

Informix Geodetic DataBlade Module

User's Guide

Version 2.1
June 1997
Part No. 000-3711

Published by INFORMIX® Press

Informix Software, Inc.
4100 Bohannon Drive
Menlo Park, CA 94025-1032

Copyright © 1981-1997 by Informix Software, Inc. or its subsidiaries, provided that portions may be copyrighted by third parties, as set forth in documentation. All rights reserved.

The following are worldwide trademarks of Informix Software, Inc., or its subsidiaries, registered in the United States of America as indicated by “®,” and in numerous other countries worldwide:

INFORMIX®; DataBlade®

All other marks or symbols are registered trademarks or trademarks of their respective owners.

ACKNOWLEDGMENTS

Documentation Team: Michael Kline, Tai Day, Martha McDevitt

Contributors: David Ashkenas, Dave Brown, Antonin Guttman, Kumar Ramaiyer, Christiane Renck,
Robert Uleman, Jim Vaudagna, Jie Zhang

To the extent that this software allows the user to store, display, and otherwise manipulate various forms of data, including, without limitation, multimedia content such as photographs, movies, music and other binary large objects (blobs), use of any single blob may potentially infringe upon numerous different third-party intellectual and/or proprietary rights. It is the user's responsibility to avoid infringements of any such third-party rights.

RESTRICTED RIGHTS/SPECIAL LICENSE RIGHTS

Software and documentation acquired with US Government funds are provided with rights as follows: (1) if for civilian agency use, with Restricted Rights as defined in FAR 52.227-19; (2) if for Dept. of Defense use, with rights as restricted by vendor's standard license, unless superseded by negotiated vendor license as prescribed in DFAR 227.7202. Any whole or partial reproduction of software or documentation marked with this legend must reproduce the legend.

Table of Contents

Introduction

About This Manual	3
Organization of This Manual	3
Types of Users	4
Software Dependencies	4
Standards Compliance	5
Documentation Conventions	5
Typographical Conventions	6
Icon Conventions	7
Function Syntax Conventions	7
Naming Conventions	8
Additional Documentation	8
Printed Documentation	9
Release Notes, Documentation Notes, Machine Notes	10
Related Reading	10
Informix Welcomes Your Comments	11

Chapter 1

Overview of the Informix Geodetic DataBlade Module

Concepts and Principles	1-3
Geodetic Datums	1-4
Geodetic Coordinate Reference System	1-4
The Altitude Range	1-5
The Time Range	1-6
Distance	1-6
Internal Representation of Latitude and Longitude	1-7
Spatial Access Method	1-8

Chapter 2

Using the Informix Geodetic DataBlade Module

Informix Geodetic DataBlade Module Data Types	2-3
Spatio-Temporal Data Types	2-4

Domain-Enforcing Data Types	2-4
Range Data Types	2-5
Internal Representation of Spatio-Temporal Data.	2-5
Large Objects	2-6
Inserting Spatio-Temporal Objects into a Table	2-6
Geodetic Functions	2-7
Client Validation Program	2-9
Selecting Objects by Altitude or Time	2-10
Setting Date and Time Precision	2-11
Default Setting	2-11
Effect of GL_DATETIME on Input and Output	2-13
Using DATETIME Casts	2-15

Chapter 3

Geodetic Data Types

Domain-Enforcing Data Types	3-3
Range Data Types	3-4
Spatio-Temporal Data Types	3-4
Data Type Reference	3-4
GeoAltitude	3-5
GeoAltrange	3-7
GeoAngle	3-11
GeoAzimuth	3-16
GeoBox	3-19
GeoCircle	3-22
GeoCoords	3-25
GeoDistance	3-27
GeoEllipse	3-29
GeoLatitude	3-33
GeoLineseg.	3-36
GeoLongitude.	3-39
GeoObject	3-41
GeoPoint	3-44
GeoPolygon	3-47
GeoRing.	3-53
GeoString	3-56
GeoTimeRange	3-59

Chapter 4

Geodetic Functions

Overview of Geodetic Functions	4-3
Accessor Functions	4-4
Constructors and Conversion Functions	4-4

Operator Functions	4-5
Data Validation Functions	4-5
System Function	4-5
Function Reference	4-6
AltRange	4-7
Azimuth	4-8
Begin	4-9
Beyond.	4-10
Bottom.	4-13
Coords.	4-14
End	4-15
GeoAltRange	4-16
GeoBox	4-17
GeoCircle	4-18
GeoCoords	4-20
GeoEllipse	4-21
GeoLineseg	4-23
GeoPoint	4-24
GeoPolygon	4-25
GeoRelease	4-26
GeoRing	4-27
GeoString.	4-28
GeoTimeRange	4-29
Inside	4-30
Intersect	4-32
IsGeoBox	4-34
IsGeoCircle	4-35
IsGeoEllipse	4-36
IsGeoLineseg	4-37
IsGeoPoint	4-38
IsGeoPolygon	4-39
IsGeoRing.	4-40
IsGeoString	4-41
IsValidGeometry	4-42
IsValidSDTS	4-44
Latitude	4-45
Longitude.	4-46
Major	4-47
Minor	4-48
NPoints	4-49
NRings.	4-50
Outside	4-51

	Radius	4-53
	Ring	4-54
	SetAltRange	4-55
	SetDist	4-56
	SetTimeRange	4-57
	TimeRange	4-58
	Top	4-59
	Within	4-60
Chapter 5	Spatial Operators	
	Handling of Horizontal Time and Altitude Dimensions	5-4
	Spatial Operators That Use R-tree Indexes	5-6
Chapter 6	Using R-tree Indexes	
	About Indexes	6-3
	Access Methods	6-4
	The B-tree Access Method.	6-4
	The R-tree Access Method.	6-4
	Operator Classes	6-5
	Operator Class GeoObject_ops	6-5
	Operator Class rtree_ops	6-6
	Creating an R-tree Index	6-6
	How Spatial Operators Use R-tree Indexes	6-7
	Rewriting Ternary Operator Expressions	6-7
	Example Query Using Within	6-8
	Example Query Using Beyond	6-9
Appendix A	Geodetic Header File	
Appendix B	Example Programs	
	Index	

Introduction

About This Manual	3
Organization of This Manual	3
Types of Users	4
Software Dependencies	4
Standards Compliance	5
Documentation Conventions	5
Typographical Conventions	6
Icon Conventions	7
Function Syntax Conventions	7
Naming Conventions.	8
Additional Documentation	8
Printed Documentation	9
Release Notes, Documentation Notes, Machine Notes	10
Related Reading	10
Informix Welcomes Your Comments	11

T

his chapter introduces the *Informix Geodetic DataBlade Module User's Guide*. Read this chapter for an overview of the information provided in this manual and for an understanding of the conventions used throughout.

About This Manual

The Informix Geodetic DataBlade module is an extension to the INFORMIX-Universal Server object-relational database management system (ORDBMS) that enables you to store and manipulate spatio-temporal data in an INFORMIX-Universal Server database. The *Informix Geodetic DataBlade Module User's Guide* describes the data types and functions available in the Informix Geodetic DataBlade module and explains how to use them. It also describes how to create and use indexes on spatio-temporal data.

This section discusses the organization of the manual, the intended audience, and the associated software products that you must have to develop and use the DataBlade module.

Organization of This Manual

This manual includes the following chapters:

- This introduction provides an overview of the manual, describes the documentation conventions used, and explains the generic style of this documentation.
- [Chapter 1, “Overview of the Informix Geodetic DataBlade Module,”](#) describes the principles and concepts behind the Informix Geodetic DataBlade module.
- [Chapter 2, “Using the Informix Geodetic DataBlade Module,”](#) introduces the Informix Geodetic DataBlade module's data types, functions, and features.

- [Chapter 3, “Geodetic Data Types,”](#) describes the spatio-temporal data types and helper data types that this module provides.
- [Chapter 4, “Geodetic Functions,”](#) describes the Informix Geodetic DataBlade module’s accessor, constructor, conversion, operator, and system functions.
- [Chapter 5, “Spatial Operators,”](#) provides additional information about a special group of functions known as *spatial operators*.
- [Chapter 6, “Using R-tree Indexes,”](#) explains how to create an index on spatio-temporal data stored in an INFORMIX-Universal Server database.
- [Appendix A](#) lists and describes the contents of the Informix Geodetic DataBlade module header file, **geodetic.h**.
- [Appendix B](#) lists and describes the Informix Geodetic DataBlade module example files.
- A list of error messages follows [Appendix B](#), and a comprehensive index directs you to areas of particular interest.

Types of Users

This manual is written for developers creating applications that use the Informix Geodetic DataBlade module’s spatio-temporal data types and functions. This manual is also a helpful resource for database administrators managing installations that use INFORMIX-Universal Server and the Informix Geodetic DataBlade module.

Software Dependencies

To use this manual, you must be using INFORMIX-Universal Server as your database server. Check your release notes for the correct version of INFORMIX-Universal Server for this release.

Standards Compliance

The Informix Geodetic DataBlade module supports a subset of the data types from the *Spatial Data Transfer Standard (SDTS)*—*Federal Information Processing Standard 173*, as referenced by the document *Content Standard for Geospatial Metadata*, Federal Geographic Data Committee, June 8, 1994 (FGDC Metadata Standard).

Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other volumes in the documentation set.

The following conventions are covered:

- Typographical conventions
- Icon conventions
- Naming conventions




Typographical Conventions

This manual uses a standard set of conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth. The following typographical conventions are used throughout this manual.

Convention	Meaning
KEYWORD	All keywords appear in uppercase letters in a serif font.
<i>italics</i>	Within text, new terms and emphasized words appear in italics. Within syntax diagrams, values that you are to specify appear in italics.
boldface	Identifiers (names of classes, objects, constants, events, functions, program variables, forms, labels, and reports), environment variables, database names, filenames, table names, column names, icons, menu items, command names, and other similar terms appear in boldface.
<code>monospace</code>	Information that the product displays and information that you enter are printed in a monospace typeface.

Icon Conventions

Comment icons identify three types of information, as described in the following table. This information is always displayed in *italics*.

Icon	Description
	The <i>warning</i> icon identifies vital instructions, cautions, or critical information.
	The <i>important</i> icon identifies significant information about the feature or operation that is being described.
	The <i>tip</i> icon identifies additional details or shortcuts for the functionality that is being described.

Function Syntax Conventions

This guide uses the following conventions to specify the syntax for Informix Geodetic DataBlade module functions:

- Square brackets ([]) surround optional items.
- Curly brackets ({ }) surround items that can be repeated zero or more times.
- A vertical line (|) separates alternatives.
- Comma-separated lists of values are indicated by a parameter value with a “_commalist” suffix. For example, `colorlist=color_commalist` might indicate “colorlist=red, blue, green.”
- Function parameters are italicized; arguments that must be specified as shown are not italicized.

These function syntax conventions used in context look like this:

```
myfunction([optionalarg], {repeatablearg}, set | noset,  
myparam=my_comma_list)
```

Informix Geodetic DataBlade module functions are used in SQL statements to manipulate the specialized data types defined by the Informix Geodetic DataBlade module. These syntax conventions are used in the function reference chapter of this guide.

Naming Conventions

The Informix Geodetic DataBlade module uses the following naming conventions for data types and functions:

- All data types begin with the prefix *Geo*. For example, the data type for point objects is **GeoPoint**.
- Type constructor functions have the same name as the type they construct. For example, the type constructor for the **GeoPoint** data type is **GeoPoint**.
- Other functions (with the exception of **GeoRelease**) have no prefix.

All functions are presented in boldface. Data types are presented in mixed case in the normal text typeface. You can distinguish constructor functions and data types with the same name by the typeface.

Additional Documentation

This section describes the following parts of the documentation set:

- Printed documentation
- On-line documentation
- Related reading

Printed Documentation

The following related Informix documents complement the information in this manual:

- The *UNIX Products Installation Guide* helps you ensure that your database server is set up properly before you begin to work with it.
- The *BladeManager User's Guide* contains instructions for registering DataBlade modules for use in an INFORMIX-Universal Server database.
- The *INFORMIX-Universal Server Administrator's Guide* helps you understand, install, configure, and use INFORMIX-Universal Server and change its characteristics to meet your needs.
- The *DB-Access User Manual* describes how to invoke the DB-Access utility to access, modify, and retrieve information from Informix database servers.
- The *Informix Guide to SQL: Tutorial* teaches SQL as it is implemented by Informix products. It also describes the fundamental ideas and terminology for planning and implementing a relational database.
- The *Informix Guide to SQL: Reference* includes details of the Informix system catalog tables, describes Informix and common environment variables that you should set, and describes the built-in data types that INFORMIX-Universal Server supports.
- A companion volume to the Reference, the *Informix Guide to SQL: Syntax* provides a detailed description of all the SQL statements supported by Informix products. This guide also provides a detailed description of Stored Procedure Language (SPL) statements.
- The *Guide to GLS Functionality* provides information on the GL_DATETIME environment variable, which controls time range input and output formats.

Release Notes, Documentation Notes, Machine Notes

The following on-line files, located in the directory, **\$INFORMIXDIR/extend/geodetic.versno**, supplement the information in this manual.

On-Line File	Purpose
GEODC <code>nnn</code> .TXT	Documentation notes. Describes features not covered in the manuals or that have been modified since publication.
GEORL <code>nnn</code> .TXT	Release notes. Describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.
GEOMC <code>nnn</code> .TXT	Machine notes. Describes any special actions required to configure and use Informix products on your computer.

Please examine these files because they contain vital information about application and performance issues.

Related Reading

For additional technical information on database management, see *An Introduction to Database Systems*, by C. J. Date (Addison-Wesley Publishing, 1995).

To learn more about the SQL language, consider the following books:

- *A Guide to the SQL Standard*, by C. J. Date with H. Darwen (Addison-Wesley Publishing, 1993)
- *Understanding the New SQL: A Complete Guide*, by J. Melton and A. Simon (Morgan Kaufmann Publishers, 1993)
- *Using SQL*, by J. Groff and P. Weinberg (Osborne McGraw-Hill, 1990)

To learn more about map projections and datums, see:

Map Projections—A Working Manual by John P. Snyder (U.S. Geological Survey Professional Paper 1395. Washington, D.C., U.S. Government Printing Office, 1987)

To use the *Informix Geodetic DataBlade Module User's Guide*, you should be familiar with your computer operating system. If you have limited UNIX system experience, consult your operating system manual or a good introductory text before you read this manual. The following texts provide a good introduction to UNIX systems:

- *Introducing the UNIX System* by H. McGilton and R. Morgan (McGraw-Hill Book Company, 1983)
- *Learning the UNIX Operating System* by G. Todino, J. Strang, and J. Peek (O'Reilly & Associates, 1993)
- *A Practical Guide to the UNIX System* by M. Sobell (Benjamin/Cummings Publishing, 1989)
- *UNIX for People* by P. Birns, P. Brown, and J. Muster (Prentice-Hall, 1985)
- *UNIX System V: A Practical Guide* by M. Sobell (Benjamin/Cummings Publishing, 1995)

Informix Welcomes Your Comments

Please tell us what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about any corrections or clarifications that you would find useful. Please include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Write to us at the following address:

Informix Software, Inc.
Technical Publications
300 Lakeside Dr., Suite 2700
Oakland, CA 94612

If you prefer to send electronic mail, our address is:

doc@informix.com

Or, send a facsimile to Technical Publications at:

415-926-6571

We appreciate your feedback.

Overview of the Informix Geodetic DataBlade Module

Concepts and Principles	1-3
Geodetic Datums	1-4
Geodetic Coordinate Reference System	1-4
The Altitude Range	1-5
The Time Range	1-6
Distance	1-6
Internal Representation of Latitude and Longitude	1-7
Spatial Access Method	1-8

T

his chapter provides an overview of the Informix Geodetic DataBlade module and explains concepts and information you should know before you use this extension to INFORMIX-Universal Server.

With the Informix Geodetic DataBlade module, you can store and manipulate—in an INFORMIX-Universal Server database—objects in space referenced by latitude and longitude, and provide additional attributes representing an altitude range and a time range. You can create the following spatio-temporal objects with this module:

- Points
- Line segments
- Strings
- Rings
- Polygons
- Boxes
- Circles
- Ellipses

The Informix Geodetic DataBlade module provides data types that represent these spatial objects on an ellipsoidal representation of the Earth. It is designed to manage spatio-temporal data with global content, such as metadata associated with satellite images.

Concepts and Principles

The Informix Geodetic DataBlade module takes its name from the discipline known as *geodesy*. Geodesy is the study of the size and shape of the Earth. The Informix Geodetic DataBlade module is designed to handle objects defined on the Earth's surface with a high degree of precision.

To do this, it uses a latitude and longitude coordinate system on an ellipsoidal Earth model, or *geodetic datum*, rather than a planar, *x*- and *y*-coordinate system.

Geodetic Datums

A geodetic datum is a reference system that describes the Earth. Many such reference systems have been developed over the centuries as science has developed new tools for measuring the Earth. Both ground and satellite measurements have been used to create datums, which in turn are used to create flat map projections.

The Informix Geodetic DataBlade module uses a specific geodetic datum, the World Geodetic System 1984 (WGS-84) datum, in its internal calculations and representations. The WGS-84 datum takes the center of the Earth as its point of origin and maps the whole of the globe; it is an Earth-centered datum. Contrast this with a regional datum such as the North American 1927 Datum, which maps North America, starting from a point on the ground. A regional datum is accurate for the region it aims to model, but an Earth-centered geodetic datum is necessary to handle locations over the entire globe.

Geodetic Coordinate Reference System

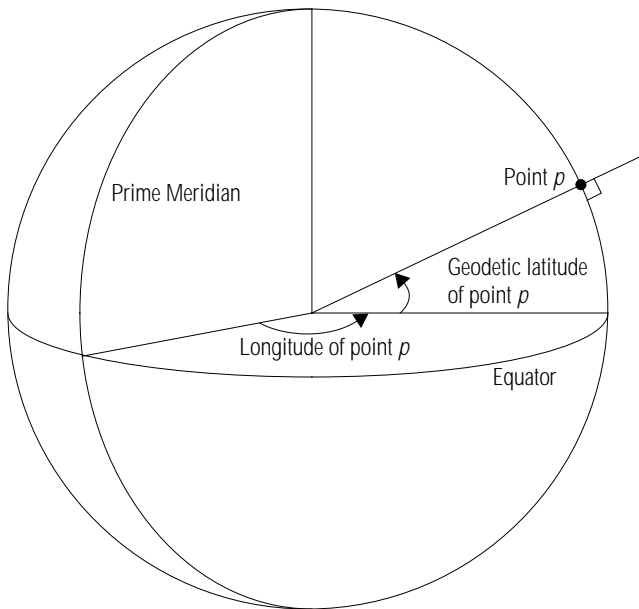
The Informix Geodetic DataBlade module's coordinate reference system uses *geodetic latitude* and *longitude* to describe locations relative to the Earth. Latitude and longitude are always based on a specific datum, in this case the WGS-84 datum.

The geodetic latitude of a point is the angle between the datum surface normal at the point, and the equatorial plane.

Longitude is the angle in the equatorial plane between the line that connects the Earth's center with the Prime Meridian, and the line that connects the center with the meridian on which the point lies. A *meridian* is a direct path that is the shortest distance between the poles.

The ellipsoid in [Figure 1-1](#) shows the angles that represent geodetic latitude and longitude.

Figure 1-1
Geodetic Latitude
and Longitude
Angles



Latitude and longitude coordinates are expressed in degrees with a decimal fraction. There are 360 degrees of longitude, starting at the Prime Meridian (0° longitude) and proceeding eastward in a positive direction through 180° , and west in negative values through -180° . Latitude degrees begin at the equator (0° latitude) and proceed to the North Pole (90° latitude) and South Pole (-90° latitude).

The Altitude Range

For each object, the Informix Geodetic DataBlade module records an altitude range. Altitude is measured in meters above or below the surface of the WGS-84 ellipsoid. The altitude at the surface of the ellipsoid is 0.

Altitude can be expressed as a single value or as a range. In either case, the top and bottom surfaces of an object have a constant altitude, rather than a truly three-dimensional shape in which the object's altitude varies. Imagine a two-dimensional object extended into a third dimension, or an object cut out with a cookie cutter. A more accurate description for these objects is 2 dimensional or pseudo three-dimensional.

Objects with a single altitude value are located at the specified altitude above or below the surface of the datum. The altitude of point objects is measured along the surface normal on the datum ellipsoid; all other objects lie on a surface that has a constant altitude, where altitude is measured from the surface of the datum at every point.

An object with an altitude range is bounded at the top and bottom by a constant altitude surface. The cross-section has the same shape at any given altitude. A point with an altitude range is actually a line segment along the surface normal to the ellipsoid.

The Time Range

Another variable that the Informix Geodetic DataBlade module allows you to associate with data is *time*. You can express time as a range with beginning and ending times, or as a single moment in time.

Generally, the time variable refers to the time the data was collected rather than the time associated with the object. For example, a time range might indicate the time at which the sea level was measured, not the time period the sea was at a particular level.

Distance

The Informix Geodetic DataBlade module measures distance between two points along a *geodesic*. A geodesic is the shortest path between two points on the ellipsoid. Distance is measured at altitude 0 on the ellipsoid regardless of the altitude range specified for a particular object.

Internal Representation of Latitude and Longitude

The Informix Geodetic DataBlade module converts geodetic latitude and longitude values to *direction cosines*. Direction cosines are the Cartesian x , y , and z components of the surface normal vector. Only the original angular values are stored in the database. Direction cosines are computed as necessary. Figure 1-2 shows the Cartesian components x_1 , y_1 , and z_1 of vector A with end point (x_1, y_1, z_1) .

Once the direction cosines have been calculated, they are used in subsequent calculations rather than the latitude and longitude values. While transparent to the user, this approach provides a significant improvement in performance and stability. Calculations that use latitude and longitude angles require the recurrent use of transcendental functions such as sine, cosine, and arc tangent. Calculations involving direction cosines use vector algebra, avoiding the use of transcendental functions. In addition, using latitude and longitude directly causes computational problems, both at the poles and at the 180° meridian, that are avoided by using direction cosines.

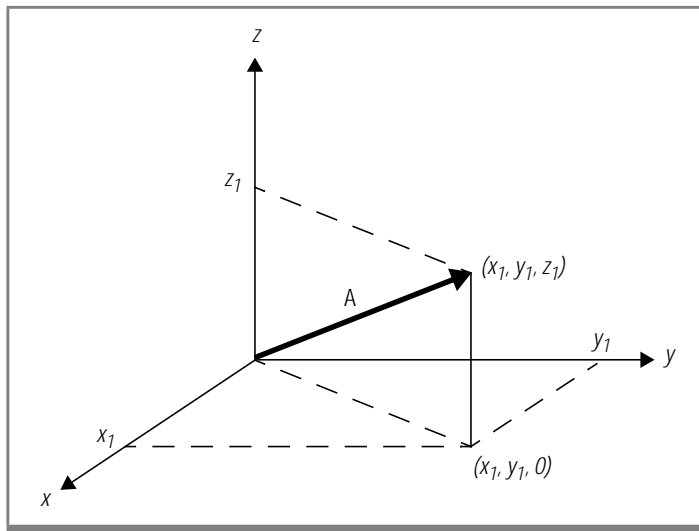


Figure 1-2
Cartesian
Components of
Vector A

Spatial Access Method

The Informix Geodetic DataBlade module includes support for the R-tree spatial access method. The R-tree access method allows you to create an index on columns containing spatio-temporal data.

For the R-tree access method, direction cosines are converted to a Cartesian *x-y-z* reference system with its origin at the center of the Earth, and with an altitude dimension and a time dimension. An index search applies an intersection test to bounding boxes created around spatial objects. When the search identifies an object of interest based on the bounding boxes, it applies an intersection or containment test to the complex objects themselves.

