XDF

a language for the definition and exchange of biological data sets

DESCRIPTION & MANUAL

Version 3.3, 2nd November 1989

Robert Allkin

Computing Section, Royal Botanic Gardens, Kew, Richmond, Surrey TW9 3AB
United Kingdom

&

Richard J. White

Biology Department, The University, Southampton SO9 5NH
United Kingdom

CONTENTS

Summary

'XDF' (the '*Exchange Data Format*') is a medium for the exchange of diverse classes of biological data between different database projects and application programs. This document provides a detailed definition of the syntax and conventions followed in the XDF language and of the principal XDF '*directives*'. The manual is intended for biologists using XDF either to take advantage of predefined, standard transfer formats or to define their own transfer format. A separate document describes one such transfer format, developed in XDF, for use between species diversity databases.

The need for XDF arises from

(i) the number of application programs used by biologists which have incompatible data formats,

(ii) the increasing number of taxonomic database projects running under different computing environments and

(iii) the growing cost to the taxonomic community of massaging data from one program, project or format to another and developing special translation software.

XDF is a language for defining biological data sets with its own syntax and vocabulary. Data sets prepared in XDF consist of text files that contain only normal printable characters, new lines and blank spaces and thus are independent of the hardware, software, data structures or character coding schemes used in any particular project.

XDF is designed for use primarily with multivariate taxonomic data sets, including textual and numeric data, but also to be extensible to new classes of biological data, such as images. Flexibility and extensibility are necessary for long term stability but do not lead to standardisation among database projects. Two important mechanisms are available within XDF, however, to encourage the adoption of data standards.

(i) XDF can be used to define specialist transfer formats for particular application areas, thus providing strict control of the classes of data and data definitions used. One such transfer format is already in use by the International Legume Database and Information Service ('Ildis').

(ii) Provision is made within XDF for default definitions of the common core elements of taxonomic data sets such as the taxonomic hierarchy and biological nomenclature.

XDF is intended to be flexible in use, so that in one guise it may be suitable for a programmer transferring large amounts of data between sophisticated databases, while in a less terse form it might be used for data capture by biologists unfamiliar with internal database structures.

The initial impetus for the development of XDF arose from the authors' own needs to transfer data between a variety of application programs, and the need of the Ildis project to merge together data sets compiled using a wide range of different computers and database programs. XDF was used to define a specialist transfer format (Allkin and Winfield, 1989) for transferring species diversity data between ILDIS and collaborating database projects (*eg* 'Tropicos' at Missouri Botanical Garden and 'Esfeds', the European Flora Documentation System at Reading University).

1 Introduction

1.1 The need for an exchange medium

between database projects An increasing number of biologists and institutes are involved in database projects that, often for good practical and historical reasons, use different software, different data structures and different machines. Data exchange between databases is therefore rarely straightforward and may require sophisticated programs to be developed -- one for every different pair of databases. Given the resources available and the time scales involved, development of such translation software often proves impractical, or at least inconvenient, thereby preventing potential information exchange. When the resources and programming expertise do exist, data may be exchanged only after a considerable investment that the biological community can ill afford.

between programs Biologists employ a variety of programs for different purposes, such as cladistic or phenetic analyses, key generation and simple data manipulation. Inevitably, program packages with similar rôles differ in the detailed facilities that they offer and users often need to pass data from one to another, *eg* to take advantage of more flexible graphics capabilities in package 'A' while benefitting from analytical techniques only available in package 'B'. More frequently still, having invested much energy in constructing a data set for one task, biologists decide to use or analyse at least part of that data for another purpose. Unfortunately, with very few exceptions, every program requires data to be prepared in quite a different format. The translation from one format to another can be time-consuming and, unless automated, error-prone. Writing programs to automate transfer is itself costly, particularly from the short term view, and may be either unsatisfactory or not done at all.

to facilitate data capture Data are normally prepared for input to individual programs in an appropriate format using a simple editor program, since specific interactive data capture routines are rarely available. Data capture is therefore time-consuming and involves the user in learning the terminology, data rules and syntax of the particular data format used by *each* individual program. Facilitating data transfer between application programs would ease this burden on the user and encourage the development of programs for interactive data entry for a commonly used format.

1.2 Aim

Our aim in developing XDF is to provide a climate within which biological data will be more readily exchanged. We envisage that instead of 'one-off' programs being required to pass individual data sets directly from database 'A' to database 'B', programs will be written to convert data from *each* database format to an intermediate form, XDF, and back again. This has the advantages that

(i) programmers need only deal with the internal mechanics (*eg* data structures and data integrity rules) of the database with which they are already familiar, and

(ii) should the users of database 'A' or 'B' then wish to communicate with database 'C' they already have a mechanism to do so and need write no further programs, once a program has been developed to convert between 'C' and XDF. The benefit to the users of 'C' in developing such a program is that at one stroke they can communicate with both databases 'A' and 'B'. Clearly the greater the number of database projects with the capability to read and write XDF, the greater the benefits for all concerned.

Different data transfer formats may still be required for different applications but if these formats were themselves defined using a common framework, then the common ground between them could be exploited. We envisage a family of distinct, albeit related, data transfer formats each written in XDF. This would have the additional and important advantage of significantly reducing software development costs to the biological community -- which can ill afford to waste meagre computing resources (Allkin, 1988).

1.3 Data standardisation: levels of compatibility

For the exchange of data between projects to be meaningful and worthwhile requires that there be a degree of compatibility between the donor and recipient databases. Compatibility may occur at a number of levels:

(*a*) common software

(*b*) common data structures

(*c*) common logical rules and data relationships

(*d*) common use and interpretation of terminology, look-up tables or dictionaries

These are arranged in a descending logical sequence with common software at the highest level of compatibility descending to common terminology at the lowest level. Meaningful information exchange at a given level requires not only standardisation at that level but also compatibility *at all lower levels.* It is fruitless, for example, for two economic botany databases to standardise on their software and data structures (levels '*a*' and '*b*') while adopting different definitions for plant use categories (level '*d*').

To date, the IUBS Commission for Plant Taxonomic Databases (referred to throughout this document by its currently better known acronym 'TDWG') has rightly concentrated on standardisation at level '*d*'. More recent initiatives, however, have begun to establish standards at level '*c*' defining logical relationships for particular application areas (*eg* the Standard Name Module for Botanical Database Systems) or even data structures at level '*b*' (*eg* the Botanic Gardens' International Transfer Format).

Standardisation is clearly desirable and in some areas it will be possible for projects, particularly those starting out from scratch, to standardise on data structures or even software. In general, however, for higher levels of compatibility it becomes increasingly difficult to impose rigid standards, and possibly self-defeating. Projects able to adopt such standards would clearly gain high benefits but many others, whose needs genuinely vary or who began building their data sets before the appropriate standard came into being, may be prevented or deterred from using even those parts of the standard to which they might otherwise conform. The danger is that many projects would then choose to take an entirely independent approach, returning us to anarchy.

XDF is not itself proposed as a data standard but as a medium in which alternative data standards can be expressed. Thus, while adoption of XDF may help to avoid unnecessary incompatibility and duplication, it cannot prevent them. The flexibility and extensibility built into XDF are necessary for its long term stability, but will not lead to standardisation among database projects since that same flexibility enables individual projects to adapt existing standards to their special requirements. Two important mechanisms do exist, however, to encourage the adoption of data standards.

(i) XDF can be used to define special transfer formats which impose strict discipline on the classes of data and data definitions used for particular application areas.

(ii) Special provision is made within XDF for the common '*core*' elements of biological data sets such as the taxonomic hierarchy and biological nomenclature. '*Default data definitions*' are provided for these core elements which users may adopt or adapt as necessary. Thus, where there is no good reason for choosing otherwise, a default data class definition will be available 'off-the-shelf'. We envisage that these default data definitions may be included in a recognised data dictionary such as that being built by TDWG.

We do hope however that XDF will prove useful as a tool in overcoming *many* classes of difficulty in converting data sets between incompatible formats.

2 The design and development of XDF

2.1 Requirements of a data definition language

XDF was designed as a data definition language for data exchange with the following requirements in mind:

(i) that it be sufficiently flexible to describe all classes of data commonly employed by biologists.

(ii) that the language be sufficiently flexible to minimise the effort involved in translating data between XDF and other data formats or data structures required by specific applications software.

(iii) that it be independent of any particular software.

(iv) that it be independent of the medium used either for data storage or data transport.

(v) that it be independent of any machine character-coding scheme, such as ASCII, ISO or EBCDIC.

(vi) that both data class definitions and the punctuation character set used be redefinable within XDF itself.

(vii) that the language itself be extensible to allow novel data classes to be accommodated

(viii) that facilities be provided for user-defined macros to enable users to improve legibility or efficiency for a particular application.

(ix) that XDF data sets would ideally be

-- both precise and concise,

-- structured so as to maximise the data verification and validation possible by application software,

-- arranged as flexibly as possible, and

-- amenable to manipulation by simple text editors and, when necessary, easily intelligible to biologists.

(x) that meaningful data sets can be generated from databases that do not contain all the classes of data defined in the appropriate standard.

Some of these requirements conflict. Conciseness, for example, is achieved at the expense of both intelligibility and the potential for data validation. The key requirement is the flexibility to enable users to employ XDF as most befits the particular application. A biologist using XDF with an editor program as a manual data-capture mechanism would choose to maximise legibility, while a programmer involved in the transfer of large volumes of data between sophisticated databases might choose to maximise either conciseness of the data or simplicity of programming.

One obvious concomitant of the need for flexibility and extensibility is that no unnecessary limitations should be placed on the lengths of fields. There is nothing to prevent the use of XDF data transfer formats that incorporate data segments having fixed-length fields within the database from which they originate. Similarly, it is possible to use XDF to pass data using fields with effectively a fixed length, such as data values in a particular field always having a length of, say, 23 characters, by ensuring that each data value is followed by the appropriate number of blank spaces and a field separator character. At present there is *no* mechanism to define XDF data segments in which fields are delimited not by a separator character but solely by their position within a record consisting of fields of predefined length. Such a possibility exists in the future, though one should consider carefully whether it is the function of a data transfer standard to impose such arbitrary restrictions.

2.2 Predecessors of XDF

XDF was developed to fulfil a perceived need and owes much to our experience with existing biological software (*eg* Adey *et al.*, 1984) and our joint research into the design of taxonomic data management software (*eg* Allkin & White, 1988).

A notable exception to the general climate of incompatible data formats has been the 'Delta' data format. Delta was developed by Dallwitz (1974, 1986) for use with his taxonomic description writing and identification programs. It was subsequently modified and adopted by Pankhurst and Payne for use with their own identification programs (Pankhurst *et al.*, 1984). Delta deals admirably with taxonomic descriptions but does not, for example, handle the taxonomic hierarchy or biological nomenclature, nor does it provide mechanisms for indicating the source of data observations. It is unclear how the format could be extended to give all of the flexibility required.

The Botanic Gardens' International Transfer Format ('BG-ITF') has been adopted by TDWG as a standard for data transfer between botanic garden databases. BG-ITF is the first of a series of TDWG transfer formats for different application areas and does not purport to be of more general use and, since for example it was designed to handle garden records concerning individual plants, it sensibly avoids the complications associated with taxonomic nomenclature. BG-ITF is actually more than just a transfer format since it was also designed to provide guidance to smaller botanic gardens starting out to implement their own databases. In the present absence of suitable general-purpose taxonomic database software, and since such institutions can rarely count on sophisticated programming expertise, BG-ITF uses fixed-length data fields, thus allowing the data class definitions included to be adopted directly as data structures within simple implementations using programs such as 'dBase III'. This simplicity for database implementation is gained at the expense of potential information loss at data transfer. An implementation of BG-ITF in the database programming language 'Revelation' is already in use at various sites in the USA (K. Walter, pers. comm.) Since Revelation itself imposes no arbitrary restrictions on field length, data transfer between sites using Revelation database implementations would suffer an unnecessary restraint on data exchange when using BG-ITF as the transfer medium.

