
# ESRI Shapefile Technical Description

***An ESRI White Paper—July 1998***

# ESRI Shapefile Technical Description

## An ESRI White Paper

**Contents** **Page**

# ESRI Shapefile Technical Description

This document defines the shapefile (.shp) spatial data format and describes why shapefiles are important. It lists the tools available in Environmental Systems Research Institute, Inc. (ESRI), software for creating shapefiles directly or converting data into shapefiles from other formats. This document also provides all the technical information necessary for writing a computer program to create shapefiles without the use of ESRI® software for organizations that want to write their own data translators.

## Why Shapefiles?

A shapefile stores nontopological geometry and attribute information for the spatial features in a data set. The geometry for a feature is stored as a shape comprising a set of vector coordinates.

Because shapefiles do not have the processing overhead of a topological data structure, they have advantages over other data sources such as faster drawing speed and edit ability. Shapefiles handle single features that overlap or that are noncontiguous. They also typically require less disk space and are easier to read and write.

Shapefiles can support point, line, and area features. Area features are represented as closed loop, double-digitized polygons. Attributes are held in a dBASE® format file. Each attribute record has a one-to-one relationship with the associated shape record.

## How Shapefiles Can Be Created

Shapefiles can be created with the following four general methods:

- Export—Shapefiles can be created by exporting any data source to a shapefile using ARC/INFO®, PC ARC/INFO®, Spatial Database Engine™ (SDE™), ArcView® GIS, or *Business*MAP™ software.
- Digitize—Shapefiles can be created directly by digitizing shapes using ArcView GIS feature creation tools.
- Programming—Using Avenue™ (ArcView GIS), MapObjects™, ARC Macro Language (AML™) (ARC/INFO), or Simple Macro Language (SML™) (PC ARC/INFO) software, you can create shapefiles within your programs.
- Write directly to the shapefile specifications by creating a program.

SDE, ARC/INFO, PC ARC/INFO, Data Automation Kit (DAK™), and ArcCAD® software provide shape-to-coverage data translators, and ARC/INFO also provides a coverage-to-shape translator. For exchange with other data formats, the shapefile specifications are published in this paper. Other data streams, such as those from global positioning system (GPS) receivers, can also be stored as shapefiles or X,Y event tables.

## Shapefile Technical Description

Computer programs can be created to read or write shapefiles using the technical specification in this section.

An ESRI shapefile consists of a main file, an index file, and a dBASE table. The main file is a direct access, variable-record-length file in which each record describes a shape with a list of its vertices. In the index file, each record contains the offset of the corresponding main file record from the beginning of the main file. The dBASE table contains feature attributes with one record per feature. The one-to-one relationship between geometry and attributes is based on record number. Attribute records in the dBASE file must be in the same order as records in the main file.

### Naming Conventions

All file names adhere to the 8.3 naming convention. The main file, the index file, and the dBASE file have the same prefix. The prefix must start with an alphanumeric character (a–Z, 0–9), followed by zero or up to seven characters (a–Z, 0–9, _, -). The suffix for the main file is .shp. The suffix for the index file is .shx. The suffix for the dBASE table is .dbf. All letters in a file name are in lower case on operating systems with case sensitive file names.

*Examples*

- Main file: counties.shp
- Index file: counties.shx
- dBASE table: counties.dbf

### Numeric Types

A shapefile stores integer and double-precision numbers. The remainder of this document will refer to the following types:

- Integer: Signed 32-bit integer (4 bytes)
- Double: Signed 64-bit IEEE double-precision floating point number (8 bytes)

Floating point numbers must be numeric values. Positive infinity, negative infinity, and Not-a-Number (NaN) values are not allowed in shapefiles. Nevertheless, shapefiles support the concept of "no data" values, but they are currently used only for measures. Any floating point number smaller than $-10^{38}$ is considered by a shapefile reader to represent a "no data" value.

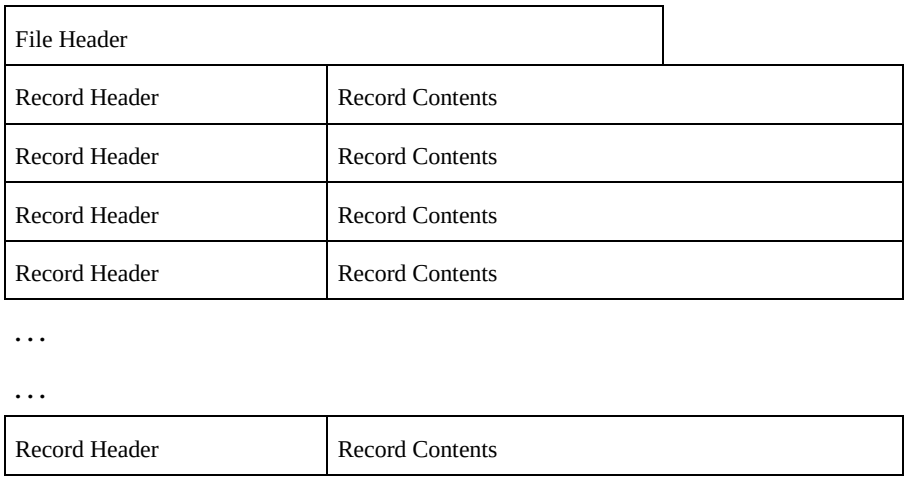
The first section below describes the general structure and organization of the shapefile. The second section describes the record contents for each type of shape supported in the shapefile.

## Organization of the Main File

The main file (.shp) contains a fixed-length file header followed by variable-length records. Each variable-length record is made up of a fixed-length record header followed by variable-length record contents. **Figure 1** illustrates the main file organization.

J-7855

**Figure 1**
**Organization of the Main File**




## Byte Order

All the contents in a shapefile can be divided into two categories:

- Data related
  - Main file record contents
  - Main file header's data description fields (Shape Type, Bounding Box, etc.)
- File management related
  - File and record lengths
  - Record offsets, and so on

The integers and double-precision integers that make up the data description fields in the file header (identified below) and record contents in the main file are in little endian (PC or Intel®) byte order. The integers and double-precision floating point numbers that make up the rest of the file and file management are in big endian (Sun® or Motorola®) byte order.

## The Main File Header

The main file header is 100 bytes long. **Table 1** shows the fields in the file header with their byte position, value, type, and byte order. In the table, position is with respect to the start of the file.

