4. Part 4: Address Data Quality

4.1 Introduction

4.1.1 Purpose

The purpose of Part Three is to help users assess the quality of their address data. It provides ways to measure the caliber of each element, attribute, and classification. Some measures compare values to Address Reference System (ARS) specifications or domains of values. Others check internal consistency, one of the most important aspects of addresses. Addresses are interdependent: the validity of all can be affected by some. Parity, for example, is an important part of address assignment. In most address reference schemes, even and odd addresses on the same side of the street disrupt normal address usage. While the assignment of each address is important it is the pattern of assignment, usually described in the (ARS) that make the system work. The measures describe ways to discover anomalies, in isolation, in relationships between data and in relationship to the ARS, and how to report the quality of the data.

4.1.2 Quality definition

Measuring quality requires a definition for quality. Existing standards provide ample source material. Those standards describing addresses, however, focus on the utility of a data set for specific purposes. The National Emergency Number Association (NENA), for example, has documented exchange standards that include a way to describe address quality relative to established Automatic Location Information (ALI) files. Similarly, the United States Postal Service (USPS) Postal Addressing Standards describe addresses as used for mailing. These application-specific assessments fulfill the purpose of their respective documents.

Assessing the quality of address data content independent of formats or specific uses, however, is a very different task. It requires information on each of the many aspects of address information. Evaluations for a specific purpose can be constructed from that information, with criteria varying according to each application. Constructing more generically useful methods for assessing address quality starts with the structure for those methods, the elements of address quality.

4.1.2.1 Elements of Address Quality

Definitions for the quality of address data content are supplied by standards for spatial metadata. These are primarily represented by two documents, the ISO 19115:2003 Geographic information -- Metadata and Content Standard for Digital Geospatial Metadata (CSDGM), Vers. 2 (FGDC-STD-001-1998). The Spatial Data Transfer Standard (SDTS: ANSI NCITS 320-1998) touches on the subject, as does ISO 19113:2002 Quality Principles.

These standards have quality reporting requirements that identify the elements of the genus spatial data, of which addresses are a species. These elements provide guidance for the various aspects of quality control, and for classifying the various measures listed in

this section. If all of them are satisfied, a data set has been thoroughly checked. The measures are listed by quality element in Appendix F.

There is remarkable agreement among the documents on the elements of spatial data quality. Each of the standards that approach that question describes the same core elements:

- Attribute (Thematic) Accuracy
- Completeness
- Lineage
- Logical Consistency
- Positional Accuracy
- Temporal Accuracy

Even the names of the elements are essentially identical. CSDGM and SDTS discuss "Attribute Accuracy", while the same content is described as "Thematic Accuracy" in ISO 19115. ISO 19113 includes temporal accuracy as one of the data quality elements, defined as "accuracy of the temporal attributes and temporal relationships of features" (ISO 19113:2002(E), 5.2.1), and this definition remains constant throughout the ISO series. Temporal attributes are not separated from other types of attributes in the CSDGM, although various types of time period entries are listed throughout the standard.

Quality Elements	Standards			
	CSDGM	SDTS	ISO 19113	ISO 19115
Dataset Purpose	†	•	•	•
Dataset Use	†	•	•	•
Attribute (Thematic) Accuracy	•	•	•	•
Logical Consistency	•	•	•	•
Temporal Accuracy	†	†	•	•
Completeness	•	•	•	•
Positional Accuracy	•	•	•	•
Lineage	•	•	•	•

• Specified by name in the standard.
† Provision made for the information in the standard, but not specified by name.

4.1.3 Anomalies: Uncertainty and Addresses

Quality control for address content is unusual in that it is normal to carry inconsistencies in a data set indefinitely. Those inconsistencies are simply the result of the addressing process. A given locality may not have always applied its Address Reference System (ARS) systematically, or the ARS may have changed. Street names may have changed, or the ground conditions changed in some other way.

These inconsistencies are called "anomalies" throughout this document. It is difficult to call them errors: the conditions that create the anomalies will persist, and many of the individual inconsistencies will never be changed. Finally, address information is essential to core, shared databases in many enterprises. Sharing the information involves

communications of all kinds, including face-to-face conversations. Plainly, the word "error" can be less than diplomatic in workplace discussions.

4.1.3.1 Using Address Anomaly Status

The Address Anomaly Status attribute provides a way of documenting and accounting for anomalies. This attribute should be assigned after an inconsistent address is researched. After it is assigned, records documented as anomalies can be excluded from quality testing. This practice reduces ambiguity, and prevents repeated research on the same addresses.

4.2 Measuring Address Quality

4.2.1 About the Measures

The quality control tests follow a simple recipe:

1. Compare address data to domains and specifications tailored for local use
2. Identify anomalies

Tests are designed to provide data quality element information as described in ISO 19115. The test specification includes:

- Scope: the elements, attributes or classifications to be tested
- Measure: a description of what the test measures.
- Procedure: a description of the test
- Script or Function: an example of the test in SQL code, or pseudocode where the exact parameters are more difficult to anticipate. The scripts and functions were written (except where noted otherwise) using standard ISO/IEC 9075-1:2008 SQL. Exact coding will vary from system to system. Spatial predicates used in the measures are described in OpenGIS Simple Features Specification for SQL. Where the code or pseudocode uses predicates beyond the SFSQL standard, they are also noted.
- Parameters for calculating anomalies as a percentage of the data set

4.2.1.1 About Anomalies

Measures are described with the understanding that records with known anomalies are excluded from related tests. New anomalies discovered should be corrected or described with Address Anomaly Status attributes.

4.2.1.2 Calculating Conforming Records as a Percentage of the Data Set

The Perc Conforming function measures the results of test for anomalies and describes the percentage of data elements that conform. Calculating the percentage of conformance requires inverting the query: the number of anomalies found is subtracted from the total number of records before calculating the percentage.

FUNCTION: CALCULATING CONFORMING RECORDS AS A PERCENT OF THE DATA SET

Description

The function receives information directly from SQL statements including the standard COUNT aggregator and calculates percentages.

Function

```
CREATE OR REPLACE FUNCTION perc_conforming( integer, integer )
RETURNS numeric as $$

DECLARE
   nonconforming alias for $1;
   total_recs alias for $2;
   calc_perc numeric;

BEGIN

SELECT INTO calc_perc
ROUND( ( ( total_recs - nonconforming )::numeric / total_recs::numeric
)  * 100, 2 );

RETURN calc_perc;

END;

$$ language 'plpgsql';
```

Pseudocode Query

```
SELECT
   perc_conforming
   (
      ( SELECT
           COUNT(*) as nonconforming
        FROM
            Address Collection
        WHERE
            condition is not met
      )::integer,
      ( SELECT
           COUNT(*) as total_recs
        FROM
           Address Collection
      )::integer
  )
;
```

Successful Result: 100% Conforming

```
 perc_conforming
-----------------
          100.00
(1 row)
```

Unsuccessful Result: 30% Conforming

```
 perc_conforming
-----------------
           30.00
(1 row)
```

4.2.1.3 Notation

The tests are described using SQL constructs and operators.

Operators used include:

Operator	Description	SQL Example Statement	Statement Result
||	concatenation	SELECT 'a' || 'b';	ab
%	modulo	SELECT 5 % 2;	1
::	type casting	SELECT '0005'::integer;	5
~	pattern matching	SELECT Complete Street Name where Complete Street Name ~ 'Main';	Main Street

4.2.2 Applying Measures to Domains of Values

Domains of values are an important tool for controlling values for address components. Measures used to test data conformance depend on the type of domain. The Content Standard for Digital Geospatial Metadata (CSDGM) classifies domains as enumerated, range, codeset and unrepresentable. The following table lists the CSDGM definition for each type of domain, as listed in *Section 5: Entity and Attribute Information*, along with the measures associated with each.

Domain Type	CSDGM Definition	Quality Measures	Quality Notes
codeset	"reference to a standard or list which contains the members of an established set of valid values."	Tabular Domain Measure Spatial Domain Measure	The U.S. Postal Service list of "Primary Street Suffix Names" is a familiar example of a codeset domain. In cases where specific street suffixes are associated with a given area in the Address Reference System, the association should also be checked with the Spatial Domain Measure.
enumerated	"the members of an established set of valid values."	Tabular Domain Measure Spatial Domain Measure	A local, validated street name list is an example of an enumerated domain. In cases where specific types of values are associated with a given area in the Address Reference System, the association should also be checked with the Spatial Domain Measure.
range	"the minimum and maximum values of a continuum of valid values."	Range Domain Measure Spatial Domain Measure Address	Range domain examples include such things as minimum and maximum Address Number values set by some jurisdictions, or a range of address values assigned to a given grid cell in the Address Reference System. In the latter case, the Spatial Domain Measure would be required to validate the location of the grid cell. Many Address

Domain Type	CSDGM Definition	Quality Measures	Quality Notes
		Number Fishbones Measure	Number values are associated with a Two Number Address Range or a Four Number Address Range. In the latter case conformance can be checked with the Address Number Fishbones Measure.
unrepresentable	"description of the values and reasons why they cannot be represented."		

4.3 How to use the Measures in a Quality Control Program

4.3.1 Preparation

The measures assume a certain body of knowledge about local addresses. Preparation for a local quality control program largely consists of assembling that information. These include:

- Tabular domains of values for street name components
- Spatial domains: jurisdiction boundaries, address reference scheme boundaries and components
- Address reference schemes: geometry and rules.

Tabular domains are frequently difficult to complete, as no organized list of street names may be available. In the latter case all available sources of street names should be compiled and checked against source documents such as plats and ordinances where possible. Official Status attributes may be helpful in assembling and maintaining domains of values. It will be normal to find a variety of street name abbreviations in use: Doctor Martin Luther King Junior Boulevard, MLK Boulevard, etc. Official Status attributes can help describe variations on street names that may appear in addresses assigned to the same location, or along the same street. As noted throughout the standard, the official name should be completely spelled out: Doctor Martin Luther King Junior Boulevard.

Address reference systems or other local conventions may govern much more than street number assignment. There may be street classification requirements for specific Street Name Post Type names. For instance, some jurisdictions may require "Courts" to be deadends, and forbid "Boulevards" on the same deadends. Street names may conform to themes in particular areas: numbers, birds, trees and presidents are some examples. The latter rule may be satisfied by applying the Spatial Domain Measure, but the former will require locally formulated tests. In any case, local conditions will require attention in drawing up a complete list of standard and local quality measures.

4.3.2 Construction

Once all of the domains and rules have been assembled, use the guidance in Address Data Content and Address Data Classification to assemble a list of measures for each aspect of your data. Informative Appendices D through G can be helpful in maintaining an overview of the process. Order the measures, beginning with the most basic: check simple elements and attributes first, then complex elements, then classifications. In cases where a quality check is beyond the scope of the standard create, name and document your own test, taking care to choose a name that does not duplicate one in the standard. It will be important to have the method completely documented both for maintenance, and in order to convey complete quality information to address users.

The table below shows how specific needs in a quality program for an E911 center match to measures.

