シェープファイルの技術情報

ESRI ホワイトペーパー・シリーズ

1998 年 7 月

(2006/07/12 update)

ESRI ジャパン株式会社

シェープファイルの技術情報

ESRI ホワイトペーパー・シリーズ

1998 年 7 月

目次

シェープファイルの技術情報

このドキュメントではシェープファイルのフォーマットを解説し、シェープファイルを使う利点について説明します。シェープファイルを直接作成したり他のデータ・フォーマットから変換してシェープファイルを作成したりできる、ESRI 社のソフトウェア製品についても紹介します。ESRI 社のソフトウェア製品を使わずにシェープファイル形式のデータを直接書き込むためのプログラムを作成するのに必要な技術情報は、すべてこのドキュメントで記載しています。

シェープファイルの利点

シェープファイルはトポロジー構造をもたない空間データの位置と形に関する情報と、その属性情報を格納するためのデータ形式です。位置と形に関する情報は、(座標値のベクトルで構成される) シェープとして格納されます。

シェープファイルではトポロジー構造の計算のためオーバーヘッドが不要なので、他のデータソースと比較して、描画が速い、データ編集が容易などの利点があります。また、オーバーラップしたり、接していないフィーチャを個別に扱うことができます。データを格納するのに必要なディスク容量は一般に少なくてすみ、リード／ライトが容易です。

シェープファイルは点、線、面のデータを扱います。面は一筆書きの閉曲線であらわされ、境界線は二重になります。属性情報は dBASE フォーマットのファイルに格納します。属性ファイルの各レコードはシェープのレコードと 1 対 1 に対応します。

シェープファイルを作成する方法

一般に、シェープファイルを作成するには以下の 4 通りの方法があります。

■エキスポートによる。ArcInfo、PC ARC/INFO、Spatial Database Engine (SDE)、ArcView、BusinessMAP などのソフトウェアを使って、データをエキスポートしてシェープファイルを作成します。

■ディジタイズによる。ArcView のデータ作成機能を使って、直接ディジタイズすることによりシェープファイルを作成します。

■プログラミングによる。Avenue(ArcView)、MapObjects、AML(ArcInfo)、SML(PC ARC/INFO)でプログラミングすることによりシェープファイルを作成します。

■プログラムを作成し、シェープファイルの仕様に沿ってデータファイルを作成します。

SDE、ArcInfo、PC ARC/INFO、Data Automation Kit(DAK)、ArcCAD のソフトウェア製品はシェープファイルからカバレッジへの変換機能を備えています。また ArcInfo はカバレッジからシェープファイルへの変換機能も備えています。その他のデータ・フォーマットとの変換には、本書の記載を参照してください。GPS(Global Positioning System)のデータなどは、シェープファイルとして格納することも XY イベント・テーブルとして保存することもできます。

シェープファイルの技術的記載

この章の技術情報を使って、シェープファイルをリード／ライトするプログラムを作成することができます。

シェープファイルはメイン・ファイル、インデックス・ファイル、属性ファイル（dBASE ファイル）で構成されます。メイン・ファイルは直接編成の可変長レコードファイルで、1 レコードごとにシェープを構成する頂点の座標リストを持っています。インデックス・ファイルには各レコードのメインファイル先頭からのオフセットを記録しています。属性ファイルには、1 フィーチャにつき 1 レコードでフィーチャの属性を格納しています。

図形情報と属性情報は、レコード番号によって 1 対 1 の対応関係が作られています。dBASE ファイル中の属性レコードは、メイン・ファイルの図形情報のレコードと同じ順序になっていなければいけません。

命名規則

すべてのファイル名は 8.3 形式の命名規則に従います[*1)]。メイン・ファイル、インデックス・ファイル、属性ファイルのプレフィクスは同じでなければなりません。プレフィクスは英数字（a-Z, 0-9）で始まり、そのあとに 7 個までの文字（a-Z, 0-9, _, -）を使うことができます。メイン・ファイルのサフィックス（拡張子）は".shp"です。またインデックス・ファイルのサフィックスは".shx"、属性ファイルのサフィックスは".dbf"です。大文字と小文字を区別するオペレーティング・システム（OS）では、ファイル名には小文字を使います。

例

■メイン・ファイル：counties.shp
■インデックス・ファイル：　counties.shx
■属性ファイル：　counties.dbf

訳注 1)これは 16 ビット OS での制限事項でした。現在は 8.3 形式より長い"ロングファイルネーム"をつけられる OS がありますが、シェープファイルが 16 ビット OS の上で動作するアプリケーション・プログラムで使われる可能性や、8.3 形式でなければファイル名として認識しないアプリケーション・プログラムもあることを考慮して、シェープファイルのファイル名は 8.3 形式に従うことをお勧めします。

数値の型

シェープファイルでは整数型と倍精度実数型の数値を使います。本書ではこれ以降、次のように数値の型を定義します。

■Integer：符号付き 32 ビット整数（4 バイト）
■Double：符号付き 64 ビット IEEE 浮動小数点数（8 バイト）

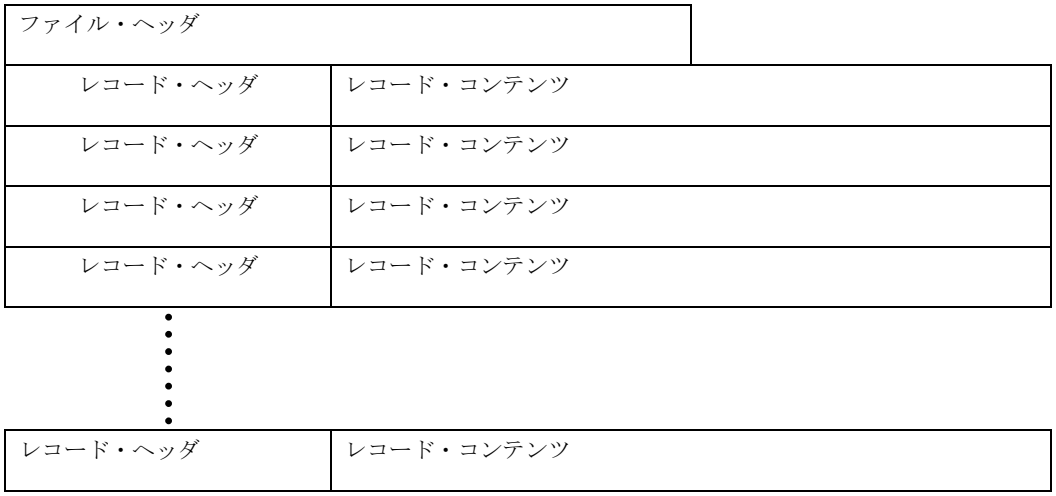
浮動小数点数は数値でなければなりません。正の無限大、負の無限大および非数（NaN）はシェープファイルでは扱えません。シェープファイルでは「NoData値」という概念をサポートしますが、現在のところmeasureにのみ使われます。-10^{38}より小さい任意の浮動小数点数は、シェープファイルを解釈するときにNoDataをあらわす値として解釈されます。