Standard data formats do exist for applications more general than biological data management. 'DXF', for example, is used to describe the simple elements of graphic images for design programs (such as 'Autocad'). Another format, 'DIF', is intended for data transfer between spreadsheet programs. The 'PostScript' programming language is widely used in the 'desk-top publishing' field to describe page layouts and images. It is highly extensible and many of its concepts have analogues in XDF.

We know of no biological data format that would be sufficiently flexible for our purpose, although we are aware of one or two similar effort in other scientific disciplines. We cannot rule out the possibility of adopting or adapting an existing format and welcome suggestions, although there is much to be said for a standard language incorporating specifically biological concepts under the control of the biological community.

2.3 The design of the XDF data definition language

Our approach in developing XDF was inspired by the relational database design philosophy (Codd, 1970) while the syntax is modelled on that of 'GPM', a macroprocessor language (Strachey, 1965). XDF is a language for describing taxonomic data sets and is designed to be independent of individual applications and to provide a medium for describing the types of data in common use within taxonomic data sets and other classes of data that have yet to be used.

Transferring information from one database to another is envisaged as a two-stage process. First, the donor database generates a device-independent ouput description of the desired data set in an appropriate data-description format. Second, a program controlling data input to a specific recipient database interprets that description. The two stages may be executed in different places and at different times. The data description format serves as an interchange standard for the transmission and storage of data.

Data formats may be considered partly on the basis of their *intrinsic* capabilities: built-in facilities such as the ability to deal with various types of data. Additionally, the degree to which the built-in facilities interact harmoniously is of considerable importance. For example, a data format that offers uniform treatment of morphological, chemical and geographical data greatly facilitates applications that must combine elements of all three in a single database.

Data formats may also be classified as *static* or *dynamic*.

(i) A *static* data format provides some fixed set of intrinsic facilities together with a syntax for their specification. Static formats have been in existence since computers were first used to process data sets. Classic examples are the rigid data formats expected by programs such as 'Taxir' and 'Clustan'. The 'Delta' format mentioned previously is another example. Historically, static formats have been designed to express the capabilities of a specific class of program, and have subsequently evolved as new features are added to the programs.

(ii) A *dynamic* data-description language allows considerably more flexibility than a static format. The set of functions may be extensible and the precise meaning of a function may not be known until it is actually encountered. A data set described in a dynamic format is more appropriately thought of as a *program* to be executed than as data to be consumed. A dynamic data-description language contains elements of programming languages such as procedures, variables, control constructs and so forth.

A data format that was primarily static but that purported to describe complex data sets containing disparate classes of data, including nomenclatural synonymy, dependent characters, source references and hierachies of place-names, would need a proliferation of special-purpose syntactic details. A dynamic language that allows primitive operations to be combined according to the needs of the database application will be superior to a static format that tries to anticipate all possible needs.

The XDF design takes the dynamic data-description language approach. The language includes a set of primitive operators which it allows to be combined in any possible manner. It not only has variables but also has programming language control structures to specify arbitrary computations to be performed during interpretation of the data.

There will be situations in which the way that a data description is processed must depend on information about the specific database to which it is to be imported, for example the order of fields in a record in the recipient database. This information is not necessarily known at the time the data description is composed, but only when it is imported to the recipient database. Thus it is convenient for a data description, when being processed for input, to be preceded by information about its destination database environment, to allow computations and transformations based on that information.

These considerations lead us to the XDF language, a dynamic data format in which data sets can be thought of as programs to be executed by an *interpreter*. XDF programs may resemble data sets in a static format, that is an uninterrupted sequence of basic data elements. XDF data sets will often have such a simple repetitive nature when generated from, or intended for, databases with a simple structure. When necessary, however, the power is available to be exploited by the knowledgeable user, data manager or programmer.

We describe XDF as a language advisedly -- it has a vocabulary, punctuation and syntax. It is a procedural language, like most conventional programming languages (Pascal, C, dBase *etc*), in which programs are read as a logical sequence of actions. It is processed by an interpreter which carries out the actions specified in the program. However, XDF is also a declarative language in that each data record in an XDF data set makes a statement of fact and it is the combined effect of these statements that conveys information. The order of data items and segments therefore has comparatively little importance within an XDF data set. The same data set could be defined in a variety of ways using XDF and, in this respect, XDF is similar to languages (*eg* Prolog) used for artificial intelligence applications. In contrast to such languages, however, XDF has been tailored for easy definition of data formats and hence programming of format conversions.

Although XDF is a general-purpose programming *language* it is not a completely self-contained programming *environment* because it lacks an editor and other tools required for program development. Interacting directly with the XDF interpreter (called 'XGPM', see later) is useful mainly for experimenting with its capabilities and for trying out XDF data sets under development.

An XDF data set may optionally be constructed in two parts.

(i) The *first part* contains the application-specific definitions used in the data. It is normally written manually by the designer of a specific data format and specified as a *prologue* to every data set generated for that application.

(ii) The *second part* is usually the data set itself, generated manually using a text editor or automatically by an application program such as an interactive data editor or a program generating output for data transfer from a donor database. It consists of references to XDF primitives and to definitions made in the first part, interspersed with operands and data required by those operations. This second part, unlike the first, is usually very stylised, repetitive and simple. The data set in the second part is processed according to the prologue.

XDF is analogous to a programming language such as Fortran, and an XDF prologue written to implement a particular data transfer format is analogous to a program written in Fortran. Different special transfer formats for particular purposes can be implemented by writing different XDF prologues.

In designing XDF we have made a conscious effort to avoid unnecessary restrictions and to provide mechanisms by which XDF may be extended by users. Such extensions may be to define and label new data classes (such as alternative NAMES data classes defined according to the needs of particular classes of user). Like any other language, we anticipate that XDF will evolve with use and we hope that this evolution will occur by the addition of extra prologue '*modules*' of specialist definitions rather than by requiring alterations to the core of XDF itself.

3 An example data set

Though the following data set consists of a single data block, it otherwise illustrates the overall structure of an XDF data set. Comments are included (between curly braces "{" and "}") to explain the use of the various directives *etc*. The data classes given (*eg* USES, GEOGRAPHY *etc*) are as defined for use within the Ildis project and are *not* integral to XDF itself, which is simply the vehicle for the data. Elsewhere, possibly in a header data block, the class definitions will also have been expressed in XDF.

```
(XDF)                         { Compulsory command at the start of a data set }

                              { Comments such as this may occur anywhere within
                                a data set delimited by a chosen character-pair;
                                the default pair used here is '{}' }

(HEADING)                     { The first data segment begins with this HEADING
                                declaration.  The segment contains records which
                                give the whole data set a TITLE, AUTHOR, etc. }
```

```
TITLE   : Example to show the overall structure of an XDF data set
AUTHOR  : Bob Allkin
VERSION: 2.2
DATE    : 23rd September 1989
(END : HEADING)            { A data segment may finish with an optional END
                             command; this one names the segment being ended }

(TAXA)                     { This TAXA data segment declares two taxa and
                             assigns names to them; note the use of ":" to
                             mark field boundaries in data records }
86  : Species : [Vicia : L. : graminea    : Smith]
102 : Species : [Vicia : L. : epetiolaris : Burkart]
(END)                      { An un-named END command ends the current segment }

(SYNONYMS)                 { Two synonyms are now attached to taxon 86
                             (V. graminea, already described).  In both cases
                             the information was obtained from reference 108
                             (described later).  There is some doubt about
                             the status of the second synonym }
#86 :Synonym:OK      :Species   : [Vicia : L. : selloi    : Vogel] : #108
#86 :Synonym:Doubtful:Subspecies: [Vicia : L. : pauciflora:/
                               Gillies ex Hook. & Arn. : pauciflora] : #108
                           { The continuation character "/" in the data record
                             above allows it to occupy more than one line }
(END : SYNONYMS)

(USES)      #86 : Medicine : #12
                           { Record a use for V. graminea from source 12 }

(GEOGRAPHY) #86 : Country : Native : Brazil, Argentina, Uruguay : #12
                           { Record V. graminea distribution from source 12 }

(BIBLIOGRAPHY)             { Two reference citations are given.  To avoid
                             repetition these reference numbers may be used
                             instead of full citations }

108 : Allkin, R., D.J. Goyder, F.A. Bisby and R.J. White : 1986 :/
      Names and synonyms of species and subspecies in the Vicieae /
      (issue 3) : Vicieae Database Project Publication No. 7.

12  : Burkart, A. : 1966 : Notas sobre las especies argentinas de /
      Vicia (Leguminosae) del area mesopota\'mico-pampeana :/
      Darwiniana 14 pp. 161 -193
                           { Note the use of the accent marker character "\"
                             in reference 12 to represent the character "á" }

(END : BIBLIOGRAPHY)

(END : XDF)                { Compulsory command to terminate an XDF data set }
```

4 How will XDF be used?

Using XDF, there are a number of different ways in which data sets might be manipulated.

(i) The traditional approach is that special-purpose, *stand-alone* application programs might be written to perform particular data analyses or data transformations, just as, for example, 'Pankey' generates dichotomous keys from 'Delta' data sets.

(ii) A software *toolkit* might be developed for use within the biological community to provide processing routines for reading, writing, interpreting and manipulating XDF data sets. This library of routines would then be available for incorporation within individual application programs or database systems as they are developed by different authors.

(iii) A *processor* program with similar functionality to the above toolkit could be used to read XDF files and generate the input data files necessary for existing application programs, in a similar way to that in which 'Confor' manipulates 'Delta' data sets.

While any or all of these implementation routes may occur, we favour the second and third and a prototype XDF processor program (called 'XGPM') and a toolkit of software routines, now being tested, will shortly be made available (see Appendix E). XGPM is already in use within the Ildis project (figure 1) and is used to validate a variety of data sets sent to Ildis from other databases (*eg* Tropicos and Esfeds) having diverse data structures, before they are passed to the 'Alice' database merging software, 'Sam' (see figure 1). A second more generally applicable version of Sam is now being developed, in association with Alice, which will use XGPM as a pre-processor.