Description	Measure
Number/percent of MSAG-valid street segments without street names	RelatedNotNullMeasure
Number/percent of MSAG-valid street segments without address ranges	RelatedNotNullMeasure
Sort by distinct and count: various street name elements	UniquenessMeasure
Street name elements must be MSAG-valid	Related Element Value Measure
All street segments must be broken and snapped at street intersections, ESN boundaries, and community boundaries	Beyond the scope of the standard
Direction of street segment must follow real-world ranges	AddressRangeDirectionalityMeasure
Ranges may not overlap in the same community	OverlappingRangesMeasure
“To” range must be greater than “From” range on each street segment	LowHighAddressSequenceMeasure
Street segments must be free of parity errors	LeftRightOddEvenParityMeasure
Divided highways, freeways, and streets (divided by median) must be depicted as two line segments	Beyond the scope of the standard
The ALI database must have at least a 95% match rate to the GIS centerline layer (98% in other states)	AddressNumberFishbonesMeasure
Centerline address ranges should include ranges in MSAG (i.e. - actual vs. theoretical)	RangeDomainMeasure
Other comparisons (other address DBs)	Related Element Value Measure
Reference: Control points, Other accurate layers (aerial imagery, parcel boundaries, etc.)	Beyond the scope of the standard

-- Example QC program courtesy of Adam Iten

4.3.3 Testing

Construct SQL statements specific to your system from the code or pseudocode given in the standard, and test them. Run all the measures on a test data set to make sure they

produce believable results. Where known address problems are not discovered by the measures, review how the measures are applied and double check the SQL. Check to make sure that all the characteristics of quality are thoroughly tested: attribute (thematic) accuracy, logical consistency, temporal accuracy, completeness, positional accuracy and lineage. Where there is insufficient information to check a given aspect of quality the address process may need review.

4.3.4 Interpreting Results

The measures are written to produce sets of identifiers. In practice it's important to see data in context. In a normalized relational database it is most often easiest to construct a view to display the data you want to see. The AddressPtCollection and StCenterlineCollection views can provide the ancillary information to describe query results.

4.3.5 Implementation

Once a suite of measures has been constructed and tested, implementation consists of deciding when it will be run, and how to handle the results. For example, where a number of datasets from separate organizations are assembled to create a master address repository, a complete suite of tests may be run on each individual dataset before acceptance. The data may only be incorporated into the repository when the anomalies are either attributed and accepted as part of the data set, or resolved. Once the data are incorporated, it is risky to believe the combined data will test identically to each individual data set. While the street name components may be identical, other aspects may be affected by the inherent interdependence of addresses. The results of Address Number Fishbones Measure, for example, may be very different when new data are added. Quality control implementation, therefore, may require developing several suites of quality measures to support each part of the address process.

User confidence in a data set depends on an effective program. Test thoroughly, and document the process as well as the results. Recording dates and times is often an important part of that documentation. Users will question aspects of the data. Knowing the condition of the data over time simplifies response, and increases both the reality and perception of the value of the data.

4.3.6 Maintenance

Addressing is a dynamic process. Just as the construction of testing suites is based on the process, testing suites need to be reexamined each time an addressing process changes.

4.4 How to Prepare Data for Quality Control

No specific database design is required for using the quality measures presented here. The measures rely on the views, tables and pseudocode descriptions listed below.

View, Table or Description	Type	Description
AddressPtCollection	View	A view comprising elements and attributes for thoroughfare addresses, including point geometry.
StCenterlineCollection	View	A view of street centerline segments, including Two Number Address Range or Four Number Address Range attributes and line geometry
Pseudocode data	Descriptions	In some cases a measure may apply broadly. The UniquenessMeasure is one example: it may apply in many circumstances. Data is described in a more general way with pseudocode.
Elevation Polygon Collection	Table	A set of polygons built from contours of elevation, used in the Address Elevation Measure
StreetsNodes	Table	Startpoints and endpoints of street segments, with associated street name information. This table has a foreign key relationship with the Nodes table.
Nodes	Table	Unique point locations found in StreetsNodes. These are, effectively, intersection and endpoints. Where two streets intersect, for example, four records in StreetsNodes have the same Nodes foreign key value

Creating the views and tables described below will simplify using quality control measures. The majority of queries use the views, in the interest of supporting a variety of database designs. These are often useful to maintain in a database for general use. Although they are not in and of themselves a database design, they can assist in the use of a more normalized set of tables. Any of the optional elements or attributes listed in the table may be omitted if they are not required for the data set itself. The views are "wide" and, depending on the complexity of the underlying database may be best supported as materialized views. A materialized view is a table, often maintained by triggers or queries, created instead of a view in the interest of efficiency.

The AddressPtCollection and StCenterlineCollection views are listed below, followed by a brief discussion of the Streets Nodes and Nodes tables required for some of the measures. Finally, the Elevation Polygon Collection required for the Address Elevation Measure is described.

4.4.1 Views

The queries for composing those views will vary according to the design of the underlying database.

4.4.1.1 Address Point Collection (AddressPtCollection)

Field Name	Description
Address ID	Address attribute
AddressCompleteStreetNameID	A unique identification number assigned to each Complete Street Name
Related Transportation Feature ID	Address Transportation Feature ID
Complete Address Number	Address element
Address Number Prefix	Address element
Address Number	Address element
AddressNumberAttached	Example of an Attached Element. Can be inserted between Complete Address Number or Complete Street Name components as needed.
Address Number Suffix	Address element
AddressCompleteStreetName	Address element: Complete Street Name
Street Name Pre Modifier	Address element
Street Name Pre Directional	Address element
Street Name Pre Type	Address element
Separator Element	Address element
Street Name	Address element
Street Name Post Type	Address element
Street Name Post Directional	Address element
Street Name Post Modifier	Address element
Place Name	Address element
Zip Code	Address element
Address Side Of Street	Address attribute
Address Number Parity	Address attribute
Address Authority	Address attribute
Address Feature Type	Address attribute
Address Elevation	Address attribute
Address Lifecycle Status	Address attribute
Official Status	Address attribute
BuildingPermit	A Boolean field describing whether or not a building permit has been issued.
Address Start Date	Address attribute
Address End Date	Address attribute
Address XCoordinate	Address attribute
Address YCoordinate	Address attribute
Address Longitude	Address attribute
Address Latitude	Address attribute
Delivery Address Type	Address attribute
USNational Grid Coordinate	Address attribute
AddressPtGeometry	The point geometry for the address

4.4.1.2 Street Centerline Collection (StCenterlineCollection)

Field Name	Description
Address Transportation Feature ID	Address Transportation Feature ID
StCenterlineCompleteStreetNameID	The unique identification number assigned to the Complete Street Name pertaining to each centerline or transportation feature associated with the address
Range.Low	A placeholder for low values in a Four Number Address Range or Two Number Address Range
Range.High	A placeholder for high values in a Four Number Address Range or Two Number Address Range
StCenterlineCompleteStreetName	Address element: Complete Street Name
StCenterlineStreetNamePreModifier	Address element: Street Name Pre Modifier
StCenterlineStreetNamePreDirectional	Address element: Street Name Pre Directional
StCenterlineStreetNamePreType	Address element: Street Name Pre Type
StCenterlinePreTypeAttachedElement	Address element: Attached Element. This is an example. Attached elements may occur anywhere a Complete Address Number or Complete Street Name.
StCenterlineStreetName	Address element: Street Name
StCenterlineStreetNamePostType	Address element: Street Name Post Type
StCenterlineStreetNamePostDirectional	Address element: Street Name Post Directional
StCenterlineStreetNamePostModifier	Address element: Street Name Post Modifier
PlaceNameLeft	Address element: Place Name
PlaceNameRight	Address element: Place Name
ZipCodeLeft	Address element: Zip Code
ZipCodeRight	Address element: Zip Code
StCenterlineGeometryDirection	The cardinal direction of the line: "east-west" or "north-south". This attribute is part of Address Reference System (ARS) rule set in many areas, included as an example. Elements or attributes required in a given ARS that may be included in this view to support quality control.
Address Range Directionality	Address attribute
StCenterlineGeometry	Line geometry for the street centerline or transportation feature associated with the address

4.4.2 Tables

4.4.2.1 Nodes and StreetNodes

4.4.2.2 About Nodes

Nodes are the end points for each road segment. They are used throughout Address Data Quality in checking features at intersections. The code examples below show how to

create and fill one version of the tables required. There are a wide variety of variations that will work. For example, in a more normalized database the Complete Street Name field may be replaced by a foreign key. The specifics will vary across systems.

The tables are:

1. StreetsNodes, a table correlating nodes with the street names assigned to segments connecting at those nodes.

2. Nodes, a table to hold the nodes themselves.

4.4.2.3 Nodes

Where street segments intersect, multiple segment ends will share the same node geometry. This table selects unique node points. The geometries are matched back to the StreetsNodes table so that each record has a node identifier referencing an unique geometry.

The following transaction creates the table. The Nodes table must be created before StreetsNodes to provide for the reference to it in the latter table.

```
begin;
```

Create a table with a primary key

```
create table Nodes
(
   id serial primary key
)
;
```

Add a geometry column. In most cases, -1 will be replaced by an Address Coordinate Reference System ID

```
select addgeometrycolumn( 'nodes', 'nodes', 'Geom',-1,'POINT',2);

end;
```

4.4.3.4 StreetsNodes

The transaction below creates and fills the table.

```
begin;
```

Create a table for the StreetsNodes, ideally holding only intersections and dead ends. It will also hold pseudonodes where they occur. In many cases it will be desirable to include an identifier for the Complete Street Name value in addition to the text.

```
create table StreetsNodes
(
   id serial primary key,
   Nodesfk integer references Nodes,
   RelatedTransportationFeatureID integer,
   CompleteStreetName varchar(100),
   SegmentEnd varchar(4)
)
;
```

Add a geometry column for StreetsNodes. As with the Nodes table, the -1 will likely be replaced with an Address Coordinate Reference System ID value.

```
select addgeometrycolumn( 'nodes', 'StreetsNodes', 'Geom',-
1,'POINT',2);
```

Fill the StreetsNodes table with data from the StCenterlineCollection

```
insert into StreetsNodes( RelatedTransportationFeatureID,
CompleteStreetName, SegmentEnd, Geom )
(
  select
     id,
     CompleteStreetName,
     'from',
     st_startpoint( a.StCenterlineGeometry )
  from
     StCenterlineCollection
)
union
(
  select
     id,
     CompleteStreetName,
     'to',
     st_endpoint( a.StCenterlineGeometry )
  from
     StCenterlineCollection
)
;
end;
```

Fill the Nodes table from the data captured in StreetsNodes

```
insert into
   Nodes( Geom )
select distinct
   geom
from
   StreetsNodes
;
```

Finally, the statement below fills the nodesfk field in the StreetsNodes table.

```
update
   StreetsNodes
set
   Nodesfk = foo.Nodesfk
from
   (
     select
        a.id as Nodesfk,
        b.id
     from
        Nodes a,
        StreetsNodes b
     where
        equals( a.Geom, b.Geom )
   ) as foo
```

```
where
   foo.id = StreetsNodes.id
;

end;
```

4.4.3.5 Elevation Polygon Collection

Field Name	Description
ElevationPolygonID	Primary key
AddressElevationMin	Lowest elevation of the contours bounding the polygon
AddressElevationMax	Highest elevation of the contours bounding the polygon
ElevationPolygonGeometry	Polygon geometry

4.5 Quality Measures

4.5.1 AddressCompletenessMeasure

4.5.1.2 Measure Name
Address Completeness Measure

4.5.1.3 Measure Description
This measure compares the number of addressable objects with the address information recorded. There are a number of circumstances where more than one address is assigned to an addressable object. Addressable objects without addresses, however, are anomalies unless described by Address Anomaly Status attributes.