Using the Informix Geodetic DataBlade Module

Informix Geodetic DataBlade Module Data Types	2-3
Spatio-Temporal Data Types	2-4
Domain-Enforcing Data Types	2-4
Range Data Types	2-5
Internal Representation of Spatio-Temporal Data	2-5
Large Objects	2-6
Inserting Spatio-Temporal Objects into a Table	2-6
Geodetic Functions	2-7
Client Validation Program	2-9
Selecting Objects by Altitude or Time	2-10
Setting Date and Time Precision	2-11
Default Setting	2-11
Effect of GL_DATETIME on Input and Output	2-13
Using DATETIME Casts.	2-15

T

his chapter discusses Informix Geodetic DataBlade module data types, functions and features. It describes:

- the data types and functions.
- how the Informix Geodetic DataBlade module stores data internally and represents objects to a user.
- how to insert and load spatio-temporal data.
- how the Informix Geodetic DataBlade module handles large objects.
- the **geovalidate** client program.
- how the Informix Geodetic DataBlade module interacts with the `GL_DATETIME` environment variable.

Informix Geodetic DataBlade Module Data Types

The Informix Geodetic DataBlade module provides three kinds of data types:

- Spatio-temporal
- Domain-enforcing
- Range

You can use spatio-temporal types to represent objects. Domain-enforcing and range data types are *building blocks* that simplify construction and manipulation of spatio-temporal objects.

The Informix Geodetic DataBlade module's data types are user-defined data types. Refer to the `CREATE OPAQUE TYPE` statement in the [Informix Guide to SQL: Syntax](#) for more information on opaque data types.

Spatio-Temporal Data Types

Spatio-temporal data types allow you to store data that represent spatial objects with a time component in an INFORMIX-Universal Server database. The time component associates the object with a time period or moment in time. Spatio-temporal types are defined in a type hierarchy, with a supertype—GeoObject—that has the time and altitude attributes common to all spatio-temporal types.

With the exception of two types that have a more general purpose, spatio-temporal types are defined by one or more pairs of latitude and longitude coordinates on the datum ellipsoid, time values, and altitude values. Some spatio-temporal types also have one or more parametric values that define the shape and extent of the object.

The exceptions are GeoCoords and supertype GeoObject. The GeoCoords data type describes two-dimensional coordinate pairs in terms of position on the WGS-84 datum. It does not have a temporal component but can be used to construct spatio-temporal objects.

You can use the Informix Geodetic DataBlade module to represent the following objects in an INFORMIX-Universal Server database.

Object	Data Type
Point	GeoPoint
Line segment	Geolineseg
String	GeoString
Ring	GeoRing
Polygon	GeoPolygon
Box	GeoBox
Circle	GeoCircle
Ellipse	GeoEllipse
Coordinate pair	GeoCoords

Domain-Enforcing Data Types

Domain-enforcing types are user-defined data types represented internally as double-precision floating-point numbers and constructed using casts from numeric built-in types.

Domain-enforcing types provide angular components such as latitude and longitude, and linear components such as distance. You can use domain-enforcing types to define spatio-temporal objects. Many of the functions described in [Chapter 4](#) return domain-enforcing types or take them as arguments.

Domain-enforcing types support double-precision built-in operators. The behavior of some of the arithmetic operators has been modified so that they perform in useful ways on multidimensional data. [Chapter 3](#) describes each domain-enforcing type and explains the behavior of the arithmetic operators.

Range Data Types

Range data types specify the altitude and time dimensions of spatio-temporal data. The following are range data types:

- GeoAltRange
- GeoTimeRange

Internal Representation of Spatio-Temporal Data

Internally, a spatio-temporal object is represented by a C structure that contains, among other things, the geodetic coordinates, direction cosines, reference ellipsoid, and an *x-y-z* bounding box used by the R-tree access method.

Each spatio-temporal data type is supported by a collection of internal support functions. These include the input function and output function of the data type. The input function for a spatio-temporal data type converts the external representation to the internal representation. The output function converts the internal representation to the external representation.

Other support functions perform tasks such as casting the data type to other data types. Support functions are used internally and are not documented in this guide. For more information on support functions, see the CREATE OPAQUE TYPE statement in [Informix Guide to SQL: Syntax](#).

Large Objects

The Informix Geodetic DataBlade module includes support for arbitrarily large values (in terms of the number of bytes needed for the internal region). Small objects are stored in-row. If an object is larger than a defined threshold value, it is stored separately from the table in a *large object*. In this case, the row stores a reference to the large object but not the data itself.

This strategy allows you to store very large objects in a single row, even if the objects are larger than the maximum row size.

When you insert a spatio-temporal object in a table, internal support functions determine whether the object is large or small and store it appropriately. These support functions are called automatically whenever you perform an insert; no special user interaction is required to store or access these objects. If you modify an object and its size exceeds the defined threshold, the support functions promote the in-row object to a large object.

If your database contains large spatio-temporal objects, it is possible that some values may be stored as large objects, while other values are stored conventionally in rows. Since storage and access of these objects is the same whether they are stored in-row or as large objects, this feature requires no modification of your queries or applications. However, you must create an **sbospace** to store large objects. Refer to the **onspaces** utility in *INFORMIX-Universal Server Administrator's Guide* for details.

For general information about large objects, see Chapter 3 of the *Informix Guide to SQL: Reference*.

Refer to *INFORMIX-OnLine Dynamic Server Performance Guide* for information on planning configuration of your INFORMIX-Universal Server.

Inserting Spatio-Temporal Objects into a Table

To insert a spatio-temporal object into a table with the INSERT statement, put the text description of the object in quotes.

Another way to copy spatial data into a table is with the LOAD statement. The LOAD statement enables you to copy data from a client file into a database. See *Informix Guide to SQL: Syntax* for details on how to use the LOAD statement. You can also copy data from a database table to a file. See the UNLOAD statement in *Informix Guide to SQL: Syntax* for more information.

You can also use the High-Performance Loader (HPL) to load data. Refer to the [Guide to the High-Performance Loader](#) for more information.

Geodetic Functions

The Geodetic can be grouped in the following categories:

- **Accessor functions.** Accessors allow you to access information about objects stored in a database. For example, the **Latitude** function returns—or accesses—the latitude value of an object.
Type verification functions are included in this group.
- **Constructor and conversion functions.** Constructor functions are the basis for the data types described in [Chapter 3](#). Many of these functions are overloaded, with a conversion syntax in addition to a constructor. The conversion syntax converts an object from the GeoObject supertype back to its original representation.
- **Spatial operators.** Operators take spatio-temporal objects and test them for proximity or intersection. Spatial operators return Boolean `TRUE` or `FALSE`.
- **Data verification functions.** This group of functions performs verification of GeoRing and GeoPolygon objects.
- **System function.** The **GeoRelease** function is a system function.

The following table lists the functions by category.

Function Type	Function Name	
Accessor	AltRange	Latitude
	Azimuth	Longitude
	Begin	Major
	Bottom	Minor
	Coords	NPoints
	End	NRings
	IsGeoBox	Radius
	IsGeoCircle	Ring
	IsGeoEllipse	SetAltRange
	IsGeoLineseg	SetDist
	IsGeoPoint	SetTimeRange
	IsGeoPolygon	TimeRange
	IsGeoRing	Top
	IsGeoString	
Constructor and conversion	GeoAltRange	GeoPoint
	GeoBox	GeoPolygon
	GeoCircle	GeoRing
	GeoCoords	GeoString
	GeoEllipse	GeoTimeRange
	GeoLineseg	
Spatial operator	Beyond	
	Inside	
	Intersect	
	Outside	
	Within	
Data validation	IsValidGeometry	
	IsValidSDTS	
System	GeoRelease	

Client Validation Program

The Informix Geodetic DataBlade module provides a client utility called **geovalidate** that you can use to validate GeoObject data. This utility verifies the correctness of your input data and performs the same tests as the **IsValidGeometry** and **IsValidSDTS** server functions. You can use the **geovalidate** utility to verify that:

- any GeoObject is specified correctly.
- the vertices of a GeoRing form a valid g-ring.
- the vertices of a GeoPolygon form a valid g-ring.
- the g-rings of a GeoPolygon do not intersect one another.
- consecutive points of a GeoRing or GeoPolygon are distinct.
- a GeoPolygon meets the SDTS definition of a g-polygon. (Refer to [“IsValidSDTS” on page 4-44](#) for the SDTS definition of a g-polygon).

The **geovalidate** utility takes a standard LOAD file as input. You can specify the following options:

- Column(s) to verify (the default is the first column)
- Data type (the default is GeoPolygon)
- Column delimiter
- Allow/disallow GeoPolygons that do not meet the SDTS definition of a g-polygon (the default allows such polygons)
- An output file containing all valid input lines

Unlike the server-side functions **IsValidGeometry** and **IsValidSDTS**, the **geovalidate** utility continues processing when it encounters a violation. For each violation, it writes an error message to the output file. This enables you to process an entire input data set and examine the violations after processing completes.

The **geovalidate** utility syntax is:

```
geovalidate [-c nT] [-d delim] [-s] [-o outfile] filename
```

filename is the name of the file to validate.

The options are:

- c	<p><i>n</i> is a column of data that you wish to validate, where 1 is the first column in the table, 2 is the second, and so forth. The default is 1.</p> <p><i>T</i> is the data type. The types that are allowed are: R (GeoRing), P (GeoPolygon), and O (GeoObject). The default is P.</p> <p>To validate several columns of data, simply repeat the - c arguments for each column.</p>
- d	<p><i>delim</i> is the column delimiter character. You must enclose the delimiter in quotes. The default is set in the DBDELIMITER environment variable. If DBDELIMITER is not set, the default is the vertical bar (). The delimiter cannot be a backslash (\), new line, or hexadecimal number.</p>
- s	<p>allows only GeoPolygons that conform to the SDTS definition of g-polygon. If this option is not specified, disjoint polygons with multiple nesting levels are allowed.</p>
- o	<p><i>outfile</i> is the name of the file to which valid input lines are written.</p>

The expressions in square brackets ([]) are optional.

Selecting Objects by Altitude or Time

You can use the value ANY with the **GeoAltRange** function and the **GeoTimeRange** function to search for all objects that have non-null altitude or time ranges. This is useful when you search for objects that match in the spatial, or horizontal, dimension, while altitude or time is unimportant.

In the spatial dimension, on the other hand, you cannot use the value `ANY` to describe latitude or longitude. You can, however, still query the database for objects that match a specific time or altitude range and any latitude or longitude.

To do this, use a `GeoBox` data type to specify the entire Earth as your area of spatial interest. Your `GeoBox` extends from latitude `-90` to `90`, and longitude `-180` to `180`—the minimum and maximum latitude and longitude values allowed. The Informix Geodetic DataBlade module will return all spatio-temporal objects that match the time and altitude ranges in your query.

For more information on using the `GeoBox` data type and an example of how you can specify any latitude or longitude, refer to [“GeoBox” on page 3-19](#).

Setting Date and Time Precision

The Informix Geodetic DataBlade module uses the `GeoTimeRange` data type to specify a time range for spatio-temporal objects. The input, storage, and display of `GeoTimeRange` data are affected by the setting of the `GL_DATETIME` environment variable. This section describes the interactions of `GL_DATETIME` with the Informix Geodetic DataBlade module.

Default Setting

THE `GL_DATETIME` environment variable specifies the end-user formats of values in `DATETIME` columns.

A `GL_DATETIME` format can contain:

- one or more white-space characters.
- an ordinary character—for example, a colon to separate hours and minutes—other than the `%` symbol or white-space character.
- a formatting directive, which is composed of the `%` symbol followed by a conversion character that specifies the required replacement.

For example, the following is a typical format for `YEAR TO FRACTION (5)`, the maximum precision you can set:

```
%Y- %d- %m %H: %M: %S. %F5
```



The default precision for `GL_DATETIME` is `YEAR TO SECOND`. You can change the setting for `GL_DATETIME` using the **SetNet32** utility on a Windows NT machine or using the `setenv` command on a UNIX machine.

Tip: You can change other characteristics of `GL_DATETIME` besides the precision. For example, you can specify slashes instead of dashes as the delimiter between years, months, and days. For general information on `GL_DATETIME`, see the *Guide to GLS Functionality*.

Regardless of the setting for `GL_DATETIME`, when you insert `GeoTimeRange` data, you must use at least the `YEAR TO SECOND` format, and you must follow the seconds with a decimal point.

For example, assuming the precision is set to the default, `YEAR TO SECOND`, the following shows a correct input format. The results follow the example.

```
-- GL_DATETIME="%Y-%m-%d %H:%M:%S"
create table t (tr GeoTimeRange);
insert into t values (GeoTimeRange('1997-01-02 3:04:05.'));
select * from t;

tr (1997-01-02 03:04:05,1997-01-02 03:04:05)
```



Tip: The output for this example and the following examples shows the input value duplicated because `GeoTimeRange` represents a range. If you do not insert a second value, it is assumed to be the same as the first value.

Also note that this example creates a table, `t`, with a `GeoTimeRange` column. Subsequent examples in this and the following subsections do not explicitly create a table but assume that this same table has already been created.

In the following example, the first INSERT statement returns an error because it uses an input format of YEAR TO DAY, which is less than the required precision of YEAR TO SECOND. The second INSERT statement returns an error because there is no trailing decimal point.

```
insert into t values (GeoTimeRange('1997-01-02'));

1262: Non-numeric character in datetime or interval.
Error in line 1
Near character position 48

insert into t values (GeoTimeRange('1997-01-02 3:04:05'));

1262: Non-numeric character in datetime or interval.
Error in line 1
Near character position 56
```

Effect of GL_DATETIME on Input and Output

If you insert GeoTimeRange data with a precision less than that of GL_DATETIME, they are rejected as an error. In the following example, GL_DATETIME is set to YEAR TO FRACTION, but the GeoTimeRange data are inserted with a precision of YEAR TO SECOND, so an error is returned.



Important: To change the setting of GL_DATETIME, you can execute a `setenv` command from the UNIX prompt before launching the `dbaccess` program to execute SQL statements. You can also change the setting of GL_DATETIME using the Informix `Setnet32` utility program before executing SQL statements in SQL Editor.

```
-- GL_DATETIME="%Y-%m-%d %H:%M:%S.%F5"
insert into t values (GeoTimeRange('1997-01-02 3:04:05'));

1262: Non-numeric character in datetime or interval.
Error in line 1
Near character position 56
```

If you insert data with a precision greater than that of GL_DATETIME, the data are truncated to the precision specified by GL_DATETIME. For example, if GL_DATETIME is set to a precision of YEAR TO SECOND and data are entered at a precision of YEAR TO FRACTION, the input is truncated.

```
-- GL_DATETIME="%Y-%m-%d %H:%M:%S"
insert into t values (GeoTimeRange('1997-01-02 3:04:05.01234'));
select * from t;

tr  (1997-01-02 03:04:05,1997-01-02 03:04:05)
```

The setting of GL_DATETIME also affects the format of the GeoTimeRange data that you retrieve from the database. If you store data at a higher precision, the data are truncated on output. If you store data at a lower precision, the data are zero-extended on output.

In the following example, the data are stored at YEAR TO FRACTION (5) precision. The first SELECT statement retrieves the data at this precision. The precision is changed to YEAR TO SECOND before the second SELECT statement, so the output is truncated to the precision of YEAR TO SECOND.

```
-- GL_DATETIME="%Y-%m-%d %H:%M:%S.%F5"
insert into t values (GeoTimeRange('1997-01-02 3:04:05.01234'));
select * from t;

1 row(s) retrieved.

-- GL_DATETIME="%Y-%m-%d %H:%M:%S"
select * from t;

tr  (1997-01-02 03:04:05,1997-01-02 03:04:05)
```


In the following example, the data are input at YEAR TO FRACTION (5) precision. However, because GL_DATETIME is set to YEAR TO SECOND, the first SELECT statement retrieves the data at this precision. The precision is changed to YEAR TO FRACTION (5) before the second SELECT statement, so the output is displayed at this precision.

```
-- GL_DATETIME="%Y-%m-%d %H:%M:%S"
insert into t values (GeoTimeRange('1997-01-02 3:04:05.01234'));
select * from t;

tr (1997-01-02 03:04:05,1997-01-02 03:04:05)

-- GL_DATETIME="%Y-%m-%d %H:%M:%S.%F5"
select * from t;

tr (1997-01-02 03:04:05.01234,1997-01-02 03:04:05.01234)
```



Tip: You can use the **Begin** and **End** functions to return the beginning or ending time of a range. Each of these functions returns DATETIME values with five decimal digits (YEAR TO FRACTION (5)), regardless of the number of digits you provide on input and regardless of the setting of the GL_DATETIME environment variable.

Using DATETIME Casts

You can use the double colon operator (: :) to cast GeoTimeRange values to a specific precision. This allows you to specify a less precise input specification than the GL_DATETIME setting. It does not, however, override the GL_DATETIME setting, nor does it extend text input.

In the first part of the following example, GL_DATETIME is set to YEAR TO DAY, and the DATETIME cast allows you to specify a GeoTimeRange value with a precision less than the minimum default of YEAR TO SECOND. In the second part of the example, the cast allows you to specify less than the GL_DATETIME precision. The data, however, are zero-extended on input and output to YEAR TO SECOND.

```
-- GL_DATETIME="%Y-%m-%d"
insert into t values (GeoTimeRange('1997-01-02'::datetime year to day));
select * from t;

tr (1997-01-02,1997-01-02)

-- GL_DATETIME="%Y-%m-%d %H:%M:%S"
insert into t values (GeoTimeRange('1997-01-02'::datetime year to day));
select * from t;

tr (1997-01-02 00:00:00,1997-01-02 00:00:00)
```



Tip: *Instead of casting, you could insert data with hours, minutes, and seconds set to 0 (or any value). The hours, minutes, and seconds would be truncated to conform to the GL_DATETIME setting of YEAR TO DAY.*

In the following example, the INSERT statement returns an error message because the cast is to YEAR TO DAY, while YEAR TO SECOND data are inserted:

```
-- GL_DATETIME="%Y-%m-%d %H:%M:%S"
insert into t values (GeoTimeRange('1997-01-02 03:04:05'::datetime year to day));
select * from t;

1264: Extra characters at the end of a datetime or interval.
Error in line 1
Near character position 79
```

Geodetic Data Types

Domain-Enforcing Data Types	3-3
Range Data Types	3-4
Spatio-Temporal Data Types	3-4
Data Type Reference	3-4
GeoAltitude	3-5
GeoAltrange.	3-7
GeoAngle.	3-11
GeoAzimuth.	3-16
GeoBox	3-19
GeoCircle.	3-22
GeoCoords	3-25
GeoDistance.	3-27
GeoEllipse	3-29
GeoLatitude.	3-33
GeoLineseg	3-36
GeoLongitude	3-39
GeoObject	3-41
GeoPoint	3-44
GeoPolygon	3-47
GeoRing	3-53
GeoString.	3-56
GeoTimeRange	3-59

T

his chapter contains reference information on Informix Geodetic DataBlade module data types. Three categories of data types are provided with this module:

- Domain-enforcing data types
- Range data types
- Spatio-temporal data types

Domain-Enforcing Data Types

The following are *angular types* provided in the Informix Geodetic DataBlade module:

- GeoAngle
- GeoAzimuth
- GeoLatitude
- GeoLongitude

With the exception of the GeoAngle type, these types are defined relative to an absolute origin of North (GeoAzimuth), the equator (GeoLatitude), or the Greenwich Meridian (GeoLongitude).

The unit of measure for angular types is degrees (°) with a decimal fraction.

The following are *linear types* provided in the Informix Geodetic DataBlade module:

- GeoAltitude
- GeoDistance

The unit of measure for linear types is meters.

Range Data Types

Range data types specify the altitude and time dimensions of spatio-temporal data. The following are range data types:

- GeoAltrange
- GeoTimeRange

Spatio-Temporal Data Types

The following are spatio-temporal data types:

- GeoBox
- GeoCircle
- GeoCoords (latitude/longitude only)
- GeoEllipse
- GeoLineseg
- GeoPoint
- GeoPolygon
- GeoRing
- GeoString

Data Type Reference

The following sections contain reference information on Informix Geodetic DataBlade module data types. The data types are listed alphabetically. The functions referenced in this chapter are described in [Chapter 4, “Geodetic Functions.”](#)

GeoAltitude

GeoAltitude is the domain-enforcing type for altitude. Its domain is $-\infty$ to $+\infty$.

GeoAltitude is a “building block” for GeoAltrange, which defines altitude and altitude range for the spatio-temporal objects described later in this chapter. Refer to [“GeoAltrange” on page 3-7](#) for more information on this range data type.

Functions that Return this Type

The **Bottom** and **Top** functions take GeoAltrange data and return the bottom or top altitude value as GeoAltitude. For more information, refer to [Bottom on page 4-13](#) and [Top on page 4-59](#).

Usage

Operations involving GeoAltitude cannot result in an out-of-range error, but some operators are not allowed with all combinations of operands. For example, multiplying two GeoAltitude values is not supported.

The following matrix shows the operators you can use with GeoAltitude, and the results that the operations yield.

Left Operand	Operator	Right Operand	Result
GeoAltitude	= < <= > >= <>	GeoAltitude	Boolean
Double-precision	*	GeoAltitude	GeoAltitude
GeoAltitude		Double-precision	GeoAltitude
GeoAltitude	/	Double-precision	GeoAltitude
GeoAltitude		GeoAltitude	Double-precision
None	+	GeoAltitude	GeoAltitude
GeoAltitude		GeoAltitude	GeoAltitude
None	-	GeoAltitude	GeoAltitude
GeoAltitude		GeoAltitude	GeoAltitude

Examples

The following examples demonstrate valid values and the use of operators for GeoAltitude. The results follow the examples.

```
create table altitudes (altitude GeoAltitude);
insert into altitudes values (0);-- all values allowed
insert into altitudes values (20E+3);
```

```
insert into altitudes values (-20.0E+03);
select * from altitudes;
```

```
altitude  0.00
altitude  20000.000000000000000000
altitude  -20000.000000000000000000
```

```
create table altops (alt1 GeoAltitude,
                    num double precision,
                    alt2 GeoAltitude,
                    alt3 GeoAltitude);
```

```
insert into altops values (20E3::GeoAltitude / 1E3,
                          20E3::GeoAltitude / 2E3::GeoAltitude,
                          20E3::GeoAltitude + 40E3::GeoAltitude,
                          20E3::GeoAltitude - 10E3::GeoAltitude);
```

```
select * from altops;
```

```
alt1  20.000000000000000000
num    10.000000000000
alt2  60000.000000000000000000
alt3  10000.000000000000000000
```


GeoAltrange

GeoAltrange is a range data type that you can use to specify the altitude range for a spatio-temporal object. It consists of lower and upper GeoAltitude values representing the altitude range for the object, or a single GeoAltitude value representing the altitude of the object above the datum.

External Representation

The external representation of GeoAltrange has the following forms:

```
( alt1, alt2)
( alt)
ANY
```

where

alt1 is the lower altitude.

alt2 is the upper altitude.

Both **alt1** and **alt2** must be non-null GeoAltitude values, and **alt2** must be greater than or equal to **alt1**.

alt is a single value representing altitude. This syntax can only be used on input. **alt** is a non-null GeoAltitude value.

ANY matches any other range, except **NULL**. On input, **ANY** is not case sensitive; 'any', 'ANY', and 'Any' yield the same result.

The **alt1** and **alt2** length values can be specified in any way that is allowed for a double-precision value: 0, +17.2358, and 358.600E+03 are all possible values. On output, they appear as floating-point numbers; for example, 0.00, 17.2358000000000000, and 358600.0000000000000000.

For more information on specifying altitude values, refer to [“GeoAltitude” on page 3-5](#).