この次の節ではシェープファイルの全般的な構造と構成を解説します。2 番目の節でシェープファイルがサポートするシェープ・タイプごとにレコードの内容を記述します。

メイン・ファイルの構造

メイン・ファイルは固定長のファイル・ヘッダと可変長のレコードで構成されます。個々のレコードは固定長のレコード・ヘッダと、これに続く可変長のレコード・コンテンツから作られます。図１にメイン・ファイルの構成を示します。

図１　メイン・ファイルの構成



バイト順

シェープファイルが含む内容は、次の２種類のいずれかに分類されます。

■データ関連
　メイン・ファイルのレコード・コンテンツ
　メイン・ファイルのヘッダのデータ記述フィールド（シェープ・タイプ、バウンディング・ボックスなど）
■ファイル管理関連
　ファイル長、レコード長
　レコード・オフセットなど

ファイルのヘッダのデータ記述フィールドで使われる整数と倍精度整数およびメイン・ファイルのレコード・コンテンツは、リトル・エンディアンのバイト順（PC、Intel 方式）になっています。このほかの整数と倍精度浮動小数点数およびファイル管理関連項目はビッグ・エンディアンのバイト順（Sun、Motorola 方式）になっています。

メイン・ファイル・ヘッダ

メイン・ファイル・ヘッダは 100 バイトの長さがあります。表１にファイル・ヘッダのフィールドと開始位置、値、型、バイト順を示します。開始位置はファイル先頭からのバイト数です。

表１：メイン・ファイル・ヘッダの構成

開始位置	フィールド	値	型	バイト順
Byte 0	ファイル・コード	9994	Integer	ビッグ
Byte 4	未使用	0	Integer	ビッグ
Byte 8	未使用	0	Integer	ビッグ
Byte 12	未使用	0	Integer	ビッグ
Byte 16	未使用	0	Integer	ビッグ
Byte 20	未使用	0	Integer	ビッグ
Byte 24	ファイル長	ファイル長	Integer	ビッグ
Byte 28	バージョン	1000	Integer	リトル
Byte 32	シェープ・タイプ	シェープ・タイプ	Integer	リトル
Byte 36	バウンディング・ボックス	Xmin	Double	リトル
Byte 44	バウンディング・ボックス	Ymin	Double	リトル
Byte 52	バウンディング・ボックス	Xmax	Double	リトル
Byte 60	バウンディング・ボックス	Ymax	Double	リトル
Byte 68*	バウンディング・ボックス	Zmin	Double	リトル
Byte 76*	バウンディング・ボックス	Zmax	Double	リトル
Byte 84*	バウンディング・ボックス	Mmin	Double	リトル
Byte 92*	バウンディング・ボックス	Mmax	Double	リトル

*measure なし、あるいは Z 値を持たないシェープ・タイプでは、未使用フィールドとして値 0.0 が入る。

ファイル長の値はワード単位（16 ビットを 1 ワードとする）で、ヘッダの 50 ワードを含む全ファイルの長さです。

ひとつのシェープファイル内では、Null Shape を除いてすべて同じシェープ・タイプでなければなりません。シェープ・タイプは、それぞれ次の値であらわします。

値	シェープ・タイプ
0	Null Shape
1	Point
3	PolyLine
5	Polygon
8	MultiPoint
11	PointZ
13	PolyLineZ
15	PolygonZ
18	MultiPointZ
21	PointM
23	PolyLineM
25	PolygonM
28	MultiPointM
31	MultiPatch

上の表で値を与えていないシェープ・タイプは、将来の拡張のための予約領域です。
現行のシェープファイルでは、シェープ・タイプはすべて同じでなければならないという制限があります。将来は、異なるシェープ・タイプの混在を許すように拡張される可能性があります。異なるシェープ・タイプの混在が実装されるときには、シェープ・タイプのフィールドには組み合わせに応じたフラグをたてるようになります。

メイン・ファイル・ヘッダのバウンディング・ボックスには、ファイル中のシェープが存在する座標範囲（すなわち、X 軸と Y 軸に平行な辺ですべてのシェープを囲むような最小の長方形）の値を格納しています。M 座標と Z 座標についても同様です。シェープがない場合（すなわちレコードがない場合）には Xmin、Ymin…などの値は不定になります。measure つきのシェープ・タイプで実際には measure の値を与えないとき、Mmin と Mmax は「NoData 値」を取り得ます（3 ページの「数値の型」を参照）。

レコード・ヘッダ

各レコードのレコード・ヘッダには、レコード番号とレコード・コンテンツの長さが格納されています。レコード・ヘッダは 8 バイトの固定長です。表 2 に、レコード・ヘッダのフィールドと開始位置、値、型、バイト順を示します。開始位置はレコードの先頭からのバイト数であらわしています。

表２　メイン・ファイルのレコード・ヘッダ

開始位置	フィールド	値	型	バイト順
Byte 0	レコード番号	レコード番号	Integer	ビッグ
Byte 4	コンテンツ長	コンテンツ長	Integer	ビッグ

レコード番号は 1 から始まります。

コンテンツ長は、各レコードのレコード・コンテンツの部分の長さをワード単位であらわしたものです。ファイル・ヘッダの 24 バイト目に示されているファイル長に対して、各レコードは（4 ＋コンテンツ長）ワード分だけ寄与していることになります。

メイン・ファイルのレコード・コンテンツ

シェープファイルのレコード・コンテンツは、シェープ・タイプとそのシェープの形状を記述するデータとで構成されます。レコード・コンテンツの長さは、シェープを構成するパートと頂点の個数によって変わります。この後の節では、各シェープ・タイプごとにシェープの定義を解説し、つづいてレコード・コンテンンツとしてどのような持ち方をするかを説明します。表 3 から表１６では、開始位置はレコード・コンテンツの先頭からのバイト数であることに注意してください。

Null Shape

シェープ・タイプ 0 は Null Shape です。このシェープ・タイプには位置と形状に関するデータは付随しません。各フィーチャ・タイプ（ポイント、ライン、ポリゴンなど）は Null をサポートし、同じシェープファイル内で例えば Point と Null Point の共存が許されています。多くの場合 Null Shape はシェープファイルを生成する際にプレースホールダとして使われます。この場合は、シェープファイルが作成されれば Null Shape は直ちに位置と形状をもつ通常のシェープで置き換えられます。

表３　Null Shape のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	0	Integer	1	リトル

XY 空間における シェープのタイプ

Point

Point は X,Y の順に並んだ倍精度の座標値で構成されます。

```
Point
{
        Double    X        //X 座標値
        Double    Y        //Y 座標値
}
```

表４　Point のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	1	Integer	1	リトル
Byte 4	X	X	Double	1	リトル
Byte 12	Y	Y	Double	1	リトル

MultiPoint

MultiPoint は Point の集合で、下記のように表現されます。

```
MultiPoint
{
        Double[4]          Box          //バウンディング・ボックス
        Integer            NumPoints    //Point の個数
        Point[NumPoints]   Points       //集合に含まれる Point
}
```