4.5.1.4 Report
Completeness

4.5.1.5 Evaluation Procedure
Compare the number of addressable objects with the address information recorded.

4.5.1.6 Spatial Data Required
Geometry describing addressable objects attributed with Address ID, and polygon(s) describing Address Reference System extent. The example below uses the AddressPtCollection view.

4.5.1.7 Code Example: Testing Records
Note that this query assumes that both the feature types and the addresses are assumed to be within a given Address Reference System Extent.

```
SELECT
   a.AddressFeatureType
FROM
   AddressFeatureType a
      LEFT JOIN AddressPtCollection b
         ON  a.AddressFeatureType = b.AddressFeatureType
```

```
                INTERSECTS( a.Geometry, b.AddressPtGeometry )
WHERE
   b.AddressID is null
```

4.5.1.8 Code Example: Testing the Conformance of a Data Set

FUNCTION
See Perc Conforming for the sample query

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   COUNT(*)
FROM
   AddressFeatureType a
      LEFT JOIN AddressPtCollection b
         ON  b.AddressFeatureType = a.AddressFeatureType
             INTERSECTS( a.Geometry, b.AddressPtGeometry )
WHERE
   b.AddressID is null
```

count_of_total_records

```
SELECT
   COUNT( a.* )
FROM
   FeatureType a
```

4.5.1.9 Result Report Example

Tested Address Completeness Measure at 87% conformance.

4.5.2 AddressElevationMeasure

4.5.2.1 Measure Name

AddressElevationMeasure

4.5.2.2 Measure Description

This measure checks each elevation in an address point collection against polygons created from contours of elevation.

4.5.2.3 Report

Attribute (Thematic) Accuracy

4.5.2.4 Evaluation Procedure

Check each elevation identified by the measure as outside the range defined by the polygons.

4.5.2.5 Spatial Data Required

AddressPtCollection, Elevation Polygon Collection

4.5.2.6 Code Example: Testing Records

```
SELECT
   a.AddressID
FROM
   AddressPtCollection a
      LEFT JOIN ElevationPolygonCollection b
         ON INTERSECTS( a.AddressPtGeometry, b.ElevationPolygonGeometry
)
WHERE
   NOT( a.AddressElevation BETWEEN b.AddressElevationMin and
b.AddressElevationMax )
```

4.5.2.7 Code Example: Testing the Conformance of the Data Set

FUNCTION

See Perc Conforming for the sample query

FUNCTION PARAMETERS

count_of_non_conforming_records

```
      SELECT
         COUNT(*)
      FROM
          AddressPtCollection a
       LEFT JOIN ElevationPolygonCollection b
          ON INTERSECTS( a.AddressPtGeometry,
   b.ElevationPolygonGeometry )
      WHERE
         NOT( a.AddressElevation BETWEEN b.AddressElevationMin and
   b.AddressElevationMax )
```

count_of_total_records

```
      SELECT
         COUNT( * )
      FROM
         AddressPtCollection
```

4.5.2.8 Result Report Example

Tested AddressElevationMeasure at 90% conformance.

4.5.3 AddressLeftRightMeasure

4.5.3.1 Measure Name

AddressLeftRightMeasure

4.5.3.2 Measure Description

This measure checks stored values describing left and right against those found by geometry. Left and right attributes are frequently entered by hand, an error-prone process. It is important to confirm the actual locations of the addresses. Even where the initial left/right information was derived from the geometry, edits to the data may have changed the parity relationships. This information is central to confirming the conformance of the address assignment to the local Address Reference System.

Note that the measure is constructed with overlapping ranges where an address is found precisely aligned with the road centerline. In these cases two records will be generated: one describing the point on the left side of the centerline, another describing it on the right. In these few cases it is simply practical to use the record that conforms to the Address Reference System and eliminate the other.

Address Left Right Measure is a prerequisite to Left Right Odd Even Parity Measure. An example of finding these duplicate records is included, as well as an example of comparing the mathematically determined sides against those recorded by hand.

Remember when examining the results that the side is determined by the Address Range Directionality of the centerline geometry. If all the from ends of the centerline segments are at the low addresses and the to ends of the centerline segments at the high addresses, then the results will be consistent. In the latter case, it is possible to evaluate whether the odd and even Address Number values are consistently on the left or right of the segment without also accounting for Address Range Directionality. Where Address Range Directionality is inconsistent, however, it must also factor into left/right evaluation.

4.5.3.3 Report

Logical Consistency

4.5.3.4 Evaluation Procedure

Determine the left/right status of the location of each address point. Where there is a value recorded in the database, check it against the side as calculated.

4.5.3.5 Spatial Data Required

Street centerline (or other transportation feature) and address point locations. The Address Transportation Feature ID for the transportation feature associated with each address must be recorded with the address points

4.5.3.6 Code Example: Assembling Data from Views

```
--
-- Calculate the angle at which a line drawn from the address point to
the
-- closest point along the centerline meets a specific segment and
determine
-- right and left from that angle.
--
-- Insert the left/right results, along with all the relevant
identifiers, into a table.
--
```

```
CREATE TABLE AddressLeftRight
(
   id serial primary key,
   "AddressID" text,
   "AddressTransportationFeatureID" text,
   "Side"
)
;


INSERT INTO AddressLeftRight
   (
      "AddressID",
      "AddressTransportationFeatureID",
      "Side"
   )
SELECT DISTINCT
   foo."AddressID",
   foo."AddressTransportationFeatureID",
   CASE
      WHEN
         (
           degrees( azimuth( foo."Pt1", foo."AddressPtGeometry" ) )
           -
           degrees( azimuth( foo."Pt1", foo."Pt2" ) )
         ) between 0 and 180
      THEN 'right'
      WHEN
         (
           degrees( azimuth( foo."Pt1", foo."AddressPtGeometry" ) )
           -
           degrees( azimuth( foo."Pt1", foo."Pt2" ) )
         ) between 180 and 360
      THEN 'left'
      WHEN
         (
           degrees( azimuth( foo."Pt1", foo."AddressPtGeometry" ) )
           -
           degrees( azimuth( foo."Pt1", foo."Pt2" ) )
         ) between -180 and 0
      THEN 'left'
      WHEN
         (
           degrees( azimuth( foo."Pt1", foo."AddressPtGeometry" ) )
           -
           degrees( azimuth( foo."Pt1", foo."Pt2" ) )
         ) between -360 and -180
      THEN 'right'
   END as "Side"
FROM
   --
   -- Calculate the point on the related centerline closest to the
address.
   -- Calculate the start point and end point of each segment of the
centerline.
   --
   (
```

```
     SELECT
        a."AddressID",
        b."AddressTransportationFeatureID",
        a."AddressPtGeometry",
        st_line_interpolate_point
           ( b."StCenterlineGeometry",
             st_line_locate_point( b."StCenterlineGeometry",
a."AddressPtGeometry" )
           ) as "ClosestPtOnStCenterline",
        pointn
           ( b."StCenterlineGeometry",
             generate_series( 1, ( numpoints( b."StCenterlineGeometry"
) - 1 ) )

           ) as "Pt1",
        pointn
           ( b."StCenterlineGeometry",
             generate_series( 2, numpoints( b."StCenterlineGeometry" )
)
           ) as "Pt2"
     FROM
        address."AddressPtCollection" a
           inner join address."StCenterlineCollection" b
             on a."RelatedTransporationFeatureID" =
b."AddressTransportationFeatureID"
   ) as foo
WHERE
   st_intersects( st_expand( foo."ClosestPtOnStCenterline", 1 ),
               st_makeline( foo."Pt1", foo."Pt2" )
             )
;
```

4.5.3.7 Notes

The query to assemble left-right information contains a number of functions proprietary to PostGIS and PostgreSQL as listed below.

st_line_locate_point

Linear referencing function to determine the location of the closest point on a given linestring to a given point.

st_line_interpolate_point

Linear referencing function to create a point at a specified location along a linestring.

st_makeline

Geometry constructor.

generate_series

A set-returning function that generates a series of values.

4.5.3.8 Code Example: Checking for Address Points with both Left and Right Records

This query should describe few, if any records. Such records occur when an address point is perfectly align with one end of a centerline, for example at the end of a cul-de-sac. These addresses should be resolved in favor of the local left-right parity rules before proceeding with queries based on left/right data.

```
SELECT
    foo.AddressID,
    bar.Side
FROM
    (
      SELECT
         Address ID
      FROM
         AddressLeftRight
      GROUP BY
         Address ID
      HAVING
          count( Address ID ) > 1
    ) as foo,
    AddressLeftRight as bar
WHERE
    foo.AddressID = bar.AddressID
;
```

4.5.3.9 Code Example: Checking Left/Right Attributes

This query produces a list of Address ID values for which the left/right attribute cannot be checked by the left/right information in the table populated by the queries above, or where the left/right attribute conflicts with query results.