Constructors

GeoAltrange has two type constructors:

```
GeoAltRange(GeoAltitude, GeoAltitude)
GeoAltRange(GeoAltitude)
```

The type constructors take the GeoAltitude domain-enforcing type and return GeoAltrange.

For more information on the GeoAltrange type constructors, refer to [GeoAltRange on page 4-16](#).

Functions

The **AltRange** function takes a spatio-temporal object and returns the object's altitude range as GeoAltrange. For more information, refer to [AltRange on page 4-7](#).

To change the altitude range of a spatio-temporal object, use the **SetAltRange** function. **SetAltRange** takes a spatio-temporal object and a GeoAltrange. For more information, refer to [SetAltRange on page 4-55](#).

The **Bottom** and **Top** functions take a GeoAltrange and return the bottom or top altitude value, respectively. For more information, refer to [Bottom on page 4-13](#) and [Top on page 4-59](#).

Usage

The upper value of a GeoAltrange cannot be less than its lower value. The upper and lower values can be the same; in this case, the altitude range for the object is treated as a single altitude value.

On input, you can specify the value `ANY` to indicate that the object does not have any restrictions in the altitude dimension.

When you retrieve spatio-temporal objects, `ANY` intersects all other altitude ranges, except `NULL`. If there are no objects with an altitude range value of `NULL`, searching for `NULL` returns an empty result set.

Examples

The following examples show the `GeoAltRange` constructors. The results follow each set of examples.

```
create table altranges (altrange GeoAltRange);

insert into altranges values ('(0, 0)');
insert into altranges values ('(1E3, 2E3)');
insert into altranges values (null::GeoAltRange);
insert into altranges values ('any');
insert into altranges values ('(0)'::GeoAltRange);
insert into altranges values (GeoAltRange(0));
insert into altranges values (GeoAltRange(-200, 0));
insert into altranges values (GeoAltRange(-200, null));
insert into altranges values (GeoAltRange(10000.0));

select * from altranges;

altrange  (0.0,0.0)
altrange  (1000.0,2000.0)
altrange
altrange  ANY
altrange  (0.0,0.0)
altrange  (0.0,0.0)
altrange  (-200.0,0.0)
altrange
altrange  (10000.0,10000.0)
```

The following example creates a table named **altranges2**, inserts an altitude range consisting of an upper and lower value, and inserts a range consisting of a single value. It then uses the **Bottom** and **Top** functions to retrieve the upper and lower altitude components.

```
create table altranges2 (altrange GeoAltRange);
insert into altranges2 values (GeoAltRange(1E3, 2E3));
insert into altranges2 values (GeoAltRange(-200));
select Bottom(altrange), Top(altrange) from altranges2;

(expression)  1000.0000000000000000
(expression)  2000.0000000000000000
(expression)  -200.0000000000000000
(expression)  -200.0000000000000000
```

The following statements generate an error because the altitude range is invalid:

```
insert into altranges2 values (GeoAltRange(-100, -200));
-- bottom > top
insert into altranges2 values ('(1E5, 0)');
-- bottom > top

(UGEOF) - Invalid altitude range (-1.000000E+02, -
2.000000E+02): The top altitude must be greater than or equal
to the bottom altitude.
Error in line 1
Near character position 54

(UGEOF) - Invalid altitude range (1.000000E+05,
0.000000E+00): The top altitude must be greater than or equal
to the bottom altitude.
Error in line 1
Near character position 42
```

Refer to [“Error Messages”](#) for a list of error messages reported by the Informix Geodetic DataBlade module.

GeoAngle

GeoAngle is a domain-enforcing data type. A GeoAngle is a generic angle you can use as the second operand in expressions with angular data types.

Usage

The domain for GeoAngle is $-\infty$ to $+\infty$. However, because angular degrees are only unique modulo 360 degrees, GeoAngle values less than 0 or greater than or equal to 360 are automatically adjusted to the corresponding value between 0 and 360. (The range includes the value 0 but excludes the value 360.)

For example, if an operation results in an angle value of 405 degrees, which is equivalent to the angle 45, the result is adjusted and the GeoAngle value 45 is stored.

Expressions that subtract angular values from one another return a GeoAngle. For example, subtracting two GeoLongitude values from one another returns a GeoAngle.

Unary plus (+) has no effect on GeoAngle values, and unary minus (-) subtracts the GeoAngle value from 360.

Subtracting two GeoAngle values returns the unsigned difference and does not depend on operand order.

The following matrix shows the operators you can use with the GeoAngle type, and the results that the operations yield.

Left Operand	Operator	Right Operand	Result
GeoAngle	= < <= > >= <>	GeoAngle	Boolean
Double-precision GeoAngle	*	GeoAngle	GeoAngle
		Double-precision	GeoAngle
GeoAngle	/	Double-precision	GeoAngle
GeoAngle		GeoAngle	Double-precision
Any angular type		GeoAngle	Double-precision
None	+	GeoAngle	GeoAngle
GeoAngle		GeoAngle	GeoAngle
Any angular type		GeoAngle	Angular type
GeoAngle		Any angular type	Angular type
None	-	GeoAngle	GeoAngle
GeoAngle		GeoAngle	GeoAngle
Any angular type		GeoAngle	Angular type

Examples

The following examples demonstrate the results of using GeoAngle values with decimal fractional digits, values greater than 360, and negative values. The results follow each example.

```
create table angles (angle GeoAngle);
insert into angles values (0);-- decimal point optional
insert into angles values (90.245);-- decimal fraction ok
insert into angles values (360);-- wraps to 0
insert into angles values (575.0);-- adjusted mod 360
select * from angles;
```

```
angle 0.00
angle 90.245000000000000000
angle 0.00
angle 215.000000000000000000
```

```
create table angles2 (angle GeoAngle);
insert into angles2 values (-17);-- all of these are
insert into angles2 values (-360);-- adjusted mod 360
insert into angles2 values (-575.862);
select * from angles2;
```

```
angle 343.000000000000000000
angle 0.00
angle 144.138000000000000000
```

The following statements show the use of operators for GeoAngle:

```
create table angops (angle1 GeoAngle, angle2 GeoAngle);
insert into angops values (180,90);-- 180 > 90?
insert into angops values (180,-180);-- 180 > 180?
insert into angops values (180,425);-- 180 > 65?
insert into angops values (180,625);-- 180 > 265?
select angle1 > angle2 from angops;
```

```
(expression)
```

```
t
f
t
f
```

```
create table angles3 (angle1 GeoAngle, angle2 GeoAngle,
number double precision);
insert into angles3 values (180::GeoAngle * 3,
180::GeoAngle / 3,
180::GeoAngle / 60::GeoAngle);
select * from angles3;
```

```
angle1  180.0000000000000000
angle2  60.0000000000000000
number  3.0000000000000000
```

```
create table angles4 (angle GeoAngle);
insert into angles4 values
(-90::GeoAngle);-- casting a negative number
insert into angles4 values
(-(90::GeoAngle));-- negating a GeoAngle
insert into angles4 values
(180::GeoAngle - 90::GeoAngle);-- operand order not
insert into angles4 values
(90::GeoAngle - 180::GeoAngle);-- important
select * from angles4;
```

```
angle  270.0000000000000000
angle  270.0000000000000000
angle   90.0000000000000000
angle   90.0000000000000000
```


The following statement results in an error because the operator is not defined for these combinations of types:

```
select angle1 + 20 from angles3;-- can't add a number
select angle1 - 20 from angles3;-- can't subtract a number

9700: Routine (plus) ambiguous - more than one routine
resolves to given signature.
Error in line 1
Near character position 30

(UGEOJ) - Function Minus(GeoAngle,double precision) with the
specified type signature not supported.
Error in line 1
Near character position 1
```

Refer to [“Error Messages”](#) for a list of error messages reported by the Informix Geodetic DataBlade module.

GeoAzimuth

GeoAzimuth is a domain-enforcing data type.

Azimuth is the angle between a vector tangent to the datum surface and geodetic North at a given point. Azimuth is measured in degrees, clockwise (eastward) from North.

GeoAzimuth is the domain-enforcing data type for azimuth. Its domain is 0 to 360; 0 is included in the domain, but 360 is not included.

GeoAzimuth values less than 0 or greater than or equal to 360 are automatically adjusted to the corresponding value between 0 and 360 (in the same manner as GeoAngle).

For example, if an operation results in an angle value of 405 degrees, which is equivalent to the angle 45, the result is adjusted and the GeoAzimuth value 45 is stored.

Functions that Return this Type

The **Azimuth** function returns the azimuth of the major axis of an ellipse as a GeoAzimuth. For details, refer to [Azimuth on page 4-8](#).

Usage

GeoAzimuth is one of the arguments used to construct a GeoEllipse. For more information, refer to [GeoEllipse on page 4-21](#).

When you subtract two azimuths, the value that results is the smaller of the two possible angles between the two directions and does not depend on operand order.

The following matrix shows the operators you can use with the GeoAzimuth type, and the results that the operations yield.

Left Operand	Operator	Right Operand	Result
GeoAzimuth	= < <= > >= <>	GeoAzimuth	Boolean
Double-precision GeoAzimuth	*	GeoAzimuth Double-precision	GeoAzimuth GeoAzimuth
GeoAzimuth GeoAzimuth	/	Double-precision GeoAngle	GeoAzimuth Double-precision
None GeoAzimuth GeoAngle	+	GeoAzimuth GeoAngle GeoAzimuth	GeoAzimuth GeoAzimuth GeoAzimuth
None GeoAzimuth GeoAzimuth	-	GeoAzimuth GeoAzimuth GeoAngle	GeoAzimuth GeoAngle GeoAzimuth

Examples

The following examples demonstrate the values you can use with GeoAzimuth. The results follow each example.

```
create table azimuths (azimuth GeoAzimuth);
insert into azimuths values (0);-- all of these are
insert into azimuths values (90);-- accepted as-is
insert into azimuths values (270);
insert into azimuths values (180);

select * from azimuths;

azimuth 0.00
azimuth 90.0000000000000000
azimuth 270.0000000000000000
azimuth 180.0000000000000000
```

Examples

```
create table azops (azimuth GeoAzimuth,  
                   number    double precision,  
                   angle1    GeoAngle,  
                   angle2    GeoAngle,  
                   angle3    GeoAngle,  
                   azimuth2  GeoAzimuth);  
  
insert into azops values (90::GeoAzimuth / 3,  
                          90::GeoAzimuth / 30::GeoAngle,  
                          135::GeoAzimuth - 45::GeoAzimuth,  
                          135::GeoAzimuth - 270::GeoAzimuth,  
                          270::GeoAzimuth - 135::GeoAzimuth,  
                          135::GeoAzimuth - 90::GeoAngle);  
  
select * from azops;  
  
azimuth    30.000000000000000  
number     3.000000000000000  
angle1     90.000000000000000  
angle2     135.000000000000000  
angle3     135.000000000000000  
azimuth2   45.000000000000000
```

.

GeoBox

GeoBox is a spatio-temporal data type. A GeoBox is an area bounded by four orthogonal edges parallel to the coordinate axes. The edges are two meridians (constant longitude) and two parallels (constant latitude). The corners must be unique points and the GeoBox cannot degenerate to a point or line. A GeoBox includes its boundary and interior area.

Two coordinate pairs representing the southwest (minimum latitude and longitude) and northeast (maximum latitude and longitude) corners define a GeoBox.

A GeoBox is internally constructed as a polygon. The GeoBox constructor creates a polygonal representation of the GeoBox by inserting additional points in the southern and northern boundaries, no more than 0.5 degrees apart. The additional points are needed to form edges of constant latitude. The deviation of a geodesic segment from the parallel it approximates is always less than 50 meters.

Because a GeoBox is internally represented by a polygon, processing a GeoBox is no faster than processing a GeoPolygon.

Usage

To indicate an area of interest, it is generally better to use a GeoCircle or GeoPolygon rather than GeoBox. A GeoCircle is more easily constructed internally and yields better performance; it is the best choice if precision is not important. To specify an area more precisely, use a GeoPolygon and define intermediate points as needed.

For more information on using GeoCircle and GeoPolygon, refer to [“GeoCircle” on page 3-22](#) and [“GeoPolygon” on page 3-47](#).

You can use a GeoBox to specify the entire Earth. This is useful if you want to construct a query that searches for items with specific time and/or altitude ranges, and *any* latitude or longitude. Use the coordinate values `(-90, -180)`, `(90, 180)` to indicate the entire Earth.

External Representation

GeoBox has two external representations. The second format is preceded by the type name, GeoBox:

```
((latS, longW), (latN, longE), altrange, timerange)
GeoBox((latS, longW), (latN, longE), altrange, timerange)
```

where You can use either format to input data, but you must use the second

lat_S is the southern (minimum) latitude of the GeoBox.

long_W is the western longitude of the GeoBox.

lat_N is the northern (maximum) latitude of the GeoBox.

The northern latitude value must be greater than the southern latitude value.

long_E is the eastern longitude of the GeoBox.

altrange is the altitude range.

timerange is the time range.

You can use either format to input data, but you must use the second format if you are entering data in a GeoObject column. The second format is also the output format for data stored in a GeoObject column. For more information on GeoObject input and output formats, see [“GeoObject” on page 3-41](#).

Important: *Reversing the latitude values results in an error (the southern latitude cannot be greater than the northern latitude value). Reversing the longitude values, on the other hand, simply specifies a different box and does not result in an error.*



Constructors

The GeoBox type constructor syntax is:

```
GeoBox(GeoCoords, GeoCoords, GeoAltRange, GeoTimeRange)
```

The type constructor takes GeoCoords and domain-enforcing data types and returns GeoBox. For more information, see [GeoBox on page 4-17](#).

Conversion from GeoObject

The syntax for conversion from GeoObject to GeoBox is:

```
GeoBox( GeoObject )
```

For more information on the **GeoBox** conversion function, refer to [GeoBox on page 4-17](#).

Functions

The following functions take GeoBox as an argument or return GeoBox.

Functions that Take GeoBox Argument

[Coords on page 4-14](#)

Functions that Return GeoBox

[GeoBox on page 4-17](#)

***Tip:** Generally, functions that take GeoObject as an argument can also take a GeoBox or other spatio-temporal object. Exceptions are noted in [Chapter 4](#). Refer to [“GeoObject” on page 3-41](#) for a list of functions that take GeoObject as an argument.*



GeoCircle

GeoCircle is a spatio-temporal data type. GeoCircle is an area enclosed by a boundary that is the locus of all points equidistant from a specified center point. GeoCircle includes both its boundary and interior area. It is defined by the latitude and longitude coordinates of its center point and the length of its radius. The radius must be greater than 0 and is measured as a distance along a geodesic.

External Representation

GeoCircle has two external representations. The second format is preceded by the type name, GeoCircle:

```
(lat, long, r, altrange, timerange)
GeoCircle(lat, long, r, altrange, timerange)
```

where

lat is the latitude of the center point.

long is the longitude of the center point.

r is the length of the radius.

The length of the radius must be greater than 0 and less than or equal to 2 million meters (2,000 kilometers)

altrange is the altitude range.

timerange is the time range.

You can use either format to input data, but you must use the second format if you are entering data in a GeoObject column. The second format is also the output format for data stored in a GeoObject column. For more information on GeoObject input and output formats, see [“GeoObject” on page 3-41](#).

Constructors

The `GeoCircle` type constructor syntax is:

```
GeoCircle(GeoCoords, GeoDistance, GeoAltrange, GeoTimeRange)
```

The type constructor takes `GeoCoords` and domain-enforcing data types and returns `GeoCircle`. For more information on the **GeoCircle** type constructor, refer to [GeoCircle on page 4-18](#).

Conversion from GeoObject

The syntax for conversion from `GeoObject` to `GeoCircle` is:

```
GeoCircle(GeoObject)
```

For more information on the **GeoCircle** conversion function, refer to [GeoCircle on page 4-18](#).

Functions

The following functions take `GeoCircle` as an argument or return `GeoCircle`.

Functions that Take `GeoCircle` Argument

[Coords on page 4-14](#)

[Radius on page 4-53](#)

Functions that Return `GeoCircle`

[GeoCircle on page 4-18](#)



***Tip:** Generally, functions that take `GeoObject` as an argument can also take a `GeoCircle` or other spatio-temporal object. Exceptions are noted in [Chapter 4](#). Refer to [“GeoObject” on page 3-41](#) for a list of functions that take `GeoObject` as an argument.*

Examples

The following examples use GeoCircle to specify the location and extent of four airports in the United States. The results follow the examples.

```
create table AirPortCoverage (code char(3), city varchar(100), reach GeoCircle);

insert into AirPortCoverage values ('ORD', 'Chicago', 'GeoCircle((41.98, -87.90),
50000.0, ANY, ANY)');
insert into AirPortCoverage values ('SFO', 'San Francisco', '((37.619, -122.37),
70000, ANY, ANY)');
--Omit the prefix GeoCircle, since the column type is GeoCircle.
insert into AirPortCoverage values ('JFK', 'New York', '((40.777, -73.87), 100000,
ANY, ANY)');
insert into AirPortCoverage values ('IAH', 'Houston', '((29.98, -95.34), 80000,
ANY, ANY)');

select * from AirPortCoverage;
drop table AirPortCoverage;

code    ORD
city    Chicago
reach   GeoCircle((41.98,-87.90000000000001),50000.0,ANY,ANY)

code    SFO
city    San Francisco
reach   GeoCircle((37.619,-122.37),70000.0,ANY,ANY)

code    JFK
city    New York
reach   GeoCircle((40.777,-73.87),100000.0,ANY,ANY)

code    IAH
city    Houston
reach   GeoCircle((29.98,-95.34),80000.0,ANY,ANY)
```

GeoCoords

The GeoCoords type consists of a pair of latitude and longitude coordinates. GeoCoords is a “building block” for spatio-temporal objects. Its purpose is to relate spatio-temporal objects to the WGS-84 datum. A GeoCoords object represents a point location on the WGS-84 datum.

External Representation

The external representation of GeoCoords is:

(lat, long)

where

- *lat* is a latitude coordinate.
- *long* is a longitude coordinate.

Constructor

The GeoCoords type constructor syntax is:

`GeoCoords(GeoLatitude, GeoLongitude)`

The constructor takes GeoLatitude and GeoLongitude domain-enforcing types and returns GeoCoords. For more information on the **GeoCoords** type constructors, refer to [GeoCoords on page 4-20](#).

Functions

The following functions take GeoCoords as an argument or return GeoCoords.

Functions that Take GeoCoords Argument

GeoBox on page 4-17

GeoCircle on page 4-18

GeoEllipse on page 4-21

GeoPoint on page 4-24

Latitude on page 4-45

Longitude on page 4-46

Functions that Return GeoCoords

Coords on page 4-14

GeoCoords on page 4-20

GeoDistance

GeoDistance is the domain-enforcing type for distance. Its domain is 0 to 2 million meters (2,000 kilometers), inclusive.

Functions

The proximity functions **Beyond** and **Within** take GeoDistance as an argument. The **SetDist** function takes a GeoDistance and adds a distance parameter to a spatio-temporal object.

The functions **Major** and **Minor** return the length of the semi-major and semi-minor axes of an ellipse, respectively, as a GeoDistance.

Refer to [Chapter 4, “Geodetic Functions,”](#) for more information on these functions.

Usage

You can use GeoDistance to specify the radius of a GeoCircle and the semi-major and semi-minor axes of a GeoEllipse. GeoCircle and GeoEllipse are discussed in detail later in this chapter.

The following matrix shows the operators you can use with GeoDistance, and the results that the operations yield.

Left Operand	Operator	Right Operand	Result
GeoDistance	= < <= > >= <>	GeoDistance	Boolean
Double-precision	*	GeoDistance	GeoDistance
GeoDistance		Double-precision	GeoDistance
GeoDistance	/	Double-precision	GeoDistance
GeoDistance		GeoDistance	Double-precision
GeoDistance	+	GeoDistance	GeoDistance
None		GeoDistance	GeoDistance
GeoDistance	-	GeoDistance	GeoDistance

Examples

The examples below show the use of valid GeoDistance values. The results follow the examples.

```
create table distances (distance GeoDistance);
insert into distances values (0);-- non-negative values only
insert into distances values (20E+3);
select * from distances;
```

```
distance  0.00
distance  20000.000000000000000000
```

The following statement results in an error because the distance is a negative value, and therefore outside the range of allowed values for GeoDistance:

```
insert into distances values (-20E+3);-- no negative values

(UGE09) - Distance -20000 out of range: The value must be non-
negative and can be at most 2,000,000 m.
Error in line 1
Near character position 1
```

The following generates errors because the distances that result are outside the domain for GeoDistance:

```
insert into distances values (20E3::GeoDistance * (-3));
insert into distances values
(20E3::GeoDistance - 30E3::GeoDistance)

(UGE09) - Distance -20000 out of range: The value must be non-
negative and can be at most 2,000,000 m.
Error in line 1
Near character position 1
```

Refer to [“Error Messages”](#) for a list of error messages reported by the Informix Geodetic DataBlade module.

GeoEllipse

GeoEllipse is a spatio-temporal data type. GeoEllipse is an area enclosed by a boundary and defined by the latitude and longitude coordinates of its center point, the length of its semi-major and semi-minor axes, and the azimuth of the major axis. The length of the axes is measured along a geodesic and must be greater than 0. The length of the semi-minor axis must be less than or equal to the length of the semi-major axis.

A GeoEllipse includes both its boundary and interior area. It is internally constructed as a polygon. The outer ring of the polygon is the boundary of the GeoEllipse, and the four end points of the major and minor axes are included in the outer ring, with intermediate points inserted at reasonable intervals.

Because the vertices of the polygon are on the boundary of the original ellipse, its edges are slightly inside the boundary of the original ellipse, and the area of the resulting polygon is slightly smaller than the area of the original ellipse.



Tip: If both axes of a GeoEllipse are of the same length, the object is a circle. However, because GeoCircle and GeoEllipse are internally represented in different ways, the results of spatial operators using circular GeoEllipse objects will differ from the same operation using GeoCircle objects. Refer to [“GeoCircle” on page 3-22](#) for information on how a GeoCircle is represented.

External Representation

GeoEllipse has two external representations. The second format is preceded by the type name, GeoEllipse:

```
((lat, long), semi_maj, semi_min, az, altrange, timerange)  
GeoEllipse((lat, long), semi_maj, semi_min, az, altrange, timerange)
```

where

<i>lat</i>	is the latitude of the center point.
<i>long</i>	is the longitude of the center point.
<i>semi_maj</i>	is the length of the semi-major axis (the length must be greater than 0 and less than 2 million meters).
<i>semi_min</i>	is the length of the semi-minor axis (the length must be greater than 0 and less than or equal to the length of the semi-major axis).
<i>az</i>	is the azimuth of the major axis.
<i>altrange</i>	is the altitude range.
<i>timerange</i>	is the time range.

You can use either format to input data, but you must use the second format if you are entering data in a GeoObject column. The second format is also the output format for data stored in a GeoObject column. For more information on GeoObject input and output formats, see [“GeoObject” on page 3-41](#).

Constructors

The GeoEllipse type constructor syntax is:

```
GeoEllipse(GeoCoords, GeoDistance, GeoDistance, GeoAzimuth,  
GeoAltrange, GeoTimeRange)
```

The type constructor takes GeoCoords and domain-enforcing data types and returns GeoEllipse. For more information on the **GeoEllipse** type constructor, refer to [GeoEllipse on page 4-21](#).

Conversion from GeoObject

The syntax for conversion from GeoObject to GeoEllipse is:

```
GeoEllipse(GeoObject)
```

For more information on the **GeoEllipse** conversion function, refer to [GeoEllipse on page 4-21](#).

Functions

The following functions take GeoEllipse as an argument or return GeoEllipse.