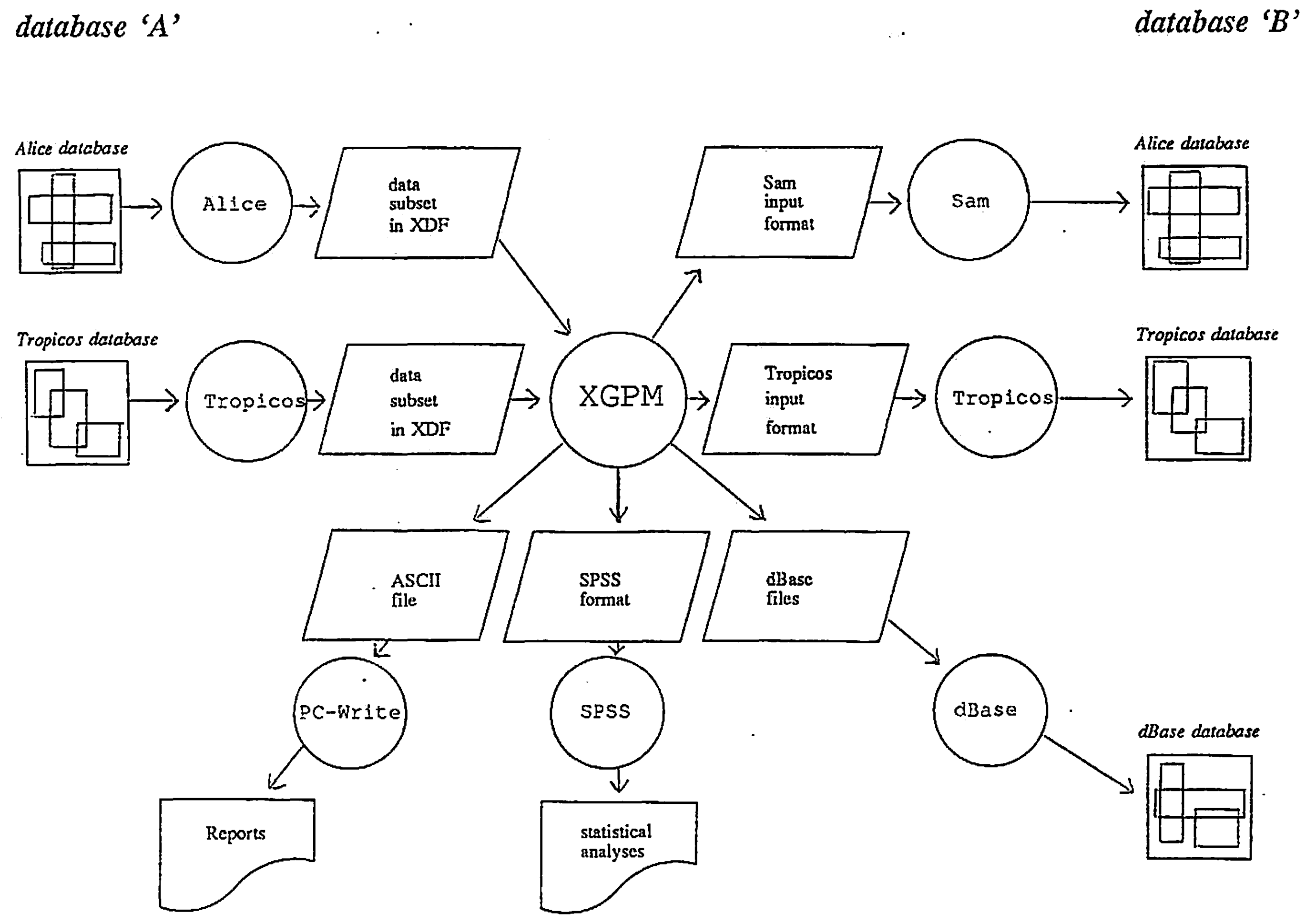
In transferring data from database 'A' to database 'B' we anticipate that those most familiar with the workings of database A will write the data subset to be transferred in an XDF format most convenient to them. XGPM will then be used either as a stand-alone program or as part of another program, to manipulate the data into a second format, also in XDF, more suitable for input to database B. Finally a transfer program will be needed to read the modified XDF data set into database B. Incorporation of the XGPM data manipulation functions within the import program will simplify things for users of the programs.

Given the above scenario in which XDF data sets are created or read in the form most convenient for the current stage of data transfer, the routines to generate XDF data sets will be comparatively small programs simple to write. These programs will be able to generate '*flat file*' output (one data segment for each donor database file) dressed up in an XDF syntax.

Programs for reading and merging data sets with recipient databases, however, may well need to be more sophisticated to undertake the necessary data validation (for example, "is this a legal entry?") and verification (logical comparisons with the existing data set). This complexity arises from the need to control data entry with as much care as during normal data entry and need only be as sophisticated as the recipient database requires. This complexity would occur whatever medium were used for data transfer.

Should fundamental changes in the definition of XDF be necessary, then a new release of a program such as XGPM will be required, *but* once such a processor program is available then an XDF data set, unlike a Delta data set for example, may be translated into quite novel data formats simply by appropriate use of the XDF language, since the way in which XDF data sets are interpreted, processed and reformatted can be altered by appropriate directives in the XDF data itself or in an associated XDF prologue. Similarly, new classes of data can be defined and different punctuation protocols initiated by the user without *any* further programming effort by the author of XGPM or the translation program. Thus the user community is not dependent on individual programmers or on limited programming resources.

Figure 1: Use of XGPM to translate XDF data files



5 Structure of XDF data sets

An XDF data set is constructed from building blocks that represent the hierarchical structure of the information contained in the data set. The following two sections describe the structure of XDF data sets and the associated XDF syntax rules, first from the perspective of the logical structure of the information itself and then by describing the physical structure of an XDF data set.

5.1 Logical structure

This section describes a hierarchy of logical building blocks that constitute an XDF data set, in descending order of size, from an entire XDF data set down to the elemental particles: the data values themselves.

XDF data sets An XDF data set is complete and self-contained. It comprises all or part of one, or more, operating system files (see section 5.2, *Physical structure*, below). A data set begins with an XDF command and finishes with an END command. A HEADING data segment (see later) may occur at the outset of any XDF data set to label it appropriately.

data blocks Each XDF data set consists of one or more discrete '*blocks*' of data, which normally represent a major logical subdivision of the data, such as phytochemical data, the character definitions or a particular recent update. Many smaller data sets will consist of only one data block. Each data block starts with a BEGIN command and ends with a matching END command. Any one data block must begin and end in the same physical data file although parts may reside in other files.

Different data blocks in the same XDF data set may occur in the same data file, in different data files on the same storage device or in files spread across different devices, limited only by the facilities provided by the user's operating system to manage those files and devices. Data blocks within the same data file are distinguishable since each must be named in its opening BEGIN command. The order in which data blocks are read may be specified by use of the GET command (see below).

data segments Each data block may contain any number of discrete data segments. Each segment contains records of *identical structure* and with data of a particular class. A data segment is delimited by the initial data segment declaration or definition directive and optionally by a matching END command. If no END command is included then the initial directive of the following data segment will automatically close the current segment.

records Each segment comprises from one to many identically structured data records, the format of which is implied by the initial segment directive, since this will refer to a previous implicit or explicit data CLASS definition. Each logical record is subdivided into a predefined number of fields separated by the '*field separator*' character. The types of fields and their sequence within the record are outlined in the definitions of the data classes (see later).

A logical record will ordinarily end with the '*new line*' character but may extend over any number of consecutive *physical* data records (or lines) by placing the continuation character at the end of each physical record, immediately preceding the new line character, for each record to be continued. Each record must start on a new line, assuming the default value of the new line character, although not necessarily in the first character position.

fields Within records, fields are separated by the field separator character, and contain data values. Since XDF directives may occur anywhere in a data set they may replace or occur within individual data values. Blank spaces immediately preceding or following field separators are ignored. The final field in any record does *not* require a following field separator character. Separator characters placed at the end of records will be ignored. Missing fields at the end of records will be assumed to be '*null*' entries, whether the field separation character is appended or not. A null value is one whose length is zero.

A data field that is compulsory and that has no data value entered will have a default data value assumed which will be defined in the class definition. Where data fields are not compulsory and have no default value assigned, blank or missing data entries will be assumed to take a null value.

data values Data values are the smallest units of information and are placed within data fields of the appropriate type. Individual data values may have blank spaces within them (*eg* "`very long`"). Normally, each field will contain a single data value although mechanisms exist within XDF to enable single fields to contain the following

code values stand instead of a data value or record of the same type defined elsewhere in the same data set. Declaration and definition of codes should occur before their use in other data segments. When code values occur in a given field they are distinguished from genuine data values by the '*code value*' character (whose default value is "#") in the first character position of the value. Blanks preceding the code value marker are ignored but blanks following the code value character but preceding the data value will *not* be ignored.

multiple data values of the same type can be entered into a single data field provided that they are separated by the '*list separator*' character. A list of data values may contain a mix of genuine data values and code values.

compound data values (or '*embedded records*') consist of more than one data value placed in a sequence of data sub-fields, typically of different types, separated by the field separator character. Compound data values are enclosed within a special pair of delimiter characters (see Appendix B). It is, therefore, as if a single data field from the current record had an entire record from a data segment of a different type embedded within it.

comments Superimposed on the strictly hierarchical logical structure inherent in XDF data sets, comments may be attached with a great deal of flexibility. Two classes of comment may occur within XDF data sets: '*notes*' and '*remarks*', each type of comment being delimited by the appropriate comment delimiter pair. Comments of either type may occur either within or between logical records.

notes form part of the data itself and convey permanent information not otherwise expressed within the data set.

remarks	convey additional transient material not of a biological nature but management information concerning either the data set itself or its transfer. Remarks, unlike notes, are ignored during data transfer, and do not pass with the data when read into an application program or converted to another format.

5.2 Physical structure

XDF data sets can be physically subdivided into the following units:

files Files (physical, external data storage areas provided by the computer operating system) may be used to divide a data set into convenient portions for use, transfer, *etc.* Normally the arrangement of the physical files would correspond to the logical structure of the data. `INCLUDE` commands (see later) can be used to read the contents of another file as if they were entered physically into the current file in place of the `INCLUDE` command.

Different operating systems have different file naming conventions. XDF uses whatever local conventions are applicable to file names or identifiers. Complex data sets containing explicit reference to file names may therefore cause difficulties if transferred to a different operating system. With care, however, these references can be isolated in one part of the data set for ease of modification.

lines Physical records or lines are best restricted to a maximum of 80 characters in length. Blank spaces preceding the first non-blank character in a line and blank spaces following the last non-blank character in each line are ignored, while blank spaces immediately preceding a continuation character will be included.

punctuation A set of special XDF '*punctuation*' characters is defined (see Appendix B). Each punctuation character serves a different syntactic purpose within XDF data sets. Although each punctuation character has a value (an ordinary printable character) assigned to it by default, the actual characters used can be changed within an XDF data set using the `SET` command.

Thus only three characters " (", ") " and " : " are needed, that is the default values for the characters used in the initial XDF command and the `SET` command. Should some of the default punctuation characters be unavailable or be used within the data itself, `SET` can be used to change them to printable characters that *are* available.

other characters XDF data sets consist only of ordinary printable characters. The precise character set used may vary with the application. An XDF command allows definition of the character set used in any particular data set (see later).

accents Accented characters can cause problems in exchanging data sets since they are handled differently by different machines and programs. Where the sender and recipient are able to adopt the same character set, such as an ISO national set or the IBM PC extended ASCII, these problems disappear and characters may be transferred directly within XDF. Where two projects use incompatible systems, however, or where a data set is to be sent to many recipients, we recommend the use of character substitution codes. Each accented character is replaced by a three-character sequence consisting of the base letter, the accent marker (whose default value is "\") and finally the appropriate accent or secondary character. Thus for example the characters "é", "ñ" and "Å" will for the purposes of data transfer become "e\'", "n\~" and "A\o" respectively. The following list is a recommended set of character substitution codes for use in transferring XDF data sets. The list may be extended to allow other similar characters to be built up in a similar fashion.

á	a\'	ë	e\"	ò	o\`	ä	a\"	¿	?\?	ε	e\/
â	a\^	í	i\'	º	o_	Æ	A\E	¡	!\!	Θ	h\/
à	a\`	î	i\^	ú	u\‾'	æ	a\e	½	1\2	μ	m\/
ª	a_	ì	i\`	û	u\^	Ö	O\"	¼	1\4	π	P\/
Ç	C\‾,	ï	i\"	ù	u\`	ö	o\"	∞	$\'	Σ	S\/
ç	c\,	ÿ	y\"	Ü	U\"					σ	s\/
É	E\'	Ñ	N\~	ü	u\"	£	L\-	α	a\/	τ	t\/
é	e\'	ñ	n\~	Å	A\o	¥	Y\=	β	b\/	Φ	F\/
ê	e\^	ó	ȯ/'	å	a\o	₧	P\t	ϒ	g\/	Ø	f\/
è	e\`	ô	o\^	Ä	A\"	¢	c\/	δ	d\/	Ω	O\/

6 XDF data set processing

The XDF interpreter is case-sensitive although if necessary data files can be produced and interpreted entirely in one case. General rules specify order of the components from which data sets are assembled.

(i) Directives and punctuation characters can be defined and redefined within data sets but redefinition must precede their use.

(ii) Data records can only be processed if the segment containing them is of a *class* that is intrinsic to XDF or has defined previously.

As XDF data sets are read they are processed as follows.

(i) When the initial XDF command is encountered the default header data block is read from the file XDF.DEF.

(ii) Any directives or other macro calls are processed and removed, possibly generating replacement text which is processed as if it were inserted into the input stream.

(iii) Remarks are simply removed.