バウンディング・ボックスには Xmin、Ymin、Xmax、Ymax の順で値を格納します。

表５　MultiPoint のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	8	Integer	1	リトル
Byte 4	ボックス	Box	Double	4	リトル
Byte 36	NumPoints	NumPoints	Integer	1	リトル
Byte 40	Points	Points	Point	NumPoints	リトル

PolyLine

PolyLine は順序をつけた頂点の集合で、1 個または複数のパートで構成されます。パートは 2 個以上の Point を順に結んだもので、パート同士が接続していることもあれば離れていることもあります。また、パート同士が交差することもあります。

仕様では、連続する 2 個の Point が全く同じ座標値を持つ場合を禁じていないので、シェープファイルを利用する場合はこのことを想定しておく必要があります。しかし、長さゼロの、縮退したパートは許されないことに注意してください。

```
PolyLine
{
        Double[4]           Box         //バウンディング・ボックス
        Integer             NumParts    //パートの個数
        Integer             NumPoints   //Point の総数
        Integer[NumParts]   Parts       //パートの最初のPoint へのインデックス
        Point[NumPoints]    Points      //すべてのパートに関する Point
}
```

各フィールドの詳細は以下の通りです。

Box : Xmin、Ymin、Xmax、Ymax の順で値を格納したバウンディング・ボックス。

NumParts : PolyLine に含まれるパートの個数。

NumPoints : すべてのパートを合計した Point の総数。

Parts : 長さ NumParts の配列。配列 Point 中の、各パートの最初の Point へのインデックスを格納する。配列の添字は 0 から始まる。

Points : 長さ NumPoints の配列。PolyLine を構成する Points をすべて格納する。パート 1 の Point につづいてパート 2、パート 3…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。

表 6　PolyLine のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	3	Integer	1	リトル
Byte 4	ボックス	Box	Double	4	リトル
Byte 36	NumParts	NumParts	Integer	1	リトル
Byte 40	NumPoints	NumPoints	Integer	1	リトル
Byte 44	Parts	Parts	Integer	NumParts	リトル
Byte X	Points	Points	Point	NumPoints	リトル

注：X=44+(4×NumParts)

Polygon

Polygon は 1 個または複数のリングで構成されます。ここでリングとは、4 個以上の Point を順につないだもので、自分自身に交差しない閉曲線のことを指します。Polygon ではいくつかのリングが入れ子になっていることがあります。リングを構成する頂点の順番、すなわちリングの向きによって、リングのどちら側を Polygon の内部とみなすかが決まります。頂点の順にリングを辿るとき、右側にある方を Polygon の内部と定義します。

Polygon 内部の穴をあらわす場合、リングの向きは反時計回りになります。1 本のリングで構成される Polygon の場合には、リングの向きは常に時計回りになります。このリングのことを、Polygon のパートと呼びます。

仕様では、連続する 2 個の Point が全く同じ座標値を持つ場合を禁じていないので、シェープファイルを利用する場合はこのことを想定しておく必要があります。しかし、長さゼロ・面積ゼロの、縮退したパートは許されないことに注意してください。

Polygon の構造は PolyLine と同じで、以下のようになります。

```
Polygon
{
        Double[4]           Box         //バウンディング・ボックス
        Integer             NumParts    //パートの個数
        Integer             NumPoints   //Point の総数
        Integer[NumParts]   Parts       //パートの最初のPoint へのインデックス
        Point[NumPoints]    Points      //すべてのパートに関する Point
}
```

各フィールドの詳細は以下の通りです。

Box：	Xmin、Ymin、Xmax、Ymax の順で値を格納したバウンディング・ボックス。
NumParts：	Polygon に含まれるリングの個数。
NumPoints：	すべてのリングを合計した Point の総数。
Parts：	長さ NumParts の配列。配列 Point 中の、各リングの最初の Point へのインデックスを格納する。配列の添字は 0 から始まる。
Points：	長さ NumPoints の配列。Polygon を構成する Points をすべて格納する。 リング 1 の Point につづいてリング 2、リング 3…の順になる。配列 Parts には、各リングの最初の Point を格納している Points 要素の添字の値を持っている。リングとリングの間にデリミタはない。

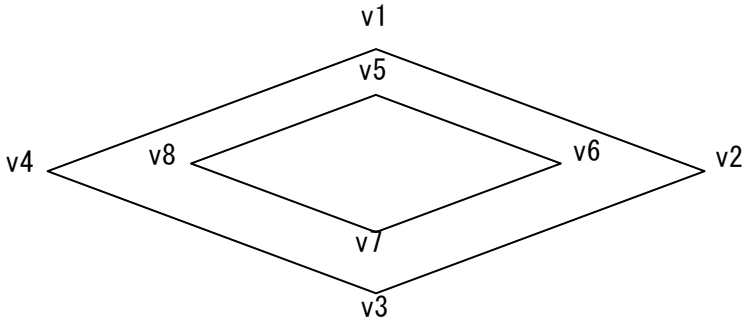
図 2 に Polygon のデータの表現方法を示します。

Polygon ポリゴン・シェープについて重要な点を以下に列挙します。

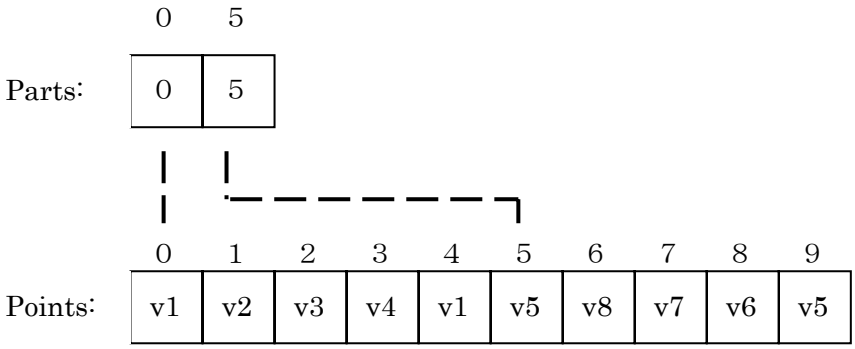
■リングが閉じていること（始点と終点が同一である）
■配列 Point 中のリングの順序はポリゴンの構造に影響しない
■シェープファイルとして保存する Polygon は「クリーン」でなければならない。ここでクリーンな Polygon とは次の 2 点を満たすことをいう。

1．自分自身と交差しない。これは、あるリングの一部が他のリングの一部に交差することを禁止する。リングは頂点で他のリングと接してもよい。しかし線分の一部を他のリングと共有することは許されない。
2．ポリゴンを構成するラインの「正しい」側に内部領域がある。頂点の順にリングを辿るとき、右側にある方を Polygon の内部と定義する。1 本のリングで構成される Polygon の場合には、リングの向きは常に時計回りになる。Polygon 内部の穴をあらわす場合、リングの向きは反時計回りになる。Polygon 内部の穴をあらわすべきリングを時計回りに作ってしまうと、内部の領域が重なった不正な Polygon になってしまう。