Functions that Take GeoEllipse Argument

[Azimuth on page 4-8](#)

[Coords on page 4-14](#)

[Major on page 4-47](#)

[Minor on page 4-48](#)

Functions that Return GeoEllipse

[GeoEllipse on page 4-21](#)



***Tip:** Generally, functions that take GeoObject as an argument can also take a GeoEllipse or other spatio-temporal object. Exceptions are noted in [Chapter 4](#). Refer to “GeoObject” on page 3-41 for a list of functions that take GeoObject as an argument.*

Examples

The following example uses GeoEllipse to specify the location and extent of four airports in the United States:

```
create table AirPortCoverage (code      char(3),
                             city      varchar(100),
                             reach     GeoEllipse);

insert into AirPortCoverage values ('ORD', 'Chicago', 'GeoEllipse((41.98, -87.90),
50000.0, 40000, 90.0, ANY, ANY)');
insert into AirPortCoverage values ('SFO', 'San Francisco', '((37.619, -122.37),
70000, 50000, 90.0, ANY, ANY)');
--Omit the prefix GeoEllipse, since the column type is GeoEllipse.
insert into AirPortCoverage values ('JFK', 'New York', '((40.777, -73.87), 100000,
60000, 90.0, ANY, ANY)');
insert into AirPortCoverage values ('IAH', 'Houston', '((29.98, -95.34), 80000,
50000, 90.0, ANY, ANY)');

select * from AirPortCoverage;

code   ORD
city   Chicago
reach  GeoEllipse((41.98,-87.90000000000001),50000.0,40000.0,90.0,ANY,ANY)

code   SFO
city   San Francisco
reach  GeoEllipse((37.619,-122.37),70000.0,50000.0,90.0,ANY,ANY)

code   JFK
city   New York
reach  GeoEllipse((40.777,-73.87),100000.0,60000.0,90.0,ANY,ANY)

code   IAH
city   Houston
reach  GeoEllipse((29.98,-95.34),80000.0,50000.0,90.0,ANY,ANY)
```

GeoLatitude

GeoLatitude is a domain-enforcing data type. It represents geodetic latitude. The GeoLatitude domain is -90 through +90, inclusive.

Functions that Return this Type

The **Latitude** function described in [Chapter 4](#) returns GeoLatitude.

Usage

You can use GeoLatitude and GeoLongitude to construct a GeoCoords object. For details, refer to [GeoCoords on page 4-20](#).

Subtracting two latitudes returns the unsigned difference and does not depend on operand order.

The following matrix shows the operators you can use with GeoLatitude, and the results that the operations yield.

Left Operand	Operator	Right Operand	Result
GeoLatitude	= < <= > >= <>	GeoLatitude	Boolean
Double-precision	*	GeoLatitude	GeoLatitude
GeoLatitude		Double-precision	GeoLatitude
GeoLatitude	/	Double-precision	GeoLatitude
GeoLatitude		GeoAngle	Double-precision
None	+	GeoLatitude	GeoLatitude
GeoLatitude		GeoAngle	GeoLatitude
GeoAngle		GeoLatitude	GeoLatitude
None	-	GeoLatitude	GeoLatitude
GeoLatitude		GeoLatitude	GeoAngle
GeoLatitude		GeoAngle	GeoLatitude

Examples

The following examples demonstrate the values you can use with GeoLatitude. The results follow the examples.

```
create table latitudes (latitude GeoLatitude);
insert into latitudes values (0);
insert into latitudes values (90);-- upper bound included
insert into latitudes values (-90);-- lower bound included
select * from latitudes;
```

```
latitude 0.00
latitude 90.0000000000000000
latitude -90.0000000000000000
```

The following statements result in an error because the latitude is out of range:

```
insert into latitudes values (180);-- too big
insert into latitudes values (-120);-- too big (negative)

(UGE07) - Latitude 180 out of range: A GeoLatitude value must
be in the range -90 (inclusive) to 90 (inclusive).
Error in line 1
Near character position 1

(UGE07) - Latitude -120 out of range: A GeoLatitude value must
be in the range -90 (inclusive) to 90 (inclusive).
Error in line 1
Near character position 1
```

The example below shows the subtraction logic for GeoLatitude:

```
create table llops (angle GeoAngle);

insert into llops values (60::GeoLatitude - 30::GeoLatitude);
insert into llops values (30::GeoLatitude - 60::GeoLatitude);

select * from llops;

angle 30.0000000000000000
angle 30.0000000000000000
```

The following statement results in an error because the operator is not defined for this combination of types:

```
insert into llops values (90::GeoLatitude - 45::GeoLongitude);  
drop table llops;
```

```
9700: Routine (minus) ambiguous - more than one routine resolves to given  
signature.
```

```
Error in line 1
```

```
Near character position 60
```

Refer to “[Error Messages](#)” for a list of error messages reported by the Informix Geodetic DataBlade module.



GeoLineSeg

GeoLineSeg is a spatio-temporal data type. GeoLineSeg is a direct line between two points. It is defined by the latitude and longitude of its end points, an altitude range, and a time range.

A line segment is represented as a geodesic. On an ellipsoidal datum such as the one used by this DataBlade module, a geodesic is the shortest path between two points.

The geodesic represented by a GeoLineSeg is the shorter of the two possible ways that the end points can be connected. The distance measured along the GeoLineSeg is always less than half the circumference of the Earth.

Tip: To represent a line that is not a geodesic—such as a parallel—use a *GeoString* object to insert additional intermediate points.

External Representation

GeoLineSeg has two external representations. The second format is preceded by the type name, GeoLineSeg:

```
((lat1, long1), (lat2, long2), altrange, timerange)
GeoLineSeg((lat1, long1), (lat2, long2), altrange, timerange)
```

where

lat1 and *long1* are the latitude and longitude coordinates of the line segment's beginning point.

lat2 and *long2* are the latitude and longitude coordinates of the line segment's ending point.

altrange is the altitude range.

timerange is the time range.

You can use either format to input data, but you must use the second format if you are entering data in a GeoObject column. The second format is also the output format for data stored in a GeoObject column. For more information on GeoObject input and output formats, see [“GeoObject” on page 3-41](#).

Constructor

The `GeoLineSeg` type constructor syntax is:

```
GeoLineSeg(GeoCoords, GeoCoords, GeoAltrange, GeoTimeRange)
```

The type constructor takes `GeoCoords` and domain-enforcing data types and returns `GeoLineSeg`. For more information on the **GeoLineSeg** type constructor, refer to [GeoLineSeg on page 4-23](#).

Conversion from GeoObject

The syntax for conversion from `GeoObject` to `GeoLineSeg` is:

```
GeoLineSeg(GeoObject)
```

For more information on the **GeoLineSeg** conversion function, refer to [GeoLineSeg on page 4-23](#).

Functions

The following functions take `GeoLineSeg` as an argument or return `GeoLineSeg`.

Functions that Take `GeoLineSeg` Argument
[Coords on page 4-14](#)

Functions that return `GeoLineSeg`
[GeoLineSeg on page 4-23](#)



***Tip:** Generally, functions that take `GeoObject` as an argument can also take a `GeoLineSeg` or other spatio-temporal object. Exceptions are noted in [Chapter 4](#). Refer to “[GeoObject](#)” on page 3-41 for a list of functions that take `GeoObject` as an argument.*

Examples

The following example uses GeoLineseg to represent the routes between three pairs of airports. The results follow the example.

```
create table AirRoutes (origin      char(3),
                        destination char(3),
                        route       GeoLineseg);

insert into AirRoutes values ('SFO', 'ORD', '((37.619, -122.37), (41.98, -87.90),
ANY, ANY)');
insert into AirRoutes values ('ORD', 'JFK', 'GeoLineseg((41.98, -87.90), (40.777,
-73.87), ANY, ANY)');
insert into AirRoutes values ('SEA', 'BOS', 'GeoLineseg((47.45, -122.31), (42.36,
-71.01), ANY, ANY)');

select * from AirRoutes;
```

origin	SFO
destination	ORD
route	GeoLineseg((37.619,-122.37),(41.98,-87.90000000000001),ANY,ANY)
origin	ORD
destination	JFK
route	GeoLineseg((41.98,-87.90000000000001),(40.777,-73.87),ANY,ANY)
origin	SEA
destination	BOS
route	GeoLineseg((47.45,-122.31),(42.36,-71.01000000000001),ANY,ANY)

GeoLongitude

GeoLongitude is the domain-enforcing type for longitude. Its domain is -180 to +180, inclusive.

Functions that Return this Type

The **Longitude** function described in [Chapter 4](#) returns GeoLongitude.

Usage

You can use the GeoLatitude and GeoLongitude types to construct a GeoCoords object. For details, refer to [GeoCoords on page 4-20](#).

Subtracting two longitudes is sensitive to operand order. Subtracting returns the positive angle between the two, traveling east from the second longitude to the first longitude.

The following matrix shows the operators you can use with the GeoLongitude type, and the results that the operations yield.

Left Operand	Operator	Right Operand	Result
GeoLongitude	= < <= > >= <>	GeoLongitude	Boolean
Double-precision GeoLongitude	*	GeoLongitude Double-precision	GeoLongitude GeoLongitude
GeoLongitude GeoLongitude	/	Double-precision GeoAngle	GeoLongitude Double-precision
None GeoLongitude GeoAngle	+	GeoLongitude GeoAngle GeoLongitude	GeoLongitude GeoLongitude GeoLongitude
None GeoLongitude GeoLongitude	-	GeoLongitude GeoLongitude GeoAngle	GeoLongitude GeoAngle GeoLongitude

Examples

The following examples demonstrate the values you can use with GeoLongitude. The results follow the examples.

```
create table longitudes (longitude GeoLongitude);
insert into longitudes values (0);
insert into longitudes values (90);
insert into longitudes values (180);-- upper bound included
insert into longitudes values (-180);-- lower bound included
select * from longitudes;

longitude  0.00
longitude  90.000000000000000000
longitude  180.0000000000000000
longitude -180.0000000000000000
```

The following statements result in an error because the longitude is out of range:

```
insert into longitudes values (575);-- too big
insert into longitudes values (-575);-- too big (negative)

(UGE08) - Longitude 575 out of range: A GeoLongitude value
must be in the range -180 (inclusive) to 180 (inclusive).
Error in line 1
Near character position 1

(UGE08) - Longitude -575 out of range: A GeoLongitude value
must be in the range -180 (inclusive) to 180 (inclusive).
Error in line 1
Near character position 1
```

Refer to [“Error Messages”](#) for a list of error messages reported by the Informix Geodetic DataBlade module.

GeoObject

GeoObject is the supertype for spatio-temporal objects. GeoObject represents the elements that are common to all spatio-temporal objects:

- Altitude range
- Time range

GeoObject has a text representation and is supported by output, import, and export functions. A GeoObject is always associated with a derived type. The external representation contains the name of the derived type.

The types that derive from GeoObject are:

- GeoBox
- GeoCircle
- GeoEllipse
- GeoLineseg
- GeoPoint
- GeoPolygon
- GeoRing
- GeoString

Usage

You can create a GeoObject column, but you can only store types derived from GeoObject in such a column. You cannot directly insert a GeoObject in a database table.

When you insert a derived spatio-temporal object in a GeoObject column, the object is cast to a GeoObject. The cast is transparent to your application and does not alter the values that you store in any way.

On output, data in a GeoObject column consists of:

- data type name.
- the original, type-specific external representation.

For example, the representation of a `GeoPoint` stored in a `GeoObject` column is:

```
GeoPoint(external_representation)
```

where *external_representation* contains the external representation of a `GeoPoint` object.

To manipulate objects stored in a `GeoObject` column, you must use a function to convert the object back to its original representation. Each spatio-temporal type has a conversion function for this purpose, described later in this chapter.

Functions

You can use any of the spatio-temporal types derived from `GeoObject` as an argument in functions that take `GeoObject`. That is, a function that lists `GeoObject` as an allowed argument will take any of the `GeoObject`-derived spatio-temporal types listed in [“GeoObject” on page 3-41](#).

The following functions described in [Chapter 4](#) take `GeoObject` as an argument:

Functions that Take `GeoObject` Argument

[AltRange on page 4-7](#)
[Beyond on page 4-10](#)
[GeoBox on page 4-17](#)
[GeoCircle on page 4-18](#)
[GeoEllipse on page 4-21](#)
[GeoPoint on page 4-24](#)
[GeoPolygon on page 4-25](#)
[GeoString on page 4-28](#)
[Inside on page 4-30](#)
[Intersect on page 4-32](#)
[IsGeoBox on page 4-34](#)
[IsGeoCircle on page 4-35](#)
[IsGeoEllipse on page 4-36](#)
[IsGeoLineseg on page 4-37](#)
[IsGeoPoint on page 4-38](#)
[IsGeoPolygon on page 4-39](#)
[IsGeoRing on page 4-40](#)
[IsGeoString on page 4-41](#)

Functions that Return `GeoObject`

[SetAltRange on page 4-55](#)
[SetDist on page 4-56](#)
[SetTimeRange on page 4-57](#)

Functions that Take GeoObject Argument

Outside on page 4-51

SetAltRange on page 4-55

SetDist on page 4-56

SetTimeRange on page 4-57

TimeRange on page 4-58

Within on page 4-60

Functions that Return GeoObject

The following functions return GeoObject:

SetTimeRange on page 4-57

GeoPoint

GeoPoint is the spatio-temporal data type for four-dimensional point objects. GeoPoint is a GeoObject derived type. It consists of a latitude, longitude pair and an altitude range value that together specify the location of the point relative to the datum and a time range that associates the point with a time.

External Representation

The GeoPoint external representations are:

```
((lat, long), altrange, timerange))
GeoPoint((lat, long), altrange, timerange))
```

where

lat and *long* are the latitude and longitude of the GeoPoint.

altrange is the altitude range.

timerange is the time range.

Constructor

GeoPoint has two external representations. The second format is preceded by the type name, GeoPoint:

```
GeoPoint(GeoCoords, GeoAltrange, GeoTimeRange)
```

You can use either format to input data, but you must use the second format if you are entering data in a GeoObject column. The second format is also the output format for data stored in a GeoObject column. For more information on GeoObject input and output formats, see [“GeoObject” on page 3-41](#).

The type constructor takes GeoCoords and domain-enforcing data types and returns GeoPoint. For more information on the **GeoPoint** type constructor, refer to [GeoPoint on page 4-24](#).

Conversion from GeoObject

The syntax for conversion from GeoObject to GeoPoint is:

```
GeoPoint(GeoObject)
```

For more information on the **GeoPoint** conversion function, refer to [GeoPoint on page 4-24](#).

Functions

The following functions take GeoPoint as an argument or return GeoPoint.

Functions that Take GeoPoint Argument

[Coords on page 4-14](#)

Functions that Return GeoPoint

[GeoPoint on page 4-24](#)



***Tip:** Generally, functions that take GeoObject as an argument can also take a GeoPoint or other spatio-temporal object. Exceptions are noted in [Chapter 4](#). Refer to “GeoObject” on page 3-41 for a list of functions that take GeoObject as an argument.*

Example

The following example uses GeoPoint to specify the location of four airports in the United States. The results follow the example.

```
create table AirPorts (code      char(3),
                      city      varchar(100),
                      location   GeoPoint);

insert into AirPorts values ('ORD', 'Chicago', 'GeoPoint((41.98, -87.90), ANY,
ANY)');
insert into AirPorts values ('SFO', 'San Francisco', '((37.619, -122.37), ANY,
ANY)');
--Omit the prefix GeoPoint, since the column type is GeoPoint.
insert into AirPorts values ('JFK', 'New York', '((40.777, -73.87), ANY, ANY)');
insert into AirPorts values ('IAH', 'Houston', '((29.98, -95.34), ANY, ANY)');

select * from AirPorts;

code      ORD
city      Chicago
location  GeoPoint((41.98,-87.90000000000001),ANY,ANY)

code      SFO
city      San Francisco
location  GeoPoint((37.619,-122.37),ANY,ANY)

code      JFK
city      New York
location  GeoPoint((40.777,-73.87),ANY,ANY)

code      IAH
city      Houston
location  GeoPoint((29.98,-95.34),ANY,ANY)
```


GeoPolygon

GeoPolygon is a spatio-temporal data type that consists of one or more outer rings and zero or more inner rings. GeoPolygon is defined by the latitude and longitude coordinates of the points that make up its ring(s). No inner or outer ring can intersect or touch any other ring of the same GeoPolygon.

The Informix Geodetic DataBlade module's definition of a GeoPolygon differs from the SDTS definition of g-polygon. In the SDTS definition of g-polygon:

- only one outer ring is allowed.
- the first ring specified is the outer ring.
- inner rings cannot be nested.

The Informix Geodetic DataBlade module allows:

- multiple nesting levels of inner rings.
- multiple outer rings.
- rings to be specified in any order.

Figure 3-1 shows an example of a valid GeoPolygon that *does not* meet the SDTS definition of a g-polygon.

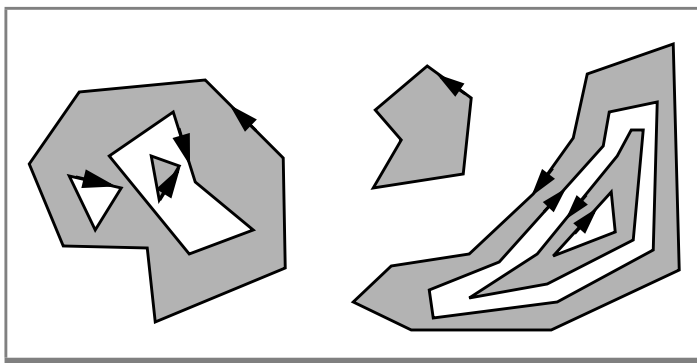


Figure 3-1
*GeoPolygon with
Three Outer Rings
and Nested Inner
Rings*

The GeoPolygon input function does not verify that your data meets these criteria. You can, however, perform data validation in a separate step. Separating validation from input provides better performance and enables you to skip validation if your data has already been validated or if you know it to be correct.

The functions that perform validation are:

- **IsValidGeometry on page 4-42.** Validates that data meets the GeoPolygon definition.
- **IsValidSDTS on page 4-44.** Checks that data meets the SDTS g-polygon definition.

You can also use the **geovalidate** utility to validate that your data meets the GeoPolygon or SDTS definitions. See “[Client Validation Program](#)” on [page 2-9](#) for more information on the **geovalidate** utility.

External Representation

GeoPolygon has two external representations. The second format is preceded by the type name, GeoPolygon:

```
((((lat11, long11), (lat21, long21), ... (latn1, longn1)),
 ((lat12, long12), (lat22, long22), ... (latn2, longn2))), ...
 ((lat1N, long1N), (lat2N, long2N), ... (latnN, longnN))),
 altrange, timerange)

GeoPolygon(((lat11, long11), (lat21, long21), ... (latn1, longn1)),
 ((lat12, long12), (lat22, long22), ... (latn2, longn2))), ...
 ((lat1N, long1N), (lat2N, long2N), ... (latnN, longnN))),
 altrange, timerange)
```

where

N is the number of rings.

altrange is the altitude range.

timerange is the time range.

The points of each ring are denoted as follows:

$latn_N$ and $longn_N$ are the latitude and longitude coordinates of the vertex points ring.

Vertices are numbered 1 through n , consecutively. Each ring must have at least 3 vertices.

The subscript identifies the ring. Rings can be entered in any order, and are numbered 1 through N .

You can use either format to input data, but you must use the second format if you are entering data in a GeoObject column. The second format is also the output format for data stored in a GeoObject column. For more information on GeoObject input and output formats, see [“GeoObject” on page 3-41](#).

Usage

You must specify the points in the outer rings in a *counterclockwise* direction, assuming that you are viewing the polygon from above the Earth. In this convention, the interior is on the left as you traverse the outer ring.

The purpose of specifying the outer rings in this way is to allow you to describe a bounded area on an ellipsoidal Earth where both the interior and exterior are valid areas. This convention enables you to distinguish between the representation of the Northern and Southern hemispheres, where both regions have the same outer ring—the equator. If the equator is defined by points in an eastward sequence, the polygon describes the Northern hemisphere; a westward sequence of points describes a polygon representing the Southern hemisphere.



Tip: The Informix Geodetic DataBlade module allows you to define any polygon on the Earth, regardless of its size. You can use the **IsValidGeometry** function to help you identify polygons in which the vertex points of the outer ring may have been incorrectly defined. For more information, see [IsValidGeometry on page 4-42](#).

If a polygon has inner rings, the direction in which they are specified alternates between clockwise and counterclockwise. First-level inner rings define exclusion areas, cutting a hole in the polygon; their points are specified in *clockwise* order. As with the outer ring, the interior of the polygon is on the left as you traverse the inner ring in a clockwise direction. The exclusion area is on the right.

The points that make up the outer and inner rings must be unique; vertices cannot be repeated. A polygon must have a nonzero interior area and cannot collapse to a point or line.

The first and last points in each ring must be different; the last point connects with the first point to close the ring.

Important: Do not repeat the first point of a ring to enforce closure. Doing so results in an error.



Conversion from GeoObject

The GeoPolygon conversion function is:

```
GeoPolygon( GeoObject )
```

For more information on this function, see [GeoPolygon on page 4-25](#).

Functions

The following functions take GeoPolygon as an argument or return GeoPolygon.

Functions that Take GeoPolygon Argument

[Coords on page 4-14](#)

[NPoints on page 4-49](#)

[NRings on page 4-50](#)

Functions that Return GeoPolygon

[GeoPolygon on page 4-25](#)



***Tip:** Generally, functions that take GeoObject as an argument can also take a GeoPolygon or other spatio-temporal object. Exceptions are noted in [Chapter 4](#). Refer to “GeoObject” on page 3-41 for a list of functions that take GeoObject as an argument.*

Example

The following example uses a GeoPolygon to create three sales regions bounded by sets of airports. The results follow the examples.

```
create table SalesRegion (name      char(4),
                        region      GeoPolygon);

--Region 1: SFO, ORD, JFK, IAH, SFO
insert into SalesRegion values ('REG1', '((((37.619, -122.37), (41.98, -87.90),
(40.777, -73.87), (29.98, -95.34))), ANY, ANY)');
--Add the prefix GeoPolygon in front of the input for clarity
--Region 2: LAX, SEA, SAN, MIA, LAX
insert into SalesRegion values ('REG2', 'GeoPolygon((((33.94, -118.41), (47.45, -
122.31), (32.73, -117.19), (25.79, -80.29))), ANY, ANY)');
--Region 3: (BOS, DTW, CIN), (IAH, MIA, SAN, ATL)
insert into SalesRegion values ('REG3', 'GeoPolygon(((42.36, -71.01), (42.21, -
83.35), (39.05, -84.66)), ((29.98, -95.34), (25.79, -80.29), (32.73, -117.19),
(33.64, -84.43))), ANY, ANY)');

select * from SalesRegion;

name      REG1
region    GeoPolygon((((37.619,-122.37),(41.98,-87.90000000000001),(40.777,-73.87
),(29.98,-95.34))),ANY,ANY)

name      REG2
region    GeoPolygon((((33.94,-118.41),(47.45,-122.31),(32.73,-117.19),(25.79,-80
.2900000000000001))),ANY,ANY)

name      REG3
region    GeoPolygon((((42.36,-71.01000000000001),(42.21,-83.34999999999999),(39.
05,-84.66)),((29.98,-95.34),(25.79,-80.29000000000001),(32.73,-117.19),
(33.64,-84.43000000000001))),ANY,ANY)
```

GeoRing

GeoRing is a spatio-temporal data type. A GeoRing is a sequence of non-intersecting line segments with closure. It consists of the boundary but not the interior area. This interior must be a nonzero area—it cannot be collapsed to a line or point.

A GeoRing must have at least three points, and all points in the GeoRing must be unique. The line segments in a GeoRing cannot intersect or touch each other. The first and last points in a GeoRing are assumed to be unique and connect to create the ring.