(iv) Compound data values (embedded records) are replaced by a code value in the appropriate field of the current record and a new record in the appropriate data segment

(v) Multiple data values cause the entire record to be repeated, once for each value in the multiple data field, with each copy containing *one* of the data values only.

(vi) Sequences of characters containing only spaces, tab characters and those characters, such as carriage return and line feed, used by the operating system to mark the separation between lines with at least one record continuation character, will be replaced by the number of space characters immediately preceding the first record continuation character. Records may therefore be conveniently broken into separate lines between words.

(vii) Any sequence of space characters immediately preceding or following either a field separator character or a list separator character is ignored.

(viii) When a record separator character is read, or a directive implying the end of a data segment is processed, the preceding input is processed as a data record.

7 Directives: declarations, definitions and commands

An XDF data set contains a number of directives, with particular meaning within the XDF language, which affect the processing of the data and control the way the data are to be interpreted. They are implemented as macro calls processed by the XGPM program. They may be classified conveniently into three types, *declarations*, *definitions* and *commands*.

declarations	Ordinary descriptive data and quite a lot of other information, including 'metadata' which is data which itself describes or 'manages' the real biological data, exists as data records in an XDF data set. For these records to be interpreted correctly, they are packaged into *data segments*. Each segment contains a single *class* of data record. A segment is introduced by a data class *declaration* which says what class of data records it contains, in other words what sort of data it is and what its format is. Some of these data class declarations are implicitly predefined, *ie* built into the XDF language definition, such as `DATA` and `HEADING`.
definitions	Other data class declarations are not built in, but have to be explicitly *defined* using XDF, either by the user or packaged into a prologue module by someone else, before they can be used to declare or introduce data segments of the new class.
commands	Directives which neither declare data segments nor define new data classes, are called *commands* and perform a miscellany of other functions, including data and file management and the extension of the XDF language itself using fairly conventional programming language concepts.

Formal definitions are given for the most important of these later in this document. A brief summary of some of them is given here.

7.1 Declarations

Only a very small number of built-in classes of data segment is provided. Two of them are:

`DATA`	a directive which introduces a general-purpose data segment
`HEADING`	a directive for attaching a title, date, author *etc* to a particular data set

7.2 Definitions

A wide range of facilities is provided to enable users and programmers to define classes of data segment to suit their own specialised requirements. The following are only a selection.

`DESCRIPTORS`	allows the user to define new characters in the taxonomic sense, such as a descriptor to express the habit of a plant
`HIERARCHY`	permits the definition of hierarchical character-states, useful for taxonomic levels, geographical areas, *etc*
`CLASS`	is used to define a new kind or *class* of data segment, perhaps using descriptors created using the `HIERARCHY` and `DESCRIPTORS` definition facilities

7.3 Commands

Commands are directives, other than data segment declarations and directives used to define new data classes. They are used to control data flow and the way in which data sets are processed. Example commands are:

`XDF`	a command initiating an XDF data set
`INCLUDE`	a command to read the contents of another file as if they were placed in the current file in place of the `INCLUDE` command.

Some commands are suitable for more sophisticated users.

`SET`	provides the user with the power to change default values and mechanisms within an XDF data set, such as the punctuation character set
`DEFINE`	is a general mechanism by which users can define new macros (*ie* new directives)

8 Data classes

A data class declaration must precede the records in each data segment and establishes the structure that the following data records will have, by reference to a definition in a `CLASS` segment of that name elsewhere. Novel classes of data may themselves be defined or existing definitions amended within an XDF data set. It would be usual for all such data class definitions to occur together, probably at the outset of the XDF data set and possibly in a separate data block or prologue.

Standard data transfer formats could therefore be defined in a separate XDF data block that nevertheless formed part of each individual data set prepared according to that standard. This '*header data block*' would define all the data classes used and establish the default punctuation and other XDF rules to be followed.

A second mechanism for facilitating the adoption of standards within the biological community would be the adoption of the recommended list of default data class definitions from TDWG. Such a list would encompass those data classes commonly used in biological database projects, such as

`TAXA`	`NAMES`	`TAXONOMIC HIERARCHY`
`CHARACTERS`	`SYNONYMS`	`BIBLIOGRAPHY`
`SPECIMENS`	`COLLECTORS`	*etc*

Debate is required to establish which data classes should be included in such a recommended list and what should be the default data definition for each. In some cases the latter task is being undertaken by different TDWG subgroups for data standards covering particular application areas. A recommended list would help to avoid alternative definitions for the same data type occurring unnecessarily.

Nevertheless, even with the best intentions in the world there may be valid reasons for alternative definitions for individual data classes. An economic database will find the example `USES` data class used in the example data set (page 10) very inadequate. If two application areas require different class definitions for essentially the same kind of data, the TDWG Data Dictionary Subgroup may propose a list containing both, or a single master definition from which variants may be defined (Allkin & Bisby, 1988).

It is not proposed that XDF should attempt to enforce the use of such default definitions, since there will be many occasions when the needs of individuals differ. Nevertheless, the existence of 'off-the-shelf' data definitions should promote greater compatibility between database projects and, consequently, greater opportunities for information exchange.

9 Conclusions: the data transfer problem

XDF is a medium for the transfer of biological data, particularly of a systematic nature. It is a language providing a framework within which biologists may define their own transfer data formats or make use of existing standard formats to transfer data between different databases, between collaborators working on a joint project or between different application programs. Adoption of XDF will reduce unecessary program development, particularly in the longer term and will increase the opportunities for data exchange, but is far from being a solution to *all* obstacles to data transfer.

XDF does *not* dictate data standards, either for the terminology used or for the data structures employed within the data sets to be exchanged, so that meaningful exchange of information between projects clearly further requires that common standards be established and jointly adhered to.

XDF offers users the flexibility of modifying existing data standards as they see fit, thereby potentially creating new limitations on information exchange. This ability of users to define their own data classes or adapt existing definitions *where necessary* will however make adoption of existing data standards possible to many who would otherwise be excluded.

Transfer of data between two sophisticated databases, inevitably with different structures, is necessarily a complex and demanding process requiring great attention to detail and sophisticated transfer software. Data transfer between simpler databases will not require such sophisticated software but nevertheless may face numerous hurdles. Successful data transfer between two database projects requires solutions to the incompatibilities potentially caused by differences in any of the following:

physical medium	A data storage or transfer medium prepared and written according to a particular data format, *eg* 3.5" IBM-compatible floppy discs formatted to the Dos 1.44 Mb standard *or* via RS-232 serial communications at 9600 baud.

character coding scheme	A recognised scheme for digital representation of alphanumeric characters, *eg* ISO, ASCII or EBCDIC.
data terminology	Common vocabularies and use and definition of terms, *eg* "`Spain`", "`Timber`" or "`Suspected synonym`".
data classes	Inconsistent definition of data classes. 'Plant use records' in the two databases, for example, may not have identical information. Some information may still be passed, however, if the two definitions share some common elements such as plant name, type of use, part of plant used, *etc*.
data rules	Logical constraints upon the data, such as "A given scientific name (Latin binomial) may occur only once in a given data set -- either as the accepted valid name for a particular taxon *or* as a synonym -- unless it is also recorded as having been previously misapplied to another taxon."
internal data structures	Two databases may store the same information in quite different internal data structures. Transfer software is required to convert between one set of structures and another.
external data structures	Since transfer software will rarely be able to operate simultaneously on both databases, data must be represented in an intermediate external form. A common external data structure intermediate between the two internal structures may be sought, perhaps establishing that in plant-use records 'plant name' should precede 'type of use', 'part of plant' and 'source citation' in that sequence. Programming limitations or other constraints may prevent agreement on common transfer structures.
syntax	The syntax used to mark the end of one record and the start of the next, to distinguish one data field from another or to establish the presence of embedded records of different classes. A 'new line' character followed by a line feed character might indicate a new record, and adjacent fields might be separated by colons. Alternatively fields within records might *not* be separated by marker characters but instead be distinguished from one another by their *position* within records of a given type.

XDF is designed to surmount the difficulties associated with syntax and provides mechanisms to allow users to establish external data structures. It will further allow the translation of one external data structure into another containing the identical information. This enables users to define external data structures which mirror their own database's internal data structures, thereby saving a great deal of programming effort.

XDF can do nothing however, to facilitate data transfer between databases unable to find common solutions to the other hurdles mentioned above. If no physical medium or format is available to both then data transfer cannot take place. If the two databases follow different conventions regarding terminology, data class definitions or data constraint rules then, although data may be physicaly transferred, there will inevitably be a loss of information or the transfer of erroneous information. Only where there is an inconsistency of data class definition may XDF assist since, if divergent data classes have elements in common, it may be possible for users to define new data classes containing only those common elements to enable transfer of as much information as possible.

It is worth underlining that any difficulties caused by physical media or format are, in today's computing environment, trivial in comparison with the headaches and barriers to information transfer caused by differences in data definition.

10 What do you think?

We are very keen to receive prompt feedback of any kind about this document. In many of the directive definitions given later, there are comments about unresolved design matters. We also particularly draw your attention to the following unresolved or debatable issues.

fixed-length fields	How valuable would it be to allow in the future for data record classes to be defined to have genuinely fixed-length fields, without field separator characters, rather than always relying on the separation of fields by field separator characters?
example data classes	The classes TAXA, NAMES and SYNONYMS are included in this document for illustrative purposes. The definition of such data classes is not the business of this document and we have used for illustration definitions used by the Ildis project. Debate as to the detail of these definitions will occur elsewhere (*eg* within TDWG) but is it helpful that such illustrative basic data classes appear in this document, perhaps to help to understand the examples provided?
	If it is helpful then what other basic taxonomic data classes should be included?
DATA *class*	There is a need for a general DATA class. This class is central to the aims of XDF. Is the given definition adequate? Is there a need for other definitions? Should these be restricted to specific areas of application?
DESCRIPTIONS	More specifically, is there a need for the suggested DESCRIPTIONS data class that allows for the entire descriptive data for one taxon to occur within a single record?
alternative field labels	The present system for defining a data class does not allow for alternative labels for fields to allow different user views. Where might such a facility be useful, and would you use it?
CHARACTER-SET	The CHARACTER-SET directive was originally considered as a means to enable application software to test the readability of a particular data set at the outset and as an additional mechanism to allow redundant information within a data set being used for data validation. The need for such a directive is debatable. Do you have any views?
low-level macros	At a few places within this manual we refer to low-level macros not actually defined within this manual. These tend to be more primitive but more efficient versions of directives that are defined here. These low-level macros are generally of little interest to most XDF users but since they may be combined together in different ways to build other directives, they could be useful to more knowledgeable individuals interested in extending the scope of XDF. Since these lower-level macros are implemented within the XGPM processor program, should they be described in the current document, at the risk of confusing the normal user, *or* should their documentation be restricted either to an Advanced Manual for XDF or to the XGPM User's Guide?
	Currently we follow the convention that lower-level macro names are all in lower case, while XDF directives have names in upper case. In instances where an XDF directive and a low-level macro share similar functionality then the two have been given the same name (for example close and CLOSE). Is this acceptable?

data blocks

Do you think the *data block* concept and the corresponding BEGIN command are useful? If so, what use would you make of data blocks? If you don't use data blocks, would you mind if they were deleted from the design?

terminology

Many of the words we use in this manual to describe XDF concepts have existing meanings in everyday speech or in computing or both. Suggestions for easily-understood alternatives will be welcomed. In particular, *block, segment, class* and "data type" have various computing or biological overtones. Should we use words like "part", "chapter", "volume", "section", "division", "structure" or "component" to describe some of the present *block, segment* or *class* concepts?

The two related concepts of the group of similar adjacent data records, now referred to as a *data segment*, and the description of the contents and format of a certain logical *data class*, of which one or more data segments may exist as physical instances, were originally both referred to indiscriminately as a "data type". Is this distinction helpful for clarity? Would other terms be preferable for these concepts?

data class definitions

The CLASS mechanism for defining a new data type, as described in this manual and especially if extended to incorporate alternative field labels and fixed-width fields, specifies two sorts of information. There is both a indication of the kinds of data allowed and a specification of the format of the data records which will occur in data segments of this class. Perhaps the complexity of CLASS definitions could be reduced if the record format information could be separated out into another class of segment, perhaps called STRUCTURE or FORMAT.

