

下記は NetCD-2 の関数名と対応する NetCDF-3 関数の対応表です。NetCDF-2 関数の引数のリストは NetCDF-2 User's Guide に載っています。

ncabort	nc_abort
ncattcopy	nc_copy_att
ncattdel	nc_del_att
ncattget	nc_get_att_double, nc_get_att_float, nc_get_att_int, nc_get_att_long, nc_get_att_schar, nc_get_att_short, nc_get_att_text, nc_get_att_uchar
ncattinq	nc_inq_att, nc_inq_attid, nc_inq_attlen, nc_inq_atttype
ncattname	nc_inq_attname
ncattput	nc_put_att_double, nc_put_att_float, nc_put_att_int, nc_put_att_long, nc_put_att_schar, nc_put_att_short, nc_put_att_text, nc_put_att_uchar

ncattrename	nc_rename_att
ncclose	nc_close
nccreate	nc_create
ncdimdef	nc_def_dim
ncdimid	nc_inq_dimid
ncdiminq	nc_inq_dim, nc_inq_dimlen, nc_inq_dimname
ncdimrename	nc_rename_dim
ncendef	nc_enddef
ncinquire	nc_inq, nc_inq_natts, nc_inq_ndims, nc_inq_nvars, nc_inq_unlimdim
ncopen	nc_open
ncrecget	*(none)*
ncrecinq	*(none)*
ncrecput	*(none)*
ncredef	nc_redef
ncsetfill	nc_set_fill
ncsync	nc_sync
nctypelen	*(none)*
ncvardef	nc_def_var
ncvarget	nc_get_vara_double, nc_get_vara_float, nc_get_vara_int, nc_get_vara_long, nc_get_vara_schar, nc_get_vara_short, nc_get_vara_text, nc_get_vara_uchar
ncvarget1	nc_get_var1_double, nc_get_var1_float, nc_get_var1_int, nc_get_var1_long, nc_get_var1_schar, nc_get_var1_short, nc_get_var1_text, nc_get_var1_uchar
ncvargetg	nc_get_varm_double, nc_get_varm_float, nc_get_varm_int, nc_get_varm_long, nc_get_varm_schar, nc_get_varm_short, nc_get_varm_text, nc_get_varm_uchar, nc_get_vars_double, nc_get_vars_float, nc_get_vars_int, nc_get_vars_long, nc_get_vars_schar, nc_get_vars_short, nc_get_vars_text, nc_get_vars_uchar
ncvarid	nc_inq_varid

ncvarinq	nc_inq_var, nc_inq_vardimid, nc_inq_varname, nc_inq_varnatts, nc_inq_varndims, nc_inq_vartype
ncvarput	nc_put_vara_double, nc_put_vara_float, nc_put_vara_int, nc_put_vara_long, nc_put_vara_schar, nc_put_vara_short, nc_put_vara_text, nc_put_vara_uchar
ncvarput1	nc_put_var1_double, nc_put_var1_float, nc_put_var1_int, nc_put_var1_long, nc_put_var1_schar, nc_put_var1_short, nc_put_var1_text, nc_put_var1_uchar
ncvarputg	nc_put_varm_double, nc_put_varm_float, nc_put_varm_int, nc_put_varm_long, nc_put_varm_schar, nc_put_varm_short, nc_put_varm_text, nc_put_varm_uchar, nc_put_vars_double, nc_put_vars_float, nc_put_vars_int, nc_put_vars_long, nc_put_vars_schar, nc_put_vars_short, nc_put_vars_text, nc_put_vars_uchar
ncvarrename	nc_rename_var
(none)	nc_inq_libvers
(none)	nc_strerror