The GeoRing input function does not verify that GeoRing data meets these criteria. You can perform validation in a separate step with the **IsValidGeometry** function or the **geovvalidate** utility. Separating validation from input provides better performance and enables you to skip validation if your data has already been validated or if you know it to be correct. For more information, see [IsValidGeometry on page 4-42](#) and [“Client Validation Program” on page 2-9](#).



Important: Do not repeat a ring's first point to enforce closure. Doing so results in an error.

External Representation

GeoRing has two external representations. The second format is preceded by the type name, GeoRing:

```
((lat1, long1), (lat2, long12), ...(latn, longn)),  
altrange, timerange)
```

```
GeoRing(((lat1, long1), (lat2, long12), ...(latn, longn)),  
altrange, timerange)
```

where

lat1 and *long1* are the latitude and longitude coordinates of the first point in the ring.

Subsequent points are numbered 2 through *n*; the number of points in the ring is *n*.

altrange is the altitude range and *timerange* is the time range.

You can use either format to input data, but you must use the second format if you are entering data in a GeoObject column. The second format is also the output format for data stored in a GeoObject column. For more information on GeoObject input and output formats, see [“GeoObject” on page 3-41](#).

Conversion from GeoObject

The syntax for conversion from GeoObject to GeoRing is:

```
GeoRing(GeoObject)
```

For more information on the **GeoRing** conversion function, refer to [GeoRing on page 4-27](#).

Functions

The following functions take GeoRing as an argument or return GeoRing.

Functions that Take GeoRing Argument

[Coords on page 4-14](#)

[NPoints on page 4-49](#)

Functions that Return GeoRing

[GeoRing on page 4-27](#)



Tip: Generally, functions that take *GeoObject* as an argument can also take a *GeoRing* or other spatio-temporal object. Exceptions are noted in [Chapter 4](#). Refer to “*GeoObject*” on page 3-41 for a list of functions that take *GeoObject* as an argument.

Examples

The following example uses a *GeoRing* to create sales routes for two different months, based on sets of airports. The results follow the example.

```
create table SalesBoundary (month      char(3),
                           boundary   GeoRing);
--Route: SFO, ORD, JFK, IAH, SFO
insert into SalesBoundary values ('Jan', '(((37.619, -122.37), (41.98, -87.90),
(40.777, -73.87), (29.98, -95.34))), ANY, ANY)');
--Add the prefix GeoRing in front of the input for clarity
--Route: LAX, SEA, SAN, MIA, LAX
insert into SalesBoundary values ('Apr', 'GeoRing(((33.94, -118.41), (47.45, -
122.31), (32.73, -117.19), (25.79, -80.29))), ANY, ANY)');
select * from SalesBoundary;

month      Jan
boundary   GeoRing(((37.619,-122.37),(41.98,-87.90000000000001),(40.777,-73.87),
(29.98,-95.34))),ANY,ANY)

month      Apr
boundary   GeoRing(((33.94,-118.41),(47.45,-122.31),(32.73,-117.19),(25.79,-80.2
90000000000001))),ANY,ANY)
```



GeoString

GeoString is a spatio-temporal data type. A GeoString is a connected, nonbranching sequence of line segments. A GeoString may intersect itself.

Tip: *Line segments are computed as geodesics (the shortest distance on the ellipsoid between two points). To represent a path that is not a geodesic, you may need to insert additional intermediate points.*

External Representation

GeoString has two external representations. The second format is preceded by the type name, GeoString:

```
((lat1, long1), (lat2, long2), ..., (latn, longn)), altrange,
timerange)
```

```
GeoString(((lat1, long1), (lat2, long2), ..., (latn, longn)),
altrange, timerange)
```

where

lat1 and *long1* are the latitude and longitude coordinates of the first point in the string.

Subsequent points are numbered 2 through *n*, consecutively. Consecutive points must be different from one another. The minimum number of points is two.

The number of points in the string is *n*.

altrange is the altitude range.

timerange is the time range.

You can use either format to input data, but you must use the second format if you are entering data in a GeoObject column. The second format is also the output format for data stored in a GeoObject column. For more information on GeoObject input and output formats, see [“GeoObject” on page 3-41](#).

Conversion from GeoObject

The syntax for conversion from GeoObject to GeoString is:

```
GeoString(GeoObject)
```

For more information on the **GeoString** conversion function, refer to [GeoString on page 4-28](#).

Functions

The following functions take GeoString as an argument or return GeoString.

Functions that Take GeoString as an Argument

[Coords on page 4-14](#)

[NPoints on page 4-49](#)

Functions that Return GeoString

[GeoString on page 4-28](#)



***Tip:** Generally, functions that take GeoObject as an argument can also take a GeoString or other spatio-temporal object. Exceptions are noted in [Chapter 4](#). Refer to “GeoObject” on page 3-41 for a list of functions that take GeoObject as an argument.*

Examples

The following example uses the GeoString data type to create sales routes for two different months, based on sets of airports. The results follow the example.

```
create table SalesRoutes (month      char(3),
                        route      GeoString);
--Route: SFO, ORD, JFK, IAH
insert into SalesRoutes values ('Jan', '(((37.619, -122.37), (41.98, -87.90),
(40.777, -73.87), (29.98, -95.34))), ANY, ANY)');
--Add the prefix GeoString in front of the input for clarity
--Route: LAX, SEA, SAN, MIA
insert into SalesRoutes values ('Apr', 'GeoString(((33.94, -118.41), (47.45, -
122.31), (32.73, -117.19), (25.79, -80.29))), ANY, ANY)');

select * from SalesRoutes;

month  Jan
route  GeoString(((37.619,-122.37),(41.98,-87.90000000000001),(40.777,-73.87),(
29.98,-95.34))),ANY,ANY)

month  Apr
route  GeoString(((33.94,-118.41),(47.45,-122.31),(32.73,-117.19),(25.79,-80.29
000000000001))),ANY,ANY)
```

GeoTimeRange

GeoTimeRange is a range data type used to specify a spatio-temporal object's time range. It consists of a single time value, or beginning and ending values, and represents the time or time period associated with the object, usually the time the data was collected.

GeoTimeRange uses the Informix DATETIME YEAR TO FRACTION (5) data type. The decimal fraction for seconds consists of one to five digits and is optional. For more information on DATETIME values, refer to [Informix Guide to SQL: Reference](#).



Important: The storage and output of DATETIME values are affected by the setting of the `GL_DATETIME` environment variable. See [“Setting Date and Time Precision” on page 2-11](#) for information on the interaction of `GL_DATETIME` with the Informix Geodetic DataBlade module.

External Representation

The external representation of GeoTimeRange is:

```
( time1, time2 )
( time )
ANY
```

where

`time1` is the begin time.

`time2` is the end time.

Both `time1` and `time2` are non-null DATETIME YEAR TO FRACTION (5) values.

`time` is a single value representing time. This syntax can only be used on input. `time` is a non-null DATETIME YEAR TO FRACTION (5) value.

`ANY` matches any other ranges, except `NULL`. On input, `ANY` is not case sensitive; `'any'`, `'ANY'`, and `'Any'` yield the same result.

Constructors

There are two type constructors for `GeoTimeRange`:

```
GeoTimeRange(DATETIME YEAR TO FRACTION(5), DATETIME YEAR TO  
FRACTION(5))
```

```
GeoTimeRange(DATETIME YEAR TO FRACTION(5))
```

The type constructors take `DATETIME` values and return `GeoTimeRange`.

For more information on the `GeoTimeRange` type constructors, refer to [GeoTimeRange on page 4-29](#).

Functions

The **TimeRange** function takes a spatial object and returns its time range as `GeoTimeRange`. For more information on this function, refer to [TimeRange on page 4-58](#).

The **SetTimeRange** function allows you to modify the time range of a spatial object. For more information, refer to [SetTimeRange on page 4-57](#).

You can use the **Begin** and **End** functions to return the beginning or ending time of a time range. For more information, refer to [Begin on page 4-9](#) and [End on page 4-15](#).

Usage

The upper value of a `GeoTimeRange` cannot be less than its lower value. If the upper and lower values are the same, the object has a single time value.

When you retrieve spatio-temporal objects, `ANY` intersects all other time ranges, except `NULL`. Searching for a `NULL` time range returns an empty result set.

Input Values

On input, seconds have an optional decimal fraction of up to five digits.

You can specify the value `ANY` on input to indicate that the object does not have any restrictions in the time dimension.

Output Values

The format of the output value depends on the setting of the `GL_DATETIME` environment variable. See [“Setting Date and Time Precision” on page 2-11](#) for information on the interaction of `GL_DATETIME` with the Informix Geodetic DataBlade module.

Examples

The following examples show the **GeoTimeRange** constructors. The results follow the examples.

```
create table timeranges (timerange GeoTimeRange);

insert into timeranges values ('(1996-08-05 00:00:00., 1996-08-05 23:59:59.99999)');
insert into timeranges values ('(1996-08-05 17:30:00.)');
insert into timeranges values (null::GeoTimeRange);
insert into timeranges values ('any');
insert into timeranges values (
GeoTimeRange('1996-08-05 17:30:00.));
insert into timeranges values (
GeoTimeRange('1996-08-05 17:30:00.', null));
select * from timeranges;

timerange  (1996-08-05 00:00:00,1996-08-05 23:59:59)
timerange  (1996-08-05 17:30:00,1996-08-05 17:30:00)
timerange
timerange  ANY
timerange  (1996-08-05 17:30:00,1996-08-05 17:30:00)
timerange
```

The following example creates the **timeranges2** table and inserts time range values in a **GeoTimeRange** column. It then uses the **Begin** and **End** functions to retrieve the beginning and ending time values for the items in this column.

```
create table timeranges2 (timerange GeoTimeRange);

insert into timeranges2 values (
    GeoTimeRange('1996-08-05 00:00:00.', '1996-08-05 23:59:59.99999'));
insert into timeranges2 values (GeoTimeRange('2000-01-01 00:00:00.'));
select Begin(timerange), End(timerange) from timeranges2;

(expression)                (expression)

1996-08-05 00:00:00.00000    1996-08-05 23:59:59.99999
2000-01-01 00:00:00.00000    2000-01-01 00:00:00.00000
```

Refer to [Begin on page 4-9](#) and [End on page 4-15](#) for information on the **Begin** and **End** functions.

The following statement generates an error because the time range is invalid:

```
insert into timeranges2 values (
    GeoTimeRange('2000-01-01 00:00:00.',
    '1996-08-05 17:30:00.'))-- begin > end

(UGEOG) - Invalid time range (2000-01-01 00:00:00, 1996-08-05 17:30:00): The end
time must be later than or equal to the begin time.
Error in line 3
Near character position 23
```

Refer to [“Error Messages”](#) for a list of error messages reported by the Informix Geodetic DataBlade module.

Geodetic Functions

Overview of Geodetic Functions	4-3
Accessor Functions	4-4
Constructors and Conversion Functions	4-4
Operator Functions	4-5
Data Validation Functions	4-5
System Function	4-5
Function Reference	4-6
AltRange	4-7
Azimuth	4-8
Begin	4-9
Beyond	4-10
Bottom.	4-13
Coords.	4-14
End	4-15
GeoAltRange	4-16
GeoBox	4-17
GeoCircle.	4-18
GeoCoords	4-20
GeoEllipse	4-21
GeoLineseg	4-23
GeoPoint	4-24
GeoPolygon	4-25
GeoRelease	4-26
GeoRing	4-27
GeoString.	4-28
GeoTimeRange	4-29
Inside	4-30

Intersect	4-32
IsGeoBox	4-34
IsGeoCircle	4-35
IsGeoEllipse	4-36
IsGeoLineSeg	4-37
IsGeoPoint	4-38
IsGeoPolygon	4-39
IsGeoRing.	4-40
IsGeoString	4-41
IsValidGeometry	4-42
IsValidSDTS	4-44
Latitude	4-45
Longitude.	4-46
Major	4-47
Minor	4-48
NPoints	4-49
NRings.	4-50
Outside	4-51
Radius	4-53
Ring.	4-54
SetAltRange	4-55
SetDist	4-56
SetTimeRange	4-57
TimeRange	4-58
Top	4-59
Within	4-60

T

his chapter provides information on Informix Geodetic DataBlade module functions. You can execute these functions interactively using DB-Access or send them from an application using the DataBlade API function **mi_exec**.

For information about DB-Access, see the [DB-Access User Manual](#). For information about the DataBlade API, see the [INFORMIX-Universal Server DataBlade API User Manual](#).

Overview of Geodetic Functions

You can use the functions described in this manual to construct spatial objects, access information and convert spatial objects from one data type to another, return information about the dimension (size) of a spatial object, and compare the position of spatial objects.

The functions can be grouped in the following categories:

- Accessor functions
- Constructor and conversion functions
- Operator functions
- Data validation functions
- System function

The sections that follow describe the Informix Geodetic DataBlade module's functions. The functions are presented in alphabetical order. Function names are boldfaced in text.

***Tip:** The text and many of the examples in this section show function names in mixed (upper and lower) case. Because INFORMIX-Universal Server is case insensitive, you do not need to enter function names exactly as shown. You can use upper, lower, or any combination of uppercase and lowercase.*



Accessor Functions

The following are the Informix Geodetic DataBlade module accessor functions:

[AltRange](#) on page 4-7
[Azimuth](#) on page 4-8
[Begin](#) on page 4-9
[Bottom](#) on page 4-13
[Coords](#) on page 4-14
[End](#) on page 4-15
[IsGeoBox](#) on page 4-34
[IsGeoCircle](#) on page 4-35
[IsGeoEllipse](#) on page 4-36
[IsGeoLineseg](#) on page 4-37
[IsGeoPoint](#) on page 4-38
[IsGeoPolygon](#) on page 4-39
[IsGeoRing](#) on page 4-40
[IsGeoString](#) on page 4-41
[Latitude](#) on page 4-45
[Longitude](#) on page 4-46
[Major](#) on page 4-47
[Minor](#) on page 4-48
[NPoints](#) on page 4-49
[NRings](#) on page 4-50
[Radius](#) on page 4-53
[Ring](#) on page 4-54
[SetAltRange](#) on page 4-55
[SetDist](#) on page 4-56
[SetTimeRange](#) on page 4-57
[TimeRange](#) on page 4-58
[Top](#) on page 4-59

Constructors and Conversion Functions

The following are the Informix Geodetic DataBlade module constructor and conversion functions:

[GeoAltRange](#) on page 4-16
[GeoBox](#) on page 4-17
[GeoCircle](#) on page 4-18
[GeoCoords](#) on page 4-20
[GeoEllipse](#) on page 4-21
[GeoLineseg](#) on page 4-23
[GeoPoint](#) on page 4-24
[GeoPolygon](#) on page 4-25
[GeoRing](#) on page 4-27
[GeoString](#) on page 4-28
[GeoTimeRange](#) on page 4-29

Operator Functions

The following are the Informix Geodetic DataBlade module operator functions:

[Beyond](#) on page 4-10
[Inside](#) on page 4-30
[Intersect](#) on page 4-32
[Outside](#) on page 4-51
[Within](#) on page 4-60

Data Validation Functions

The following are the Informix Geodetic DataBlade module data validation functions:

[IsValidGeometry](#) on page 4-42
[IsValidSDTS](#) on page 4-44

System Function

The Informix Geodetic DataBlade module system function is [GeoRelease](#) on page 4-26.

Function Reference

The following sections contain reference information on the Informix Geodetic DataBlade module functions. The functions are listed in alphabetical order.

AltRange

The **AltRange** function returns the altitude range of a spatio-temporal object as a GeoAltrange data type.

Syntax

```
AltRange( GeoObject )
```

For more information, refer to [“GeoObject” on page 3-41](#).

Return Type

The **AltRange** function returns a GeoAltrange data type. For more information on the GeoAltrange data type, refer to [“GeoAltrange” on page 3-7](#).

Azimuth

The **Azimuth** function takes an ellipse as its argument and returns the azimuth of the major axis.

Syntax

```
Azimuth(GeoEllipse)
```

For more information, refer to [“GeoEllipse” on page 3-29](#).

Return Type

The **Azimuth** function returns a GeoAzimuth data type. Refer to [“GeoAzimuth” on page 3-16](#) for more information on this data type.

Related Topics

[Coords on page 4-14](#)

[Major on page 4-47](#)

[Minor on page 4-48](#)

Begin

The **Begin** function returns the beginning time of a time range. **Begin** returns DATETIME values with five decimal digits, regardless of the number of digits you provide on input and regardless of the setting of the GL_DATETIME environment variable.

Syntax

```
Begin ( GeoTimeRange )
```

Return Type

The **Begin** function returns DATETIME YEAR TO FRACTION(5). If the time range is ANY, **Begin** returns 0001-01-01 00:00.00000.

Refer to “[GeoTimeRange](#)” on page 3-59 for information on the GeoTimeRange data type. See [Informix Guide to SQL: Reference](#) for information on the DATETIME data type.

Related Topics

[End on page 4-15](#)

Beyond

The **Beyond** function takes two spatio-temporal objects and a GeoDistance data type as arguments and returns `t` (TRUE) if *all* of the following are true:

- The objects do not spatially intersect.
- The point or segment on one object that is nearest any point or segment on the other object is farther away than the distance specified in the third argument.
- The time ranges of the objects intersect.

Otherwise, **Beyond** returns `f` (FALSE).

Syntax

```
Beyond(GeoObject, GeoObject, GeoDistance)
```

For more information, refer to [“GeoObject” on page 3-41](#) and [“GeoDistance” on page 3-27](#).

Usage

The altitude range of the objects is ignored. **Beyond** measures distance along a geodesic on the datum surface at altitude 0, regardless of the altitude values of the objects.

If two objects both have a null time range, the null ranges are considered equal—and if the objects otherwise qualify for a TRUE result—**Beyond** returns TRUE.

This function does not consider the boundaries and vertices to be part of the objects when determining if one lies beyond the specified distance of the other.

Important: *Keep in mind that the boundaries of objects are known exactly only at specified vertices. You cannot rely on results of this function that depend on locations of boundaries away from vertices.*

The negator of **Beyond** is **Within**.



Ternary operators such as **Beyond** cannot make use of indexes that may be associated with their arguments. You can, however, rewrite a query that uses **Beyond** by substituting the **Outside** binary operator and the **SetDist** function so that an index can be used.

The following query with three arguments does not use the index on **location**:

```
SELECT name
FROM cities
WHERE Beyond(location, '((37.8, -122.4), (100, 200)),
(1896-08-01 00:00:00, 1996-08-01 00:00:00)'
::GeoPoint, 1E5);
```

Rephrasing the query with **Outside** and **SetDist** allows the query to make use of an index. The following query uses the index on **location** in the time dimension:

```
SELECT name
FROM cities
WHERE Outside(location,
SetDist('((37.8, -122.4), (100, 200)),
(1896-08-01 00:00:00, 1996-08-01 00:00:00)'
::GeoPoint, 1E5);
```



Important: For best performance, apply the distance parameter to the *GeoObject* argument that does not have an index associated with it. In this example, **location** has an index so **SetDist** is applied to a constant that is not indexed. Applying a distance parameter to an argument associated with an index will disable the index.

For more information, see [“Rewriting Ternary Operator Expressions” on page 6-7.](#)

Return Type

Beyond returns Boolean *t* (TRUE) or *f* (FALSE).

Related Topics

[“Handling of Horizontal Time and Altitude Dimensions” on page 5-4](#)

[“How Spatial Operators Use R-tree Indexes” on page 6-7](#)

[Outside on page 4-51](#)

[SetDist on page 4-56](#)

[Within on page 4-60](#)



Bottom

The **Bottom** function takes an altitude range and returns its lower (bottom) altitude as a GeoAltitude data type.

***Tip:** If the altitude range was constructed from a single value, **Bottom** and **Top** return the same value.*

Syntax

```
Bottom(GeoAltrange)
```

For more information, refer to [“GeoAltrange” on page 3-7](#).

Return Type

Bottom returns a GeoAltitude data type. For more information on the GeoAltitude data type, refer to [“GeoAltitude” on page 3-5](#).

If the altitude range is ANY, **Bottom** returns the value of the constant DBL_MAX (typically -1.7976931348623158e+308) from the ANSI C **float.h** header file.

Related Topics

[Top on page 4-59](#)

Coords

The **Coords** function takes a spatio-temporal object and returns a GeoCoords latitude/longitude coordinate pair.

Syntax

```
Coords(GeoLineSeg, INTEGER)
Coords(GeoString, INTEGER)
Coords(GeoRing, INTEGER)
Coords(GeoPolygon, INTEGER)
Coords(GeoBox, INTEGER)
Coords(GeoPoint)
Coords(GeoCircle)
Coords(GeoEllipse)
```

For more information on these data types, refer to [Chapter 3](#).

Usage

For GeoLineSeg, GeoString, GeoRing, GeoPolygon, and GeoBox objects, **Coords** returns the *n*th coordinate pair, where *n* is the integer specified in the second argument. For example, if the second argument is the integer 1, **Coords** returns the first coordinate pair in the object's external representation.

For GeoCircle and GeoEllipse objects, **Coords** returns the coordinates of the center point of the circle or ellipse.

For GeoPoint objects, **Coords** returns the coordinates of the point.

Return Value

Coords returns a GeoCoords data type. For more information on this data type, refer to [“GeoCoords” on page 3-25](#).

End

The **End** function returns the end time of a time range.

Syntax

```
End ( GeoTimeRange )
```

Usage

The **End** function returns DATETIME values with five decimal digits, regardless of the number of digits you provide on input and regardless of the setting of the GL_DATETIME environment variable.

Return Type

End returns DATETIME YEAR TO FRACTION(5). If the time range is ANY, **End** returns 9999-12-31 23:59:59:99999.

Refer to “[GeoTimeRange](#)” on page 3-59 and *[Informix Guide to SQL: Reference](#)* for information on the DATETIME data type.

Related Topics

[Begin](#) on page 4-9

GeoAltRange

The **GeoAltRange** function is the constructor for the GeoAltrange data type.

Syntax

The **GeoAltRange** function takes one or two GeoAltitude values as arguments.

```
GeoAltRange(GeoAltitude, GeoAltitude)  
GeoAltRange(GeoAltitude)
```

For more information on GeoAltitude, refer to [“GeoAltitude” on page 3-5](#).

Return Type

GeoAltRange returns a GeoAltrange data type. Refer to [“GeoAltrange” on page 3-7](#) for a description of the external representation of this data type.

Related Topics

[Bottom](#) on page 4-13

[Top](#) on page 4-59

GeoBox

The **GeoBox** function is the constructor for the GeoBox spatio-temporal type.

It also converts a **GeoObject** data type to a **GeoBox** data type, if the original object was **GeoBox**.

Type Constructor Syntax

```
GeoBox(GeoCoords, GeoCoords, GeoAltrange, GeoTimeRange)
```

Refer to [“GeoCoords” on page 3-25](#), [“GeoAltrange” on page 3-7](#), and [“GeoTimeRange” on page 3-59](#) for a description of the external representation of these types.

GeoObject Conversion Syntax

```
GeoBox(GeoObject)
```

The **GeoBox** function converts a **GeoBox** data type that was cast to **GeoObject** data type back to **GeoBox**, and returns **NULL** if the original object was not **GeoBox**.

Return Type

The **GeoBox** type constructors and conversion syntax return a **GeoBox** data type. If the original object was not a **GeoBox** data type, the conversion function returns **NULL**.

Refer to [“GeoBox” on page 3-19](#) for a description of the external representation of this data type.

GeoCircle

The **GeoCircle** function is the constructor for the GeoCircle spatio-temporal type.

It also converts a GeoCircle that was cast to GeoObject back to its original GeoCircle representation.

Type Constructor Syntax

```
GeoCircle(GeoCoords, GeoDistance, GeoAltrange, GeoTimeRange)
```

The **GeoCircle** type constructor constructs a GeoCircle data type from a GeoCoords latitude, longitude coordinate pair representing the center of the circle's center, followed by a GeoDistance value representing the GeoCircle's radius, a GeoAltrange altitude range, and a GeoTimeRange time range:

Refer to [“GeoCoords” on page 3-25](#), [“GeoTimeRange” on page 3-59](#), [“GeoAltrange” on page 3-7](#), and [“GeoTimeRange” on page 3-59](#) for a description of the external representation of these types.