Acknowledgements

Peter Winfield was involved in early discussion of the structure and use of XDF and Mark Coode made useful criticisms of an early draft of the introduction. XDF development has been encouraged by the IUBS Commission for Plant Taxonomic Databases (referred to throughout this document by its earlier acronym TDWG). In particular, several members of a special TDWG subgroup have commented on an earlier version of this document.

References

XDF bibliography

Allkin, R. & R.J. White (1988) *XDF: a language for the definition & exchange of biological data sets.* An introductory document submitted to the IUBS Commission for Plant Taxonomic Databases at St. Louis, Missouri, USA.

Allkin, R. & P.J. Winfield (1989) *Manual for ILDIS Data Transfer Format. Experimental use of XDF and the ILDIS datatype definitions.*

Further documents will need to be produced to list the TDWG recommended default data class definitions and give more details of the use of the XGPM program.

Other literature cited

Adey, M.E., R. Allkin, F.A. Bisby, R.J. White & T.D. Macfarlane (1984) The Vicieae Database: an Experimental Taxonomic Monograph. *In* R. Allkin & F.A. Bisby (eds.) *Databases in Systematics*. Systematics Association Special Volume 26, Academic Press, London.

Allkin, R. (1988) Taxonomically intelligent database programs. *In* D.L. Hawksworth (ed.) *Prospects in Systematics*. Systematics Association Special Volume 36, Oxford University Press.

Allkin, R. & F.A. Bisby (1988) The structure of monographic databases. *Taxon* 37: 756-763.

Allkin, R. & R.J. White (1988) Data management models for biological classification. *In* H.H. Bock (ed.) *Classification and Related Methods of Data Analysis*. North Holland, Amsterdam.

Codd, F.E. (1970) A Relational Model for Large Shared Data Banks. *CACM* 13 (6).

Dallwitz, M.J. (1974) A flexible computer program for generating identification keys. *Syst. Zool.* 23: 50-57.

Dallwitz, M.J. (1986) *User's Guide to the DELTA system. A general system for processing taxonomic descriptions*, 3rd ed. Division of Entomology Report No. 13, CSIRO, Canberra.

Pankhurst, R.J., M.J. Dallwitz & R.W. Payne (1984) Notice to users of computing programs for identification and description. *Taxon* 33: 556.

Strachey, C. (1965) A general-purpose macrogenerator. *Computer Journal* 8: 225-241.

Appendix A: Glossary

There follows a list of terms used in this manual along with their definitions.

block	see data block
character	An individual letter, digit, sign etc. A member of the allowed character set for a given XDF data set, eg "3", "x", "Å", or "!".
class	see data class
command	An XDF directive, other than those used as data class declarations or definitions, used to control the flow of data and the way in which data are processed. Most commands are verbs (GET, INCLUDE, SET, etc).
comment	see remark, note
data block	An XDF data set consists of one or more discrete 'blocks' of data which normally represent a major logical subdivision of the data, eg phytochemical data, the character definitions or a particular recent update. Many smaller data sets will consist of only one data block. Each data block starts with a BEGIN command and ends with a matching END command. Each data block must begin and end in the same physical data file.
data segment	A subdivision of a data block. Each segment contains data records of identical structure and with data of a particular class. A data segment is delimited by the initial data class declaration and optionally by a matching END command.
data set	An entire, self-contained set of information gathered together for some particular purpose.
data class	The formal description of the content and arrangement of the information associated with a particular kind of data. The definition of a data class stipulates how data records are to be constructed for use in data segments of that class.
data record	One of a series of logical records in a data segment, all of which have essentially the same format, appropriate to the class of the segment.
declaration	A directive used to introduce a data segment. May be built-in to XDF or defined in the data set or in a prologue module.

definition	The process of defining new data classes using the appropriate definition directives. The definition of a data class is described by the use of data records.
descriptor	The formal definition of a variable or property for use in describing objects, eg taxa. Descriptors may be bistate, multistate (ordered or unordered), quantitiative (continuous or discrete), dictionaries, hierarchies or simple text descriptors.
descriptor set	A named collection of descriptors that may be referred to or manipulated as a whole.
dictionary	A list of allowed terms for a particular use. Each term may be assigned a code value for use elsewhere. Definitions may be attached as notes.
directive	A word of particular significance expressed as a macro call within the XDF language that controls the flow of data and the way in which data are processed. Directives are either commands with simple parameter lists (GET, INCLUDE, SET, etc) or are data class declarations or definitions which introduce data segments.
field	A sub-component of a record. A field will contain one or more data values of the same kind.
file	A discrete, named, portion of long-term data storage maintained by the computer's operating system.
header data block	A data block commonly stored in a separate file that contains a suite of XDF data class definitions, macro definitions and default settings for punctuation characters for use in a particular class of application. Many different data sets prepared for such applications may then INCLUDE the same header data block.
line	A subdivision (or 'physical record') of a file. Each line is separated from the next by a specific character or characters used by the operating system for this purpose and known as the 'new line' character.
macro	see directive
new line	see line
note	A comment that forms part of the data itself and conveys permanent information not otherwise expressed within the data set.
operating system	The software environment of the particular computer in use. The operating system has a user command language of its own and controls file management, output devices, etc.
physical record	see line
prologue	A file or 'module' of XDF definitions, especially new class definitions for use in specialised data sets.
punctuation	The use of special characters to impose a logical structure on a data set by separating records, fields and data values, etc.
record	see data record, line
remark	A comment that conveys additional transient information not of a biological nature but management information concerning either the data set itself or its transfer. Remarks, unlike notes, are ignored during data transfer, and do not pass with the data when read into an application program or converted to another format.
segment	see data segment
source	The origin of information. This may be a publication, unpublished manuscript, specimen, letter, photo, etc.

Appendix B: Punctuation characters

The following is the list of punctuation characters recognised in XDF. For each character or character pair the default value is given along with the XDF control name used to refer to each. The latter enables the SET command to be used to change the default value of a punctuation character.

Punctuation character	XDF macro name	default value
New line character(s)	NEW-LINE[2]	new line[1]
Record continuation character	CONTINUE	/
Record separator	RECORD-SEPARATOR	(NEW-LINE)
Field separator	FIELD-SEPARATOR	:
List separator	LIST-SEPARATOR	,
Directive (macro) delimiters	MACRO	( )
Directive (macro) parameter separator	MACRO-SEPARATOR	:
Macro definition string delimiters	QUOTE	` '
Macro parameter substitution character	PARAMETER	~
Compound data field (embedded record)	COMPOUND	[ ][3]
Code value character	CODE	#
Note delimiters	NOTE	< >
Remark delimiters	REMARK	{ }
Escape character	ESCAPE	!
Accent substitution character	ACCENT	\

Notes

1 A 'new line' is whatever character or characters are used by the operating system to separate records in a text file.

2 This directive is provided so that punctuation characters may be redefined using the SET command. eg the 'record continuation' character may be defined as the NEW-LINE character and the 'record separator' as some other printable character such as a semi-colon (" ; ").

3 Square brackets are used in the manual section of the present document simply as a device within this manuscript to indicate optional entries. Wherever they are used properly to delimit compound records this is indicated in the text.

4 Two punctuation characters that may occur in the same context should not simultaneously take identical values.

5 Any character currently used to represent one of the punctuation characters may still be used as an ordinary ASCII character with no special effect, by preceding it by the escape character.

An example using the default values for punctuation characters:

```
(BIBLIOGRAPHY)  213 : Ali, S.I.: 1967 : Bot. J. Linn. Soc. 64!: 236 - 238
```

The escape character "!" precedes a colon which forms part of the actual data itself and is not to be interpreted as a field separator character. Thus the data record in this example contains 4, not 5, separate data values:

213	Ali, S.I.	1967	Bot. J. Linn. Soc. 64: 236 - 238

Appendix C: Declarations, definitions and commands

This section contains the formal specifications of the most commonly used XDF directives and more important predefined XDF data class definitions. The specifications appear in alphabetic sequence by name, not always starting on a new page.

Punctuation characters are represented throughout this manual using the default values given in Appendix B.

Data values or punctuation characters that are optional (<u>not</u> compulsory) are surrounded by a pair of square brackets ('[' and ']'). Where the characters "[" and "]" appear in their own right within the examples this will be indicated.

While specifications are given in alphabetic sequence for ease of reference, they may be logically grouped together as follows:

Declarations

XDF data set labelling	`HEADING, CHARACTER-SET` (as defined at present)
Data classes (segments)	`DATA, DESCRIPTIONS, NAMES, SYNONYMS, TAXA, SOURCES` (not defined in this document)

Definitions

Data definition	`CLASS, DESCRIPTORS, DESCRIPTOR-SET, DICTIONARY, HIERARCHY`

Commands

Entire XDF data set	`END, XDF`
File handling	`APPEND, CLOSE, ERASE, INCLUDE`
Data blocks	`BEGIN, END`
XDF configuration	`DEFINE, SET`

Name:	APPEND
Version:	1.3, 15th May 1989
Use:	to close the current output file and prepare a new output file for data to be appended to the end
Position:	anywhere within a data set
Format:	`(APPEND : file-name)`
	where file-name is the operating system identifier for the file to which data is to be written
Rules:	-
Example 1:	`(APPEND : myfile)`
Example 2:	`(APPEND : c!:\data\test.out)`
	Note the use of the escape character before the colon that forms part of the file name.
Related directives:	INCLUDE
Comments:	If the file named does not already exist then it will be created.

APPEND is defined by means of lower-level macros. APPEND closes the current output file automatically before opening the specified one, and thus avoids any potential problem with exceeding any limit set by the operating system on the number of simultaneously open files, and reduces the potential data loss if a system crash occurs. However, unnecessary file closing (and later re-opening) may be inefficient with some operating systems. If necessary, a more efficient version of APPEND could be defined that does not close files.

Name:	BEGIN
Version:	1.4, 14th May 1987
Use:	to name and mark the start of a new XDF data block
Position:	only at the start of a data block
Format:	`(BEGIN : name)`
	where name is the name of the data block
Rules:	Each BEGIN command must be matched by an END command at the end of the data block.

No BEGIN command is necessary if the XDF data set consists of only one data block. BEGIN commands are compulsory if a data set has more than one data block.

The name string used should not normally include white space characters (blank, line feed, carriage return or tab characters).

Example:

```
(BEGIN : FRED)
. . .
(END : FRED)
```

Related directives: END, INCLUDE, XDF

Comments: The block name may also be specified in a HEADING segment.

Name:	**CHARACTER-SET**
Version:	1.3, 14th May 1989
Use:	to define sets of characters for use within XDF data sets
Position:	normally within the default header data block, but also elsewhere within a data set

Format:

```
(CHARACTER-SET [: name] )
[n] : 'c1c2c3c4c5......cn'
[(END [: CHARACTER-SET] )]
```

where	name	is a name given to a particular set of characters
	n	is the number of characters in the set
	c1, c2, ..., cn	are the characters themselves

Rules: Only one CHARACTER-SET data segment is allowed in each data set.

Example:

```
(CHARACTER-SET : Reduced Set)
73 : 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/
     'abcdefghijklmnopqrstuvwxyz'/
     '0123456789'/
     ' ,:!!/()<>.-'
(END)
```

Notice that in the character set described there is specified, on the fourth line of characters, a space character before the comma and a <u>single</u> " ! " character: the first " ! " is an escape character which causes the second " ! " to be taken literally and not as an escape character.

Related directives: XDF, SET

Comments: The need for and function of this directive is debatable.