```
SELECT
   a.AddressID
FROM
   AddressPtCollection a
      LEFT JOIN AddressLeftRight b
         ON a.AddressID = b.AddressID
WHERE
    a.Side != b.Side
;
```

4.5.3.10 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_nonconforming_records

```
SELECT
   count( a.AddressID )
FROM
   AddressPtCollection a
      left join AddressLeftRight b
      on a.AddressID = b.AddressID
```

```
WHERE
   a.Side != b.Side
;
```

count_of_total_records

```
SELECT
   count(*)
FROM
   AddressPtCollection
;
```

4.5.3.11 Result Report Example

Tested Address Left Right Measure at 85% conformance.

4.5.4 AddressLifecycleStatusDateConsistencyMeasure

4.5.4.1 Measure Name

AddressLifecycleStatusDateConsistencyMeasure

4.5.4.2 Measure Description

This measure tests the agreement of the Address Lifecycle Status with the development process. This query is far more conceptual than many others in this section of the standard for the simple reason that the development process, and the data it generates, vary considerably from place to place.

It is common to track the starting and ending dates of each Address Lifecycle Status value. Address Start Date and Address End Date are notably different, not directly attached to any given Address Lifecycle Status value. Checking the validity of any given Address Lifecycle Status requires checking both the Address Start Date and Address End Date values, and data from the development process.

The query for testing records assumes a process where the issuance of a building permit describes the transition of an address from potential or proposed to active. Any given development process is likely to have a longer, more complicated list of conditions. Reports using this measure should include the final query used.

4.5.4.3 Report

Temporal Accuracy and/or Logical Consistency

4.5.4.4 Evaluation Procedure

Check entries where the Address Lifecycle Status conflicts with the Address Start Date or Address End Date, or with the development process. Refine the query as needed to track the development process as it affects Address Lifecycle Status and make sure the records are contemporaneous.

4.5.4.5 Spatial Data Required

None

4.5.4.6 Code Example: Testing Records

```
SELECT
   AddressID
FROM
   AddressPtCollection
WHERE
   (
      AddressLifecycleStatus = 'Potential'
      AND
      ( BuildingPermit IS NOT NULL
        OR
        AddressStartDate IS NULL
        OR
        AddressEndDate IS NOT NULL
      )
   )
   OR
   (
      AddressLifcycleStatus = 'Proposed'
      AND
      ( BuildingPermit IS NOT NULL
        OR
        AddressStartDate IS NULL
        OR
        AddressEndDate IS NOT NULL
   )
   OR
   (
      AddressLifecycleStatus = 'Active'
      OR
      AddressStartDate IS NULL
      OR
      AddressEndDate IS NOT NULL
   )
   OR
   (
      AddressLifecycleStatus = 'Retired'
      AND
      AddressEndDate IS NULL
   )
```

4.5.4.7 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

- count_of_non_conforming_records

```
SELECT
   count( * )
FROM
```

```
      AddressPtCollection
   WHERE
      (
         AddressLifecycleStatus = 'Potential'
         AND
         ( BuildingPermit IS NOT NULL
           OR
           AddressStartDate IS NULL
           OR
           AddressEndDate IS NOT NULL
         )
      )
      OR
      (
         AddressLifcycleStatus = 'Proposed'
         AND
         ( BuildingPermit IS NOT NULL
           OR
           AddressStartDate IS NULL
           OR
           AddressEndDate IS NOT NULL
      )
      OR
      (
         AddressLifecycleStatus = 'Active'
         OR
         AddressStartDate IS NULL
         OR
         AddressEndDate IS NOT NULL
      )
      OR
      (
         AddressLifecycleStatus = 'Retired'
         AND
         AddressEndDate IS NULL
      )
```

count_of_total_records

```
      SELECT
         COUNT( * )
      FROM
         AddressPtCollection
```

4.5.4.8 Result Report Example

Tested AddressLifecycleStatusDateConsistencyMeasure at 65% conformance.

4.5.5 AddressNumberFishbonesMeasure

4.5.5.1 Measure Name

AddressNumberFishbonesMeasure

4.5.5.2 Measure Description

This measure generates lines between addressed locations and the corresponding locations along the matching Overlapping Ranges Measure to check the spatial sequence of Address Number locations. The pattern created by these lines frequently resembles a fishbone.

This query is most often used where the Two Number Address Range or Four Number Address Range values are present and trusted. If those values are not present or are suspect the geocoded points may be produced without reference to the ranges. For example, points may be generated along the closest street centerline with a matching Complete Street Name value, directly opposite the addresses. This process, with the results diligently checked, allows the ranges themselves to be checked against an inventory of the address numbers actually located along the segment.

In addition to checking Address Number sequence anomalies, this query can be used to fill the Related Transportation Feature ID field in the AddressPtCollection.

4.5.5.3 Report

Logical Consistency

4.5.5.4 Evaluation Procedure

Fishbones will reflect the Address Reference System applied in a given area, and the points used.

Examples of addresses to check include those where:

- there is no fishbone
    - This may show an address with a Complete Street Name value that doesn't match anything in the StCenterlineCollection
- the fishbone touches one or more other fishbones, either crossing them or intersecting in some other way
    - Address Number values may have been assigned out of order. Another possibility, especially in more sparsely settled areas, is that the address assignment as determined by the location of the property access, and the fishbones are being drawn from buildings on the property. It's important to generate fishbones that reflect assignment practice.
- the fishbone crosses street centerlines
    - There may be inconsistencies in the Complete Street Name values recorded in the AddressPtCollection and the StCenterlineCollection.
- the fishbone extends further than expected. In many areas fishbone lines >= 1000 feet (304.8 meters) require investigation
    - These may indicate variations in street names that need to be resolved, especially when a fishbone crosses a jurisdiction. Alternatively, there may be segments missing or unnamed the StCenterlineCollection.

- Bunch at the end of a street segment, forming a bowtie
    - These frequently indicate address ranges that inappropriately begin with zero (0).

4.5.5.5 Spatial Data Required

AddressPtCollection and a set of geocoded points along the street centerline, called GeocodedPtZeroOffset in the query.

4.5.5.6 Code Example: Testing Records

CREATING A TABLE TO HOLD THE FISHBONES

```
CREATE TABLE Fishbones
(
   id SERIAL PRIMARY KEY,
   AddressID INTEGER NOT NULL REFERENCES AddressPtCollection,
   RelatedTransportationFeatureID TEXT REFERENCES
StCenterlineCollection,
   Geometry geometry
)
```

QUERY TO TEST RECORDS

```
INSERT INTO
   Fishbones
   ( AddressID,
     RelatedTransportationFeatureID,
     Geometry
   )
SELECT
   a.AddressID,
   b.RelatedTransportationFeatureID,
   ST_Makeline( a.Geometry, b.Geometry)
FROM
   AddressPtCollection a
      INNER JOIN GeocodedPtZeroOffset b
         ON a.AddressID = b.AddressID
```

4.5.5.7 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

Many anomalous fishbones are most easily located visually. Those should be added to your FishboneAnomalies table.

CREATE A TABLE TO HOLD THE POTENTIAL ANOMALIES

```
CREATE TABLE FishboneAnomalies
(
   id SERIAL PRIMARY KEY,
   AddressID TEXT,
   AddressNumber INTEGER,
```

```
   CompleteStreetName TEXT,
   Anomaly Text
)
```

COLLECT ADDRESSES WITHOUT FISHBONES

```
INSERT INTO FishboneAnomalies
   (
     AddressID,
     AddressNumber,
     CompleteStreetName,
     Anomaly
   )
SELECT
   a.AddressID,
   a.AddressNumber,
   a.CompleteStreetName,
   'No fishbone'::TEXT as "Anomaly"
FROM
   AddressPtCollection a
      LEFT JOIN Fishbones b
         ON a.AddressID = b.AddressID
WHERE
   b.AddressID is null
```

COLLECT ADDRESSES WITH FISHBONES THAT TOUCH OTHER FISHBONES

Run this query for each fishbone.

```
INSERT INTO FishboneAnomalies
   (
     AddressID,
     AddressNumber,
     CompleteStreetName,
     Anomaly
   )
SELECT
   a.AddressID,
   a.AddressNumber,
   a.CompleteStreetName,
   'Fishbone touches'::TEXT as "Anomaly"
FROM
   Fishbones a
      INNER JOIN fishbones b
         ON TOUCHES( a.Geometry, b.AddressID )
WHERE
   a.AddressID = [ __fill in AddressID value__ ]
```

COLLECT ADDRESSES WITH FISHBONES THAT CROSS CENTERLINES

```
INSERT INTO FishboneAnomalies
   (
     AddressID,
     AddressNumber,
     CompleteStreetName,
     Anomaly
   )
SELECT
   a.AddressID,
   a.AddressNumber,
   a.CompleteStreetName,
```

```
   'Fishbone touches'::TEXT as "Anomaly"
FROM
   Fishbones a
      INNER JOIN StCenterlineCollection b
         ON CROSSES( a.Geometry, b.StCenterlineGeometry )
```

COLLECT ADDRESSES WITH LONG FISHBONES

```
INSERT INTO FishboneAnomalies
   (
     AddressID,
     AddressNumber,
     CompleteStreetName,
     Anomaly
   )
SELECT
   AddressID,
   AddressNumber,
   CompleteStreetName,
   'Long fishbone'::TEXT as "Anomaly"
FROM
   Fishbones
WHERE
   ST_Length( Geometry ) >= 1000
```

COLLECT ADDRESSES WITH SUSPECTED BOWTIE FISHBONES

```
INSERT INTO FishboneAnomalies
   (
     AddressID,
     AddressNumber,
     CompleteStreetName,
     Anomaly
   )
SELECT
   a.AddressID,
   a.AddressNumber,
   a.CompleteStreetName,
   'Bowtie fishbone'::TEXT as "Anomaly"
FROM
   Fishbones a
      INNER JOIN
        ( SELECT
            st_startpoint( Geometry ) as Geometry
          from
             Fishbones
          group by
             st_startpoint( Geometry )
          having
             count( st_startpoint( Geometry ) ) > 1
         ) as b
         on st_startpoint( a.Geometry ) = b.Geometry
;
```

COUNT OF NON-CONFORMING RECORDS

After examining your results in the FishboneAnomalies and discarding those that were identified in error, count the number of non-conforming records.

```
SELECT
   COUNT(*)
```

```
FROM
   FishboneAnomalies
```

```
COUNT_OF_TOTAL_RECORDS
SELECT
   COUNT(*)
FROM
   AddressPtCollection
```

4.5.5.8 Result Report Example

Tested AddressNumberFishbonesMeasure at 80% conformance.

4.5.6 AddressNumberParityMeasure

4.5.6.1 Measure Name

AddressNumberParityMeasure

4.5.6.2 Measure Description

Test agreement of the odd/even status of the numeric value of an address number with the Address Number Parity attribute. The arithmetic listed in the pseudocode substitutes for a modulo operator(%) that may be unfamiliar, and is not always available. An alternate WHERE clause with a modulo is:

```
WHERE
   ( ( AddressNumber % 2 ) = 1
     AND
     AddressNumberParity = 'even'
   )
   OR
   ( ( AddressNumber % 2 ) = 0
     AND
     AddressNumberParity = 'odd'
   )
```

4.5.6.3 Report

Logical Consistency

4.5.6.4 Evaluation Procedure

Compare the odd/even status of the numeric value of an address number with the Address Number Parity attribute.

4.5.6.5 Spatial Data Required

None

4.5.6.6 Code Example: Testing Records

```
SELECT
   AddressID,
   AddressNumber
FROM
   AddressPtCollection
```

```
WHERE
   ( AddressNumber - ( ( AddressNumber / 2 ) * 2 ) = 0
     AND
     AddressNumberParity = 'odd'
   )
   OR
   ( AddressNumber - ( ( AddressNumber / 2 ) * 2 ) = 1
     AND
     AddressNumberParity = 'even'
   )
```

4.5.6.7 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   COUNT( AddressID )
FROM
   AddressPtCollection
WHERE
   ( AddressNumber - ( ( AddressNumber / 2 ) * 2 ) = 0
     AND
     AddressNumberParity = 'odd'
   )
   OR
   ( AddressNumber - ( ( AddressNumber / 2 ) * 2 ) = 1
     AND
     AddressNumberParity = 'even'
   )
```

count_of_total_records

```
SELECT
   COUNT(*)
FROM
   AddressPtCollection
```

4.5.6.8 Result Report Example

Tested AddressNumberParityMeasure at 92% conformance.