**Table 1**
**Description of the Main File Header**

| Position | Field | Value | Type | Byte Order |
|---|---|---|---|---|
| Byte 0 | File Code | 9994 | Integer | Big |
| Byte 4 | Unused | 0 | Integer | Big |
| Byte 8 | Unused | 0 | Integer | Big |
| Byte 12 | Unused | 0 | Integer | Big |
| Byte 16 | Unused | 0 | Integer | Big |
| Byte 20 | Unused | 0 | Integer | Big |
| Byte 24 | File Length | File Length | Integer | Big |
| Byte 28 | Version | 1000 | Integer | Little |
| Byte 32 | Shape Type | Shape Type | Integer | Little |
| Byte 36 | Bounding Box | Xmin | Double | Little |
| Byte 44 | Bounding Box | Ymin | Double | Little |
| Byte 52 | Bounding Box | Xmax | Double | Little |
| Byte 60 | Bounding Box | Ymax | Double | Little |
| Byte 68* | Bounding Box | Zmin | Double | Little |
| Byte 76* | Bounding Box | Zmax | Double | Little |
| Byte 84* | Bounding Box | Mmin | Double | Little |
| Byte 92* | Bounding Box | Mmax | Double | Little |

* Unused, with value 0.0, if not Measured or Z type

The value for file length is the total length of the file in 16-bit words (including the fifty 16-bit words that make up the header).

All the non-Null shapes in a shapefile are required to be of the same shape type. The values for shape type are as follows:

| Value | Shape Type |
|---|---|
| 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 |

Shape types not specified above (2, 4, 6, etc., and up to 33) are reserved for future use. Currently, shapefiles are restricted to contain the same type of shape as specified above. In the future, shapefiles may be allowed to contain more than one shape type. If mixed shape types are implemented, the shape type field in the header will flag the file as such.

The Bounding Box in the main file header stores the actual extent of the shapes in the file: the minimum bounding rectangle orthogonal to the X and Y (and potentially the M and Z) axes that contains all shapes. If the shapefile is empty (that is, has no records), the values for Xmin, Ymin, Xmax, and Ymax are unspecified. Mmin and Mmax can contain "no data" values (see Numeric Types on page 2) for shapefiles of measured shape types that contain no measures.

### Record Headers

The header for each record stores the record number and content length for the record. Record headers have a fixed length of 8 bytes. **Table 2** shows the fields in the file header with their byte position, value, type, and byte order. In the table, position is with respect to the start of the record.

**Table 2**
**Description of Main File Record Headers**

| Position | Field | Value | Type | Byte Order |
|---|---|---|---|---|
| Byte 0 | Record Number | Record Number | Integer | Big |
| Byte 4 | Content Length | Content Length | Integer | Big |

Record numbers begin at 1.

The content length for a record is the length of the record contents section measured in 16-bit words. Each record, therefore, contributes (4 + content length) 16-bit words toward the total length of the file, as stored at Byte 24 in the file header.

## Main File Record Contents

Shapefile record contents consist of a shape type followed by the geometric data for the shape. The length of the record contents depends on the number of parts and vertices in a shape. For each shape type, we first describe the shape and then its mapping to record contents on disk. In **Tables 3** through **16,** position is with respect to the start of the record contents.

### Null Shapes

A shape type of 0 indicates a **null** shape, with no geometric data for the shape. Each feature type (point, line, polygon, etc.) supports nulls—it is valid to have points and null points in the same shapefile. Often null shapes are place holders; they are used during shapefile creation and are populated with geometric data soon after they are created.

**Table 3**
**Null Shape Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 0 | Integer | 1 | Little |

## *Shape Types in X,Y Space*

### Point

A point consists of a pair of double-precision coordinates in the order X,Y.

```
Point
{
    Double      X       // X coordinate
    Double      Y       // Y coordinate
}
```

**Table 4**
**Point Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 1 | Integer | 1 | Little |
| Byte 4 | X | X | Double | 1 | Little |
| Byte 12 | Y | Y | Double | 1 | Little |

### MultiPoint

A MultiPoint represents a set of points, as follows:

```
MultiPoint
{
    Double[4]           Box         // Bounding Box
    Integer             NumPoints   // Number of Points
    Point[NumPoints]    Points      // The Points in the Set
}
```

The Bounding Box is stored in the order Xmin, Ymin, Xmax, Ymax.

**Table 5**
**MultiPoint Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 8 | Integer | 1 | Little |
| Byte 4 | Box | Box | Double | 4 | Little |
| Byte 36 | NumPoints | NumPoints | Integer | 1 | Little |
| Byte 40 | Points | Points | Point | NumPoints | Little |

## PolyLine

A PolyLine is an ordered set of vertices that consists of one or more parts. A part is a connected sequence of two or more points. Parts may or may not be connected to one another. Parts may or may not intersect one another.

Because this specification does not forbid consecutive points with identical coordinates, shapefile readers must handle such cases. On the other hand, the degenerate, zero length parts that might result are not allowed.

```
PolyLine
{
    Double[4]          Box          // Bounding Box
    Integer            NumParts     // Number of Parts
    Integer            NumPoints    // Total Number of Points
    Integer[NumParts]  Parts        // Index to First Point in Part
    Point[NumPoints]   Points       // Points for All Parts
}
```

The fields for a PolyLine are described in detail below:

Box — The Bounding Box for the PolyLine stored in the order Xmin, Ymin, Xmax, Ymax.

NumParts — The number of parts in the PolyLine.

NumPoints — The total number of points for all parts.

Parts — An array of length NumParts. Stores, for each PolyLine, the index of its first point in the points array. Array indexes are with respect to 0.

Points — An array of length NumPoints. The points for each part in the PolyLine are stored end to end. The points for Part 2 follow the points for Part 1, and so on. The parts array holds the array index of the starting point for each part. There is no delimiter in the points array between parts.

**Table 6**
**PolyLine Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 3 | Integer | 1 | Little |
| Byte 4 | Box | Box | Double | 4 | Little |
| Byte 36 | NumParts | NumParts | Integer | 1 | Little |
| Byte 40 | NumPoints | NumPoints | Integer | 1 | Little |
| Byte 44 | Parts | Parts | Integer | NumParts | Little |
| Byte X | Points | Points | Point | NumPoints | Little |

Note: $X = 44 + 4 * NumParts$

## Polygon

A polygon consists of one or more rings. A ring is a connected sequence of four or more points that form a closed, non-self-intersecting loop. A polygon may contain multiple outer rings. The order of vertices or orientation for a ring indicates which side of the ring is the interior of the polygon. The neighborhood to the right of an observer walking along the ring in vertex order is the neighborhood inside the polygon. Vertices of rings defining holes in polygons are in a counterclockwise direction. Vertices for a single, ringed polygon are, therefore, always in clockwise order. The rings of a polygon are referred to as its parts.