図２：ポリゴン・シェープの例



この例では NumParts = 2、NumPoints = 10 となる。ドーナツ（穴）を構成する頂点の向きが逆転していることに注意。



Polygon

表 7　Polygon のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	5	Integer	1	リトル
Byte 4	ボックス	Box	Double	4	リトル
Byte 36	NumParts	NumParts	Integer	1	リトル
Byte 40	NumPoints	NumPoints	Integer	1	リトル
Byte 44	Parts	Parts	Integer	NumParts	リトル
Byte X	Points	Points	Point	NumPoints	リトル

注：X=44+(4×NumParts)

XY 空間における measure つきのシェープのタイプ

このタイプのシェープでは M 座標が追加されています。M 座標に関しては NoData 値を取り得ることに注意してください（3 ページの「数値の型」を参照）。

PointM

PointM は X,Y の順に並んだ倍精度の座標値に加え、measure の値 M で構成されます。

```
PointM
{
        Double    X        //X 座標値
        Double    Y        //Y 座標値
        Double    M        //measure
}
```

表 8　PointM のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	21	Integer	1	リトル
Byte 4	X	X	Double	1	リトル
Byte 12	Y	Y	Double	1	リトル
Byte 20	M	M	Double	1	リトル

MultiPointM

MultiPointM は PointM の集合で、下記のように表現されます。

```
MultiPointM
{
        Double[4]              Box          //バウンディング・ボックス
        Integer                NumPoints    //Point の個数
        Point[NumPoints]       Points       //集合に含まれる Point
        Double[2]              M Range      //measure の値の範囲
        Double[NumPoints]      M Array      //各 Point の measure の値
}
```

各フィールドの詳細は以下の通りです。

Box :	Xmin、Ymin、Xmax、Ymax の順で値を格納したバウンディング・ボックス
NumPoints :	Point の総数
Points :	長さ NumPoints の Point の配列
M Range :	Mmin、Mmax の順で値を格納した M 値の最小値および最大値
M Array :	長さ NumPoints の measure の配列

表９　MultiPointM のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	28	Integer	1	リトル
Byte 4	ボックス	Box	Double	4	リトル
Byte 36	NumPoints	NumPoints	Integer	1	リトル
Byte 40	Points	Points	Point	NumPoints	リトル
Byte X*	Mmin	Mmin	Double	1	リトル
Byte X+8*	Mmax	Mmax	Double	1	リトル
Byte X+16*	Marray	Marray	Double	NumPoints	リトル

注：X=40+(16×NumPoints)

*このフィールドは存在しない場合がある

PolyLineM

PolyLineMは1個または複数のパートで構成されます。パートは2個以上のPointを順に結んだもので、パート同士が接続していることもあれば離れていることもあります。また、パート同士が交差することもあります。

```
PolyLineM
{
        Double[4]              Box          //バウンディング・ボックス
        Integer                NumParts     //パートの個数
        Integer                NumPoints    //Point の総数
        Integer[NumParts]      Parts        //パートの最初のPoint へのインデックス
        Point[NumPoints]       Points       //すべてのパートに関する Point
        Double[2]              M Range      //measure の値の範囲
        Double[NumPoints]      M Array      //各 Point の measure の値
}
```

各フィールドの詳細は以下の通りです。

Box：	Xmin、Ymin、Xmax、Ymax の順で値を格納したバウンディング・ボックス。
NumParts：	PolyLineM に含まれるパートの個数。
NumPoints：	すべてのパートを合計した Point の総数。
Parts：	長さ NumParts の配列。配列 Point 中の、各パートの最初の Point へのインデックスを格納する。配列の添字は 0 から始まる。
Points：	長さ NumPoints の配列。PolyLineM を構成する Points をすべて格納する。パート 1 の Point につづいてパート 2、パート 3…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。
M Range：	Mmin、Mmax の順で値を格納した measure の最小値および最大値。
M Array ：	長さ NumPoints の measure の配列。パート 1 の Point につづいてパート 2、パート 3…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。

表１０　PolyLineM のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	23	Integer	1	リトル
Byte 4	ボックス	Box	Double	4	リトル
Byte 36	NumParts	NumParts	Integer	1	リトル
Byte 40	NumPoints	NumPoints	Integer	1	リトル
Byte 44	Parts	Parts	Integer	NumParts	リトル
Byte X	Points	Points	Point	NumPoints	リトル
Byte Y*	Mmin	Mmin	Double	1	リトル
Byte Y+8*	Mmax	Mmax	Double	1	リトル
Byte Y+16*	Marray	Marray	Double	NumPoints	リトル

注：X=44+(4×NumParts)、Y=X+(16×NumPoints)

*このフィールドは存在しない場合がある

PolygonM

PolygonM は 1 個または複数のリングで構成されます。ここでリングとは、自分自身に交差しない閉曲線のことを指します。交差の判定は XYM 空間ではなく XY 空間で計算することに注意してください。PolygonM ではいくつかのリングが入れ子になっていることがあります。リングのことを PolygonM のパートと呼びます。

PolygonM の構造は PolyLineM と同じで、以下のようになります。

```
PolygonM
{
        Double[4]           Box         //バウンディング・ボックス
        Integer             NumParts    //パートの個数
        Integer             NumPoints   //Point の総数
        Integer[NumParts]   Parts       //パートの最初のPoint へのインデックス
        Point[NumPoints]    Points      //すべてのパートに関する Point
        Double[2]           M Range     //measure の値の範囲
        Double[NumPoints]   M Array     //各 Point の measure の値
}
```

各フィールドの詳細は以下の通りです。

Box：	Xmin、Ymin、Xmax、Ymax の順で値を格納したバウンディング・ボックス。
NumParts：	PolyLineM に含まれるパートの個数。
NumPoints：	すべてのパートを合計した Point の総数。
Parts：	長さ NumParts の配列。配列 Point 中の、各パートの最初の Point へのインデックスを格納する。配列の添字は 0 から始まる。
Points：	長さ NumPoints の配列。PolygonM を構成する Points をすべて格納する。パート 1 の Point につづいてパート 2、パート 3…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。
M Range：	Mmin、Mmax の順で値を格納した measure の最小値および最大値。
M Array ：	長さ NumPoints の measure の配列。パート 1 の Point につづいてパート 2、パート 3…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。

PolygonM シェープについて重要な点を以下に列挙します。

■リングが閉じていること（始点と終点が同一である）。
■配列 Point 中のリングの順序はポリゴンの構造に影響しない。

PolygonM

表１１　PolygonM のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	25	Integer	1	リトル
Byte 4	ボックス	Box	Double	4	リトル
Byte 36	NumParts	NumParts	Integer	1	リトル
Byte 40	NumPoints	NumPoints	Integer	1	リトル
Byte 44	Parts	Parts	Integer	NumParts	リトル
Byte X	Points	Points	Point	NumPoints	リトル
Byte Y*	Mmin	Mmin	Double	1	リトル
Byte Y+8*	Mmax	Mmax	Double	1	リトル
Byte Y+16*	Marray	Marray	Double	NumPoints	リトル

注：X=44+(4×NumParts)、Y=X+(16×NumPoints)