4.5.7 AddressNumberRangeCompletenessMeasure

4.5.7.1 Measure Name

AddressNumberRangeCompletenessMeasure

4.5.7.2 Measure Description

Check for a low and high value in each Two Number Address Range or Four Number Address Range pair. This test assumes that you are checking ranges for which ranges should be complete in order to conform with the Address Reference System.

Systems that use addresses, such as Computer Aided Dispatch (CAD), or any given Address Reference System may have requirements regarding null or zero numbers on ranges.

4.5.7.3 Report

Logical Consistency

4.5.7.4 Evaluation Procedure

Check for a non-zero value for both low and high each range.

4.5.7.5 Spatial Data Required

None. Although the query references the StCenterlineCollection it does not use geometry. The StCenterlineGeometry field need not be populated to use the measure.

4.5.7.6 Pseudocode Example: Testing records

QUERY

The query below is identical for features using either Two Number Address Range or Four Number Address Range.

- One range type must be used consistently throughout the query.
- When using Four Number Address Range each side must be checked independently, and remain constant throughout the query.
- Fill in the appropriate field names for range values where you see Range.Low or Range.High.
- Fill in the name of the primary key field for the ranges where you see Range.

```
SELECT
   AddressTransportationFeatureID,
   Range.Low,
   Range.High
FROM
   StCenterlineCollection
WHERE
   ( Range.Low is null
     OR
     Range.Low = 0
   )
   OR
   ( Range.High is null
     OR
     Range.High = 0
   )
```

4.5.7.7 Pseudocode Example: Checking the Conformance of a Data Set

FUNCTION
See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_nonconforming_records

```
SELECT
   count( * )
FROM
   StCenterlineCollection
WHERE
   ( Range.Low is null
     OR
     Range.Low = 0
   )
  OR
  ( Range.High is null
    OR
    Range.High = 0
  )
```

count_of_total_records

```
SELECT
   count( * )
FROM
   StCenterlineCollection
```

4.5.7.8 Result Report Example

Tested AddressNumberRangeCompletenessMeasure at 50% conformance.

4.5.8 AddressNumberRangeParityConsistencyMeasure

4.5.8.1 Measure Name

AddressNumberRangeParityConsistencyMeasure

4.5.8.2 Measure Description

Test agreement of the odd/even status of the numeric value of low and high address numbers. The arithmetic listed in the pseudocode substitutes for a modula operator(%) that may be unfamiliar, and is not always available. Two versions of the query are listed, one using a modula and one without.

4.5.8.3 Report

Logical Consistency

4.5.8.4 Evaluation Procedure

Compare the odd/even status of the numeric value of each address number in a Two Number Address Range or one side of a Four Number Address Range.

4.5.8.5 Spatial Data Required

None.

4.5.8.6 Pseudocode Example: Testing records

Queries for this measure are identical for features using either Two Number Address Range or Four Number Address Range.

- One range type must be used consistently throughout the query.
- When using Four Number Address Range each side must be checked independently, and remain constant throughout the query.
- Fill in the appropriate field names for range values where you see Range.Low or Range.High.

QUERY USING MODULA

```
SELECT
    AddressTransportationFeatureID,
    Range.Low,
    Range.High
FROM
    StCenterlineCollection
WHERE
   ( Range.Low % 2 ) != ( Range.High % 2 )
```

QUERY WITHOUT MODULA

```
SELECT
    AddressTransportationFeatureID,
    Range.Low,
    Range.High
FROM
   StCenterlineCollection
WHERE
   ( Range.Low - ( truncate( Range.Low  / 2 ) * 2 ) )
   !=
  ( Range.High - ( truncate( Range.High/ 2 ) * 2 ) )
```

EXAMPLE RESULTS

- If the query does not return any records, there are no anomalies. Conformance is 100%
- Example results with anomalies

Range.ID	Range.Low	Range.High
27	2	99
1142	501	598

4.5.8.7 Pseudocode Example: Checking the Conformance of a Data Set

FUNCTION
See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_nonconforming_records with modula

```
SELECT
   count(*)
FROM
   StCenterlineCollection
WHERE
  ( Range.Low % 2 ) != ( Range.High % 2 )
```

count_of_nonconforming_records without modula

```
 SELECT
   count(*)
FROM
   StCenterlineCollection
WHERE
   ( Range.Low - ( truncate( Range.Low  / 2 ) * 2 ) )
   !=
  ( Range.High - ( truncate( Range.High/ 2 ) * 2 ) )
```

count_of_total_records

```
 SELECT
    count( * )
 FROM
    StCenterlineCollection
```

4.5.8.8 Result Report Example

Tested AddressNumberRangeParityConsistencyMeasure at 90% consistency.

4.5.9 Address Range Directionality Measure

4.5.9.1 Measure Name

AddressRangeDirectionalityMeasure

4.5.9.2 Measure Description

This measure derives Address Range Directionality values, allowing update to and/or checks of values stored in the database. It requires that the thoroughfare to which the address is related be specifically identified. In the AddressPtCollection view this relationship is identified by the Related Transportation Feature ID. The Address Side Of Street information is also required.

The geometry chosen to represent the addresses determines the effectiveness of this test. In urban areas the distance from a building or other addressed feature to a street is short and spatial relationships simple. Rural areas with long driveways may find relative

positions of addresses misrepresented, and therefore the directionality of the related street centerlines confused. In the latter case it may be helpful to use points describing access from the street to represent the addresses and determine Address Range Directionality.

4.5.9.3 Report

Logical Consistency

4.5.9.4 Evaluation Procedure

Determine the AddressRangeDirectionality value of each segment. Where there is a value recorded in the database, check it against the AddressRangeDirectionality as calculated.

4.5.9.5 Spatial Data Required

StCenterlineCollection, AddressPtCollection, and fishbones (see Address Number Fishbones Measure).

4.5.9.6 Code Example: Assembling Data from Views

CREATE A TABLE FOR CALCULATED ADDRESSRANGEDIRECTIONALITY VALUES

```
CREATE TABLE "AddressRangeDirectionalityTable"
(
   id serial primary key,
   "AddressTransportationFeatureID" integer,
   "AddressRangeDirectionality" text
)
;
```

CALCULATE ADDRESSRANGEDIRECTIONALITY VALUES

This query calculates the Address Range Directionality value of a single centerline segment. Insert the same Related Transportation Feature ID value in the three places indicated by [RelatedTransportationFeatureID value].

```
--
-- Insert the results of the query into the table
--
INSERT INTO "AddressRangeDirectionality"
   (
     "AddressTransportationFeatureID",
     "AddressRangeDirectionality"
   )
--
-- Assemble the AddressRangeDirectionality phrase
--
SELECT
   g."AddressTransportationFeatureID",
   CASE
      WHEN
         g.directionality_left = g.directionality_right
      THEN
         g.directionality_left
      WHEN
         g.directionality_left is not null
         AND
```

```
         g.directionality_right is null
      THEN
         g.directionality_left
      WHEN
         g.directionality_left is null
         AND
         g.directionality_right is not null
      THEN
         g.directionality_right
      WHEN
         g.directionality_left is not null
         AND
         g.directionality_right is not null
         AND
         g.directionality_left != g.directionality_right
      THEN
         g.directionality_left || '-' || g.directionality_right
   END as "AddressRangeDirectionality"
FROM
   (
     --
     -- Calculate the orientation of each side of the line
     --
     SELECT
        f."AddressTransportationFeatureID",
        CASE
           WHEN
              e."LeftMinAddressNumber" IS NOT NULL
              AND
              e."LeftMaxAddressNumber" IS NOT NULL
              AND
              e."LeftMinAddressNumber" != e."LeftMaxAddressNumber"
              AND
              ( st_distance( st_startpoint( f."StCenterlineGeometry" ),
e."LeftMinAddressPtGeometry" )
                <
                st_distance( st_endpoint( f."StCenterlineGeometry" ),
e."LeftMaxAddressPtGeometry" )
              )
           THEN
              'with'
           WHEN
              e."LeftMinAddressNumber" IS NOT NULL
              AND
              e."LeftMaxAddressNumber" IS NOT NULL
              AND
              e."LeftMinAddressNumber" != e."LeftMaxAddressNumber"
              AND
              ( st_distance( st_startpoint( f."StCenterlineGeometry" ),
e."LeftMinAddressPtGeometry" )
                >
                st_distance( st_endpoint( f."StCenterlineGeometry" ),
e."LeftMaxAddressPtGeometry" )
              )
           THEN
              'against'
        END as "directionality_left",
```

```
        CASE
           WHEN
              e."RightMinAddressNumber" IS NOT NULL
              AND
              e."RightMaxAddressNumber" IS NOT NULL
              AND
              e."RightMinAddressNumber" != e."RightMaxAddressNumber"
              AND
              ( st_distance( st_startpoint( f."StCenterlineGeometry" ),
e."RightMinAddressPtGeometry" )
                <
                st_distance( st_endpoint( f."StCenterlineGeometry" ),
e."RightMaxAddressPtGeometry" )
              )
           THEN
              'with'
           WHEN
              e."RightMinAddressNumber" IS NOT NULL
              AND
              e."RightMaxAddressNumber" IS NOT NULL
              AND
              e."RightMinAddressNumber" != e."RightMaxAddressNumber"
              AND
              ( st_distance( st_startpoint( f."StCenterlineGeometry" ),
e."RightMinAddressPtGeometry" )
                >
                st_distance( st_endpoint( f."StCenterlineGeometry" ),
e."RightMaxAddressPtGeometry" )
              )
           THEN
              'against'
         END as "directionality_right"
      FROM
        "StCenterlineCollection" f
        INNER JOIN
           (
             --
             -- Match the selected addresses to point geometry
             --
             SELECT
                bim."RelatedTransportationFeatureID",
                a."AddressID" as "LeftMinAddressID",
                bim."LeftMinAddressNumber",
                a."AddressPtGeometry" as "LeftMinAddressPtGeometry",
                b."AddressID" as "LeftMaxAddressID",
                bim."LeftMaxAddressNumber",
                b."AddressPtGeometry" as "LeftMaxAddressPtGeometry",
                c."AddressID" as "RightMinAddressID",
                bim."RightMinAddressNumber",
                c."AddressPtGeometry" as "RightMinAddressPtGeometry",
                d."AddressID" as "RightMaxAddressID",
                bim."RightMaxAddressNumber",
                d."AddressPtGeometry" as "RightMaxAddressPtGeometry"
             FROM
                (
                  --
```

```
                  -- Select the highest and lowest numbers on the left
and right side of the street
                  --
                  SELECT
                     [ RelatedTransportationFeatureID value ] as
"RelatedTransportationFeatureID",
                     min( foo."AddressNumber" ) as
"LeftMinAddressNumber",
                     max( foo."AddressNumber" ) as
"LeftMaxAddressNumber",
                     min( bar."AddressNumber" ) as
"RightMinAddressNumber",
                     max( bar."AddressNumber" ) as
"RightMaxAddressNumber"
                  FROM
                     ( SELECT
                          "AddressNumber"
                       FROM
                          address."AddressPtCollection"
                       WHERE
                          "RelatedTransportationFeatureID" = [
RelatedTransportationFeatureID value ]
                          AND
                          "AddressSideOfStreet" = 'left'
                      ) AS foo,
                      ( SELECT
                           "AddressNumber"
                       FROM
                          address."AddressPtCollection"
                       WHERE
                          "RelatedTransportationFeatureID" = [
RelatedTransportationFeatureID value ]
                          AND
                          "AddressSideOfStreet" = 'right'
                      ) AS bar
               ) AS bim
                  INNER JOIN address."AddressPtCollection" a
                     ON ( bim."RelatedTransportationFeatureID" =
a."RelatedTransportationFeatureID"
                          AND
                          bim."LeftMinAddressNumber" =
a."AddressNumber"
                        )
                  INNER JOIN address."AddressPtCollection" b
                     ON ( bim."RelatedTransportationFeatureID" =
b."RelatedTransportationFeatureID"
                          AND
                          bim."LeftMaxAddressNumber" =
b."AddressNumber"
                        )
                  INNER JOIN address."AddressPtCollection" c
                     ON ( bim."RelatedTransportationFeatureID" =
c."RelatedTransportationFeatureID"
                          AND
                          bim."RightMinAddressNumber" =
c."AddressNumber"
                        )
```

```
                  INNER JOIN address."AddressPtCollection" d
                     ON ( bim."RelatedTransportationFeatureID" =
d."RelatedTransportationFeatureID"
                              AND
                              bim."RightMaxAddressNumber" =
d."AddressNumber"
                            )
           ) AS e
           ON f."AddressTransportationFeatureID" =
e."RelatedTransportationFeatureID"
   ) AS g
;
```