Name:	**CLOSE**
Version:	1.3, 15th May 1989
Use:	to close a operating system file
Position:	anywhere within a data set

Format:

```
(CLOSE : file-name)
```

where	file-name	is the operating system name for the file to be closed

Rules: -

Example 1:

```
(CLOSE : myfile)
```

Example 2:

```
(CLOSE : c!:\data\test.out)
```

Note the use of the escape character before the colon that forms part of the file name.

Related directives: APPEND, ERASE

Comments: Given that the APPEND command automatically closes the previous output file before opening the new one and that any files still open will be closed when the XGPM program terminates, this command may not be necessary.

Name:	CLASS
Version:	2.2, 19th October 1989
Use:	to declare the existence of a new data class (a new kind of data segment) and to define it in terms of its component data fields and permissible data values
Position:	anywhere in a data set but before any attempted use of the new data class

Format:

```
(CLASS : class-name [: no-of-fields] )
. . .
[id] : field-type : field-name [: option-list]
. . .
[(END [: CLASS [: class-name]] )]
```

where		
	class-name	is the name used for the class being defined, which will also become the label for the corresponding field if this data class is used in the definition of another class
	no-of-fields	is the number of fields in the new class, and the number of records in the definition segment being introduced
	id	is a unique identifier for the field
	field-type	is one of the following: DESCRIPTOR, DICTIONARY, GLOSSARY, CLASS or UNDEFINED
	field-name	is the name to be used for the field, normally the name of a descriptor, a DICTIONARY or HIERARCHY segment or another data class
	option-list	is a list containing one or more of the following: a data value to be used as the default value, UNIQUE, OPTIONAL, REMEMBER, MULTIPLE, SINGLE or COMPULSORY (default)

Rules:

The definition of a data class must precede any attempted use of it.

The class name must not be the same as that of any other class or directive.

Use field-type DICTIONARY or HIERARCHY if the named dictionary or hierarchy is a complete list of the permitted data values for use in this field of data records belonging to this class.

Use field-type GLOSSARY if the named dictionary is an incomplete list and data records may have data values legally entered which do <u>not</u> occur in the named dictionary.

Use field-type UNDEFINED for fields which will be permitted to take <u>any</u> single data value without restriction. Such fields are not further defined elsewhere.

The field-name must be the name of a descriptor or data segment of the appropriate class elsewhere in the data set.

If UNIQUE is specified it will be necessary for each and every data record in the data set to have a value entered into the data field which is unique for <u>all</u> records, otherwise it will be permitted for a data value entered into or generated for a particular field to occur in any number of different records.

If OPTIONAL is specified it will be possible to omit the data value from the corresponding field of a data record, in which case the empty data field will generate a null data value.

If REMEMBER is specified the corresponding field within a data record will, if empty, assume a default data value from the equivalent field in the previous record of the same data class, possibly in a previous segment of this class.

If MULTIPLE is specified (or implied by omission) it will be possible to enter multiple data values in the data field, otherwise if SINGLE is specified this will not be allowed.

If COMPULSORY is specified (or implied by a missing option-list) an empty value in the corresponding field in a data record of this class will cause an error, otherwise if the option-list field contains any other value or values these values will be used as the default data values whenever no value is given for this field in records of this class.

Example 1: A new data class is defined, called USES and consisting of four fields. The taxon identification will be remembered from one record to the next, and the default source will be #999, if no other source is given.

```
(CLASS : USES : 4)
1 : CLASS       : taxon : REMEMBER {taxon identication}
2 : DICTIONARY  : use              {which use?}
3 : HIERARCHY   : place            {where used?}
4 : CLASS       : source : #999    {who said so}
(END : CLASS : USES)
```

Example 2: A definition of the TAXA data class included in this manual is shown. The taxon status is a two-state descriptor, assumed to be defined elsewhere, which may take data values ACCEPTED or PROVISIONAL, and the name is specified to be UNIQUE since an accepted name can refer to only one taxon and SINGLE since only one accepted name is allowed for each taxon.

```
(CLASS : TAXA : 5)
1 : UNDEFINED   : Taxon-identifier : SINGLE, UNIQUE
2 : DESCRIPTOR  : Taxon-status     : SINGLE
3 : DICTIONARY  : Taxon-rank
4 : CLASS       : Name             : SINGLE, UNIQUE
5 : CLASS       : Source           : OPTIONAL
(END)
```

Related Directives: DICTIONARY, HIERARCHY, DESCRIPTOR

Comments: Should we allow data types with fixed-width fields and no field separators by implementing an optional field-width specification within the CLASS record for the fixed-width field?

It is possible that the combination of MULTIPLE and UNIQUE for a field might not be allowed. Should UNIQUE therefore imply SINGLE? If so, the only reason to permit a list of values for the option-list field would be to allow for multiple default values to be substituted in case of an empty data field.

If a DICTIONARY name is specified as a GLOSSARY, should the dictionary be updated as new values are encountered in data segments of the new class? Or should a new option-list key-word called UPDATE be defined to specify this?

Name:	**DATA**
Version:	1.4, 18th October 1989
Use:	a general mechanism to attach observations to taxa
Position:	anywhere within a data block

Format:

```
(DATA [: descriptor-set-name] )
. . .
taxon-list : descriptor : data-value-list [: source-list]
. . .
[(END [: DATA] )]
```

where

descriptor-set-name	is the name of the descriptor set which defines the descriptors allowed in the second field in the data records of this DATA segment
taxon-list	is a list of one or more taxon codes for which the data following in a given data record applies
descriptor	is the name or code of a descriptor in the specified descriptor set
data-value-list	is a list of values (character states) observed for this descriptor in the given taxon-list
source-list	is an optional list of sources which support these observations

Rules: -

Example:

```
(DATA : ILDIS)
#23             : #1       : Herb       : #34
                : Height   : 1.5
#24, #45,       : Habit    : Tree
#12, #13, #14 :            : Shrub      : #67
#12, #13        : #4       : rare       : #67
[Eminia : Taubert : holubii: (Hemsley) Taubert] : /
                  Lifespan : perennial : #67, #78
(END)
```

Note the use of " [" and "] " to enclose a compound data value (an embedded record of class NAMES), and the use of blank entries in the taxon-list and descriptor fields to imply the corresponding entry in the previous record.

Related Directives: DESCRIPTIONS, DESCRIPTORS

Comments: Previously called DESCRIPTIONS.

If more than one data value occurs in a data-value-list, should this be taken to mean that these values occur as alternative states in separate specimens?

Other definitions of DATA could be used. The alternative DESCRIPTIONS data class may be used to allow an entire taxon description in a single record by permitting the states of several descriptors to be specified within one record. Which is preferable?

Name:	DEFINE
Version:	2.3, 30th July 1989
Use:	to define new macros (directives) for local use
Position:	anywhere in a data set, preceding the use of the defined macro
Format:	(DEFINE : macro-name : comment : 'macro definition')

where

macro-name	is the name of the new macro
comment	is any comment text, which will be ignored
macro-definition	is the defining string of the new macro, the text which will be processed when the macro is later called

Rules: Definition of a macro means that the new definition becomes effective immediately even if a macro of the same name already exists, thereby normally making the old definition inaccessible.

Blank spaces are significant within a macro definition and for the most part therefore should be avoided.

The macro definition is normally enclosed in quote characters to prevent its expansion at the time of macro definition.

Example 1: To rename the END command FINISH. Note that any parameter associated with a macro must also be explicitly passed to the new macro using the ~1 notation. Following this definition either END or FINISH may be used identically.

```
(DEFINE : FINISH : Make FINISH mean END :'(END:~1)')
```

Example 2: Similarly this definition will enable users to use CHANGE instead of SET. Note the syntax for passing multiple parameters.

```
(DEFINE : CHANGE :
   Make CHANGE a synonym for SET
:'(SET:~1:~2)')
```

Example 3: This example is the definition of the APPEND command described elsewhere in this manual. Definition is based on low-level macros available within XDF (append, close, divert and output). Note the distinction between low-level macros of relatively little interest to users and their more easily used, if sometimes less efficient, high-level counterparts described in this manual and that most users will deal with. The low-level macros have been consistently given names consisting entirely of lower-case letters. High-level macros have been given names consisting of entirely upper-case letters.

```
(DEFINE : APPEND :
   A High level XDF directive
:'(divert:(append:~1))(close:(output))')
```

Related directives: -

Comments: Most users of XDF will never need concern themselves with the DEFINE macro but the flexibility it provides allows XDF to be developed further for use in ways not currently envisaged. Facilities exist, such as for arithmetic, which are not described in this manual.

New macros will commonly be defined together in a separate header data block.

Name:	DESCRIPTIONS
Version:	1.1, 18th October 1989
Use:	another general mechanism to attach observations to taxa
Position:	anywhere within a data block

Format:

```
(DESCRIPTIONS [: descriptor-set-name] )
. . .
taxon-list : data-list [: source-list]
. . .
[(END [: DESCRIPTIONS] )]
```

where

descriptor-set-name	is the name of the descriptor set which defines the descriptors allowed in the second field in the data records of this DATA segment
taxon-list	is a list of one or more taxon codes for which the data following in a given data record applies
data-list	is a list of compound data values, each containing the name or code of a descriptor in the specified descriptor set, a list of values (character states) observed for this descriptor in the given taxon-list, and an optional list of sources which support the observations in that compound data value
source-list	is an optional list of sources which support all these observations

Rules: Descriptors must have been defined in a previous DESCRIPTORS segment.

Example: This is a data segment exactly equivalent in its effect to that given as an example for the DATA class.

```
(DESCRIPTIONS : ILDIS)
#23       : [#1      : Herb : #34], [Height : 1.5]
#24, #45  : [Habit   : Tree       ]
#12, #13  : [Habit   : Shrub      ], [#4 : rare] : #67
#14       : [Habit   : Shrub      ]              : #67
[Eminia : Taubert : holubii: (Hemsley) Taubert] /
          : [Lifespan : perennial ]              : #67, #78
(END)
```

Related directives: DATA, DESCRIPTORS

Comments: Other definitions of DESCRIPTIONS could be used. If it is too cumbersome to present all of a taxon's descriptor values in a single record, the alternative DATA class might be preferred.

Name:	DESCRIPTORS
Version:	1.5, 12th June 1989
Use:	to define descriptors of any sort, most commonly to be attached to taxa
Position:	anywhere within a data set prior to the use of the defined descriptors

Format:

```
(DESCRIPTORS [: n] )
. . .
[id] : name : type : range [: label-list [: variable]]
. . .
[(END [: DESCRIPTORS] )]
```

where	n	is the number of descriptors to be defined
	id	is a unique identifier for each descriptor defined
	name	is a unique name for the descriptor
	type	is the kind of descriptor being defined, namely one of the following classes: TWO-STATE, ORDERED, UNORDERED, DICTIONARY, HIERARCHY, DISCRETE, CONTINUOUS or TEXT
	range	is a detailed definition of the data values or states allowed for the descriptor. The format for this definition varies according to the type of descriptor:

TWO-STATE qualitative:
first-state, second-state

ORDERED or UNORDERED qualitative:
first-state, second-state, third-state, ...

DICTIONARY or HIERARCHY:
segment-name

DISCRETE or CONTINUOUS quantitative:
minimum-value, maximum-value

TEXT: [max-length]

	label-list	is a list of labels to be suffixed to data values (states), such as the unit of measurement for a quantitative descriptor, eg mm. A list of alternatives may be given.
	variable	is either VARIABLE (default) or NOT VARIABLE

Rules: Descriptor names or descriptor state names may contain blank spaces.

If VARIABLE is specified (or implied by omission) multiple data values are meaningful for this descriptor. For example, a plant species may occur both as a `tree` and as a `shrub`, but it may not simultaneously be both a monocotyledon and a dicotyledon.