Because this specification does not forbid consecutive points with identical coordinates, shapefile readers must handle such cases. On the other hand, the degenerate, zero length or zero area parts that might result are not allowed.

The Polygon structure is identical to the PolyLine structure, as follows:

```
Polygon
{
    Double[4]          Box         // Bounding Box
    Integer            NumParts    // Number of Parts
    Integer            NumPoints   // Total Number of Points
    Integer[NumParts]  Parts       // Index to First Point in Part
    Point[NumPoints]   Points      // Points for All Parts
}
```

The fields for a polygon are described in detail below:

Box — The Bounding Box for the polygon stored in the order Xmin, Ymin, Xmax, Ymax.

NumParts — The number of rings in the polygon.

NumPoints — The total number of points for all rings.

Parts — An array of length NumParts. Stores, for each ring, the index of its first point in the points array. Array indexes are with respect to 0.

Points — An array of length NumPoints. The points for each ring in the polygon are stored end to end. The points for Ring 2 follow the points for Ring 1, and so on. The parts array holds the array index of the starting point for each ring. There is no delimiter in the points array between rings.

The instance diagram in **Figure 2** illustrates the representation of polygons. This figure shows a polygon with one hole and a total of eight vertices.

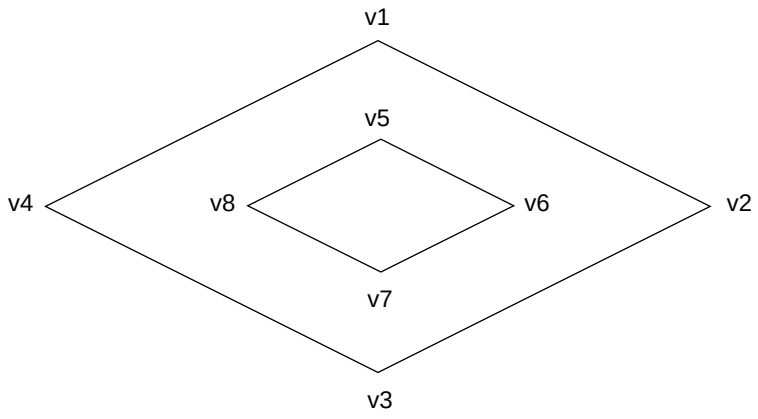
The following are important notes about Polygon shapes.

- The rings are closed (the first and last vertex of a ring MUST be the same).
- The order of rings in the points array is not significant.
- Polygons stored in a shapefile must be clean. A clean polygon is one that

1. Has no self-intersections. This means that a segment belonging to one ring may not intersect a segment belonging to another ring. The rings of a polygon can touch each other at vertices but not along segments. Colinear segments are considered intersecting.

2. Has the inside of the polygon on the "correct" side of the line that defines it. The neighborhood to the right of an observer walking along the ring in vertex order is the inside of the polygon. Vertices for a single, ringed polygon are, therefore, always in clockwise order. Rings defining holes in these polygons have a counterclockwise orientation. "Dirty" polygons occur when the rings that define holes in the polygon also go clockwise, which causes overlapping interiors.

**Figure 2**
**An Example Polygon Instance**



For this example, NumParts equals 2 and NumPoints equals 10. Note that the order of the points for the donut (hole) polygon are reversed below.

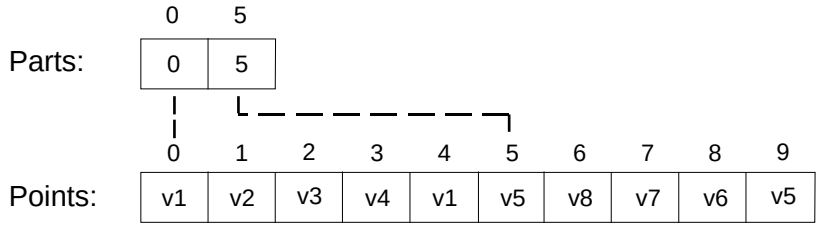



**Table 7**
**Polygon Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 5 | Integer | 1 | Little |
| Byte 4 | Box | Box | Double | 4 | Little |
| Byte 36 | NumParts | NumParts | Integer | 1 | Little |
| Byte 40 | NumPoints | NumPoints | Integer | 1 | Little |
| Byte 44 | Parts | Parts | Integer | NumParts | Little |
| Byte X | Points | Points | Point | NumPoints | Little |

Note: X = 44 + 4 ∗ NumParts

## *Measured Shape Types in X,Y Space*

Shapes of this type have an additional coordinate—M. Note that "no data" value can be specified as a value for M (see Numeric Types on page 2).

### PointM

A PointM consists of a pair of double-precision coordinates in the order X, Y, plus a measure M.

```
PointM
{
    Double      X       // X coordinate
    Double      Y       // Y coordinate
    Double      M       // Measure
}
```

**Table 8**
**PointM Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 21 | Integer | 1 | Little |
| Byte 4 | X | X | Double | 1 | Little |
| Byte 12 | Y | Y | Double | 1 | Little |
| Byte 20 | M | M | Double | 1 | Little |

## MultiPointM

A MultiPointM represents a set of PointMs, as follows

```
MultiPointM
{
    Double[4]          Box          // Bounding Box
    Integer            NumPoints    // Number of Points
    Point[NumPoints]   Points       // The Points in the Set
    Double[2]          M Range      // Bounding Measure Range
    Double[NumPoints]  M Array      // Measures
}
```

The fields for a MultiPointM are

Box — The Bounding Box for the MultiPointM stored in the order Xmin, Ymin, Xmax, Ymax

NumPoints — The number of Points

Points — An array of Points of length NumPoints

M Range — The minimum and maximum measures for the MultiPointM stored in the order Mmin, Mmax

M Array — An array of measures of length NumPoints

**Table 9**
**MultiPointM Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 28 | Integer | 1 | Little |
| Byte 4 | Box | Box | Double | 4 | Little |
| Byte 36 | NumPoints | NumPoints | Integer | 1 | Little |
| Byte 40 | Points | Points | Point | NumPoints | Little |
| Byte X* | Mmin | Mmin | Double | 1 | Little |
| Byte X+8* | Mmax | Mmax | Double | 1 | Little |
| Byte X+16* | Marray | Marray | Double | NumPoints | Little |

Note: X = 40 + (16 * NumPoints)
* optional

## PolyLineM

A shapefile PolyLineM consists of one or more parts. A part is a connected sequence of two or more points. Parts may or may not be connected to one another. Parts may or may not intersect one another.

```
PolyLineM
{
    Double[4]          Box          // Bounding Box
    Integer            NumParts     // Number of Parts
    Integer            NumPoints    // Total Number of Points
    Integer[NumParts]  Parts        // Index to First Point in Part
    Point[NumPoints]   Points       // Points for All Parts
    Double[2]          M Range      // Bounding Measure Range
    Double[NumPoints]  M Array      // Measures for All Points
}
```

The fields for a PolyLineM are

Box — The Bounding Box for the PolyLineM stored in the order Xmin, Ymin, Xmax, Ymax.