*このフィールドは存在しない場合がある

XYZ 空間における シェープのタイプ

このタイプのシェープでは M 座標が追加されています。M 座標に関しては NoData 値を取り得ることに注意してください（3 ページの「数値の型」を参照）。

PointZ

PointZ は倍精度の X、Y、Z 座標および M 座標の組で構成されます。

```
PointZ
{
        Double    X        //X 座標
        Double    Y        //Y 座標
        Double    Z        //Z 座標
        Double    M        //measure
}
```

表１２　PointZ のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	11	Integer	1	リトル
Byte 4	X	X	Double	1	リトル
Byte 12	Y	Y	Double	1	リトル
Byte 20	Z	Z	Double	1	リトル
Byte 28	M	M	Double	1	リトル

注：X=44+(4×NumParts)、Y=X+(16×NumPoints)、Z=Y+16+(8×NumPoints)

MultiPointZ

MultiPointZ は PointZ の集合です。

```
MultiPointZ
{
        Double[4]              Box           //バウンディング・ボックス
        Integer                NumPoints     //Point の個数
        Point[NumPoints]       Points        //集合に含まれる Point
        Double[2]              Z Range       //Z 値の範囲
        Double[NumPoints]      Z Array       //各 Point の Z 値
        Double[2]              M Range       //measure の値の範囲
        Double[NumPoints]      M Array       //各 Point の measure の値
}
```

バウンディング・ボックスは Xmin、Ymin、Xmax、Ymax の順に値を格納しています。
Z 値の範囲は Zmin、Zmax の順、 measure の値の範囲は Mmin、Mmax の順に値を格納しています。

表１３　MultiPointZ のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	18	Integer	1	リトル
Byte 4	ボックス	Box	Double	4	リトル
Byte 36	NumPoints	NumPoints	Integer	1	リトル
Byte 40	Points	Points	Point	NumPoints	リトル
Byte X	Zmin	Zmin	Double	1	リトル
Byte X+8	Zmax	Zmax	Double	1	リトル
Byte X+16	Zarray	Zarray	Double	NumPoints	リトル
Byte Y*	Mmin	Mmin	Double	1	リトル
Byte Y+8*	Mmax	Mmax	Double	1	リトル
Byte Y+16*	Marray	Marray	Double	NumPoints	リトル

注：X=40+(16×NumPoints)、Y=X+16+(8×NumPoints)
*このフィールドは存在しない場合がある

PolyLineZ

PolyLineZ は 1 個または複数のパートで構成されます。パートは 2 個以上の Point を順に結んだもので、パート同士が接続していることもあれば離れていることもあります。また、パート同士が交差することもあります。

```
PolyLineZ
{
        Double[4]              Box          //バウンディング・ボックス
        Integer                NumParts     //パートの個数
        Integer                NumPoints    //Point の総数
        Integer[NumParts]      Parts        //パートの最初のPoint へのインデックス
        Point[NumPoints]       Points       //すべてのパートに関する Point
        Double[2]              Z Range      //Z 値の範囲
        Double[NumPoints]      Z Array      //各 Point の Z 値
        Double[2]              M Range      //measure の値の範囲
        Double[NumPoints]      M Array      //各 Point の measure の値
}
```

各フィールドの詳細は以下の通りです。

Box：	Xmin、Ymin、Xmax、Ymax の順で値を格納したバウンディング・ボックス。
NumParts：	PolyLineZ に含まれるパートの個数。
NumPoints：	すべてのパートを合計した Point の総数。
Parts：	長さ NumParts の配列。配列 Point 中の、各パートの最初の Point へのインデックスを格納する。配列の添字は 0 から始まる。
Points：	長さ NumPoints の配列。PolyLineZ を構成する Points をすべて格納する。パート 1 の Point につづいてパート 2、パート 3…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。
Z Range：	Zmin、Zmax の順で値を格納した Z 値の最小値および最大値。
Z Array ：	長さ NumPoints の Z 値の配列。パート 1 の Point につづいてパート 2、パート 3…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。
M Range：	Mmin、Mmax の順で値を格納した measure の最小値および最大値。
M Array ：	長さ NumPoints の measure の配列。パート 1 の Point につづいてパート 2、パート 3…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。

表１４　PolyLineZ のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	13	Integer	1	リトル
Byte 4	ボックス	Box	Double	4	リトル
Byte 36	NumParts	NumParts	Integer	1	リトル
Byte 40	NumPoints	NumPoints	Integer	1	リトル
Byte 44	Parts	Parts	Integer	NumParts	リトル
Byte X	Points	Points	Point	NumPoints	リトル
Byte Y	Zmin	Zmin	Double	1	リトル
Byte Y+8	Zmax	Zmax	Double	1	リトル
Byte Y+16	Zarray	Zarray	Double	NumPoints	リトル
Byte Z*	Mmin	Mmin	Double	1	リトル
Byte Z+8*	Mmax	Mmax	Double	1	リトル
Byte Z+16*	Marray	Marray	Double	NumPoints	リトル

注：X=44+(4×NumParts)、Y=X+(16×NumPoints)、Z=Y+16+(8×NumPoints)
*このフィールドは存在しない場合がある

PolygonZ

PolygonZ は 1 個または複数のリングで構成されます。ここでリングとは、自分自身に交差しない閉曲線のことを指します。PolygonZ ではいくつかのリングが入れ子になっていることがあります。リングのことを PolygonZ のパートと呼びます。

PolygonZ の構造は PolyLineZ と同じで、以下のようになります。

```
PolygonZ
{
        Double[4]              Box          //バウンディング・ボックス
        Integer                NumParts     //パートの個数
        Integer                NumPoints    //Point の総数
        Integer[NumParts]      Parts        //パートの最初の Point へのインデックス
        Point[NumPoints]       Points       //すべてのパートに関する Point
        Double[2]              Z Range      //Z 値の範囲
        Double[NumPoints]      Z Array      //各 Point の Z 値
        Double[2]              M Range      //measure の値の範囲
        Double[NumPoints]      M Array      //各 Point の measure の値
}
```

各フィールドの詳細は以下の通りです。

Box：	Xmin、Ymin、Xmax、Ymax の順で値を格納したバウンディング・ボックス。
NumParts：	PolygonZ に含まれるパートの個数。
NumPoints：	すべてのパートを合計した Point の総数。
Parts：	長さ NumParts の配列。配列 Point 中の、各パートの最初の Point へのインデックスを格納する。配列の添字は 0 から始まる。
Points：	長さ NumPoints の配列。PolygonZ を構成する Points をすべて格納する。パート 1 の Point につづいてパート２、パート３…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。
Z Range：	Zmin、Zmax の順で値を格納した Z 値の最小値および最大値。
Z Array　：	長さ NumPoints の Z 値の配列。パート 1 の Point につづいてパート２、パート３…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。
M Range：	Mmin、Mmax の順で値を格納した measure の最小値および最大値。
M Array　：	長さ NumPoints の measure の配列。パート 1 の Point につづいて パート２、パート３…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。