4.5.9.7 Code Example: Testing records

This query tests previously stored Address Range Directionality values against information derived from the geometry. Values that have changed, or those that are not simply **with** may be anomalies.

```
SELECT
   a."RelatedTransportationFeatureID",
   a."AddressRangeDirectionality",
   b."AddressRangeDirectionality"
FROM
   "AddressRangeDirectionalityTable" a
      LEFT JOIN "PreviousAddressRangeDirectionalityTable" b
         ON a."RelatedTransportationFeatureID" =
b."RelatedTransportationFeatureID"
WHERE
   b."AddressRangeDirectionality" is null
   OR
   a."AddressRangeDirectionality" != b."AddressRangeDirectionality"
   OR
   a."AddressRangeDirectionality" != 'with'
```

4.5.9.8 Pseudocode Example: Checking the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_nonconforming_records

```
     SELECT
         count(*)
     FROM
        "AddressRangeDirectionalityTable" a
           LEFT JOIN "PreviousAddressRangeDirectionalityTable" b
              ON a."RelatedTransportationFeatureID" =
   b."RelatedTransportationFeatureID"
     WHERE
         b."AddressRangeDirectionality" IS NULL
         OR
         a."AddressRangeDirectionality" !=
   b."AddressRangeDirectionality"
         OR
         a."AddressRangeDirectionality" != 'with'
```

count of total records

```
SELECT
    count( a.*)
FROM
    "StCenterlineCollection" a
       INNER JOIN ( SELECT DISTINCT
                                    "RelatedTransportationFeatureID"
                                FROM
                                  "AddressPtCollection"
                               ) AS b
                ON a."AddressTransportationFeatureID" =
b."RelatedTransportationFeatureID"
```

4.5.9.9 Result Report Example

Tested Address Range Directionality Measure at 94% conformance.

4.5.10 Address Reference System Axes Point Of Beginning Measure

4.5.10.1 Measure Name

AddressReferenceSystemAxesPointOfBeginningMeasure

4.5.10.2 Measure Description

This measure checks for a common point to describe the intersection of the Address Reference System Axis elements, and the location of the Address Reference System Axis Point Of Beginning against that common point. The measure query assumes that the data are stored topologically, so each axis is split where it intersects the others. It makes, however, no assumptions about the directionality of the axis lines themselves. The query returns TRUE results if the axes meet. Once a TRUE result is achieved the query may be altered to return the Address Reference System Axis Point Of Beginning if it does not already exist.

The query assumes 4 axes, identifiable as north, south, east and west. It can be altered for other kinds of Address Reference System Axis geometries.

4.5.10.3 Report

Logical Consistency

4.5.10.4 Evaluation Procedure

Make sure the axes meet at the Address Reference System Axis Point Of Beginning.

4.5.10.5 Spatial Data Required

Address Reference System Axis Point Of Beginning, Address Reference System Axis

4.5.10.5 Code Example: Testing Records

```
SELECT
   CASE
      WHEN EQUALS( foo.x_origin, foo.y_origin ) = TRUE
                  AND
                  EQUALS( foo.x_origin,
AddressReferenceSystemAxisPointOfBeginning ) = TRUE
       THEN 'Conforms'
       ELSE 'Does not conform'
   END as "Evaluation",
   EQUALS( foo.x_origin, foo.y_origin ) as "axis_end_points",
   EQUALS( foo.x_origin, AddressReferenceSystemAxisPointOfBeginning )
as "pob_check"
FROM
   (
     SELECT
        CASE
           WHEN EQUALS( east.start, west.start ) THEN east.start
           WHEN EQUALS( east.start, west.end ) THEN east.start
           WHEN EQUALS( east.end, west.start ) THEN east.end
           WHEN EQUALS( east.end, west.end ) THEN east.end
        END AS "x_origin",
        CASE
           WHEN EQUALS( north.start, south.start ) THEN north.start
           WHEN EQUALS( north.start, south.end ) THEN north.start
           WHEN EQUALS( north.end, south.start ) THEN north.end
           WHEN EQUALS( north.end, south.end ) THEN north.end
        END AS "y_origin"
     FROM
        (
          SELECT
             ST_Startpoint( Geometry ) as "start"
             ST_Endpoint( Geometry ) as "end"
          FROM
             AddressReferenceSystemAxis
          WHERE
             Axis = 'North'
        ) as north,
        (
          SELECT
             ST_Startpoint( Geometry ) as "start"
             ST_Endpoint( Geometry ) as "end"
          FROM
             AddressReferenceSystemAxis
          WHERE
             Axis = 'South'
        ) as south,
        (
          SELECT
             ST_Startpoint( Geometry ) as "start"
             ST_Endpoint( Geometry ) as "end"
          FROM
             AddressReferenceSystemAxis
          WHERE
             Axis = 'East'
        ) as east,
```

```
      (
        SELECT
           ST_Startpoint( Geometry ) as "start"
           ST_Endpoint( Geometry ) as "end"
        FROM
           AddressReferenceSystemAxis
        WHERE
           Axis = 'West'
      ) as west
) as foo
```

4.5.10.6 Testing the Conformance of a Data Set

This measure produces a result that conforms 100% or 0%, as noted in the query results.

4.5.10.7 Result Report Example

Tested AddressReferenceSystemAxesPointOfBeginningMeasure at 100% conformance.

4.5.11 Address Reference System Rules Measure

4.5.11.1 Measure Name

AddressReferenceSystemRulesMeasure

4.5.11.2 Measure Description

Address Reference System layers are essential for both address assignment and quality control, particularly in axial systems. The exact use is dependent on Address Reference System Rules and will be different for each locality. The example query given here describes checking one frequently used rule that the beginning Address Number values for each street are determined by the grid cell in which the start point of the street is located. Local Address Reference System Rules will shape the final query or queries used.

4.5.11.3 Report

Logical Consistency

Given the variability involved in testing it will be important to report the queries actually used along with the results.

4.5.11.4 Evaluation Procedure

For the example, examine each Two Number Address Range or Four Number Address Range for streets where the lowest range numbers are not within the ranges described for the corresponding grid cell.

4.5.11.5 Spatial Data Required

For the example, StCenterlineCollection with Two Number Address Range or Four Number Address Range values, and an Address Reference System layer are required. In the example below the Address Reference System layer has low values for east-west and north-south trending roads beginning within the area covered by each grid cell. For the

purposes of this example, the east-west or north-south direction of each street segment is recorded in the database.

4.5.11.6 Pseudocode Example: Testing Records

```
SELECT DISTINCT ON ( Range.Low )
   a.CompleteStreetName,
   a.TransportationFeatureID,
   b.GridCellID
   a.Range.Low,
   b.EastWestLowRangeNumber,
   b.NorthSouthLowRangeNumber,
   CASE
      WHEN ( a.Range.Low = b.EastWestLowRangeNumber
             AND
             a.GeometryDirection = 'east-west'
           )
           OR
           ( a.Range.Low = b.NorthSouthLowRangeNumber
             AND
             a.GeometryDirection = 'north-south'
           )
      THEN 'ok'
      ELSE 'anomaly'
   END AS "RangeAnomaly"
FROM
   StCenterlineCollection a
      INNER JOIN AddressReferenceSystem b
         ON INTERSECTS( St_Startpoint( a.Geometry ), b.Geometry )
ORDER BY
   a.Range.Low
```

4.5.11.7 Pseudocode Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
     SELECT
        COUNT(a.*)
     FROM
        StCenterlineCollection a
           INNER JOIN AddressReferenceSystem b
              ON INTERSECTS( St_Startpoint( a.Geometry ), b.Geometry
)
     WHERE
        ( a.Range.Low != b.EastWestLowRangeNumber
          AND
          a.GeometryDirection = 'east-west'
        )
        OR
        ( a.Range.Low != b.NorthSouthLowRangeNumber
          AND
          a.GeometryDirection = 'north-south'
```

```
        )
```

count_of_total_records

```
SELECT
   COUNT(*)
FROM
   StCenterlineCollection
```

4.5.11.8 Result Report Example

Tested AddressReferenceSystemRulesMeasure at 65% conformance.

Rule tested: [insert rule description here]

Query used: [list query here]

4.5.12 Check Attached Pairs Measure

4.5.12.1 Measure Name

CheckAttachedPairsMeasure

4.5.12.2 Measure Description

This measure describes how to check Attached Element attributes set to "attached" for matching values describing adjacent Complete Street Name or Complete Address Number components.

4.5.12.3 Report

Logical Consistency

4.5.12.4 Evaluation Procedure

Run the query for each Attached Element attribute. Attached Element attributes will be present or absent according to the needs of each locality. If the query is successful it will return an empty result set. Anomalies returned should be researched and corrected.

4.5.12.5 Spatial Data Required

None

4.5.12.6 Code Example: Testing Records

Attached Element may occur almost anywhere in the Complete Street Name or Complete Address Number elements. The query may be changed to check whichever pair of elements are affected.