NumParts — The number of parts in the PolyLineM.

NumPoints — The total number of points for all parts.

Parts — An array of length NumParts. Stores, for each part, the index of its first point in the points array. Array indexes are with respect to 0.

Points — An array of length NumPoints. The points for each part in the PolyLineM are stored end to end. The points for Part 2 follow the points for Part 1, and so on. The parts array holds the array index of the starting point for each part. There is no delimiter in the points array between parts.

M Range — The minimum and maximum measures for the PolyLineM stored in the order Mmin, Mmax.

M Array — An array of length NumPoints. The measures for each part in the PolyLineM are stored end to end. The measures for Part 2 follow the measures for Part 1, and so on. The parts array holds the array index of the starting point for each part. There is no delimiter in the measure array between parts.

**Table 10**
**PolyLineM Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 23 | Integer | 1 | Little |
| Byte 4 | Box | Box | Double | 4 | Little |
| Byte 36 | NumParts | NumParts | Integer | 1 | Little |
| Byte 40 | NumPoints | NumPoints | Integer | 1 | Little |
| Byte 44 | Parts | Parts | Integer | NumParts | Little |
| Byte X | Points | Points | Point | NumPoints | Little |
| Byte Y* | Mmin | Mmin | Double | 1 | Little |
| Byte Y + 8* | Mmax | Mmax | Double | 1 | Little |
| Byte Y + 16* | Marray | Marray | Double | NumPoints | Little |

Note: X = 44 + (4 * NumParts), Y = X + (16 * NumPoints)
* optional

## PolygonM

A PolygonM consists of a number of rings. A ring is a closed, non-self-intersecting loop. Note that intersections are calculated in X,Y space, *not* in X,Y,M space. A PolygonM may contain multiple outer rings. The rings of a PolygonM are referred to as its parts.

The PolygonM structure is identical to the PolyLineM structure, as follows:

```
PolygonM
{
    Double[4]          Box          // Bounding Box
    Integer            NumParts     // Number of Parts
    Integer            NumPoints    // Total Number of Points
    Integer[NumParts]  Parts        // Index to First Point in Part
    Point[NumPoints]   Points       // Points for All Parts
    Double[2]          M Range      // Bounding Measure Range
    Double[NumPoints]  M Array      // Measures for All Points
}
```

The fields for a PolygonM are

| | |
|---|---|
| Box | The Bounding Box for the PolygonM stored in the order Xmin, Ymin, Xmax, Ymax. |
| NumParts | The number of rings in the PolygonM. |
| NumPoints | The total number of points for all rings. |
| Parts | An array of length NumParts. Stores, for each ring, the index of its first point in the points array. Array indexes are with respect to 0. |
| Points | An array of length NumPoints. The points for each ring in the PolygonM are stored end to end. The points for Ring 2 follow the points for Ring 1, and so on. The parts array holds the array index of the starting point for each ring. There is no delimiter in the points array between rings. |
| M Range | The minimum and maximum measures for the PolygonM stored in the order Mmin, Mmax. |
| M Array | An array of length NumPoints. The measures for each ring in the PolygonM are stored end to end. The measures for Ring 2 follow the measures for Ring 1, and so on. The parts array holds the array index of the starting measure for each ring. There is no delimiter in the measure array between rings. |

The following are important notes about PolygonM shapes.

- The rings are closed (the first and last vertex of a ring MUST be the same).
- The order of rings in the points array is not significant.

**Table 11**
**PolygonM Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 25 | Integer | 1 | Little |
| Byte 4 | Box | Box | Double | 4 | Little |
| Byte 36 | NumParts | NumParts | Integer | 1 | Little |
| Byte 40 | NumPoints | NumPoints | Integer | 1 | Little |
| Byte 44 | Parts | Parts | Integer | NumParts | Little |
| Byte X | Points | Points | Point | NumPoints | Little |
| Byte Y* | Mmin | Mmin | Double | 1 | Little |
| Byte Y + 8* | Mmax | Mmax | Double | 1 | Little |
| Byte Y + 16* | Marray | Marray | Double | NumPoints | Little |

Note: X = 44 + (4 * NumParts), Y = X + (16 * NumPoints)
* optional

### *Shape Types in X,Y,Z Space*

Shapes of this type have an optional coordinate—M. Note that "no data" value can be specified as a value for M (see Numeric Types on page 2).

### PointZ

A PointZ consists of a triplet of double-precision coordinates in the order X, Y, Z plus a measure.

```
PointZ
{
    Double    X    // X coordinate
    Double    Y    // Y coordinate
    Double    Z    // Z coordinate
    Double    M    // Measure
}
```

**Table 12**
**PointZ Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 11 | Integer | 1 | Little |
| Byte 4 | X | X | Double | 1 | Little |
| Byte 12 | Y | Y | Double | 1 | Little |
| Byte 20 | Z | Z | Double | 1 | Little |
| Byte 28 | Measure | M | Double | 1 | Little |

## MultiPointZ

A MultiPointZ represents a set of PointZs, as follows:

```
MultiPointZ
{
    Double[4]           Box         // Bounding Box
    Integer             NumPoints   // Number of Points
    Point[NumPoints]    Points      // The Points in the Set
    Double[2]           Z Range     // Bounding Z Range
    Double[NumPoints]   Z Array     // Z Values
    Double[2]           M Range     // Bounding Measure Range
    Double[NumPoints]   M Array     // Measures
}
```

The Bounding Box is stored in the order Xmin, Ymin, Xmax, Ymax.

The bounding Z Range is stored in the order Zmin, Zmax. Bounding M Range is stored in the order Mmin, Mmax.