PolygonZ シェープについて重要な点を以下に列挙します。

■ リングが閉じていること（始点と終点が同一である）
■配列 Point 中のリングの順序はポリゴンの構造に影響しない

表１５　PolygonZ のレコード・コンテンツ

開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	15	Integer	1	リトル
Byte 4	ボックス	Box	Double	4	リトル
Byte 36	NumParts	NumParts	Integer	1	リトル
Byte 40	NumPoints	NumPoints	Integer	1	リトル
Byte 44	Parts	Parts	Integer	NumParts	リトル
Byte X	Points	Points	Point	NumPoints	リトル
Byte Y	Zmin	Zmin	Double	1	リトル
Byte Y+8	Zmax	Zmax	Double	1	リトル
Byte Y+16	Zarray	Zarray	Double	NumPoints	リトル
Byte Z*	Mmin	Mmin	Double	1	リトル
Byte Z+8*	Mmax	Mmax	Double	1	リトル
Byte Z+16*	Marray	Marray	Double	NumPoints	リトル

注：X=44+(4×NumParts)、Y=X+(16×NumPoints)、Z=Y+16+(8×NumPoints)
*このフィールドは存在しない場合がある

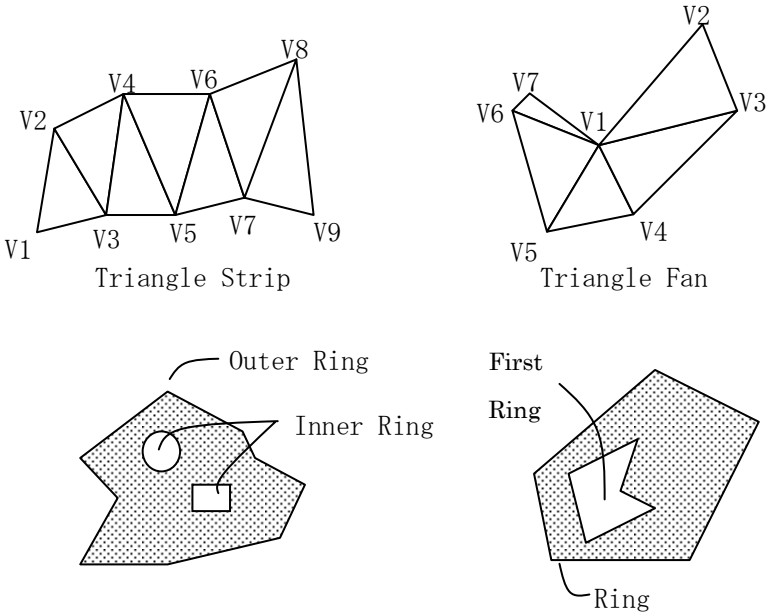
MultiPatch

MultiPatch はパッチ（patch＝面の断片の意）の集まりです。それぞれのパッチがひとつの面をあらわしています。パッチのことを MultiPatch のパートと呼び、パートの型によって、パート中の頂点をどのようにつないで面を構成するかが決まります。MultiPatch のパートの型には以下のものがあります。

トライアングル・ストリップ	三角形をつないで帯状にするもの。3 個め以降の頂点を追加するごとに、直前の 2 頂点と結んで三角形を構成する。
トライアングル・ファン	三角形をつないで扇型にするもの。3 個め以降の頂点を追加するごとに、直前の 1 頂点および最初の頂点と結んで三角形を構成する。
外周リング	ポリゴンの外周を形作るリング。
内部リング	ポリゴン内部の穴をあらわすリング。
開始リング	型の指定がないリングのうち、最初にあらわれるもの。
リング	型の指定がないリング。

トライアングル・ストリップおよびトライアングル・ファンは、それぞれ 1 個のパッチとなります。パートの型の例を図３に示します。

（図３）



リングのパートを組み合わせることで、内部に穴の開いたポリゴンのパッチを構成することができます。この場合は、1 個の外周リングと、それに続いて複数の内部リングを記述します。ポリゴンのパッチを構成する個々のリングの型を指定しないとき、これらのリングの先頭には開始リングを記述してください。開始リングが記述されていないとき、一連のリングは内部に穴をもたない外周リングとして扱われます。

パートの型を指定するコードは以下の通りです。

値	パート型
0	トライアングル・ストリップ
1	トライアングル・ファン
2	外周リング
3	内部リング
4	開始リング
5	リング

```
MultiPatch
{
        Double[4]               Box             //バウンディング・ボックス
        Integer                 NumParts        //パートの個数
        Integer                 NumPoints       //Point の総数
        Integer[NumParts]       Parts           //パートの最初のPoint へのインデックス
        Integer[NumParts]       PartTypes       //パート型
        Point[NumPoints]        Points          //すべてのパートに関する Point
        Double[2]               Z Range         //Z 値の範囲
        Double[NumPoints]       Z Array         //各 Point の Z 値
        Double[2]               M Range         //measure の値の範囲
        Double[NumPoints]       M Array         //各 Point の measure の値
}
```

各フィールドの詳細は以下の通りです。

Box：	Xmin、Ymin、Xmax、Ymax の順で値を格納したバウンディング・ボックス。
NumParts：	MultiPatch に含まれるパートの個数。
NumPoints：	すべてのパートを合計した Point の総数。
Parts：	長さ NumParts の配列。配列 Point 中の、各パートの最初の Point へのインデックスを格納する。配列の添字は 0 から始まる。
PartTypes：	長さ NumParts の配列。各パートごとに、パートの型を格納する。
Points：	長さ NumPoints の配列。MultiPatch を構成する Points をすべて格納する。パート 1 の Point につづいてパート 2、パート 3…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。
Z Range：	Zmin、Zmax の順で値を格納した Z 値の最小値および最大値。
Z Array：	長さ NumPoints の Z 値の配列。パート 1 の Point につづいてパート 2、パート 3…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。
M Range：	Mmin、Mmax の順で値を格納した measure の最小値および最大値。
M Array　：	長さ NumPoints の measure の配列。パート 1 の Point につづいてパート 2、パート 3…の順になる。配列 Parts には、各パートの最初の Point を格納している Points 要素の添字の値を持っている。パートとパートの間にデリミタはない。