GeoObject Conversion Syntax

```
GeoCircle(GeoObject)
```

The **GeoCircle** function converts a GeoCircle data type that was cast to GeoObject back to GeoCircle, and returns `NULL` if the original object was not GeoCircle.

Return Type

The **GeoCircle** type constructor and conversion syntax returns GeoCircle. If the original object was not GeoCircle, the conversion function returns `NULL`.

Refer to [“GeoCircle” on page 3-22](#) for a description of the external representation of this data type.

Related Topics

Coords on page 4-14

Radius on page 4-53

GeoCoords

The **GeoCoords** function is the constructor for the GeoCoords data type.

Syntax

```
GeoCoords(GeoLatitude, GeoLongitude)
```

GeoCoords takes a pair of latitude and longitude coordinates as arguments. The arguments can be entered as text, or as *GeoLatitude* and *GeoLongitude* values.

Refer to [“GeoAltitude” on page 3-5](#) and [“GeoLongitude” on page 3-39](#) for a description of the external representation of these data types.

Return Type

GeoCoords returns GeoCoords. Refer to [“GeoCoords” on page 3-25](#) for a description of the external representation.

GeoEllipse

The **GeoEllipse** function is the constructor for the GeoEllipse data type.

GeoEllipse also converts a GeoEllipse data type that was cast to GeoObject back to its original GeoEllipse representation.

Type Constructor Syntax

```
GeoEllipse(GeoCoords, GeoCoords, GeoDistance, GeoDistance,  
GeoAzimuth, GeoAltrange, GeoTimeRange)
```

The type constructor syntax takes two GeoCoords data types to specify the latitude and longitude of the center point, followed by nonzero GeoDistance values representing the length of the semi-major and semi-minor axes, GeoAzimuth for the azimuth of the major axis, and domain-enforcing types to define the altitude and time dimensions.

Refer to [“GeoCoords” on page 3-25](#), [“GeoTimeRange” on page 3-59](#), [“GeoAzimuth” on page 3-16](#), [“GeoAltrange” on page 3-7](#), and [“GeoTime-Range” on page 3-59](#) for descriptions of the external representation of these types.

GeoObject Conversion Syntax

```
GeoEllipse(GeoObject)
```

The **GeoEllipse** conversion syntax converts a GeoEllipse data type that was cast to GeoObject back to GeoEllipse, and returns NULL if the original object was not GeoEllipse.

Return Type

The **GeoEllipse** type constructor and conversion syntax return a GeoEllipse data type. If the original object was not GeoEllipse, the conversion function returns NULL.

Refer to [“GeoEllipse” on page 3-29](#) for a description of the GeoEllipse data type.

Related Topics

[Azimuth](#) on page 4-8

[Major](#) on page 4-47

[Minor](#) on page 4-48

GeoLineseg

The **GeoLineseg** function is the constructor for the GeoLineseg type.

It also converts a GeoLineseg that was cast to GeoObject back to its original GeoLineseg representation.

Type Constructor Syntax

```
GeoLineseg(GeoCoords, GeoCoords, GeoAltrange, GeoTimeRange)
```

The type constructor syntax takes two GeoCoords to specify the latitude and longitude coordinates of the beginning and ending points, and domain-enforcing types to define the altitude and time dimensions.

Refer to [“GeoCoords” on page 3-25](#), [“GeoAltrange” on page 3-7](#), and [“GeoTimeRange” on page 3-59](#) for a description of the external representation of these data types.

GeoObject Conversion Syntax

```
GeoLineseg(GeoObject)
```

The **GeoLineseg** conversion syntax converts a GeoLineseg that was cast to GeoObject back to a GeoLineseg, and returns NULL if the original object was not a GeoLineseg.

Return Type

The **GeoLineseg** type constructor and conversion syntax return GeoLineseg. The conversion function returns NULL if the original object was not a GeoLineseg.

Refer to [“GeoLineseg” on page 3-36](#) for a description of the external representation of GeoLineseg.

GeoPoint

The **GeoPoint** function is the constructor for the GeoPoint type.

It also converts a GeoPoint value that was cast to GeoObject back to its original representation.

Type Constructor Syntax

```
GeoPoint(GeoCoords, GeoAltrange, GeoTimeRange)
```

The type constructor syntax takes GeoCoords data types and domain-enforcing types to specify the latitude and longitude coordinates, and altitude and time ranges of the point.

Refer to [“GeoCoords” on page 3-25](#), [“GeoAltrange” on page 3-7](#), and [“GeoTimeRange” on page 3-59](#) for a description of the external representation of these data types.

GeoObject Conversion Syntax

```
GeoPoint(GeoObject)
```

The **GeoPoint** conversion syntax converts a GeoPoint value that was cast to GeoObject back to GeoPoint, and returns NULL if the original object was not of type GeoPoint.

Return Type

GeoPoint returns GeoPoint. The conversion function returns NULL if the original object was not of type GeoPoint.

Refer to [“GeoPoint” on page 3-44](#) for a description of GeoPoint.

GeoPolygon

The **GeoPolygon** function converts a GeoPolygon value that was cast to GeoObject back to its original GeoPolygon representation.

It also converts GeoBox, GeoCircle, or GeoEllipse values to GeoPolygon.

Syntax

```
GeoPolygon(GeoObject)
```

Usage

A GeoPolygon consists of an outer ring, interior area, altitude range, and time range. It can have one or more nonintersecting, non-nested inner rings that represent exclusion areas.

When you convert GeoBox or GeoEllipse to GeoPolygon, the object that results is a single-ring polygon. Converting GeoCircle to GeoPolygon results in a close approximation of the original circle.



***Tip:** Operator functions use GeoPolygon approximation of GeoBox and GeoEllipse. You can use the GeoPolygon conversion syntax described here to examine the objects used by operators. Refer to “[Operator Functions](#)” on page 4-5 for a list of these functions.*

Return Type

GeoPolygon returns GeoPolygon. It returns `NULL` if the original object was not a GeoPolygon, GeoBox, GeoCircle, or GeoEllipse.

Refer to “[GeoPolygon](#)” on page 3-47 for a description of this data type.

GeoRelease

The **GeoRelease** function returns a text string containing the installed version and release date of the Informix Geodetic DataBlade module.

Syntax

```
GeoRelease()
```

Return Type

The **GeoRelease** function returns the installed version and release date as text.

GeoRing

The **GeoRing** function converts `GeoObject` to `GeoRing`, if the original object was of type `GeoRing`. It is the `GeoObject` conversion syntax.

Syntax

```
GeoRing(GeoObject)
```

Refer to [“GeoObject” on page 3-41](#) for a description of `GeoObject`.

Return Type

GeoRing returns `GeoRing`, or `NULL` if the original object was not of type `GeoRing`. Refer to [“GeoRing” on page 3-53](#) for a description of `GeoRing`.

GeoString

GeoString converts a GeoString value that was cast to GeoObject back to its original GeoString representation.

Syntax

```
GeoString(GeoObject)
```

Refer to [“GeoObject” on page 3-41](#) for a description of GeoObject.

Return Type

GeoString returns GeoString. If the original object was not a GeoString, it returns `NULL`.

Refer to [“GeoString” on page 3-56](#) for a description of GeoString.

GeoTimeRange

The **GeoTimeRange** function is the constructor for the GeoTimeRange data type.

Syntax

```
GeoTimeRange(DATETIME YEAR TO FRACTION (5), DATETIME YEAR TO  
FRACTION (5))
```

```
GeoTimeRange(DATETIME YEAR TO FRACTION (5))
```

The **GeoTimeRange** function takes one or two DATETIME YEAR TO FRACTION (5) values to specify a time range.

For more information on DATETIME values, refer to [Informix Guide to SQL: Reference](#).

Important: The storage and output of DATETIME values are affected by the setting of the GL_DATETIME environment variable. See [“Setting Date and Time Precision” on page 2-11](#) for information on the interaction of GL_DATETIME with the Informix Geodetic DataBlade module.



Return Type

GeoTimeRange returns GeoTimeRange. Refer to [“GeoTimeRange” on page 3-59](#) for more information on this data type and its external representation.

Related Topics

[Begin on page 4-9](#)

[End on page 4-15](#)

Inside

The **Inside** function compares two spatio-temporal objects and returns Boolean *t* (TRUE) if *all* of the following are true:

- All points, line segments, and the interior area (if any) of the first object are spatially entirely inside the boundary and interior area of the second object.
- The altitude range of the first object intersects the range of the second object.
- The time ranges of the two objects intersect.

Otherwise, **Inside** returns *f* (FALSE).

Syntax

```
Inside(GeoObject, GeoObject)
```

For information on the external representation of these types, refer to [“GeoObject” on page 3-41](#).

Usage

Inside takes two spatio-temporal objects as arguments. Refer to [“GeoObject” on page 3-41](#) for more information on this data type.



Important: The second argument must be an object with an interior area. An error results if the second argument is a *GeoPoint*, *GeoLineSeg*, *GeoRing*, or *GeoString*.

The **Inside** function uses the altitude ranges of the two objects, as well as the longitude and latitude, to determine if one object is inside the other. If altitude is unimportant, specify *ANY* for the altitude range of the second object, and **Inside** ignores it.



Important: You cannot specify *ANY* for the altitude range of the first object and provide a specific range for the second object. Semantically, it is not possible for an indeterminate altitude range (specified by *ANY*) to be inside a specific range. Use *ANY* for both objects or for the second object only to ignore altitude.

If time is unimportant, specify *ANY* for time range.

If two objects each have a null time range, the null ranges are considered equal—and if the objects otherwise qualify for a TRUE result—**Inside** returns TRUE.

This function considers the boundaries and vertices to be part of the objects when determining if one is inside the other. That is, if all or part of the boundary of the first object coincides with the boundary of the second object—and if the objects otherwise qualify for a TRUE result—**Inside** returns TRUE.



Important: *Keep in mind that the boundaries of objects are known exactly only at specified vertices. You cannot rely on results of this function that depend on locations of boundaries away from vertices.*

The **Inside** function does not have a negator or commutator function and is itself not commutative; that is, if you switch the order of the arguments, you obtain a different result. However, you can apply an R-tree index to either argument to speed evaluation of queries.

Return Type

Inside returns Boolean \mathbf{t} (TRUE) or \mathbf{f} (FALSE).

Related Topics

[Beyond](#) on page 4-10

[“Handling of Horizontal Time and Altitude Dimensions”](#) on page 5-4

[“How Spatial Operators Use R-tree Indexes”](#) on page 6-7

[Intersect](#) on page 4-32

[Outside](#) on page 4-51

[Within](#) on page 4-60

Intersect

The **Intersect** function returns `t` (TRUE) if *both* of the following are true:

- Any point or partial line segment of the object in the first argument lies spatially inside of the object in the second argument.
- The time ranges for the objects intersect.

Syntax

```
Intersect(GeoObject, GeoObject)
```

For information on the external representation of spatio-temporal objects, refer to [“GeoObject” on page 3-41](#).

Usage

Intersect takes two spatio-temporal objects as arguments.

If both objects are `GeoPoints`, or `GeoPoints` that were cast to `GeoObject`, **Intersect** tests whether the two objects are equal (within the resolution of the Informix Geodetic DataBlade module—which is to approximately 50 meters).

If one object is a `GeoPoint` or `GeoPoint` cast to `GeoObject`, and the other is a linear object (`GeoLineSeg`, `GeoString`, or `GeoRing`), **Intersect** tests whether the point lies on the linear object (within the resolution of the Informix Geodetic DataBlade module).

The **Intersect** function uses the altitude ranges of the two objects, as well as the longitude and latitude, to determine if one object intersects the other. If altitude is unimportant, specify `ANY` for altitude range, and **Intersect** ignores it. Likewise, if time is unimportant, specify `ANY` for the time range of either object.

If two objects each have a null time range, the null ranges are considered equal—and if the objects otherwise qualify for a TRUE result—**Intersect** returns TRUE.



This function considers the boundaries and vertices to be part of the objects when determining if one intersects the other.

Important: *Keep in mind that the boundaries of objects are known exactly only at specified vertices. You cannot rely on results of this function that depend on locations of boundaries away from vertices.*

The **Intersect** function is commutative. If you switch the order of the arguments, you obtain the same results. You can apply an R-tree index to either argument to speed evaluation of queries. The negator function for **Intersect** is **Outside**.

Return Type

Intersect returns Boolean \mathbf{t} (TRUE) or \mathbf{f} (FALSE).

Related Topics

[“Handling of Horizontal Time and Altitude Dimensions” on page 5-4](#)

[Beyond on page 4-10](#)

[“How Spatial Operators Use R-tree Indexes” on page 6-7](#)

[Inside on page 4-30](#)

[Outside on page 4-51](#)

[Within on page 4-60](#)

IsGeoBox

The **IsGeoBox** function takes an object of type `GeoObject` and returns `t` (TRUE) if the original type was a `GeoBox`; it returns `f` (FALSE) otherwise.

Syntax

```
IsGeoBox(GeoObject)
```

`GeoObject` is the spatio-temporal supertype; refer to [“GeoObject” on page 3-41](#) for more information on this data type.

Return Type

IsGeoBox returns Boolean `t` (TRUE) or `f` (FALSE).

IsGeoCircle

The **IsGeoCircle** function takes an object of type `GeoObject` and returns `t` (TRUE) if the original type was a `GeoCircle`; otherwise, it returns `f` (FALSE).

Syntax

```
IsGeoCircle(GeoObject)
```

`GeoObject` is the spatio-temporal supertype; refer to [“GeoObject” on page 3-41](#) for more information on this data type.

Return Type

IsGeoCircle returns Boolean `t` (TRUE) or `f` (FALSE).

IsGeoEllipse

The **IsGeoEllipse** function takes an object of type `GeoObject` and returns `t` (TRUE) if the original type was a `GeoEllipse`; it returns `f` (FALSE) otherwise.

Syntax

```
IsGeoEllipse(GeoObject)
```

`GeoObject` is the spatio-temporal supertype; refer to “[GeoObject](#)” on [page 3-41](#) for more information on this data type.

Return Type

IsGeoEllipse returns Boolean `t` (TRUE) or `f` (FALSE).

IsGeoLineseg

The **IsGeoLineseg** function takes an object of type `GeoObject` and returns `t` (TRUE) if the original type was a `GeoLineseg`, `f` (FALSE) otherwise.

Syntax

```
IsGeoLineseg(GeoObject)
```

`GeoObject` is the spatio-temporal supertype; refer to “[GeoObject](#)” on [page 3-41](#) for more information on this data type.

Return Type

IsGeoLineseg returns Boolean `t` (TRUE) or `f` (FALSE).

IsGeoPoint

The **IsGeoPoint** function takes an object of type `GeoObject` and returns `t` (TRUE) if the original type was a `GeoPoint`; it returns `f` (FALSE) otherwise.

Syntax

```
IsGeoPoint(GeoObject)
```

`GeoObject` is the spatio-temporal supertype; refer to [“GeoObject” on page 3-41](#) for more information on this data type.

Return Type

IsGeoPoint returns Boolean `t` (TRUE) or `f` (FALSE).

IsGeoPolygon

The **IsGeoPolygon** function takes an object of type `GeoObject` and returns `t` (TRUE) if the original type was a `GeoPolygon`; it returns `f` (FALSE) otherwise.

Syntax

```
IsGeoPolygon(GeoObject)
```

`GeoObject` is the spatio-temporal supertype; refer to “[GeoObject](#)” on [page 3-41](#) for more information on this data type.

Return Type

IsGeoPolygon returns Boolean `t` (TRUE) or `f` (FALSE).

IsGeoRing

The **IsGeoRing** function takes an object of type `GeoObject` and returns `t` (TRUE) if the original type was a `GeoRing`; it returns `f` (FALSE) otherwise.

Syntax

```
IsGeoRing(GeoObject)
```

`GeoObject` is the spatio-temporal supertype; refer to [“GeoObject” on page 3-41](#) for more information on this data type.

Return Type

IsGeoRing returns Boolean `t` (TRUE) or `f` (FALSE).

IsGeoString

The **IsGeoString** function takes an object of type `GeoObject` and returns `t` (TRUE) if the original type was a `GeoString`; it returns `f` (FALSE) otherwise.

Syntax

```
IsGeoString(GeoObject)
```

`GeoObject` is the spatio-temporal supertype; refer to “[GeoObject](#)” on [page 3-41](#) for more information on this data type.

Return Type

IsGeoString returns Boolean `t` (TRUE) or `f` (FALSE).

IsValidGeometry

The **IsValidGeometry** function verifies GeoObject data.

Syntax

```
IsValidGeometry(GeoObject)
```

Usage

The **IsValidGeometry** function takes an object of type GeoObject. If the underlying type is GeoRing or GeoPolygon, **IsValidGeometry** verifies that the vertices form a valid g-ring, and that the g-rings of the polygon do not intersect one another. It returns `t` (TRUE) if

- the border of the GeoRing object and the constituent g-rings of the GeoPolygon object do not cross or touch themselves or one another
- the coordinates of consecutive vertices are unique (within the internal precision)

If the GeoRing or GeoPolygon object does not meet these criteria, **IsValidGeometry** returns `f` (FALSE) and posts a warning message about the violation.

If a GeoPolygon object meets these criteria but its area is greater than half the Earth, **IsValidGeometry** returns `t` (TRUE) and posts a warning. The warning may help you detect polygons with borders that were incorrectly entered. See [“GeoPolygon” on page 3-47](#) for more information.

If the underlying object is not of type GeoRing or GeoPolygon, **IsValidGeometry** returns `t` (TRUE).



Tip: You can also use the **geovvalidate** client utility to validate GeoObject data. For more information, see [“Client Validation Program” on page 2-9](#).

You can use **IsValidGeometry** in a CHECK constraint. See [Informix Guide to SQL: Syntax](#) for more information.

Warning Messages

IsValidGeometry issues a warning if you define a polygon that is larger than a hemisphere. The warning text is:

```
GeoPolygon covers more than half the earth (IsValidGeometry):  
Verify counter-clockwise orientation of the vertex points.
```

This message is intended to warn you that your data may include a polygon that was specified incorrectly. You must specify the points that make of a GeoPolygon border in counterclockwise order (the area of the polygon is on your left as you traverse the border). If you enter the points in clockwise order, the polygon is valid, but it has a different interior area.

For example, if the border of your polygon is the coast of Africa, and you specify the points in the clockwise order, you will receive a warning, because your polygon covers the entire Earth *except* Africa, rather than the African continent. If this is the intended result, you can ignore the warning message.

The warning message is intended to help you identify polygons that you *might* have entered incorrectly, but it does not prevent you from entering large polygons.

If you receive this warning and your data does not contain polygons larger than a hemisphere, verify that the border points are specified in counterclockwise order. If the polygons in question are known to be large, ignore the warning message.

Return Type

IsValidGeometry returns Boolean *t* (TRUE) or *f* (FALSE).

IsValidSDTS

The **IsValidSDTS** function verifies that an object of type `GeoPolygon` complies with the SDTS definition of a g-polygon.

Syntax

```
IsValidSDTS(GeoObject)
```

Usage

IsValidSDTS takes a `GeoObject` data type. If the underlying type is `GeoPolygon`, **IsValidSDTS** verifies that it complies with the SDTS definition of a g-polygon. The SDTS definition specifies that a g-polygon:

- can have only one outer g-ring.
- can have at most one level of nesting for inner g-rings; that is, there can be multiple inner rings within the outer g-ring, but there can be no inner rings within any of the inner rings.

IsValidSDTS returns `t` (TRUE) if the `GeoPolygon` object meets this definition. If it does not, **IsValidSDTS** returns `f` (FALSE) and posts a warning message with details of the violation.

If the underlying type is not `GeoPolygon`, **IsValidSDTS** returns `t` (TRUE).

***Tip:** You can also use the **geovvalidate** client utility to validate `GeoObject` data. For more information, see “[Client Validation Program](#)” on page 2-9.*

You can use **IsValidSDTS** in a CHECK constraint. See [Informix Guide to SQL: Syntax](#) for more information.

Return Type

IsValidSDTS returns Boolean `t` (TRUE) or `f` (FALSE).



Latitude

The **Latitude** function takes a GeoCoords coordinate pair and returns the latitude as GeoLatitude.

Syntax

```
Latitude(GeoCoords)
```

Return Type

Latitude returns a GeoLatitude. For more information, refer to [“GeoLatitude” on page 3-33](#).

Longitude

The **Longitude** function takes a GeoCoords coordinate pair and returns the pair's longitude as GeoLongitude.

Syntax

```
Longitude(GeoCoords)
```

Refer to [“GeoCoords” on page 3-25](#) for a description of the external representation.

Return Type

Longitude returns a GeoLongitude. For more information, refer to [“GeoLongitude” on page 3-39](#).

Major

The **Major** function takes a `GeoEllipse` and returns the length of its semi-major axis as a `GeoDistance`.

Syntax

```
Major (GeoEllipse)
```

See [“GeoEllipse” on page 3-29](#) for a description of the external representation of this data type.

Return Type

The **Major** function returns `GeoDistance`. For more information on this data type, refer to [“GeoDistance” on page 3-27](#).

Minor

The **Minor** function takes a `GeoEllipse` and returns the length of its semi-minor axis as a `GeoDistance`.

Syntax

```
Minor(GeoEllipse)
```

See [“GeoEllipse” on page 3-29](#) for a description of the external representation of this data type.

Return Type

The **Minor** function returns `GeoDistance`. For more information on this data type, refer to [“GeoDistance” on page 3-27](#).

Related Topics

[Azimuth on page 4-8](#)

[Coords on page 4-14](#)

[Major on page 4-47](#)



NPoints

The **NPoints** function takes a `GeoString`, `GeoRing`, or `GeoPolygon` and returns the number of points in the object. For polygons, **NPoints** returns the total number of points in the inner and outer rings of the polygon.

***Tip:** You can also use **NPoints** to determine the number of line segments in a `GeoString`, `GeoRing`, or `GeoPolygon` type object. The number of line segments in a `GeoRing` or `GeoPolygon` object is equal to the number of points returned by **NPoints**. The number of line segments in a `GeoString` is equal to the number of points, minus 1.*

Syntax

```
NPoints(GeoString)  
NPoints(GeoRing)  
NPoints(GeoPolygon)
```

Refer to [“GeoString” on page 3-56](#), [“GeoRing” on page 3-53](#), and [“GeoPolygon” on page 3-47](#) for a description of the external representation of these data types.

Return Type

The **NPoints** function returns an integer.

NRings

The **NRings** function takes a **GeoPolygon** and returns an integer representing the number of g-rings that constitute the **GeoPolygon**.

Syntax

```
NRings(GeoPolygon)
```

Refer to [“GeoPolygon” on page 3-47](#) for a description of the external representation of this data type.

Return Type

The **NRings** function returns an integer.

Outside

The **Outside** function takes two spatio-temporal objects and returns `t` (TRUE) if *both* of the following are true:

- All points and line segments and the entire interior area (if any) of the first object lie spatially outside the second object.
- The time ranges of the two objects intersect.

Otherwise, **Outside** returns `f` (FALSE).

Syntax

```
Outside(GeoObject, GeoObject)
```

For more information, refer to [“GeoObject” on page 3-41](#).

Usage

If both are GeoPoint objects, **Outside** tests for inequality (within the resolution of the Informix Geodetic DataBlade module—which is to approximately 50 meters), and returns `t` (TRUE) if the GeoPoint objects are not equal. If the points are equal, it returns `f` (FALSE).