**Table 13**
**MultiPointZ Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 18 | Integer | 1 | Little |
| Byte 4 | Box | Box | Double | 4 | Little |
| Byte 36 | NumPoints | NumPoints | Integer | 1 | Little |
| Byte 40 | Points | Points | Point | NumPoints | Little |
| Byte X | Zmin | Zmin | Double | 1 | Little |
| Byte X+8 | Zmax | Zmax | Double | 1 | Little |
| Byte X+16 | Zarray | Zarray | Double | NumPoints | Little |
| Byte Y* | Mmin | Mmin | Double | 1 | Little |
| Byte Y+8* | Mmax | Mmax | Double | 1 | Little |
| Byte Y+16* | Marray | Marray | Double | NumPoints | Little |

Note: X = 40 + (16 * NumPoints); Y = X + 16 + (8 * NumPoints)
* optional

## PolyLineZ

A PolyLineZ consists of one or more parts. A part is a connected sequence of two or more points. Parts may or may not be connected to one another. Parts may or may not intersect one another.

```
PolyLineZ
{
    Double[4]           Box         // Bounding Box
    Integer             NumParts    // Number of Parts
    Integer             NumPoints   // Total Number of Points
    Integer[NumParts]   Parts       // Index to First Point in Part
    Point[NumPoints]    Points      // Points for All Parts
    Double[2]           Z Range     // Bounding Z Range
    Double[NumPoints]   Z Array     // Z Values for All Points
    Double[2]           M Range     // Bounding Measure Range
    Double[NumPoints]   M Array     // Measures
}
```

The fields for a PolyLineZ are described in detail below:

**Box** The Bounding Box for the PolyLineZ stored in the order Xmin, Ymin, Xmax, Ymax.

**NumParts** The number of parts in the PolyLineZ.

**NumPoints** The total number of points for all parts.

**Parts** An array of length NumParts. Stores, for each part, the index of its first point in the points array. Array indexes are with respect to 0.

**Points** An array of length NumPoints. The points for each part in the PolyLineZ are stored end to end. The points for Part 2 follow the points for Part 1, and so on. The parts array holds the array index of the starting point for each part. There is no delimiter in the points array between parts.

**Z Range** The minimum and maximum Z values for the PolyLineZ stored in the order Zmin, Zmax.

**Z Array** An array of length NumPoints. The Z values for each part in the PolyLineZ are stored end to end. The Z values for Part 2 follow the Z values for Part 1, and so on. The parts array holds the array index of the starting point for each part. There is no delimiter in the Z array between parts.

**M Range** The minimum and maximum measures for the PolyLineZ stored in the order Mmin, Mmax.

**M Array** An array of length NumPoints. The measures for each part in the PolyLineZ are stored end to end. The measures for Part 2 follow the measures for Part

1, and so on. The parts array holds the array index of the starting measure for each part. There is no delimiter in the measure array between parts.

**Table 14**
**PolyLineZ Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 13 | Integer | 1 | Little |
| Byte 4 | Box | Box | Double | 4 | Little |
| Byte 36 | NumParts | NumParts | Integer | 1 | Little |
| Byte 40 | NumPoints | NumPoints | Integer | 1 | Little |
| Byte 44 | Parts | Parts | Integer | NumParts | Little |
| Byte X | Points | Points | Point | NumPoints | Little |
| Byte Y | Zmin | Zmin | Double | 1 | Little |
| Byte Y + 8 | Zmax | Zmax | Double | 1 | Little |
| Byte Y + 16 | Zarray | Zarray | Double | NumPoints | Little |
| Byte Z* | Mmin | Mmin | Double | 1 | Little |
| Byte Z+8* | Mmax | Mmax | Double | 1 | Little |
| Byte Z+16* | Marray | Marray | Double | NumPoints | Little |

Note: X = 44 + (4 * NumParts), Y = X + (16 * NumPoints), Z = Y + 16 + (8 * NumPoints)
* optional

## PolygonZ

A PolygonZ consists of a number of rings. A ring is a closed, non-self-intersecting loop. A PolygonZ may contain multiple outer rings. The rings of a PolygonZ are referred to as its parts.

The PolygonZ structure is identical to the PolyLineZ structure, as follows:

```
PolygonZ
{
    Double[4]          Box          // Bounding Box
    Integer            NumParts     // Number of Parts
    Integer            NumPoints    // Total Number of Points
    Integer[NumParts]  Parts        // Index to First Point in Part
    Point[NumPoints]   Points       // Points for All Parts
    Double[2]          Z Range      // Bounding Z Range
    Double[NumPoints]  Z Array      // Z Values for All Points
    Double[2]          M Range      // Bounding Measure Range
    Double[NumPoints]  M Array      // Measures
}
```

The fields for a PolygonZ are

| | |
|---|---|
| Box | The Bounding Box for the PolygonZ stored in the order Xmin, Ymin, Xmax, Ymax. |
| NumParts | The number of rings in the PolygonZ. |
| NumPoints | The total number of points for all rings. |
| Parts | An array of length NumParts. Stores, for each ring, the index of its first point in the points array. Array indexes are with respect to 0. |
| Points | An array of length NumPoints. The points for each ring in the PolygonZ are stored end to end. The points for Ring 2 follow the points for Ring 1, and so on. The parts array holds the array index of the starting point for each ring. There is no delimiter in the points array between rings. |
| Z Range | The minimum and maximum Z values for the arc stored in the order Zmin, Zmax. |
| Z Array | An array of length NumPoints. The Z values for each ring in the PolygonZ are stored end to end. The Z values for Ring 2 follow the Z values for Ring 1, and so on. The parts array holds the array index of the starting Z value for each ring. There is no delimiter in the Z value array between rings. |
| M Range | The minimum and maximum measures for the PolygonZ stored in the order Mmin, Mmax. |
| M Array | An array of length NumPoints. The measures for each ring in the PolygonZ are stored end to end. The measures for Ring 2 follow the measures for Ring 1, and so on. The parts array holds the array index of the starting measure for each ring. There is no delimiter in the measure array between rings. |

The following are important notes about PolygonZ shapes.

- The rings are closed (the first and last vertex of a ring MUST be the same).
- The order of rings in the points array is not significant.