MultiPatch シェープについて重要な点を以下に列挙します。

■ リングが閉じていること（始点と終点が同一である）。

■ リングの順序に注意。内部リングは外周リングの後に、ひとつのパッチを構成するリングは開始リングの後に記述する。

■パート同士が境界を共有することは許されるが、オーバーラップや交差は許されない。

表１６　MultiPatch のレコード・コンテンツ

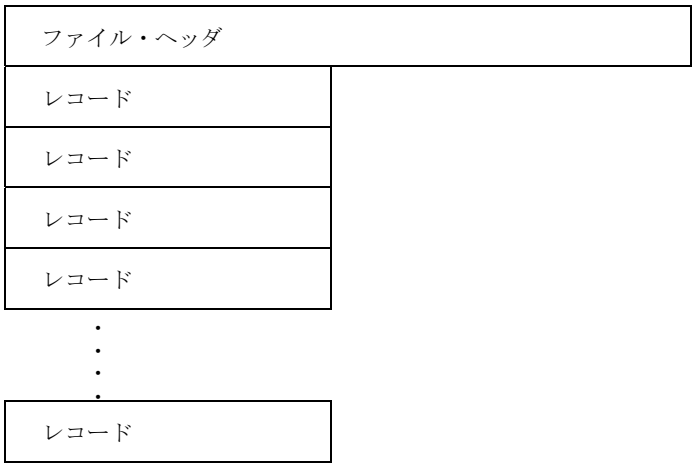
開始位置	フィールド	値	型	個数	バイト順
Byte 0	シェープ・タイプ	31	Integer	1	リトル
Byte 4	ボックス	Box	Double	4	リトル
Byte 36	NumParts	NumParts	Integer	1	リトル
Byte 40	NumPoints	NumPoints	Integer	1	リトル
Byte 44	Parts	Parts	Integer	NumParts	リトル
Byte W	PartTypes	PartTypes	Integer	NumParts	リトル
Byte X	Points	Points	Point	NumPoints	リトル
Byte Y	Zmin	Zmin	Double	1	リトル
Byte Y+8	Zmax	Zmax	Double	1	リトル
Byte Y+16	Zarray	Zarray	Double	NumPoints	リトル
Byte Z*	Mmin	Mmin	Double	1	リトル
Byte Z+8*	Mmax	Mmax	Double	1	リトル
Byte Z+16*	Marray	Marray	Double	NumPoints	リトル

注：W=44+(4×NumParts)、X=W+(4×NumParts)、Y=X+(16×NumPoints)、Z=Y+16+(8×NumPoints)
*このフィールドは存在しない場合がある

インデックス・ファイルの構造

インデックス・ファイル（*.shx）は 100 バイトのヘッダと 8 バイト固定長のレコードで構成されます。図 4 にインデックス・ファイルの構成を示します。

図４：インデックス・ファイルの構成



インデックス・ファイル・ヘッダ

インデックス・ファイルのヘッダは、先に述べたメイン・ファイルのヘッダと全く同じです。ファイル長は、ヘッダを含む全ファイルの長さを 16 ビット＝1 ワードを単位として記述します（ヘッダの 50 ワード＋4 ×レコード数)。

インデックス・レコード

インデックス・ファイルのｎ番目のレコードには、メイン・ファイルのｎ番目のレコードへのオフセットと、そのレコードのコンテンツ長が格納されています。表１７にインデックス・レコードのフィールドと開始位置、値、型、バイト順を示します。開始位置はレコードの先頭からのバイト数であらわしています。

表１７　インデックス・レコードの構成

開始位置	フィールド	値	型	バイト順
Byte 0	オフセット	オフセット値	Integer	ビッグ
Byte 4	コンテンツ長	コンテンツ長	Integer	ビッグ

ここでオフセット値は、対応するレコードのレコード・ヘッダ第 1 バイトの位置を、メイン・ファイル先頭からのワード数であらわしたものです。100 バイトのヘッダがあるので、最初のレコードのオフセットは 50 になります。

コンテンツ長の値は、メイン・ファイルのレコード・ヘッダに格納されている各レコードのコンテンツ長と同じです。

属性ファイルの構造

属性ファイル（*.dbf）は任意の属性または他のテーブルを結合するためのキーを格納しています。データ・フォーマットは多くの表計算アプリケーションで使われている dBASE 形式です。フィールドは任意に定義することができます。属性ファイルには次の条件があります。

■　ファイル名のプレフィクスはメイン・ファイル及びインデックス・ファイルと同一。サフィックス（拡張子）は「.dbf」でなければならない。(2 ページのファイル命名規則を参照)

■　シェープの 1 フィーチャごとに 1 レコードの属性をもつ。

■レコードの順序は、メイン・ファイルのフィーチャのレコード順に一致。

dBASE 形式の詳細は Borland 社のウェブ・サイト（www.borland.com）を参照してください。

（訳注）シェープファイルでは dBASE IV のフォーマットを使っています。ESRI ホワイトペーパーの原文にはありませんが、以下に dBASE IV フォーマットの仕様を記載します。なお、シェープファイルではメモフィールドはサポートしていません。

dBASE IV 2.0 テーブルのファイル・ヘッダ

ファイル構造

バイト	内容	意味
0	1byte	Valid dBASE IV file; 第 0-2 ビット バージョン番号、第 3 ビット dBASE IV メモファイルの有無、第 4-6 ビット SQL テーブルの有無、第 7 ビット dBASE III PLUS または dBASE IV メモファイルの有無
1-3	3 bytes	最終更新日; YYMMDD 形式
4-7	32-bit number	レコード数
8-9	16-bit number	ヘッダのバイト数
10-11	16-bit number	レコードのバイト数
12-13	2 bytes	予約領域（0 で埋める）
14	1 byte	トランザクション未了フラグ
15	1 byte	暗号化フラグ
16-27	12 bytes	予約領域
28	1 byte	MDX ファイル・フラグ 01H は MDX あり、00H は MDX なし
29	1 byte	言語ドライバ ID
30-31	2 bytes	予約領域（0 で埋める）
32-n*	32 bytes each	フィールド記述子配列(下記参照)
n + 1	1 byte	フィールドの終わりを示す符号（0DH）

* n はフィールド記述子配列の最終バイト。配列の数はフィールドの個数で決まる。

フィールド記述子配列

バイト	内容	意味
0-10	11 bytes※	フィールド名（ASCII）
11	1 byte	フィールド型（C, D, F, L, M, N）
12-15	4 bytes	予約領域
16	1 byte	フィールド長（バイナリ）
17	1 byte	小数部の長さ（バイナリ）
18-19	2 bytes	予約領域
20	1 byte	作業領域 ID
21-30	10 bytes	予約領域
31	1 byte	MDX フィールド・フラグ フィールドが MDX ファイルにインデックス・タグを持つとき 01H、そうでないとき 00H

※フィールド名には区切り文字としてヌル文字(0)を使用するため、実質 10 バイトの制限になります。

データベース・レコード

レコードはデータベース・ファイル中でヘッダの後に置かれる。データ・レコードの先頭の 1 バイトが半角空白（20H）のとき、このレコードは削除されていないことをあらわす。アスタリスク（2AH）のとき、 このレコードは削除されていることをあらわす。レコードにはフィールドの区切りやレコードの末尾を示す符号は含まない。ファイルの末尾には 1 バイトの end-of-file 符号 (ASCII 26、1AH) がある。

入力可能なデータ型

コード	型	データ
C	(Character)	任意の OEM コードページ文字
D	(Date)	年月日をあらわす数字および文字 （内部表現は 8 桁の YYYYMMDD 形式）
F	(Floating point binary numeric)	- . 0 1 2 3 4 5 6 7 8 9
N	(Binary coded decimal numeric)	- . 0 1 2 3 4 5 6 7 8 9
L	(Logical)	? Y y N n T t F f（初期値設定しない場合は?になる）
M	(Memo)	任意の OEM コードページ文字(内部表現は 10 桁の.DBT ブロック番号)

メモフィールドと .DBT ファイル

メモフィールドのデータは.DBT ファイルに格納される。.DBT ファイルはブロックごとに連番（0、1、2、･･･）が付けられる。SET BLOCKSIZE によりブロックの長さを指定する。.DBT ファイルの先頭ブロック（ブロック 0）はファイル・ヘッダである。