The **Outside** function uses the altitude ranges of the two objects, as well as the longitude and latitude, to determine if one object lies outside the other. If altitude is unimportant, specify `ANY` for altitude range, and **Outside** ignores it. Likewise, if time is unimportant, specify `ANY` for the time range of either object.

If two objects each have a null time range, the null ranges are considered equal—and if the objects otherwise qualify for a TRUE result—**Outside** returns TRUE.

This function considers the boundaries and vertices to be part of the objects when determining if one lies outside the other.

Important: Keep in mind that the boundaries of objects are known exactly only at specified vertices. You cannot rely on results of this function that depend on locations of boundaries away from vertices.



The **Outside** function is commutative. If you switch the order of the arguments, you obtain the same results. You can apply an R-tree index to either argument to speed evaluation of queries. The negator function for **Outside** is **Intersect**.

Return Type

Outside returns Boolean *t* (TRUE) or *f* (FALSE).

Related Topics

[Handling of Horizontal Time and Altitude Dimensions on page 5-4](#)

[Inside on page 4-30](#)

[Intersect on page 4-32](#)

Radius

The **Radius** function takes an object of type `GeoCircle` and returns the length of its radius as a `GeoDistance` value.

Syntax

```
Radius(GeoCircle)
```

Refer to [“GeoCircle” on page 3-22](#) for a description of the external representation of this data type.

Return Type

The **Radius** function returns `GeoDistance`. Refer to [“GeoDistance” on page 3-27](#) for a description of the external representation of this data type.

Related Topics

[Coords on page 4-14](#)

Ring

The **Ring** function takes an object of type `GeoPolygon` and an integer and returns the ring in the `GeoPolygon`'s represented by the integer. The integer 1 represents the first ring that was specified in the external representation, 2 refers to the second ring specified, and so forth.

Syntax

```
Ring(GeoPolygon, integer)
```

Refer to [“GeoPolygon” on page 3-47](#) for a description of the external representation of this data type.

Return Type

The **Ring** function returns `GeoRing`. Refer to [“GeoRing” on page 3-53](#) for a description of the external representation of this data type.

Related Topics

[NRings on page 4-50](#)

SetAltRange

The **SetAltRange** function updates the altitude range value for a spatio-temporal object. It takes a spatio-temporal object and a valid GeoAltrange value as arguments.

You can use **SetAltRange** on **GeoObject** or any **GeoObject** derived type.



***Tip:** **SetAltRange** automatically casts the object to **GeoObject**; to return the object to its original form, you must use the object's conversion function. For example, if the object is a **GeoPolygon**, use the **GeoPolygon** conversion function to convert the **GeoObject** returned by **SetAltRange** back to **GeoPolygon**.*

Syntax

```
SetAltRange(GeoObject, GeoAltrange)
```

Refer to “[GeoObject](#)” on page 3-41 and “[GeoAltrange](#)” on page 3-7 for a description of the external representation of these data types.

Return Value

The **SetAltRange** function returns **GeoObject**. Refer to “[GeoObject](#)” on [page 3-41](#) for more information on this data type.

SetDist

The **SetDist** function adds a distance parameter to a spatio-temporal object.

Syntax

```
SetDist(GeoObject, GeoDistance)
```

Refer to [“GeoObject” on page 3-41](#) and [“GeoDistance” on page 3-27](#) for a description of the external representation of these data types.

Usage

You can use **SetDist** to create a buffer of any size around an object, allowing the **Intersect** function to perform a proximity test rather than a simple intersection.

You can use this function with binary operators to rewrite queries containing ternary operators so that they can use indexes for better performance. For example, you can rewrite a query that uses **Beyond** by using **SetDist** with the **Outside** operator. Refer to [“Rewriting Ternary Operator Expressions” on page 6-7](#) for more information.

Return Type

The **SetDist** function returns `GeoObject`. Refer to [“GeoObject” on page 3-41](#) for more information on this data type.

Related Topics

[“Rewriting Ternary Operator Expressions” on page 6-7](#)
[Beyond on page 4-10](#)
[Within on page 4-60](#)



SetTimeRange

The **SetTimeRange** function updates the time value for a spatio-temporal object. It takes a spatio-temporal object and a **GeoTimeRange** value as arguments.

***Tip:** **SetTimeRange** automatically casts the object to **GeoObject**; to return the object to its original form, you must use the conversion function for the object. For example, if the object is a **GeoPolygon**, use the **GeoPolygon** conversion function to convert the **GeoObject** returned by **SetTimeRange** back to **GeoPolygon**.*

Syntax

```
SetTimeRange( GeoObject, GeoTimeRange )
```

Refer to “**GeoObject**” on page 3-41 and “**GeoTimeRange**” on page 3-59 for a description of the external representation of these data types.

Return Type

The **SetTimeRange** function returns **GeoObject**. Refer to “**GeoObject**” on page 3-41 for more information on this data type.

TimeRange

The **TimeRange** function takes a spatio-temporal object and returns the time range of the object.

Syntax

```
TimeRange(GeoObject)
```

Refer to [“GeoObject” on page 3-41](#) for a description of the external representation.

Return Type

The **TimeRange** function returns a GeoTimeRange. Refer to [“GeoTime-Range” on page 3-59](#) for more information on this data type.



Top

The **Top** function takes an altitude range and returns its upper altitude as GeoAltitude.

***Tip:** If the altitude range was constructed from a single value, **Bottom** and **Top** return the same value.*

Syntax

```
Top( GeoAltrange )
```

For more information, refer to [“GeoAltrange” on page 3-7](#).

Return Type

Top returns GeoAltitude. For more information on the GeoAltitude data type, refer to [“GeoAltitude” on page 3-5](#).

If the altitude range is ANY, **Top** returns the value of the constant DBL_MAX (typically 1.7976931348623158e+308) from the ANSI C **float.h** header file.

Related Topics

[Bottom on page 4-13](#)

Within

The **Within** function takes two spatio-temporal objects and an object of type `GeoDistance` and returns `t` (TRUE) if *both* of the following are true:

- The distance between the point or segment of one object that is nearest any point or segment in the other object is less than or equal to the `GeoDistance` argument.
- The time ranges of the objects intersect.

Otherwise, **Within** returns `f` (FALSE).

Syntax

```
Within(GeoObject, GeoObject, GeoDistance)
```

For more information, refer to [“GeoObject” on page 3-41](#) and [“GeoDistance” on page 3-27](#).

Usage

The altitude range of the objects is ignored. The **Within** function measures distance along a geodesic on the datum surface at altitude 0, regardless of the altitude values of the objects.

If two objects both have a null time range, the null ranges are considered equal—and if the objects otherwise qualify for a TRUE result, **Within** returns TRUE.

This function does not consider the boundaries and vertices to be part of the objects when determining if one lies beyond the specified distance of the other.

Important: Keep in mind that the boundaries of objects are known exactly only at specified vertices. You cannot rely on results of this function that depend on locations of boundaries away from vertices.

The negator of **Within** is **Beyond**.



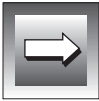
Ternary operators such as **Within** cannot make use of indexes that may be associated with their arguments. You can, however, rewrite a query that uses **Within** by substituting the **Intersect** binary operator and the **SetDist** function so that an index can be used.

The following query does not use the index on **location**:

```
SELECT name
FROM cities
WHERE Within(location, '((37.8, -122.4), ANY, ANY)'
::GeoPoint, 1E5);
```

Rephrasing the query using **Intersect** and **SetDist** allows your query to make use of an index. The following query uses the index on **location**:

```
SELECT name
FROM cities
WHERE Intersect(location, SetDist('((37.8, -122.4), ANY,
ANY)' ::GeoPoint, 1E5));
```



Important: For best performance, apply the distance parameter to the *GeoObject* argument that does not have an index associated with it. In this example, **location** has an index so **SetDist** is applied to a constant that is not indexed. Applying a distance parameter to an argument associated with an index will disable the index.

For more information, see [“Rewriting Ternary Operator Expressions” on page 6-7](#).

Return Type

The **Within** function returns Boolean *t* (TRUE) or *f* (FALSE).

Related Topics

[“Handling of Horizontal Time and Altitude Dimensions” on page 5-4](#)

[“Rewriting Ternary Operator Expressions” on page 6-7](#)

[Beyond on page 4-10](#).

[SetDist on page 4-56](#)

Spatial Operators

Handling of Horizontal Time and Altitude Dimensions	5-4
Spatial Operators That Use R-tree Indexes	5-6

This chapter describes a group of functions known as *spatial operators* and explains how you can use them to test spatio-temporal objects for proximity, inclusion, and intersection.

The spatial operators are described individually in [Chapter 4, “Geodetic Functions.”](#) This chapter describes the spatial operators as a whole, including special characteristics they have in common as well as differences between them.

The Informix Geodetic DataBlade module spatial operators are:

- **Beyond**
- **Inside**
- **Intersect**
- **Outside**
- **Within**

Those that take only two arguments—**Inside**, **Outside**, and **Intersect**—are called *binary operators*. **Within** and **Beyond** take three arguments and are therefore known as *ternary operators*. All spatial operators return Boolean `t` (TRUE) or `f` (FALSE).

The following table lists these operators and the arguments they take.

Operator	Arguments	Comment
Inside	<i>GeoObject</i> , <i>GeoObject</i>	The second argument must represent an object with interior area. Refer to Inside on page 4-30 for more information.
Intersect	<i>GeoObject</i> , <i>GeoObject</i>	Intersect is equivalent to Not Outside in the spatial dimension only.
Outside	<i>GeoObject</i> , <i>GeoObject</i>	Outside is equivalent to Not Intersect in the spatial dimension only.
Within	<i>GeoObject</i> , <i>GeoObject</i> , <i>GeoDistance</i>	Within is equivalent to <code>Intersect(GEOBJECT, SetDist(GEOBJECT, GEODISTANCE))</code> and <code>Intersect(SetDist(GEOBJECT, GEODISTANCE), GEOBJECT)</code> It is also equivalent to Not Beyond in the spatial dimension only.
Beyond	<i>GeoObject</i> , <i>GeoObject</i> , <i>GeoDistance</i>	Beyond is equivalent to <code>Outside(GEOBJECT, SetDist(GEOBJECT, GEODISTANCE))</code> and <code>Outside (SetDist(GEOBJECT, GEODISTANCE), GEOBJECT)</code> It is also equivalent to Not Within in the spatial dimension only.

Each of these functions is described in [Chapter 4, “Geodetic Functions.”](#)

Handling of Horizontal Time and Altitude Dimensions

Spatial operators take GeoObject and GeoObject-derived types as arguments. The spatial operators treat the different dimensions of these objects in different ways, as follows:

- **Horizontal dimension (latitude and longitude).** Each operator handles the horizontal dimension in the way suggested by its name.
- **Altitude dimension.** The binary operators (**Inside**, **Intersect**, and **Outside**), which handle cases where the objects may be in contact, treat the altitude dimension the same as the horizontal dimension. You can ignore altitude by specifying *ANY* (with **Inside**, you can only specify *ANY* for the second argument).
The ternary operators (**Beyond** and **Within**), which handle cases where the objects have distance between them, ignore the altitude dimension.
- **Time dimension.** All the operators require overlapping time ranges to obtain a TRUE result. You can specify *ANY* to ignore time.

The following table summarizes the information in the previous paragraph by showing how each spatial operator handles the latitude and longitude, altitude, and time dimensions.

Operator	Latitude Longitude	Altitude	Time
Inside	Inside	Inside	Intersect
Intersect	Intersect	Intersect	Intersect
Outside	Outside	Outside	Intersect
Within	Within	Ignored	Intersect
Beyond	Beyond	Ignored	Intersect

Note that in the latitude and longitude dimension, the spatial operator applies the expected operation. In the altitude dimension, **Inside**, **Intersect**, and **Outside** perform this same operation; **Within** and **Beyond** ignore the altitude dimension.

Intersect is always applied in the time dimension.



Tip: If you want, you can rewrite your query so that it applies a different operation than those defined earlier for altitude and time. For example, the following query applies the equivalent of **Inside** (rather than **Intersect**) in the time dimension:

```
select * from locations where
  Inside(loc, GeoCircle('(37.8, -122.4)::GeoCoords, 1E5::GeoDistance,
    '(0E0, 0E0)::GeoAltRange, '(1996-08-05 00:00:00,
    1996-08-05 23:59:59)::GeoTimeRange)
and
  TimeVal(TimeRange(loc), 'begin') >= '1996-08-05 00:00:00':timestamp
and
  TimeVal(TimeRange(loc), 'end') <= '1996-08-05 23:59:59':timestamp;
```

Spatial Operators That Use R-tree Indexes

The Informix Geodetic DataBlade module spatial operators belong to the **GeoObject_ops** operator class. The binary operators **Inside**, **Intersect**, and **Outside** allow a query to make use of an R-tree spatial index on either of the columns specified as arguments. An R-tree index can dramatically improve the performance of a query, especially on very large tables. The binary spatial operators are the only way to make use of R-tree indexes.

You can use an index with the binary spatial operators if either the first or second arguments represents an indexed column. This is true even for the **Inside** operator, which is not commutative.

The ternary operators, **Beyond** and **Within**, cannot use R-tree indexes. You can however, rewrite a query that contains one of these operators with a binary operator that can use an R-tree index. See the description of **Beyond** on page 4-10 and **Within** on page 4-60 or “[Rewriting Ternary Operator Expressions](#)” on page 6-7, for more information, including examples.

To create an R-tree index, you use a **CREATE INDEX** statement in which you refer to the **GeoObject_ops** operator class. This class includes the binary spatial operators. See “[Creating an R-tree Index](#)” on page 6-6 for more information.

Using R-tree Indexes

About Indexes	6-3
Access Methods	6-4
The B-tree Access Method	6-4
The R-tree Access Method	6-4
Operator Classes	6-5
Operator Class GeoObject_ops	6-5
Operator Class rtree_ops	6-6
Creating an R-tree Index	6-6
How Spatial Operators Use R-tree Indexes.	6-7
Rewriting Ternary Operator Expressions	6-7
Example Query Using Within	6-8
Example Query Using Beyond	6-9

T

his chapter explains how to create indexes on data stored with the Informix Geodetic DataBlade module in an INFORMIX-Universal Server database. An index organizes access to data so that entries can be found quickly, without searching every row.

The Informix Geodetic DataBlade module supports the R-tree access method, which enables you to index multidimensional objects.

The syntax for creating an index is described in detail in the CREATE INDEX statement in [Informix Guide to SQL: Syntax](#). This chapter focuses on creating an index using the data types and functions provided in the Informix Geodetic DataBlade module. It also explains how queries use indexes, and includes special considerations to keep in mind when writing queries involving spatio-temporal data.

About Indexes

An R-tree index is a secondary data structure that organizes access to data. Queries that use an index execute more quickly and provide a significant performance improvement.

The R-tree access method is designed for use on spatio-temporal data and is the focus of this chapter. You can also create an index on a column of non-spatial data. For more information, refer to the CREATE INDEX statement in the [Informix Guide to SQL: Syntax](#).

Tip: *An index does not change the results of your query in any way. You do not need to use an index to execute a query.*



When you create an index, you can specify an *access method*—if none is specified, the default B-tree access method is used—and an *operator class*. The access method organizes the data in a way that speeds up access. The operator class is used by the optimizer to determine whether to use an index in a query.

Access Methods

Two access methods are available to Informix Geodetic DataBlade module users:

- B-tree
- R-tree

The B-tree Access Method

The INFORMIX-Universal Server default access method is B-tree. If you do not specify an access method in the CREATE INDEX statement, B-tree is used. The B-tree access method creates a one-dimensional ordering that speeds access to traditional numeric or character data.

You can use B-tree to index a column of nonspatial data, or to create a functional index on the results of a spatial function that returns numeric or character data. For example, you could create a functional B-tree index on the **NPoints** function, because **NPoints** returns an integer that B-tree can index.

Important: *The B-tree access method indexes numeric and character data only. You cannot use the B-tree access method to index spatial data.*

The R-tree Access Method

The R-tree access method speeds access to multidimensional data. It organizes data in a tree-shaped structure, with bounding boxes at the nodes. Bounding boxes indicate the farthest extent of the data that is connected to the subtree below.



A search using an R-tree index can quickly descend the tree to find objects in the general area of interest, and then perform more exact tests on the objects themselves. An R-tree index can provide a significant performance improvement because it eliminates the need to examine objects outside the area of interest. Without an R-tree index, a query would need to evaluate every object to find those that match the query criteria.

To create an R-tree index, you must specify an operator class that supports an R-tree index on the data type you want to index.

Operator Classes

An operator class helps the query optimizer decide whether the access method can be used in the query. The operator class defines:

- operators that may be able to use an index.
- support functions used to create or modify an index.
- the access method to use.

Queries that use the operators that belong to the operator class are evaluated to determine whether to use an index.

Operator Class `GeoObject_ops`

The `GeoObject_ops` operator class associates several Informix Geodetic DataBlade module operators with the R-tree access method, enabling you to create and use an R-tree index.

The `GeoObject_ops` strategy functions are Informix Geodetic DataBlade module user-defined routines. They are:

- [Inside on page 4-30](#)
- [Intersect on page 4-32](#)
- [Outside on page 4-51](#)



Operator Class *rtree_ops*

The default operator class for the R-tree access method is ***rtree_ops***. If you create an R-tree index and do not specify an operator class, ***rtree_ops*** is used.

Important: The ***rtree_ops*** operator class supports some, but not all, of the Informix Geodetic DataBlade module strategy functions. Therefore, you must use ***GeoObject_ops*** to index spatio-temporal data.

For more information on ***rtree_ops***, see the CREATE OPCLASS statement in [Informix Guide to SQL: Syntax](#).

Creating an R-tree Index

To use the R-tree access method, you must first create an index on a column of a spatio-temporal type. The syntax for creating an R-tree index is:

```
CREATE INDEX index_name
ON table_name (column_name op_class)
USING RTREE;
```

To index spatio-temporal data, you must provide the following:

- ***Index_name***. The name you wish to give your index.
- ***Table_name***. The name of the table that contains the column you wish to index.
- ***Column_name***. The name of a GeoObject column.
- ***Op_class***. The operator class. To index spatio-temporal data, you must use operator class ***GeoObject_ops***.

See the CREATE INDEX statement in [Informix Guide to SQL: Syntax](#) for a detailed description of the syntax for creating an R-tree index.

The following statement creates an R-tree index named ***i3_DsMdOrbit*** on the ***orbitalPath*** column of the ***DsMdOrbit*** table:

```
CREATE INDEX i3_DsMdOrbit
ON DsMdOrbit
USING RTREE(orbitalPath GeoObject_ops);
```

How Spatial Operators Use R-tree Indexes

Once you have created an index, it is used by the operators for any supported combination of argument types. The index is used regardless of whether the indexed column is the first or second argument. An operator with two parameters—also known as a *binary operator*—can take advantage of an index defined on a column that serves as one of its arguments. Spatial operators are discussed in detail in [Chapter 5, “Spatial Operators.”](#)

There are some restrictions that govern the use of indexes:

- For the **Outside** operator, the index is used only on the time range.
- The index is not used in the **Within** and **Beyond** operators, because they take three arguments.

A ternary operator (operator with three arguments) is not supported by the query planner. You can, however, rephrase your query so that it will use an index, as described in the next section.

- In the bounding box, time has its own, independent dimension. An index allows the scan to skip quickly over data that does not meet the time criterion. You can use the time range **ANY** in your query to prevent an index scan from skipping over data on the basis of its time range.

Rewriting Ternary Operator Expressions

The query planner makes use of indexes associated with the arguments of binary operators, but not with ternary operators such as **Beyond** and **Within**. You can, however, substitute a binary operator in a query that uses a ternary operator so that an index can be used.

This conversion makes use of the fact that the phrase “where object *A* is within distance *d* of object *B*” is equivalent to “where object *A* intersects object *B'*, where *B'* is *B*, expanded by distance *d*.” Using this logic, you can rewrite a query involving **Within** by using **Intersect** and another function, **SetDist**, that provides the needed expansion.



The resulting statement uses a binary operator and an additional function, rather than a single ternary operator, enabling the query to make use of an R-tree index.

For queries involving **Beyond**, you can use **Outside** and **SetDist** to rewrite your query to use an index to select on time range.

Tip: The **Inside** operator ignores the results of the **SetDist** function. Consequently, the following produce the same result:

```
Inside(A, B)
Inside(SetDist(A, d), B)
```

Important: Using **SetDist** with **Inside** will not change the results of your query, but it can degrade performance. Because **SetDist** expands the object's bounding box used by the R-tree index, a greater number of candidate rows are examined, slowing down the index scan.

Refer to [“SetDist” on page 4-56](#) for information on how to use this function.

Example Query Using Within

The following query will not use the index on **location**:

```
SELECT name
FROM cities
WHERE Within(location, '((37.8, -122.4), ANY, ANY)'
::GeoPoint, 1E5);
```

Rephrasing the query using **Intersect** and **SetDist** allows the query to make use of an index. The following query uses the index on **location**:

```
SELECT name
FROM cities
WHERE Intersect(location, SetDist('((37.8, -122.4), ANY,
ANY)' ::GeoPoint, 1E5));
```

Example Query Using Beyond

The following query with three arguments will not use the index on **location**:

```
SELECT name
FROM cities
WHERE Beyond(location, '((37.8, -122.4), (100, 200)),
(1896-08-01 00:00:00, 1996-08-01 00:00:00)'
::GeoPoint, 1E5);
```

Rephrasing the query with **Outside** and **SetDist** allows the query to make use of an index. The following query uses the index on **location** in the time dimension:

```
SELECT name
FROM cities
WHERE Outside(location,
SetDist('((37.8, -122.4), (100, 200)),
(1896-08-01 00:00:00, 1996-08-01 00:00:00)'
::GeoPoint, 1E5);
```



Important: For best performance, apply the distance parameter to the *GeoObject* argument that does not have an index associated with it. In this example, **location** has an index so **SetDist** is applied to a constant that is not indexed. Applying a distance parameter to an argument associated with an index will disable the index.

Geodetic Header File

This appendix provides information about the Informix Geodetic DataBlade module header file, **geodetic.h**.

The **geodetic.h** file defines the structures used to transmit binary-format objects between a client ESQL/C or C program and the server. These objects represent all the data types defined by the Informix Geodetic DataBlade module, including GeoObject and its derived types, GeoCoords, GeoAltRange, GeoTimeRange, and so on.

See [Chapter 3, “Geodetic Data Types,”](#) for information on the Informix Geodetic DataBlade module SQL data types.

This appendix describes the GeoObject type, from which the other Informix Geodetic DataBlade module types are derived. It also lists the complete contents of the **geodetic.h** file.

GeoObject

The most complex of the Informix Geodetic DataBlade module data types is GeoObject.

GeoObject is an opaque data type containing, among other things, a variable-length data structure called **GeoObjectData**. You can extract this structure from **GeoObject** using `mi_get_vardata`, as follows:

```
MI_ROW      *row;
mi_integer  status;
mi_integer  column_no;
mi_lvarchar *retbuf;
mi_integer  retlen;
GeoObjectData *data;

row = mi_next_row ( conn, &status );
status = mi_value ( row, column_no, (MI_DATUM *) &retbuf, &retlen );
data = (GeoObjectData *) mi_get_vardata ( retbuf );
```

The GeoObjectData structure consists of a fixed length portion, followed by a variable length portion, as follows:

```
typedef struct
{
    mi_integer  version;      /* version of this structure      (4 bytes) */
    mi_integer  tag;          /* type of this object           (4 bytes) */
    mi_ref      coord_ref;    /* coordinate reference system    (16 bytes) */
    GeoAltRange alt_range;     /* altitude range                 (24 bytes) */
    GeoTimeRange time_range;   /* time range                     (56 bytes) */
    mi_char1    data[1];      /* start of multirep data        */
}
GeoObjectData;
```

The fixed length portion contains fields common to all types derived from GeoObject: the version, coordinate reference system, altitude range, and time range.