**Table 15**
**PolygonZ Record Contents**

| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 15 | Integer | 1 | Little |
| Byte 4 | Box | Box | Double | 4 | Little |
| Byte 36 | NumParts | NumParts | Integer | 1 | Little |
| Byte 40 | NumPoints | NumPoints | Integer | 1 | Little |
| Byte 44 | Parts | Parts | Integer | NumParts | Little |
| Byte X | Points | Points | Point | NumPoints | Little |
| Byte Y | Zmin | Zmin | Double | 1 | Little |
| Byte Y+8 | Zmax | Zmax | Double | 1 | Little |
| Byte Y+16 | Zarray | Zarray | Double | NumPoints | Little |
| Byte Z* | Mmin | Mmin | Double | 1 | Little |
| Byte Z+8* | Mmax | Mmax | Double | 1 | Little |
| Byte Z+16* | Marray | Marray | Double | NumPoints | Little |

Note: X = 44 + (4 * NumParts), Y = X + (16 * NumPoints), Z = Y + 16 + (8 * NumPoints)
* optional

## MultiPatch

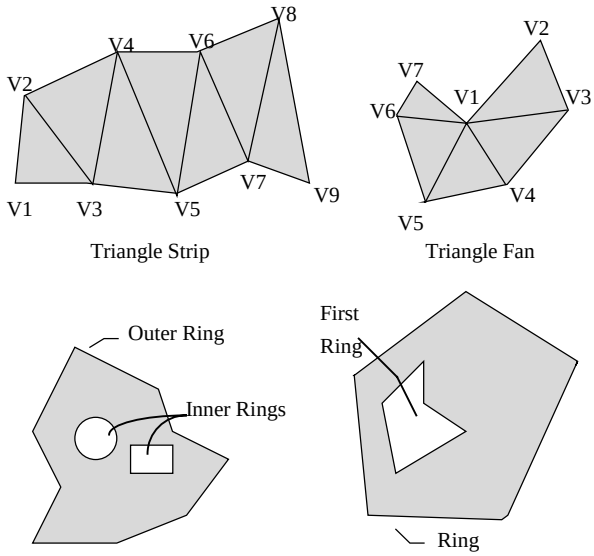
A MultiPatch consists of a number of surface patches. Each surface patch describes a surface. The surface patches of a MultiPatch are referred to as its parts, and the type of part controls how the order of vertices of an MultiPatch part is interpreted. The parts of a MultiPatch can be of the following types:

- **Triangle Strip** A linked strip of triangles, where every vertex (after the first two) completes a new triangle. A new triangle is always formed by connecting the new vertex with its two immediate predecessors.
- **Triangle Fan** A linked fan of triangles, where every vertex (after the first two) completes a new triangle. A new triangle is always formed by connecting the new vertex with its immediate predecessor and the first vertex of the part.
- **Outer Ring** The outer ring of a polygon.
- **Inner Ring** A hole of a polygon.
- **First Ring** The first ring of a polygon of an unspecified type.
- **Ring** A ring of a polygon of an unspecified type.

A single *Triangle Strip*, or *Triangle Fan*, represents a single surface patch. See **Figure 3** for examples of those part types.

A sequence of parts that are rings can describe a polygonal surface patch with holes. The sequence typically consists of an *Outer Ring*, representing the outer boundary of the patch, followed by a number of *Inner Rings* representing holes. When the individual types of rings in a collection of rings representing a polygonal patch with holes are unknown, the sequence must start with *First Ring*, followed by a number of *Rings*. A sequence of *Rings* not preceded by an *First Ring* is treated as a sequence of *Outer Rings* without holes.

**Figure 3**
**MultiPatch Part Examples**




This figure shows examples of all types of MultiPatch parts.

The values used for encoding part type are as follows:

| Value | Part Type |
|---|---|
| 0 | Triangle Strip |
| 1 | Triangle Fan |
| 2 | Outer Ring |
| 3 | Inner Ring |
| 4 | First Ring |
| 5 | Ring |

```
MultiPatch
{
    Double[4]           Box             // Bounding Box
    Integer             NumParts        // Number of Parts
    Integer             NumPoints       // Total Number of Points
    Integer[NumParts]   Parts           // Index to First Point in Part
    Integer[NumParts]   PartTypes       // Part Type
    Point[NumPoints]    Points          // Points for All Parts
    Double[2]           Z Range         // Bounding Z Range
    Double[NumPoints]   Z Array         // Z Values for All Points
    Double[2]           M Range         // Bounding Measure Range
    Double[NumPoints]   M Array         // Measures
}
```

The fields for a MultiPatch are

Box — The Bounding Box for the MultiPatch stored in the order Xmin, Ymin, Xmax, Ymax.

NumParts — The number of parts in the MultiPatch.

NumPoints — The total number of points for all parts.

Parts — An array of length NumParts. Stores, for each part, the index of its first point in the points array. Array indexes are with respect to 0.

PartTypes — An array of length NumParts. Stores for each part its type.

Points — An array of length NumPoints. The points for each part in the MultiPatch are stored end to end. The points for Part 2 follow the points for Part 1, and so on. The parts array holds the array index of the starting point for each part. There is no delimiter in the points array between parts.

Z Range — The minimum and maximum Z values for the arc stored in the order Zmin, Zmax.

Z Array — An array of length NumPoints. The Z values for each part in the MultiPatch are stored end to end. The Z values for Part 2 follow the Z values for Part 1, and so on. The parts array holds the array index of the starting Z value for each part. There is no delimiter in the Z value array between parts.

M Range — The minimum and maximum measures for the MultiPatch stored in the order Mmin, Mmax.

M Array — An array of length NumPoints. The measures for each part in the MultiPatch are stored end to end. The measures for Part 2 follow the measures for Part 1, and so on. The parts array holds the array index of the

starting measure for each part. There is no delimiter in the measure array between parts.

The following are important notes about MultiPatch shapes.

- If a part is a ring, it must be closed (the first and last vertex of a ring MUST be the same).
- The order of parts that are rings in the points array is significant: *Inner Rings* must follow their *Outer Ring*; a sequence of *Rings* representing a single surface patch must start with a ring of the type *First Ring*.
- Parts can share common boundaries, but parts must not intersect and penetrate each other.