.DBF ファイルのメモフィールドには、実際のデータが始まるブロック番号が格納される。
メモフィールドにデータを含まないときは、ブロック番号でなく半角空白（20h）を格納する。

メモフィールドのデータが変更された場合、参照するブロックが変わることがある。
それに合わせて.DBF ファイル中のブロック番号も変更しなければならない。

（出典：dBASE IV Language Reference manual, Appendix D、邦訳パスコ）

用語集

このドキュメントのキーワードを理解するための用語集です。

ArcInfo
（アークインフォ）

空間データを操作し、編集するためのツールとして開発された ESRI 社のソフトウェアです。ディジタイズ、データ編集、座標管理、ネットワーク解析、地形モデリング、サーフェス解析など、あらゆる機能を提供します。ArcInfo はエンジニアリング・ワークステーション（EWS）、ミニコン、パーソナル・コンピュータ（PC）など、多様な環境で動作します。業界標準のテクノロジーとクライアント／サーバ・アーキテクチャを採用し、ArcView のための GIS データ・サーバとしても利用されます。

ArcCAD
（アークキャド）

AutoCAD の上で GIS の機能を実現するための ESRI 社のソフトウェアです。データ・マネジメント、空間解析および表示のための広範なツールを提供します。

ARC Macro Language
（アークマクロランゲージ; AML）

ArcInfo のための構造化マクロ言語で、ArcInfo の動作を制御、自動化し、ユーザ・インタフェースを作成することができます。

ArcView
（アークビュー）

高機能で使いやすい ESRI 社のデスクトップ GIS ソフトウェアです。データの表示・検索と空間解析が手軽に実行できます。Windows 環境および多くのエンジニアリング・ワークステーションで動作します。

Avenue
（アベニュー）

ArcView のためのオブジェクト指向型プログラミング言語です。ArcView に新たな機能を追加したり、ArcView をベースとして特定用途向けアプリケーションを開発することができます。

ビッグ・エンディアン
（～のバイト順）

左から右へ並べるバイト順の方式です。Sun、HP、IBM、Data General、AViiON など、多くの UNIX システムで採用されています。

Bounding Box
（バウンディング・ボックス）

1 個のシェープ（ポリラインなど）を囲む最小の長方形。Xmin、Ymin、Xmax、Ymax で定義します。

BusinessMAP
（ビジネスマップ）

Windows 環境で動作するデータベース・マッピング・ソフトウェアです。データベースの情報を 2 次元および 3 次元の図面に表現します。

カバレッジ
（coverage）

ArcInfo のためのディジタル地図データの形式のひとつ。アーク、ノード、ポリゴン、ラベルポイントなどで構成されるベクトル型のデータで地物を抽象化してモデリングします。図郭の TIC、マップ・エクステント、リンク、注記といった複合情報も保存します。地物の属性情報を属性テーブルに保存します。

Data Automation Kit (データオートメーションキット; DAK)	高品質のディジタイズ、データ編集、トポロジー生成、データ変換、投影変換の機能を提供する ESRI 社のソフトウェア。
フィーチャ (feature)	現実世界の地物を、空間的な特徴をあらわす「シェープ」と属性の組によって抽象化したデータ要素。
インデックス・ファイル (index file)	インデックス・ファイルを利用することによりメイン・ファイル中のデータ・レコードに直接アクセスできます。
リトル・エンディアン (～のバイト順)	右から左へ並べるバイト順の方式です。DEC OSF/1, DEC OpenVMS, MS-DOS, Windows NT など、多くのオペレーション・システムで採用されています。
MapObjects (マップオブジェクツ)	ActiveX コントロール（OCX）およびプログラム可能な ActiveX オートメーション・オブジェクトとして提供される、マッピングと GIS のためのコンポーネントです。多くの標準的な Windows 開発環境で利用でき、アプリケーション・プログラムにマッピングの機能を追加します。
マルチポイント (MultiPoint)	ポイントの集合体として表現される 1 個のフィーチャで、属性ファイルの 1 レコードに対応します。各々のポイントは地理的フィーチャを表わします。
NumPoints	シェープに含まれる頂点の個数。
PC ARC/INFO (ピーシーアークインフォ)	パーソナル・コンピュータで高機能の GIS を実現した ESRI 社のソフトウェアです。ArcInfo と同様に、地図制作や地理的情報の管理と分析に関係する世界中の企業や組織で使われています。地物の特徴を記述した属性情報は dBASE 形式で保存します。
ポリライン (PolyLine)	x,y 座標で表わされる頂点を順に結んで線や境界を表現したもの。
リング	x,y 座標で表わされる頂点を順に結んだもので、最初の点と最後の点が一致するために閉曲線を形作るもの。閉じたポリラインやポリゴンと同等。
シェープファイル (Shapefile)	ArcView 用のデータセットで、道路、病院の位置、商圏、郵便番号区地域コードなどといった地理的データを表現します。ポイント、ライン、エリアのデータモデルをもち、1 個の地理的フィーチャに 1 レコードが対応します。

Simple Macro Language（シンプルマクロランゲージ; SML）

PC ARC/INFO 用のマクロ言語です。コマンドの組み合わせで構成するシンプルなプログラミング言語で、検索式の評価、入出力の制御、プログラムの流れ制御などのマクロを作成します。

テーマ（theme）

ユーザ定義の地理データセット。ArcView のテーマにはカバレッジ、GRID、イメージ、シェープファイルなど多様なデータソースが利用可能です。データソース名、注目している属性、データの分類方法、表示設定などのプロパティを設定できます。

トポロジー（topology）

アーク、ノード、ポリゴン、ポイントなど、接続または隣接するカバレッジのフィーチャ間の位置関係。例えばアークには始点ノードと終点ノードがあり、右側ポリゴンと左側ポリゴンがあります。単純な要素のトポロジーから複雑な要素が作られます。最も単純なポイントと、ポイントを接続したアークの組み合わせによって、さらに高次の要素であるエリアが定義されます。シェープファイルには、トポロジー構造を明示するレコードはありません。

カバレッジでは、地理的なフィーチャをトポロジーをもつ線グラフとして表現します。GIS の操作で、頂点の座標値を必ずしも必要としない場合にトポロジーを有効に利用することができます。例えば 2 地点間を移動する最適な経路を判定するには、互いに接続した頂点と、その間を移動するためのコストをリストアップすることが必要になります。頂点の座標値は、解析結果を画面に表示するときまで参照されることはありません。

ベクトル（vector）

x,y 座標にもとづいて作成したデータで、地理的フィーチャを扱うのに一般によく使われるデータ形式です。フィーチャは x,y 座標によって表わされる頂点で構成されます。フィーチャには属性情報が対応します。その他の重要なデータ形式としてはラスタ型（セルに属性が対応する）とサーフェスを表現するための TIN（Triangulated Irregular Networks）があります。

頂点（vertex）

ラインを構成する点のひとつで、x,y 座標で定義されます。

お問い合わせ先

ESRI ジャパン株式会社
URL: http://www.esrij.com
E-mail: gisinfo@esrij.com

2001.2 作成(2006.7 修正)