```
SELECT
   AddressID
FROM
   AddressPtCollection
WHERE
   (
```

```
      ( AddressNumberAttached IS NULL
        OR
        AddressNumberAttached = 'Not Attached'
      )
      AND
      CompleteAddressNumber ~ AddressNumber || AddressNumberSuffix
   )
   OR
   (
      AddressNumberAttached = 'Attached'
      AND
      CompleteAddressNumber ~ AddressNumber || ' ' ||
AddressNumberSuffix
   )
;
```

4.5.12.7 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
      SELECT
         COUNT(*)
      FROM
         AddressPtCollection
      WHERE
         (
            ( AddressNumberAttached IS NULL
              OR
              AddressNumberAttached = 'Not Attached'
            )
           AND
           CompleteAddressNumber ~ AddressNumber ||
   AddressNumberSuffix
         )
         OR
         (
            AddressNumberAttached = 'Attached'
            AND
            CompleteAddressNumber ~ AddressNumber || ' ' ||
   AddressNumberSuffix
         )
      ;
```

count_of_total_records

```
      SELECT
          COUNT(*)
      FROM
          AddressPtCollection
```

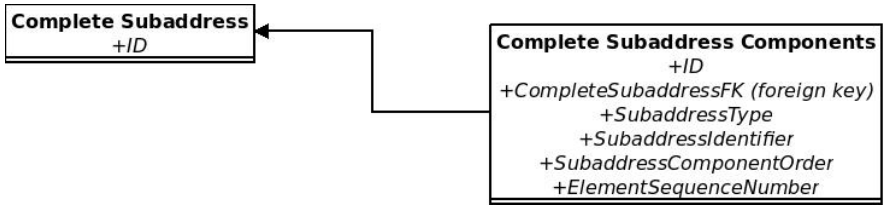
4.5.12.8 Result Report Example
Tested Check Attached Pairs Measure at 94% conformance.

4.5.13 Complex Element Sequence Number Measure

4.5.13.1 Measure Name
CompleteElementSequenceNumberMeasure

4.5.13.2 Measure Description
This measure requires assembling a complex element in order by Element Sequence Number, and testing it for completeness of the elements. The function described here is an example: the details will vary by system. It includes a function that can be used to assemble a complete complex element by sequence number. The example given is for Complete Subaddress elements, but can be applied to any series of elements ordered by an Element Sequence Number.

The example function works on a table structure for the Complete Subaddress complex element, described below.



4.5.13.3 Report
Attribute (Thematic) Accuracy

4.5.13.4 Evaluation Procedure
Check measure results for missing strings. Some localities may have a protocol for the order in which elements of a complete complex element appear. While the various types of combinations are beyond the scope of the standard, they should be considered in a local quality program.

4.5.13.5 Spatial Data Required
None.

4.5.13.6 Code Example: Testing Records

FUNCTION
```
CREATE OR REPLACE FUNCTION AssembleSubaddressString( int ) RETURNS text
AS
$BODY$
DECLARE
   id alias for $1;
   csa CompleteSubaddressComponents%rowtype;
   CompleteSubaddress text;
```

```
BEGIN
   FOR csa IN
      SELECT
         *
      FROM
         CompleteSubaddressComponents
      WHERE
         CompleteSubaddressFk = id
      ORDER BY
         element_seq
   LOOP
      IF ElementSequenceNumber = 1
         AND
         ( SubaddressComponentOrder = 1
           OR
           Subaddress Component Order = 3
         )
      THEN
         Complete Subaddress := SubaddressType || ' ' ||
SubaddressIdentifier;
      ELSIF ElementSequenceNumber = 1
            AND
            SubaddressComponentOrder = 2
      THEN
         CompleteSubaddress := SubaddressIdentifier || ' ' ||
SubaddressType;
      ELSIF ElementSequenceNumber > 1
            AND
            ( SubaddressComponentOrder = 1
              OR
              SubaddressComponentOrder = 3
            )
      THEN
         CompleteSubaddress := CompleteSubaddress || ', ' ||
SubaddressType || ' ' || SubaddressIdentifier ;
      ELSIF ElementSequenceNumber > 1
            AND
            SubaddressComponentOrder = 2
      THEN
         CompleteSubaddress := CompleteSubaddress || ', ' ||
SubaddressIdentifier || ' ' || SubaddressType ;
      END IF;
   END LOOP;
RETURN Complete Subaddress;
END
$BODY$
;
```

QUERY

```
SELECT DISTINCT
   b.CompleteSubaddressFk, AssembleSubaddressString(
b.CompleteSubaddressFk )
FROM
   CompleteSubaddress a
      LEFT JOIN Subaddress b
         ON a.PrimaryKey = b.CompleteSubaddress
WHERE
```

```
AssembleSubaddressString( b.CompleteSubaddressFk ) IS NULL
OR
b.CompleteSubaddressFk IS NULL
```

4.5.13.7 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT DISTINCT
   b.CompleteSubaddressFk,
   AssembleSubaddressString( b.CompleteSubaddressFk )
FROM
   CompleteSubaddress a
      LEFT JOIN Subaddress b
         ON a.PrimaryKey = b.CompleteSubaddressFk
WHERE
   AssembleSubaddressString( b.CompleteSubaddressFk ) IS NULL
   OR
   b.CompleteSubaddressFk IS NULL
;
```

count_of_total_records

```
SELECT
   COUNT(*)
FROM
   CompleteSubaddress
```

4.5.13.8 Result Report Example

Tested Complex Element Sequence Number Measure at 93% conformance.

4.5.14 Data Type Measure

4.5.14.1 Measure Name

DataTypeMeasure

4.5.14.2 Measure Description

This measure uses pattern matching to test for data types. It is common for delimited text files to arrive with fields that appear to be one data type or another, but may have isolated anomalies buried somewhere in the file. In this case the data are frequently loaded to a staging table with all the fields defined as TEXT. Data types need to be evaluated before loading the data into a comprehensive database. For example, this standard defines the Address Number element as integer. This technique helps to locate and resolve types that don't match. Different database systems offer functions to replace one value with another given user-defined conditions.

Data types for ASCII values can also be checked by trying to load them to a relational table. Data that do not conform to a given field definition should fail to load. This method, however, leaves anomaly resolution to other systems. Using the staging table method allows for the data to be manipulated within the system where it will be permanently deployed, while allowing the original text file to remain in its original state. History and repeatability can be maintained by saving any queries required to alter values.

Patterns are given here for integer and numeric values, as they are most often the data types that cause data loading failures. Other patterns may be added as necessary.

4.5.14.5 Report

Logical Consistency

4.5.14.6 Evaluation Procedure

Test each column in the address collection for its data type. Any elements that do not agree with the specified data type are anomalies.

4.5.14.7 Spatial Data Required

None

4.5.14.8 Code Example: Testing Records

```
SELECT
   COUNT( value_type),
   value_type
FROM
   ( SELECT
        CASE
           WHEN COALESCE( TRIM( value::TEXT ) ) ~ '^[0-9]*$'
           THEN 'integer'
           WHEN COALESCE( TRIM( value::TEXT ) ) ~ '^[0-9]*.[0-9]{1,}$'
           THEN 'numeric'
           ELSE 'other'
        END AS value_type
     FROM
        table
     WHERE
        value::TEXT ~ '[A-Za-z0-9]'
   ) AS foo
GROUP BY
   value_type
HAVING
   value_type != [fill in required type]
;
```

4.5.14.9 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   COUNT( value_type),
   value_type
FROM
   ( SELECT
        CASE
           WHEN COALESCE( TRIM( value::TEXT ) ) ~ '^[0-9]*$'
           THEN 'integer'
           WHEN COALESCE( TRIM( value::TEXT ) ) ~ '^[0-9]*.[0-
9]{1,}$'
           THEN 'numeric'
           ELSE 'other'
        END AS value_type
     FROM
        Table
     WHERE
        value::TEXT ~ '[A-Za-z0-9]'
   ) AS foo
GROUP BY
   value_type
HAVING
   value_type != [fill in required type]
;
```

count_of_total_records

```
     SELECT
        COUNT( * )
     FROM
        Table
```

4.5.14.10 Result Report Example

Tested DataTypeMeasure on Table.value with 98% conformance.

4.5.15 Delivery Address Type Subaddress Measure

4.5.15.1 Measure Name

DeliveryAddressTypeSubaddressMeasure

4.5.15.2 Measure Description

This measure checks for null Complete Subaddress values where the Delivery Address Type indicates their presence, and Complete Subaddress values where the Delivery Address Type indicates otherwise.

4.5.15.3 Report

Logical consistency

4.5.15.4 Evaluation Procedure

Check measure query results for inconsistencies.

4.5.15.5 Spatial Data Required

None

4.5.15.6 Code Example: Testing Records

Note that the query below combines the AddressPtCollection with the tables described in the Complex Element Sequence Number Measure.

```
SELECT
   a.AddressID,
   a.DeliveryAddressType,
   b.id as CompleteSubaddressForeignKey
FROM
   AddressPtCollection a
      LEFT JOIN CompleteSubaddress b
         ON a.AddressID = b.AddressID
WHERE
   ( DeliveryAddressType = 'Subaddress Included'
     AND
     b.id IS NULL
   )
   OR
   ( DeliveryAddressType = 'Subaddress Excluded'
     AND
     b.id IS NOT NULL
   )
```

5.5.15.7 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   COUNT( a.* )
FROM
   AddressPtCollection a
      LEFT JOIN CompleteSubaddress b
         ON a.AddressID = b.AddressID
WHERE
   ( DeliveryAddressType = 'Subaddress Included'
     AND
     b.id IS NULL
   )
   OR
   ( DeliveryAddressType = 'Subaddress Excluded'
     AND
     b.id IS NOT NULL
   )
```

count_of_total_records

```
SELECT
   COUNT( * )
FROM
   AddressPtCollection
```

4.5.15.8 Result Report Example
Tested DeliveryAddressTypeSubaddressMeasure at 98% conformance.

4.5.16 Duplicate Street Name Measure

4.5.16.1 Measure Name
DuplicateStreetNameMeasure

4.5.16.2 Measure Description
In many Address Reference Systems distantly disconnected street segments with the same names constitute an anomaly. This query returns Address Transportation Feature ID values for the ends of all disconnected segments. These will most often include results where the disconnected segments are close enough to mitigate the anomaly. T

The function as written is a skeleton. Local customizations typically include:

- a length test for the segments to exclude centerlines bordering traffic islands
- using identifiers for Complete Street Name values rather than text strings
- adding a test to make sure the disconnected street names are within the same jurisdiction

Regardless of customization, there will almost certainly be false positives. The final percentage of conformance should be calculated after a final set of street centerlines with duplicate street names has been finalized.

4.5.16.3 Report
Logical Consistency.

4.5.16.4 Evaluation Procedure
Examine the segments included in the results by Complete Street Name, along with the entire set of segments with the same Complete Street Name from all the street segments within the Address Reference System Extent. Take appropriate action where duplicate street names constitute a threat to public safety.

4.5.16.5 Spatial Data Required
StCenterlineCollection, Address Reference System Extent and Nodes and StreetsNodes as described in About Nodes For Quality Control.

4.5.16.6 Code Example: Testing Records

FUNCTION

```
CREATE OR REPLACE FUNCTION too_many_ends( text )
RETURNS boolean as $$
DECLARE
   this_street alias for $1;
   chk_dup boolean;
BEGIN