Example 1:

```
(DESCRIPTORS : 5)
1 : Habit        : UNORDERED : herb, shrub, tree :: VARIABLE
2 : <Climbing>   : TWO-STATE : climbing, not climbing
4 : Conservation status : UNORDERED : /
         endangered, indeterminate, insufficiently known, /
         not threatened, rare, vulnerable, extinct
5 : Height       : CONTINUOUS : 0, 100 : m.
7 : Remarks      : TEXT
(END : DESCRIPTORS)
```

Related directives: CLASS, DATA, DESCRIPTIONS, DESCRIPTOR-SET, DICTIONARY

Comments: TWO-STATE, ORDERED, UNORDERED and DICTIONARY are all different types of multi-state descriptor. HIERARCHY is a descriptor type including hierarchical structures, eg taxonomic hierarchies, gazetteers etc. DISCRETE and CONTINUOUS are genuine quantitative descriptors. TEXT descriptors contain free text.

A descriptor can be defined by reference to a previously-defined DICTIONARY name, but the latter name can be used anywhere a descriptor name could, so why bother to define a descriptor?

Are the labels only for the output of intelligible textual versions of descriptions, or will labels somehow be expected in the input? If the former, why are alternative labels allowed?

Name: DESCRIPTOR-SET

Version: 1.4, 14th May 1989

Use: to name combinations of descriptors to facilitate their use in particular applications

Position: anywhere within a data block

Format:

```
(DESCRIPTOR-SET [: n] )
. . .
[id] : name : descriptor-list
. . .
[(END [: DESCRIPTOR-SET] )]
```

where	n	is the number of sets to be defined
	id	is a unique identifier for each descriptor set
	name	is the name of the descriptor set
	descriptor-list	is a list of the names or code identifiers of the descriptors to be included in the set

Rules: A descriptor set can only be referred to after its definition.

Descriptor sets may have overlapping membership.

Example 1: Here two descriptor sets are defined, a set of four vegetative descriptors listed by their descriptor number and a set of four geographically related descriptors

```
(DESCRIPTOR-SET : 2)
: vegetative   : #45, #46, #47, #55
: geographical : gazeteer, place, introduction, habitat
(END)
```

Related directives: DESCRIPTORS

Comments: What exactly might a descriptor set be used for? Should it be used as part of a mechanism for allowing the specification of the logical dependence of sets of characters on particular observed states of controlling characters?

Name:	DICTIONARY
Version:	4.1, 12th June 1989
Use:	to define the allowable set of data values or states for a descriptor and optionally to assign unique codes to them
Position:	anywhere in a data set prior to the use of the values

Format:

```
(DICTIONARY : name [: ordered [: n]] )
. . .
[id] : data-value [: synonym-list]
. . .
[(END [: DICTIONARY [: name]] )]
```

where	name	is the name of the dictionary
	ordered	is either ORDERED or UNORDERED (default)
	n	is the number of dictionary elements to be defined
	id	is a data value code, unique within this dictionary
	data-value	is the corresponding descriptor state name
	synonym-list	is an optional list of alternative state names

Rules:

The name is used to refer to the dictionary in a HIERARCHY or DESCRIPTORS segment to define a descriptor derived from the dictionary or in a CLASS definition to assign the descriptor to a field of a data class

ORDERED or UNORDERED specifies whether or not the descriptor states are considered to form an ordered sequence, for those operations that need to know this

The number of records in the segment must match the number of dictionary elements (n) if declared in the DICTIONARY directive

Example 1:

```
(DICTIONARY : USES : UNORDERED : 3)
11 : medicine
12 : livestock fodder      {any part or all of the plant}
13 : timber : wood, beams  {note synonyms}
(END : DICTIONARY)
```

Example 2:

```
(DICTIONARY : Conservation status)
X : extinct    {Taxa which are no longer known to /
               occur in the wild}
E : endangered
V : vulnerable
R : rare        {Taxa with small world populations /
                that are not at present endangered /
                or vulnerable, but are at risk }
I : indeterminate
K : insufficiently known
N : neither rare nor threatened
(END)
```

Example 3:

```
(DICTIONARY : Taxon rank : ORDERED : 5)
1 : Tribe
2 : Genus
3 : Species     : species, sp.
4 : Subspecies  : subsp., Subsp., ssp.
5 : Variety     : var., Var.
(END : DICTIONARY : Taxon rank)
```

Example 4: In this example no codes are attached to the data values

```
(DICTIONARY : Geographic levels : ORDERED)
: Continent : continent, cont.
: Country   : country
: Area      : area
(END : DICTIONARY)
```

Related directives: HIERARCHY, DESCRIPTORS, CLASS

Comments: Dictionaries and multistate characters are slightly different alternative mechanisms for defining the same descriptor. Dictionaries offer the opportunity to define unique codes to stand instead of data values in other data segments and may themselves be used to define more than one descriptor or data class. Use of a dictionary rather than a multi-state descriptor has another advantage in that synonymous terms may be defined.

Name:	END
Version:	2.2, 14th May 1989
Use:	to mark the end of a data segment, data block or data set
Position:	optional at the end of any data segment, but compulsory at the end of each data block and at the end of the data set
Format:	`(END [: class [: name]] )`
	where class — is the name of a CLASS to end a data segment of that class or the name of a data block or XDF to end the data set
	name — is the name of the data segment being terminated, if it was named in the corresponding declaration
Rules:	If class is left blank then END is assumed to refer to the most recently opened data segment or data block which has not yet been ended.

Example 1:	Used to end a NOTES data segment, not defined in this document.
	`(NOTES)`
	`. . .`
	`(END : NOTES)`

Example 2:	Used to end a data block called `November 1989 update`.
	`(BEGIN : November 1989 update)`
	`. . .`
	`(END : November 1989 update)`

Example 3:	Used to end a data set.
	`(XDF : ILDIS)`
	`. . .`
	`(END : XDF)`

Related Directives:	BEGIN, XDF, any data class declaration.
Comments:	-

Name:	ERASE
Version:	1.2, 17th April 1989
Use:	to erase a file
Position:	anywhere within a data set
Format:	`(ERASE : file-name)`
	where file-name — is the operating system identifier or name for the file to be erased
Rules:	-
Example 1:	`(ERASE : myfile)`
Example 2:	`(ERASE : c!:\dir\junk)`
Related Directives:	CLOSE
Comments:	This command may not be appropriate or advisable for the inexperienced user.

Name:	HEADING
Version:	1.1, 15th May 1989
Position:	mandatory at the start of a data set, optional at the start of a data block
Use:	to assign a name, author and date to a data set or block

Format:

```
(HEADING)
TITLE     : text
[BLOCK    : block-name]
[AUTHOR   : text]
[VERSION  : text]
[DATE     : text]
. . .
[subject : text]
. . .
[(END [: HEADING] )]
```

where		
	text	is any arbitrary text appropriate to the record in which it appears
	block-name	is the name of the present block as specified in the BEGIN command
	subject	is a name to distinguish some kind of HEADING record, other than TITLE, BLOCK, AUTHOR, VERSION or DATE, according to local convention

Rules:

The TITLE record is mandatory, but AUTHOR, VERSION, DATE etc are optional subject key words

A BLOCK record may be used to distinguish between two data blocks within the same data set. If there are two or more data blocks, the names must be unique and match the names given in the BEGIN commands.

Example:

```
(HEADING)
TITLE   : An example data set /
               in ILDIS Exchange Data Format
AUTHOR  : Bob Allkin
DATE    : 6 April 1987
VERSION : 3.32
(END)
```

Related Directives: BEGIN

Comments:

HEADING was called TITLE in previous versions of the XDF proposals.

If a HEADING segment describes a data block, should it be positioned within that block or not? If within the block, the BLOCK record is redundant.

Should the list of predefined component records (TITLE, AUTHOR etc) be enlarged?

Name:	HIERARCHY
Version:	5.2, 15th May 1989
Use:	to define a particular hierarchical descriptor structure
Position:	anywhere before the occurrence of data segments using the defined hierarchical descripor

Format:

```
(HIERARCHY : name : rank-dict [: element-dict [: parent-dict]] )
. . .
element-list : rank : [: parent [: parent-rank [: source-list]]]
. . .
[(END [:HIERARCHY ] )]
```

where

name	is the name of the hierarchical descriptor being defined
rank-dict	is the name of the dictionary from which rank names are taken
element-dict	is the name of the dictionary from which member data elements are taken
parent-dict	is the name of the dictionary from which parent data elements are taken
element-list	is a list of element names or states, all of which have the same rank and parent element
rank	is the rank of the elements
parent	is their parent element
parent-rank	is their parent's rank
source-list	is a list of codes referring to bibliographic entries, or compound data values containing the entries themselves

Rules:

Member element rank and parent element rank must be defined in a named ORDERED dictionary data segment rank-dict.

Parent rank should be higher than the member's rank unless the member element is of the highest rank in the hierarchy being defined, in which case the parent name and rank should be blank.

The member's and parent's ranks need not be adjacent in the ordered dictionary sequence.

Data elements may be taken from a named element-dict dictionary in order to permit use of data codes or to permit data validation; if no element-dict is named for use with data elements, then data values, not codes, must be entered in the data records.

If an element-dict is specified for member element names, but no parent-dict is specified for parent element names then it will be assumed that the same dictionary is to be used.

Example 1:

This example refers to the dictionary Geographic levels defined as an example use of the DICTIONARY data segment.

```
(HIERARCHY : Gazetteer : Geographic levels)
Africa                 : Continent
Sudan, Ethiopia, Ghana : Country : Africa : Continent
Darfur                 : Area    : Sudan  : Country
(END)
```

Related Directives: DICTIONARY

Comments:

Allowing an element to belong to more than 1 parent allows flexibility for more complex hierarchies.

To avoid duplication of element rank, parent and parent rank, users can employ lists of member elements as in the second record of Example 1. This obviates the need for interpretation of default values.

Name:	**INCLUDE**
Version:	2.4, 19th October 1989
Use:	to read the contents of a file or selected data blocks as if they were physically inserted at the present location
Position:	anywhere in a data set
Format:	`(INCLUDE : file-name [: block-list] )`

where

file-name	is the name of the file containing the definitions or data to be included
block-list	is an optional list of the names of the data blocks within the named file to be included

Rules: The file-name must be unique and provide an appropriate operating-system specification of an existing data file.

Named data blocks must all exist and occur within the named data file; if no data blocks are named the entire file will be included.

Example 1: To include all definitions (including all data blocks, if any) from a prologue file called "HEADER.XDF":

```
(INCLUDE : header.xdf)
```

Example 2: The contents of data blocks DATA-A, DATA-B and DATA-C will be read from the file called "C:SUBDIR\MYFILE" and used as if they were stored in the present file in the order in which they are specified (not necessarily the order in which they occur in the file from which they are read). Note the use of the special escape character " ! " to ensure that the colon occurring within the file name is not interpreted as the parameter separator character.

```
(INCLUDE : C!:SUBDIR\MYFILE : DATA-A, DATA-B, DATA-C)
```

Related directives: BEGIN, END

Comments: INCLUDE is the most easily used way to read data from any file other than that in which the XDF data set started. INCLUDE automatically opens and closes the included file, being defined on the basis of the low-level macros `open`, `close` and `take` not documented here.

It enables data of common interest to a number of data sets to be stored physically in a separate file, and logically discrete data sets to be kept apart even within the same data file.

This version of INCLUDE incorporates the facilities previously provided by the separate GET command.

Name:	NAMES
Version:	4.1, 12th June 1989
Use:	to define scientific name strings to be attached to taxa in other segments such as TAXA or SYNONYMS
Position:	anywhere in a data set

Format:

```
(NAMES [: n] )
. . .
id : taxon-rank : name [: source-list]
. . .
[(END [: NAMES] )]
```

where	n	is the number of name strings defined in this segment
	taxon-rank	is an entry from the Taxon rank dictionary
	id	is an arbitrarily chosen unique name code
	name	is an explicit name, enclosed within compound data value delimiters " [" and "] ", containing a number of name element fields each followed by an authority name
	source-list	is an optional list of bibliographic references

Rules: If the taxon-rank is Tribe or Genus then the name-list must contain one name element, or two for a Species or three for a Subspecies or Variety.