**Table 16**
**MultiPatch Record Contents**

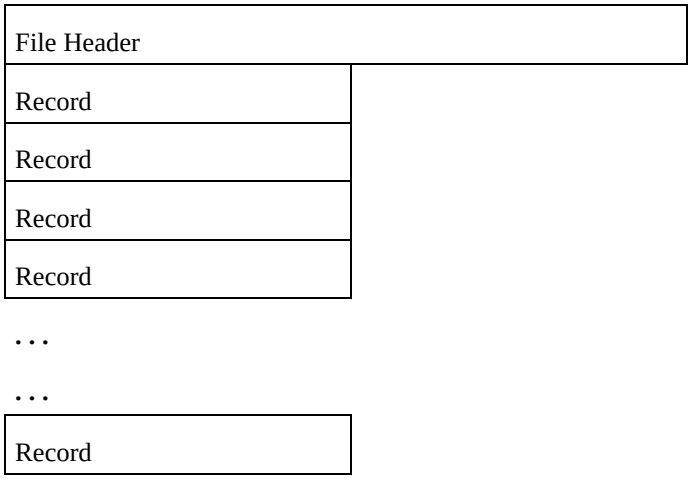
| Position | Field | Value | Type | Number | Byte Order |
|---|---|---|---|---|---|
| Byte 0 | Shape Type | 31 | Integer | 1 | Little |
| Byte 4 | Box | Box | Double | 4 | Little |
| Byte 36 | NumParts | NumParts | Integer | 1 | Little |
| Byte 40 | NumPoints | NumPoints | Integer | 1 | Little |
| Byte 44 | Parts | Parts | Integer | NumParts | Little |
| Byte W | PartTypes | PartTypes | Integer | NumParts | Little |
| Byte X | Points | Points | Point | NumPoints | Little |
| Byte Y | Zmin | Zmin | Double | 1 | Little |
| Byte Y+8 | Zmax | Zmax | Double | 1 | Little |
| Byte Y+16 | Zarray | Zarray | Double | NumPoints | Little |
| Byte Z* | Mmin | Mmin | Double | 1 | Little |
| Byte Z+8* | Mmax | Mmax | Double | 1 | Little |
| Byte Z+16* | Marray | Marray | Double | NumPoints | Little |

Note: W = 44 + (4 * NumParts), X = W + (4 * NumParts), Y = X + (16 * NumPoints), Z = Y + 16 + (8 * NumPoints)
* optional

## Organization of the Index File

The index file (.shx) contains a 100-byte header followed by 8-byte, fixed-length records. **Figure 4** illustrates the index file organization.

**Figure 4**
**Organization of the Index File**




## The Index File Header

The index file header is identical in organization to the main file header described above. The file length stored in the index file header is the total length of the index file in 16-bit words (the fifty 16-bit words of the header plus 4 times the number of records).

## Index Records

The *I'th* record in the index file stores the offset and content length for the *I'th* record in the main file. **Table 17** shows the fields in the file header with their byte position, value, type, and byte order. In the table, position is with respect to the start of the index file record.

**Table 17**
**Description of Index Records**

| Position | Field | Value | Type | Byte Order |
|---|---|---|---|---|
| Byte 0 | Offset | Offset | Integer | Big |
| Byte 4 | Content Length | Content Length | Integer | Big |

The offset of a record in the main file is the number of 16-bit words from the start of the main file to the first byte of the record header for the record. Thus, the offset for the first record in the main file is 50, given the 100-byte header.

The content length stored in the index record is the same as the value stored in the main file record header.

## Organization of the dBASE File

The dBASE file (.dbf) contains any desired feature attributes or attribute keys to which other tables can be joined. Its format is a standard DBF file used by many table-based applications in Windows™ and DOS. Any set of fields can be present in the table. There are three requirements, as follows:

- The file name must have the same prefix as the shape and index file. Its suffix must be .dbf. (See the example on page 2, in Naming Conventions.)
- The table must contain one record per shape feature.
- The record order must be the same as the order of shape features in the main (*.shp) file.
- The year value in the dBASE header must be the year since 1900.

For more information on the dBASE file format, visit the INPRISE Corp. Web site at www.inprise.com.

# Glossary

Key terms are defined below that will help you understand the concepts discussed in this document.

**ARC/INFO** ARC/INFO software is designed for users who require a complete set of tools for processing and manipulating spatial data including digitizing, editing, coordinate management, network analysis, surface modeling, and grid cell based modeling. ARC/INFO operates on a large variety of workstations and minicomputers. Using open standards and client/server architecture, ARC/INFO can act as a GIS server for ArcView GIS clients.

**ArcCAD** ArcCAD software brings the functionality of ARC/INFO GIS software to the AutoCAD environment, providing comprehensive data management, spatial analysis, and display tools.

**ARC Macro Language (AML)** ARC Macro Language is a high-level, algorithmic language that provides full programming capabilities and a set of tools to tailor the user interface of your application.

**ArcView GIS** ArcView GIS software is a powerful, easy-to-use desktop GIS that gives you the power to visualize, explore, query, and analyze data spatially. ArcView GIS operates in Windows desktop environments as well as a large variety of workstations.

**Avenue** Avenue software is an object-oriented programming language and development environment created for use with ArcView GIS software. Avenue can be used to extend ArcView GIS software's basic capabilities and customize ArcView GIS for specific applications.

**big endian byte order** Left-to-right byte ordering of an integer word. This byte-ordering method is used on many UNIX systems including Sun, Hewlett–Packard®, IBM®, and Data General AViiON®.

**Bounding Box** A Bounding Box is a rectangle surrounding each shape (e.g., PolyLine) that is just large enough to contain the entire shape. It is defined as Xmin,Ymin, Xmax,Ymax.

***Business*MAP** *Business*MAP database mapping software for Windows allows you to create custom maps and represent information in two- or three-dimensional charts. *Business*MAP reads ESRI shapefiles and works with the leading contact managers, databases, and spreadsheets.

**coverage**

1. A digital version of a map forming the basic unit of vector data storage in ARC/INFO software. A coverage stores geographic features as primary features (such as arcs, nodes, polygons, and label points) and secondary features (such as tics, map extent, links, and annotation). Associated feature attribute tables describe and store attributes of the geographic features.

2. A set of thematically associated data considered as a unit. A coverage usually represents a single theme such as soils, streams, roads, or land use.

**Data Automation Kit (DAK)**

Data Automation Kit (DAK) complements ArcView GIS and other desktop mapping software by providing high-quality digitizing and data editing, topology creation, data conversion, and map projection capabilities.

**feature**

A representation of a geographic feature that has both a spatial representation referred to as a "shape" and a set of attributes.

**index file**

An ArcView GIS shapefile index file is a file that allows direct access to records in the corresponding main file.

**little endian byte order**

Right-to-left byte ordering of an integer word. This byte-ordering method is used on many operating file systems including DEC OSF/1™, DEC OpenVMS™, MS–DOS®, and Windows NT™.