The tag field indicates the derived type of GeoObject the variable length portion represents. You use this tag to cast the variable length data to the appropriate type-specific data structure. For example, if the object is a GeoPolygon, you cast as follows:

```
GeoObjectData *d;
GeoPolygonData *p;

if ( d->tag == GeoData_Polygon )
    p = (GeoPolygonData *) d->data;
```




Important: The variable-length portion of the *GeoObjectData* structure (that is, the multirepresentational data) immediately follows the fixed-length portion in memory. You cannot use pointers. When a client program constructs a *GeoObject* for transmission to the server, there can be no pointers inside its *GeoObjectData* structure. The fixed length-portion and the variable-length portion must be a contiguous block of memory:

The *GeoAltRange* and *GeoTimeRange* fields of *GeoObjectData* are each data structures containing the two end points of the range itself, and an *rtype* field indicating various special cases. The following shows the *GeoAltRange* structure.

```
typedef struct
{
    mi_integer    rtype;        /* 4 bytes */
    mi_integer    pad;         /* 4 bytes */
    GeoAltitude   a_lo;        /* 8 bytes */
    GeoAltitude   a_hi;        /* 8 bytes */
}
GeoAltRange;
```

A client program should always check the *rtype* field because the end point data are undefined for the `null` and `any` cases.

The geodetic.h File

This section lists the contents of the **geodetic.h** file.

```

/*****
 *
 *          INFORMIX SOFTWARE, INC.
 *
 *          PROPRIETARY DATA
 *
 *      THIS DOCUMENT CONTAINS TRADE SECRET DATA WHICH IS THE PROPERTY OF
 *      INFORMIX SOFTWARE, INC.  THIS DOCUMENT IS SUBMITTED TO RECIPIENT IN
 *      CONFIDENCE.  INFORMATION CONTAINED HEREIN MAY NOT BE USED, COPIED OR
 *      DISCLOSED IN WHOLE OR IN PART EXCEPT AS PERMITTED BY WRITTEN AGREEMENT
 *      SIGNED BY AN OFFICER OF INFORMIX SOFTWARE, INC.
 *
 *      THIS MATERIAL IS ALSO COPYRIGHTED AS AN UNPUBLISHED WORK UNDER
 *      SECTIONS 104 AND 408 OF TITLE 17 OF THE UNITED STATES CODE.
 *      UNAUTHORIZED USE, COPYING OR OTHER REPRODUCTION IS PROHIBITED BY LAW.
 *
 *****/

#ifndef GEODETIC_H
#define GEODETIC_H

/*-----
 *   geodetic.h
 *
 *   This header file defines the client-side C language interface
 *   for the Informix Geodetic DataBlade module.
 *
 *-----*/

#include <mlib.h>

/*-----
 *   Domain-enforcing data types
 *-----*/

typedef mi_double_precision GeoAltitude; /* arbitrary units; [-inf,+inf] */
typedef mi_double_precision GeoAngle;    /* dec. degrees; range [0,360) */
typedef mi_double_precision GeoAzimuth;  /* dec. degrees; range [0,360) */
typedef mi_double_precision GeoDistance; /* meters; range [0, 2000000] */
typedef mi_double_precision GeoLatitude; /* dec. degrees; range [-90,90] */
typedef mi_double_precision GeoLongitude; /* dec. degrees; range [-180,180] */

/*-----
 *   Range data types
 *
 *   These structures are padded to preserve 8-byte alignment.
 *-----*/
```

```

/*
 * The following values apply to the rtype field of the GeoAltRange and
 * GeoTimeRange structures. They define various special cases of ranges.
 */
typedef enum
{
    GeoRange_Null    = 1,      /* unspecified range */
    GeoRange_Any     = 2,      /* -inf to +inf */
    GeoRange_Equal   = 3,      /* r1 == r2 */
    GeoRange_Normal  = 4,      /* r1 != r2 */
}
GeoRangeEnum;

typedef struct
{
    mi_integer    rtype;        /* really a GeoRangeEnum      (4 bytes) */
    mi_integer    pad;          /* preserves 8-byte alignment (4 bytes) */
    GeoAltitude   a_lo;         /* lower endpoint of alt range (8 bytes) */
    GeoAltitude   a_hi;         /* upper endpoint of alt range (8 bytes) */
}
GeoAltRange;

typedef struct
{
    mi_integer    rtype;        /* really a GeoRangeEnum      (4 bytes) */
    mi_integer    pad;          /* preserves 8-byte alignment (4 bytes) */
    mi_datetime   t_lo;         /* lower endpoint of time range (24 bytes) */
    mi_datetime   t_hi;         /* upper endpoint of time range (24 bytes) */
}
GeoTimeRange;

/*-----
 * GeoCoords data type.
 *
 * This is a fully functional SQL data type for storing geodetic points
 * with an associated datum.
 *-----*/

typedef struct
{
    GeoLatitude    lat;
    GeoLongitude   lon;
    mi_ref         coord_ref;    /* coordinate reference system */
} GeoCoords;

/*-----
 * Spatio-temporal data types.
 *
 * These structures are padded to preserve 8-byte alignment.
 *-----*/

```

The geodetic.h File

```
/*
 * GeoObjectData structure version number.
 * Use this in the version field of the GeoObjectData structure.
 */
#define GEO_OBJECT_VERSION 2

/*
 * The following values apply to the tag field of the GeoObjectData
 * structure. They are used to distinguish the various
 * multi-representational types of objects.
 */
typedef enum
{
    GeoData_Point    = 1,
    GeoData_Circle   = 2,
    GeoData_LineSeg  = 3,
    GeoData_String   = 4,
    GeoData_Ring     = 5,
    GeoData_Polygon  = 6,
    GeoData_Box      = 7,
    GeoData_Ellipse  = 8
}
GeoDataEnum;

/*
 * GeoObject is an opaque data type for encapsulating variable length data.
 */
typedef mi_bitvarying GeoObject;

/*
 * GeoObjectData is the structure which describes that data; it is
 * accessible via an mi_get_vardata() call.
 */
typedef struct
{
    mi_integer    version;      /* version of this structure    (4 bytes) */
    GeoDataEnum   tag;          /* type of this object        (4 bytes) */
    mi_ref        coord_ref;    /* coordinate reference system (16 bytes) */
    GeoAltRange   alt_range;    /* altitude range              (24 bytes) */
    GeoTimeRange  time_range;   /* time range                  (56 bytes) */
    mi_char1      data[8];      /* start of multirep data      (8+ bytes) */
}
GeoObjectData;

/*
 * Structures for each of the multi-representational types of objects.
 * Cast the data field of the GeoObjectData structure to the appropriate
 * structure; examine the tag field of GeoObjectData to determine which
 * one to use.
 */

typedef struct
{
    GeoLatitude    lat;
```

```

        GeoLongitude lon;
    }
    GeoPointData;

    typedef struct
    {
        GeoPointData pt[2];          /* coordinates of endpoints */
    }
    GeoLinesegData;

    typedef struct
    {
        GeoPointData sw;              /* coordinates of southwest corner */
        GeoPointData ne;              /* coordinates of northeast corner */
    }
    GeoBoxData;

    typedef struct
    {
        GeoPointData c;               /* center */
        GeoDistance r;                /* radius in meters */
    }
    GeoCircleData;

    typedef struct
    {
        GeoPointData c;               /* center */
        GeoDistance major;            /* semi-major axis, in meters */
        GeoDistance minor;            /* semi-minor axis, in meters */
        GeoAzimuth azimuth;           /* orientation of semi-major axis */
    }
    GeoEllipseData;

    /*
     * A GeoString is an ordered sequence of points which may cross itself
     */
    typedef struct
    {
        mi_integer npts;               /* number of points */
        mi_integer pad;                /* for 8-byte alignment */
        GeoPointData pt[1];           /* really pt[npts] */
    }
    GeoStringData;

    /*
     * A GeoRing is an ordered sequence of points defining a closed boundary.
     * It does not include the interior area. Closure is implied; the first
     * point in the list must not be the same as the last point.
     */
    typedef struct
    {
        mi_integer npts;               /* number of points */
        mi_integer pad;                /* for 8-byte alignment */
        GeoPointData pt[1];           /* really pt[npts] */
    }

```

The geodetic.h File

```
}
GeoRingData;

/*
 * A GeoPolygon is a set of one or more rings. Each ring defines a
 * region (a boundary plus its interior area). The interior is defined
 * by the point order of each ring; the interior is always to the left.
 * All rings are concatenated together in a single contiguous block
 * of memory; there are no pointers. In other words, the ring[] field
 * is not truly an array of GeoRingData structures; to advance from one
 * ring to the next you must use pointer arithmetic to skip over the
 * point data of a ring.
 */
typedef struct
{
    mi_integer    nrings;        /* no. of rings          */
    mi_integer    pad;          /* for 8-byte alignment */
    GeoRingData   ring[1];      /* start of ring data   */
}
GeoPolygonData;

#endif /* GEODETIC_H */
```

Example Programs

This appendix lists and briefly describes the Informix Geodetic DataBlade module sample files. See the latest Informix Geodetic DataBlade module release notes for the path and name of the directory containing the example files.

There are four categories of example files:

- Domain-enforcing types
- Range types
- Spatio-temporal types
- Client programs

A separate table lists and describes the examples in each of these categories.

Domain Enforcing Types

The following is a list of the example files for the domain-enforcing types, with a brief description of each.

Filename	Description—Domain-Enforcing Types
alt1.sql	GeoAltitude constructors
alt2.sql	Arithmetic operations with the GeoAltitude data type
angle1.sql	GeoAngle constructors
angle2.sql	GeoAngle constructors, showing modulo -360 behavior
angle3.sql	Boolean operations with the GeoAngle data type
angle4.sql	Arithmetic operations with the GeoAngle data type
angle5.sql	Arithmetic operations with the GeoAngle data type
angle6.sql	Incorrect arithmetic angle operations
azimuth1.sql	GeoAzimuth constructors
azimuth2.sql	Arithmetic operations with the GeoAzimuth data type
dist1.sql	GeoDistance constructors
dist2.sql	Incorrect GeoDistance constructors
dist3.sql	Incorrect GeoDistance constructors
lat1.sql	GeoLatitude constructors
lat2.sql	Incorrect GeoLatitude constructors
lat3.sql	Arithmetic operations with the GeoLatitude data type
lat4.sql	Incorrect arithmetic angle operations
long1.sql	GeoLongitude constructors
long2.sql	Incorrect GeoLongitude constructors
long3.sql	Arithmetic operations with the GeoLongitude data type

Range Types

The following is a list of the example files for the range types, with a brief description of each.

Filename	Description—Range Types
altrang1.sql	GeoAltRange constructors
altrang2.sql	Bottom and Top accessor functions
altrang3.sql	Incorrect altitude ranges
timerng1.sql	GeoTimeRange constructors
timerng2.sql	Begin and End accessor functions
timerng3.sql	Incorrect GeoTimeRange constructors
datetime.sh	Interaction of GL_DATETIME with the GeoTimeRange data type

Spatio-Temporal Types

The following is a list of the example files for the spatio-temporal types, with a brief description of each.

Filename	Description—Spatio-temporal Types
circle.sql	GeoCircle constructors
ellipse.sql	GeoEllipse constructors
lineseg.sql	GeoLineseg constructors
point.sql	GeoPoint constructors
polygon.sql	GeoPolygon constructors
ring.sql	GeoRing constructors
string.sql	GeoString constructors

Client Programs

The following is a list of the example client program files, with a brief description of each.

Filename	Description—Client Programs
insert.c	Construction of client GeoObjects and insertion into a table
fetch.c	Retrieval and display of data from a table
fetchetup.sql	Data set for fetch program
makefile	Makefile for compiling fetch and insert programs

Index

A

Access methods
 about 6-4
 B-tree 6-4
 R-tree
 about 1-8, 6-3 to 6-9
 operators for 5-6, 6-5
 using 6-5
 Accessor functions, list of 4-4
 Altitude range
 about 1-5
 ANY, specifying for 2-10, 3-7, 3-8,
 4-13, 4-59
 changing 3-8
 specifying 3-7
 See also GeoAltRange data type,
 GeoAltRange function
 AltRange function 3-8, 4-7
 Angular data types 3-3
 Axis
 semi-major 3-27, 3-29, 4-47
 semi-minor 3-27, 3-29, 4-48
 Azimuth
 definition 3-17
 definition of 3-17
 measurement of 3-16
 Azimuth function 3-16, 4-8

B

Begin function 4-9
 Beyond function 3-27, 4-10 to 4-12,
 5-4, 5-5, 6-9
 Binary operators 5-3, 5-6, 6-7
 Bottom function 4-13

Bounding box 1-8, 6-4, 6-7
 Box. *See* GeoBox data type.

C

Cartesian coordinate reference
 system 1-8
 Circle. *See* GeoCircle data type.
 Client validation utility 2-9
 Coordinate reference system
 Cartesian 1-8
 geodetic 1-4
 latitude/longitude 1-4
 Coords function 4-14

D

Data types 3-3 to 3-62
 angular 3-3
 domain-enforcing
 about 3-3
 definition of 2-4
 unit of measure for 3-3
 GeoAltitude 3-5 to 3-6
 GeoAltrange 3-7 to 3-10
 GeoAngle 3-11 to 3-15
 GeoAzimuth 3-3, 3-16 to 3-18, 4-8
 GeoBox 2-11, 3-19 to 3-21
 GeoCircle 3-22 to 3-24
 converting from GeoObject 4-18
 radius length 3-22
 radius length, determining 4-53
 GeoCoords 3-25
 GeoDistance 3-27
 GeoEllipse 3-29 to 3-32, 4-8
 GeoLatitude 3-3, 3-33 to 3-35

GeoLineseg 3-36 to 3-38
 GeoLongitude 3-3, 3-39 to 3-40
 GeoObject 3-41 to 3-43
 GeoPoint 3-44 to 3-46
 GeoPolygon 3-47 to 3-52
 GeoRing 3-53 to 3-55
 GeoString 3-56 to 3-58
 GeoTimeRange 3-59 to 3-62
 linear 3-3
 spatio-temporal 1-3
 using 2-3 to 2-5
 Date and time precision,
 setting 2-11 to 2-16
 Datum
 geodetic 1-4
 North American 1927 1-4
 World Geodetic System 1984 1-4,
 1-5, 2-4, 3-25
 Dimension
 altitude 1-5
 time 1-6, 6-7
 Direction cosines 1-7
 Distance 1-6, 3-27, 3-56
 Domain-enforcing data types
 about 3-3
 definition of 2-4
 unit of measure for 3-3

E

Earth
 ellipsoidal representation 1-3
 specifying the entire 2-11, 3-19
 Ellipse. *See* GeoEllipse data type.
 End function 4-15
 Equator 1-5, 3-3, 3-50
 Error messages 2
 External representation of spatio-
 temporal objects, converting to
 internal 2-5

F

Functions 4-3 to 4-61
 AltRange 3-8, 4-7
 Azimuth 3-16, 4-8
 Begin 4-9

Beyond 3-27, 4-10 to 4-12, 5-4, 5-5,
 6-9
 Bottom 4-13
 case insensitivity of 4-3
 categories of 4-3
 Coords 4-14
 End 4-15
 executing 4-3
 GeoAltRange 4-16
 GeoBox 4-17
 GeoCircle 4-18 to 4-19
 GeoCoords 4-20
 GeoEllipse 4-21 to 4-23
 GeoLineseg 4-23
 GeoPoint 4-24
 GeoPolygon 4-25 to 4-27
 GeoRelease 4-26
 GeoString 4-28
 GeoTimeRange 4-29
 Inside 4-30 to 4-31, 5-5
 Intersect 4-32, 5-5
 IsGeoBox 4-34
 IsGeoCircle 4-35
 IsGeoEllipse 4-36
 IsGeoLineseg 4-37
 IsGeoPoint 4-38
 IsGeoPolygon 4-39
 IsGeoRing 4-40
 IsGeoString 4-41
 IsValidGeometry function 4-42
 IsValidSDTS function 4-44
 Latitude 3-33, 4-45
 Longitude 3-39, 4-46
 Major 3-27, 4-47
 Minor 3-27, 4-48
 NPoints 4-49
 Outside 4-51, 5-4, 5-5
 Radius 4-53
 Ring 4-54
 SetAltRange 3-8, 4-55
 SetDist 3-27, 4-56
 SetTimeRange 3-60, 4-57
 TimeRange 3-60, 4-58
 Within 3-27, 4-60, 5-4, 5-5, 6-8

G

GeoAltitude data type 3-5 to 3-6

GeoAltRange data type 3-7 to 3-10
 GeoAltRange function 4-16
 GeoAngle data type 3-11 to 3-15
 GeoAzimuth data type 3-3,
 3-16 to 3-18, 4-8
 GeoBox data type 2-11, 3-19 to 3-21
 GeoBox function 4-17
 GeoCircle data type 3-22 to 3-24
 converting from GeoObject 4-8,
 4-18
 radius length 3-22
 radius length, determining 4-53
 GeoCircle function 4-18 to 4-19
 GeoCoords data type 2-4, 2-8, 3-25
 GeoCoords function 4-20
 Geodesic 3-22, 3-56, 4-60
 definition 1-6
 Geodesy 1-3
 Geodetic coordinate reference
 system 1-4
 Geodetic datum 1-4
 Geodetic latitude. *See* Latitude.
 geodetic.h header file A-1, A-8
 GeoDistance data type 3-27
 GeoEllipse data type 3-29 to 3-32,
 4-8
 major axis, returning with 4-47
 minor axis, returning with 4-48
 GeoEllipse function 4-21 to 4-23
 GeoLatitude data type 3-3,
 3-33 to 3-35
 allowed operators 3-33
 GeoLineseg data type 3-36 to 3-38
 GeoLineseg function 4-23
 GeoLongitude data type 3-3,
 3-39 to 3-40
 allowed operators 3-39
 GeoObject data type 3-41 to 3-43
 C interface for A-1 to A-3
 conversion function 3-42
 ESQL/C interface for A-1 to A-3
 functions that take
 GeoObject 3-42
 GeoBox function, for conversion
 of 4-17
 GeoCircle function, for
 conversion of 4-18
 GeoEllipse function, for
 conversion of 4-21

GeoLineseg function, for
 conversion of 4-23
 GeoPoint function, for conversion
 of 4-24
 inserting objects in a column
 of 3-41
 retrieving objects from a column
 of 3-42
 testing for original
 type 4-34 to 4-41
 verifying correctness of 2-9, 4-42,
 4-44
 GeoObject_ops operator class 5-6
 GeoPoint data type 3-44 to 3-46
 GeoPoint function 4-24
 GeoPolygon data type 3-47 to 3-52
 determining number of
 points 4-49
 exclusion area 3-50
 rings, determining number of
 with 4-50
 specifying the Northern and
 Southern hemispheres
 with 3-50
 verifying correctness of 2-9, 4-42,
 4-44
 GeoPolygon function 4-25 to 4-27
 GeoRelease function 4-26
 GeoRing data type
 verifying correctness of 2-9, 4-42
 GeoString data type 3-56 to 3-58
 determining number of
 points 4-49
 GeoString function 4-28
 GeoTimeRange data
 type 3-59 to 3-62
 casting to a specific precision 2-15
 effect of GL_DATETIME
 environment variable
 on 2-11 to 2-16
 precision of 2-13 to 2-15
 GeoTimeRange function 4-29
 geovalidate utility 2-9
 GL_DATETIME environment
 variable
 default precision of 2-12
 effect on GeoTimeRange
 data 2-11 to 2-16
 format of 2-11

Greenwich Meridian. *See* Prime
 Meridian.

H

Header file, geodetic.h A-1 to A-8
 Hemispheres, specifying the
 Northern and Southern 3-50
 High-Performance Loader 2-7
 HPL. *See* High Performance Loader.

I

Index scan 6-7
 Indexes
 access methods for 6-4
 creating 1-8, 6-6
 operator classes for 6-5
 R-tree
 creating 6-5
 using 6-5
 syntax for 6-3, 6-6
 using 5-6, 6-7
 Input functions 2-5
 INSERT statements 2-6
 Inside function 4-30 to 4-31, 5-5
 Internal representation of spatio-
 temporal objects
 about 2-5
 converting to external 2-5
 Internal support functions 2-5
 Intersect function 4-32, 5-5
 IsGeoBox function 4-34
 IsGeoCircle function 4-35
 IsGeoEllipse function 4-36
 IsGeoLineseg function 4-37
 IsGeoPoint function 4-38
 IsGeoPolygon function 4-39
 IsGeoRing function 4-40
 IsGeoString function 4-41
 IsValidGeometry function 4-42
 IsValidSDTS function 4-44

L

Large objects 2-6
 Latitude function 3-33, 4-45

Latitude, geodetic
 definition of 1-4
 internal representation of 1-7
 Linear data types 3-3
 LOAD statements 2-6
 Longitude 1-4
 Longitude function 3-39, 4-46
 Longitude, geodetic
 definition of 1-4
 internal representation 1-7

M

Major function 3-27, 4-47
 Map projection Intro-10
 Meridian 1-4
 Messages
 error 2
 warning 1
 Minor function 3-27, 4-48
 mi_exec DataBlade API
 function 4-3

N

North American 1927 datum 1-4
 Northern hemisphere,
 specifying 3-50
 NPoints function 4-49

O

Operator classes 5-6, 6-5
 Operators
 binary. *See* Binary operators.
 spatial. *See* Spatial operators.
 ternary. *See* Ternary operators.
 Output functions 2-5
 output functions 2-5
 Outside function 4-51, 5-4, 5-5

P

Point. *See* GeoPoint data type.
 Precision, date and time. *See* Date
 and time precision, setting.
 Prime Meridian 1-4, 3-3

Proximity test. *See* SetDist function.

Q

Queries, rewriting for ternary operators 5-6, 6-7

R

Radius function 4-53
 Ring function 4-54
 R-tree access method
 about 1-8, 6-3 to 6-9
 operators for 5-6, 6-5
 using 6-5
 rtree_ops operator class 6-6

S

SDTS definition of a polygon 4-44
 Semi-major-axis 3-29, 4-47
 Semi-major axis 3-27
 Semi-minor axis 3-27, 3-29, 4-48
 SetAltRange function 3-8, 4-55
 SetDist function 3-27, 4-56
 rewriting ternary operator query with 6-7
 SetTimeRange function 3-60, 4-57
 Southern hemisphere,
 specifying 3-50
 Spatial access method. *See* R-tree access method.
 Spatial Data Transfer Standard. *See* SDTS definition of a polygon.
 Spatial operators 5-3 to 5-6, 6-5
 spatial_ops operator class 6-5
 Spatio-temporal objects
 distance parameter, adding to 4-56
 external representation,
 converting to internal 2-5
 inserting into a table 2-6
 internal representation of 2-5
 kinds of 1-3
 storing as large objects 2-6
 time range, changing for 4-57

verifying correctness of 2-9, 4-42, 4-44

See also data types for individual objects, such as, GeoBox, GeoCircle and so on.

String. *See* GeoString data type.

Support functions 2-5

System functions 4-5

T

Ternary operators 5-6, 6-7
 Time dimension 1-6
 Time precision, setting. *See* Date and time precision, setting.
See also GeoTimeRange data type, GeoTimeRange function.
 Time range
 ANY 2-10, 3-59, 3-61
 changing an object's time range 3-60, 4-57
 selecting by time range 2-10
 TimeRange function 3-60, 4-58

U

UNIX Intro-11

V

Validating spatio-temporal data 2-9, 4-42, 4-44

W

Warning messages 1
 WGS-84 datum. *See* World Geodetic System 1984 datum.
 Whole Earth. *See* Earth, specifying the entire.
 Within function 3-27, 4-60, 5-4, 5-5, 6-8
 World Geodetic System 1984 datum 1-4, 1-5, 2-4, 3-25