If the taxon-rank is Subspecies or Variety and the species authority field is left blank then the infraspecific authority is assumed to apply to the species epithet.

The source-list is optional, and should refer to the original publication of the name.

Example:

```
(NAMES : 4)
25 : Species : [Vicia : L. : faba : L.]
26 : var.    : [Vicia : L. : faba : L. : minor : Beck]
31 : Tribe   : [Vicieae]
27 : var.    : [Vicia : L. : faba :    : faba  : L.]
(END)
```

Related directives: SYNONYMS, TAXA

Comments: This data class is included as an illustration of how a NAMES class might be defined. It is taken from Allkin & Winfield (1989) where XDF is used to define the Ildis Data Transfer Format.

Name:	SET
Version:	2.2, 3rd August 1988
Use:	to change the character value used to represent a particular punctuation character
Position:	anywhere within a data set
Format:	(SET : name : value)

where name — is the XDF macro name for the punctuation character or character pair being assigned a new character value

value — is the new character value or pair of values

Rules: The SET keyword takes effect immediately (even in mid-record) and remains in effect for all subsequent data, unless superseded by another later SET applied to the same punctuation character.

All punctuation characters may be redefined; they are listed in Appendix B of this document.

No character value may be used to represent more than one punctuation character simultaneously, if ambiguity would result.

Only single character values can be used as punctuation characters, although sometimes these are referred to in pairs.

The punctuation character values for a particular data set may be predefined in the default header file referred to by the XDF command.

Example 1: The effect of this SET command is that curly brackets take on the rôle that round brackets previously had.

```
(SET : REMARK : {})
```

Example 2: This SET command replaces the default escape character " ! " with "*".

```
(SET : ESCAPE : *)
```

Related directives: XDF

Comments: The SET command may also be used to change the values of other macros. Such a use is beyond the scope of this document.

Name:	SYNONYMS
Version:	4.0, 31st October 1989
Use:	to assign names other than accepted names to taxa
Position:	anywhere in the data set

Format:

```
(SYNONYMS [: n] )
. . .
id : class : confidence : name-rank : name-list [: source-list]
. . .
[(END [: SYNONYMS] )]
```

where		
	n	is the number of synonyms defined in this segment
	id	is the accepted taxon, as defined in the TAXA data segment, to which the synonym or synonyms apply
	class	is either Synonym, Misapplied name or Orthographic variant
	confidence	is either OK (default) or Doubtful
	name-rank	is an entry from the Taxon rank dictionary
	name-list	is a list containing name codes of names defined in a NAMES segment and explicit names enclosed within compound data value delimiters " [" and "] " and containing a number of name element fields each followed by an authority name
	source-list	is an optional list of bibliographic references

Rules: The id must be defined as a taxon code in a TAXA segment.

The name-rank must equal the taxon-rank associated with the definition of each name in a NAMES segment.

If the class of the synonym is Misapplied name or Orthographic variant then confidence should be OK.

If class is Misapplied name then the source-list should be compulsory.

Example 1:

```
(SYNONYMS)
#18 : Synonym :            : Species : #107, #108 : #216
#18 : Synonym : Doubtful : Variety : #109        : #216
END)
```

Example 2:

```
(SYNONYMS : 2)
#18 : Synonym         :: Species : /
        [Faba : L. : bona : Medik.], /
        [Faba : L. : faba : Medik.]               : #187, #188
#18 : Misapplied name :: Variety : /
        [Vicia : L. : faba : L. : minor : Beck] : #187
(END : SYNONYMS)
```

Related directives: NAMES, TAXA

Comments: This data class is included here for illustrative purposes only. A SYNONYMS class may be defined in many different ways although it is hoped that standards will be adopted.

Name:	TAXA
Version:	4.0, 12th June 1989
Use:	to declare the existence of one or more taxa and assign a single accepted name to each
Position:	anywhere within data set

Format:

```
(TAXA [: n] )
. . .
id : status : rank : name-list [: source-list]
. . .
[(END [: TAXA] )]
```

where	n	is the number of taxa defined in this segment
	id	is an arbitrarily chosen unique taxon code
	status	is either Accepted (default) or Provisional
	rank	is an entry from the Taxon rank dictionary
	name	is either the name code of a name defined in a NAMES segment or an explicit name enclosed within compound data value delimiters " [" and "] " and containing a number of name element fields each followed by an authority name
	source-list	is an optional list of bibliographic references

Rules: The rank must be appropriate to that of the accepted name string.

Example 1: An example using name codes referring to names defined in some NAMES segment.

```
(TAXA : 2)
28 : Accepted    : Species : #103 : #194, #108, #218
12 : Provisional : Species : #104 : #108, #109
(END : TAXA)
```

Example 2: An example using name strings embedded in the TAXA data segment.

```
(TAXA)
101 : Accepted    : Species    : /
      [Vicia : L. : faba : L.] : #187
28  : Accepted    : Subspecies : /
      [Vicia : L. : faba : L. : faba] : #194, #108, #218
12  : Provisional : Variety    : /
      [Vicia : L. : faba : L. : minor : Beck] : #108, #109
(END)
```

Related directives: NAMES, SYNONYMS

Comments: This data class might be defined in other ways. The definition given here is that used by the International Legume Database Project and is taken for illustrative purposes from Allkin & Winfield (1989).

Name:	XDF
Version:	3.1, 16th May 1989
Use:	to indicate the start of a data set, any prologue header file to be read and any predefined data conventions followed, such as the character set
Position:	compulsory at the start of any XDF data set and optional at the beginning of data files INCLUDEd within the data set or anywhere else to indicate a change of default header block

Format:

```
(XDF [: prologue [: char-set]] )
. . .
[(END : XDF)]
```

where	name	is the name of a default prologue or header data block whose definitions are required
	char-set	is the name of the character set used in this data set and previously defined in a CHARACTER-SET data segment

Rules: An XDF command must occur as the first directive in a data set, and must be matched by a final (END:XDF) command.

XDF commands may occur any number of times within a data set. If this happens the effect is to replace, not add to, any definitions and conventions established by the initial XDF command.

Example: See Appendix D for an example data set initiated with an XDF command that indicates use of the conventions and data classes as defined for the Ildis database project.

Related directives: BEGIN, INCLUDE, END

Comments: -

Appendix D: Example Ildis data set

This example illustrates several features of XDF, including use of more particular data class definitions taken from the Ildis data definitions.

```
(XDF : ILDIS : ILDIS-CHARACTERS)
                              { The ILDIS parameter here defines a special kind
                                of XDF data set with rules defined in a particular
                                header data block where the local definitions for
                                TAXA, SYNONYMS and REFERENCES are placed }

(SET:FIELD-SEPARATOR:|)  { Punctuation characters can be changed.  Here the
                           default field separator ":" is changed to "|" }
(SET:MACRO-SEPARATOR:|)  { The default macro parameter separator character
                           ":" is also changed to "|" }

(HEADING)
TITLE   | Example data set using ILDIS descriptors
AUTHOR  | Bob Allkin & Richard White
VERSION | 1.4
DATE    | 9 September 1988
(END | HEADING)

(DICTIONARY | Taxon rank | ORDERED)
                              { The taxonomic ranks in this data set are defined,
                                with alternative spellings in some cases }
1 | Tribe
2 | Genus
3 | Species     | species
4 | Subspecies  | subsp., Subsp.
5 | Variety     | var.,   Var.
(END | DICTIONARY | Taxon rank)

(TAXA)                        { Taxa are declared, given a rank, and a name is
                                assigned to each.  Except for the tribe
                                'Caesalpinieae', the names are indicated by code
                                values that refer to name strings defined in the
                                NAMES data segment below }
12  | Accepted    | Tribe       | [Caesalpinieae]
11  | Accepted    | Genus       | #1
19  | Provisional | species     | #9
15  | Accepted    | Genus       | #5
14  | Accepted    | species     | #4
122 | Accepted    | subspecies  | #22
123 | Accepted    | subspecies  | #23
(END | TAXA)
```

```
(NAMES)                { Names are declared and assigned code values used
                         in the corresponding taxon declarations above }
1   | Genus      | [Caesalpinia|L.]
9   | species    | [Caesalpinia|L.|bessac|Chiov]           | #125
                       { The reference #125 could be the protologue }
5   | Genus      | [Bussea|Harms]
4   | species    | [Bussea|Harms|massaiensis|(Taubert) Harms] | #106

22  | subspecies | [Bussea|Harms|massaiensis|(Taubert) Harms|/
                                        massaiensis]       | #106
23  | subspecies | [Bussea|Harms|massaiensis|(Taubert) Harms|/
                                        rhodesica|Brenan]  | #106
(END | NAMES)

(DESCRIPTORS)          { Data segment to define taxon descriptors such
                         as morphological characters }
1 | Habit               | UNORDERED  | herb, shrub, tree      | variable
2 | <Climbing>          | BISTATE    | climbing, not climbing | variable
4 | Conservation status | UNORDERED  | endangered, indeterminate, /
          insufficiently known, not threatened, rare, vulnerable, extinct
5 | Height              | CONTINUOUS | 0, 100 | m., metres, m
(END | DESCRIPTORS)

(DESCRIPTIONS)         { Descriptive observations for taxa.  Data are
                         assigned to taxa of various taxonomic rank }
#14 | #1 | tree, shrub { Taxon 14 is Bussea massaiensis, descriptor 1
                         is 'Habit' }
Caesalpinia | Habit | tree, shrub | #81, #125
                       { Sources 81 & 125 say that Caesalpinia are
                         trees and shrubs }
#122 | Conservation status | 1 | #106
                       { 'Conservation status' state 1 is 'endangered' }
#123 | #4 | 2          { 'Conservation status' = 'indeterminate' }
(END | DESCRIPTIONS)

(REFERENCES)           { A segment of a data class defined by Ildis }
106|Brenan JPM |1967| Caesalpinioideae. In: Flora of Tropical East Africa
81 |Chiovenda E|1929| Flora Somala 1
125|Roti-Michelozzi G | 1957 | /
        Webbia 13: 133-228. Advn. Fl. Aeth. Caesalpinioideae excl. Cassia
(END | REFERENCES)

(END | XDF)
```

Appendix E: Installation and support

We aim to support approaches (ii) and (iii) to the manipulation of XDF data sets, described in section 4. This support will consist, initially, of five products.

XGPM program	This is the stand-alone processor program for XDF data sets, which is used to convert data sets into other formats. The first version is for MS-Dos version 2 or later.
XGPM sampler	This is intended as a demonstration version of XGPM. However, it is a fully-working version of XGPM which is restricted only in the size of the data sets it can process.
XGPM library	XDF data set processing can be performed in your own database or application program by linking in this library of support routines, which consist essentially of the whole of the XGPM program. Initially the library will be available in compiled form, suitable for calling from any MS-Dos program which can interface to Turbo C or Microsoft C libraries.
XGPM source code	The source code for the XGPM library is written in the standard ANSI C language and is highly machine-independent. Versions translated into Fortran 77 or other languages will become available later if sufficient demand arises.
XDF definitions	All the above products require a basic set of XDF definitions to handle the facilities described in this document. Further modules of XDF definitions for particular specialised data classes and for conversions to specific output formats may be available later.

In most cases a financial contribution will be requested to cover the costs of developing, maintaining and supporting this software and to allow further developments, already in the pipeline, to continue. Development work to support particular applications may be undertaken by arrangement.

Details of all these products are available from
Richard J. White, Biology Department, The University, SOUTHAMPTON SO9 5NH, United Kingdom,
telephone Southampton (0703) 592021 (international +44 703 592021), telex 47661 SOTONU G, fax (0703) 558163 (international +44 703 558163);
or at Fair View Cottage, Penton's Hill, Hyde, FORDINGBRIDGE, Hampshire SP6 2HL, United Kingdom,
telephone Fordingbridge (0425) 54025 (international +44 425 54025).

Index

An up-to-date index has not yet been prepared for this version of the manual. We hope to include one in the published edition, possibly combined with the glossary.