**MapObjects**

MapObjects is a collection of embeddable mapping and GIS components including an Active X Control (OCX) and programmable Active X Automation objects. Use MapObjects with a variety of standard Windows development environments to build mapping applications or add mapping components into existing applications.

**MultiPoint**

A single feature composed of a cluster of point locations and a single attribute record. The group of points represents the geographic feature.

**NumPoints**

The count of the number of x,y vertices contained in a shape.

**PC ARC/INFO**

PC ARC/INFO is a full-featured GIS for PC compatibles. Like ARC/INFO software, PC ARC/INFO is used by organizations around the world for automating, managing, and analyzing geographic information. Attributes describing geographic features are stored as tabular files in dBASE format.

**PolyLine**

An ordered set of x,y vertices representing a line or boundary.

**ring**

An ordered set of x,y vertices where the first vertex is the same location as the last vertex; a closed PolyLine or a polygon.

**shapefile**

An ArcView GIS data set used to represent a set of geographic features such as streets, hospital locations, trade areas, and ZIP Code boundaries. Shapefiles can represent point, line, or area features. Each feature in a shapefile represents a single geographic feature and its attributes.

**Simple Macro Language (SML)** SML is PC ARC/INFO software's Simple Macro Language—a set of commands that constitute a simple programming language for building macros with some of the features of a high-level programming language such as expression evaluation, handling of input and output, and directing program flow of control.

**theme** A user-defined set of geographic features. Data sources for themes in ArcView GIS include coverages, grids, images, and shapefiles. Theme properties include the data source name, attributes of interest, a data classification scheme, and drawing methodology.

**topology** The spatial relationships between connecting or adjacent coverage features (e.g., arcs, nodes, polygons, and points). For example, the topology of an arc includes its from- and to-nodes and its left and right polygons. Topological relationships are built from simple elements into complex elements: points (simplest elements) and arcs (sets of connected points) are used to represent more complex features such as areas (sets of connected arcs). Shapefiles do not explicitly record topology.

Coverages represent geographic features as topological line graphs. Topology can be useful for many GIS modeling operations that do not require coordinates. For example, to find an optimal path between two points requires a list of the arcs that connect to each other and the cost to traverse each arc in each direction. Coordinates are only needed for drawing the path after it is calculated.

**vector** A Cartesian (i.e., x,y) coordinate-based data structure commonly used to represent geographic features. Each feature is represented as one or more vertices. Attributes are associated with the feature. Other data structures include raster (which associates attributes with a grid cell) and triangulated irregular networks (TINs) for surface representation.

**vertex** One of a set of ordered x,y coordinates that constitutes a line.

For more than 25 years ESRI has been helping people manage and analyze geographic information. ESRI offers a framework for implementing GIS in any organization with a seamless link from personal GIS on the desktop to enterprisewide GIS client/server and data management systems. ESRI GIS solutions are flexible and can be customized to meet the needs of our users. ESRI is a full-service GIS company, ready to help you begin, grow, and build success with GIS.

## Corporate


ESRI
380 New York Street
Redlands, California
92373-8100 USA
Telephone: 909-793-2853
Fax: 909-793-5953


For more information on ESRI software call ESRI at

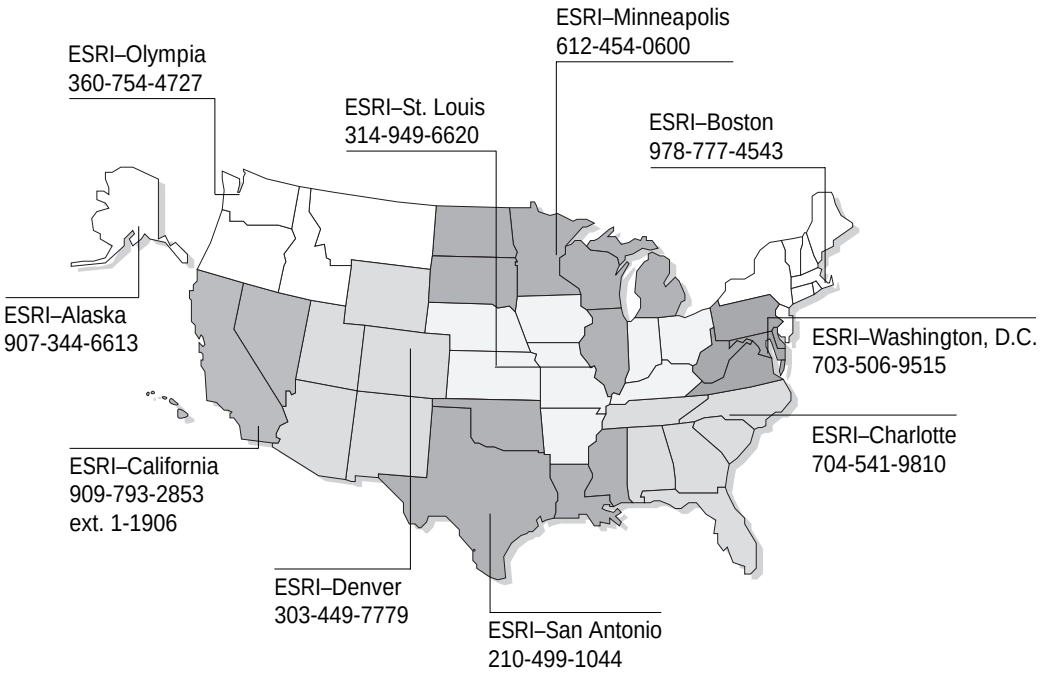
1-800-447-9778

(1-800-GIS-XPRT)

Send E-mail inquiries to
info@esri.com

Visit ESRI's Web page at
www.esri.com

## Regional




## International

Australia
61-89-242-1005

BeLux
32-2-460-7000

Canada
416-441-6035

France
33-1-46-23-6060

Germany
49-8166-677-0

Hong Kong
852-2-730-6883

India
91-11-620-3801

Italy
39-6-406-96-1

Nederland B.V.
31-10-217-0700

Poland
48-22-256-482

South Asia
65-735-8755

Spain
34-1-559-4347

Sweden
46-23-84090

Thailand
66-2-678-0707

United Kingdom
44-1-923-210450

Venezuela
58-2-285-1134

Outside the United States,
contact your local ESRI distributor.
For the number of your distributor,
call ESRI at
909-793-2853, ext.1-1235

ESRI distributor or business partner address goes here

GSA
No. GS-35F-5D86H

Printed in USA