SELECT INTO chk_dup
   CASE
     WHEN COUNT( bim.CompleteStreetName ) > 2 THEN TRUE
     ELSE FALSE
    END AS "check_for_duplicate_names"
FROM
   (
     SELECT DISTINCT
        bar.nodesfk,
        a.CompleteStreetName,
        a.RelatedTransportationFeatureID
     FROM
        StreetsNodes a
           INNER JOIN StCenterlineCollection b
              on a.RelatedTransportationFeatureID =
b.AddressTransportationFeatureID
           INNER JOIN
           (
             SELECT
                foo.nodesfk
             FROM
                ( SELECT
                     nodesfk
                  FROM
                     StreetsNodes
                  WHERE
                     CompleteStreetName = this_street
                ) as foo
             GROUP BY
                foo.nodesfk
             HAVING
                COUNT( nodesfk ) = 1
           ) as bar
           ON a.nodesfk = bar.nodesfk
  ) as bim
WHERE
  bim.CompleteStreetName = this_street
GROUP BY
   bim.CompleteStreetName
;
RETURN chk_dup;

END

$$ language 'plpgsql';
```

QUERY

```
SELECT DISTINCT
   a.RelatedTransportationFeatureID,
   a.CompleteStreetName
FROM
   StreetsNodes a
      INNER JOIN
         ( SELECT DISTINCT
              CompleteStreetName
           FROM
              StreetsNodes
         ) b
         on a.CompleteStreetName = b.CompleteStreetName
where
   too_many_ends( a.CompleteStreetName ) = TRUE
;
```

4.5.16.7 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   COUNT( DISTINCT a.RelatedTransportationFeatureID )
FROM
   StreetsNodes a
      INNER JOIN
         ( SELECT DISTINCT
              CompleteStreetName
           FROM
              StreetsNodes
          ) b
        on a.CompleteStreetName = b.CompleteStreetName
WHERE
     too_many_ends( a.CompleteStreetName ) = TRUE
```

count_of_total_records

```
SELECT
   COUNT( AddressTransportationFeatureID )
FROM
   StCenterlineCollection
```

4.5.16.8 Result Report Example

Tested Duplicate Street Name Measure at 97% conformance.

Local changes to the measure include: [descriptions of customizations].

4.5.17 Element Sequence Number Measure

4.5.17.1 Measure Name
ElementSequenceNumberMeasure

4.5.17.2 Measure Description
Element Sequence Number values must begin at 1 and increment by 1. This measure generates a sequence of integers and checks the Element Sequence Number values against them.

This example uses the same tables described for Complex Element Sequence Number Measure, but can be used for any complex element using Element Sequence Number values. The nextval construct is used as an implementation of the SQL standard NEXT VALUE FOR.

4.5.17.3 Report
Attribute (Thematic) Accuracy

4.5.17.4 Evaluation Procedure
Examine Element Sequence Number values for sequences identified by the query.

4.5.17.5 Spatial Data Required
None.

4.5.17.6 Code Example: Testing Records

FUNCTION
```
CREATE FUNCTION test_element_sequence_numbers( integer ) RETURNS
integer AS

$BODY$

DECLARE
    SubaddressID alias for $1;
    AnomalySequence integer;

BEGIN

CREATE TEMPORARY SEQUENCE TemporarySequence;

SELECT INTO AnomalySequence
   foo.CompleteSubaddressFk
FROM
   ( SELECT
        CompleteSubaddressFk,
        NEXTVAL( TemporarySequence ) as TestSequenceNumber,
        ElementSequenceNumber
     FROM
        CompleteSubaddressComponents
     WHERE
```

```
        CompleteSubaddressFk = SubaddressID
   ) as foo
WHERE
   foo.ElementSequenceNumber != TestSequenceNumber
;

DROP SEQUENCE TemporarySequence;

RETURN ( AnomalySequence );

END

$BODY$
language 'plpgsql';
```

QUERY

```
SELECT
   test_element_sequence_numbers( CompleteSubaddressFk ),
   ElementSequenceNumber
FROM
   CompleteSubaddressComponents
WHERE
   test_element_sequence_numbers( CompleteSubaddressFk ) is not null
ORDER BY
   test_element_sequence_numbers( CompleteSubaddressFk ),
   ElementSequenceNumber
;
```

4.5.17.7 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
      SELECT
         COUNT( DISTINCT CompleteSubaddressFk )
      FROM
         CompleteSubaddressComponents
      WHERE
         test_element_sequence_numbers( CompleteSubaddressFk ) is not
   null
      ORDER BY
         test_element_sequence_numbers( CompleteSubaddressFk ),
         ElementSequenceNumber
      ;
```

count_of_total_records

```
      SELECT
         COUNT(*)
      FROM
         CompleteSubaddress
```

4.5.17.8 Result Report Example
Tested Element Sequence Number Measure at 100% conformance.

4.5.18 Future Date Measure

4.5.18.1 Measure Name
FutureDateMeasure

4.5.18.2 Measure Description
This measure produces a list of dates that are in the future.

4.5.18.3 Report
Temporal Accuracy, Attribute (Thematic) Accuracy

4.5.18.4 Evaluation Procedure
Check dates.

4.5.18.5 Spatial Data Required
None

4.5.18.5 Code Example: Testing Records
```
SELECT
   AddressID,
   AddressStartDate,
   AddressEndDate
FROM
   AddressPtCollection
WHERE
   AddressStartDate > now()
   OR
   AddressEndDate > now()
```

4.5.18.6 Code Example: Testing the Conformance of a Data Set

FUNCTION
See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   COUNT(*)
FROM
   AddressPtCollection
WHERE
   AddressStartDate > now()
OR
   AddressEndDate > now()
```

count_of_total_records

```
SELECT
   COUNT( AddressID )
FROM
   AddressPtCollection
```

4.5.18.7 Result Report Example

Tested Future Date Measure at 100% conformance.

4.5.19 Intersection Validity Measure

4.5.19.1 Measure Name

IntersectionValidityMeasure

4.5.19.2 Measure Description

Check intersection addresses for streets that do not intersect in geometry.

4.5.19.3 Report

Logical Consistency

4.5.19.4 Evaluation Procedure

Check for intersection of the geometry.

4.5.19.5 Spatial Data Required

StCenterlineCollection, Nodes

4.5.19.6 Code Example: Testing Records

PREPARE DATA

Intersection addresses frequently arrive as undifferentiated strings. It is helpful to separate the Complete Street Name elements in these strings in order to check them against the geometry. The exact methods for doing this will vary across database platforms.

Create a staging table

Create a staging table with a primary key (ID) and a field for the intersection strings. It will look something like this:

```
CREATE TABLE IntersectionAddress
(
  id SERIAL PRIMARY KEY,
  IntersectionAddress text
);
```

Fill the staging table

Fill the table with your strings. Let the primary key increment automatically to create the intersection identifiers. The completed table will look something like this:

```
 id|IntersectionAddress
---+---------------------------------------------------
  1|Boardwalk and Park Place
  2|Hollywood Boulevard and Vine Street
  3|West Street & Main Street
  4|P Street && 19th Street && Mill Road
  5|Avenida Rosa y Calle 19
  6|Memorial Park, Last Chance Gulch and Memorial Drive
  7|Phoenix Village, Scovill Avenue and East 59th Street
```

Create a table for intersection address components

Create a new table for the strings to be broken into separate Complete Street Name elements. Use a foreign key from the first table to link each Complete Street Name element to its corresponding intersection.

```
CREATE TABLE IntersectionParsed
(
  id SERIAL PRIMARY KEY,
  IntersectionAddressFk INTEGER REFERENCES IntersectionAddress,
  CompleteStreetName text
);
```

Fill the new table with intersection address components

This step requires parsing the intersection addresses, and pairing each Complete Street Name value with its primary key (id) from the Intersection Address table. The resulting pairs of values are inserted into the IntersectionParsed table.

For example, this record in Intersection Address

```
 id|IntersectionAddress
---+-------------------------
  3|West Street & Main Street
```

results in these values inserted to IntersectionParsed:

```
 IntersectionAddressFk|CompleteStreetName
----------------------+------------------
           3          | West Street
           3          | Main Street
```

Methods vary from system to system. One example is:

```
INSERT INTO
   IntersectionParsed( IntersectionAddressFk, CompleteStreetName )
SELECT
   id,
   TRIM( BOTH regexp_split_to_table( IntersectionAddress,
',|and|&&|&|y'))
FROM
   IntersectionAddress
```

Check results

The results should look something like this.

```
  id| IntersectionAddressFk | CompleteStreetName
----+-----------------------+--------------------
   1|          1            | Boardwalk
   2|          1            | Park Place
   3|          2            | Hollywood Boulevard
   4|          2            | Vine Street
   5|          3            | West Street
   6|          3            | Main Street
   7|          4            | P Street
   8|          4            | 19th Street
   9|          4            | Mill Road
  10|          5            | Avenida Rosa
  11|          5            | Calle 19
  12|          6            | Memorial Park
  13|          6            | Last Chance Gulch
  14|          6            | Memorial Drive
  15|          7            | Phoenix Village
  16|          7            | Scovill Avenue
  17|          7            | East 59th Street
```

CREATE A VIEW

This view matches intersection addresses with intersecting roads using the Complete Street Name values.

```
CREATE VIEW IntersectionAddressMatch
AS
--
-- Join the node data with the intersection addresses on
-- the CompleteStreetName values and the count of CompleteStreetName
-- values found for each intersection address and each node
--
SELECT
   bam.CountNodesfk,
   bim.CountIntersectionAddressFk,
   bam.Nodesfk,
   bim.IntersectionAddressFk,
   bim.CompleteStreetName,
   a.IntersectionAddress
from
   (
     --
     -- List the NodesFk foreign key,
     -- the count of CompleteStreetName values and
     -- each CompleteStreetName meeting at the node
     --
     SELECT DISTINCT
        bar.NodesFk,
        bar.CountNodesFk,
        a.CompleteStreetName
     FROM
        (
          --
          -- Count the number of street names
          -- associated with the node
          --
          SELECT
             foo.NodesFk,
```

```
            COUNT( foo.NodesFk ) as CountNodesFk
         FROM
            (
              --
              -- Select the identifier for the intersection geometry
              -- ( nodes ) and the CompleteStreetName values for
              -- thoroughfares meeting at that point
              --
              SELECT DISTINCT
                 NodesFk,
                 CompleteStreetName
              from
                 StreetsNodes
            ) as foo
         GROUP BY
            foo.NodesFk
       ) as bar
       INNER JOIN StreetsNodes a
          ON bar.NodesFk = a.NodesFk
  ) as bam
  INNER JOIN
       (
         --
         -- List the IntersectionAddressFk foreign key,
         -- the count of CompleteStreetName values and
         -- each CompleteStreetName for the intersection
         -- addresses.
         --
         SELECT DISTINCT
            bar.IntersectionAddressFk,
            bar.CountIntersectionAddressFk,
            a.CompleteStreetName
         FROM
            (
              --
              -- Count the number of street names in the
              -- intersection address
              --
              SELECT
                 foo.IntersectionAddressFk,
                 COUNT( foo.IntersectionAddressFk ) as
CountIntersectionAddressFk
              FROM
                 (
                   --
                   -- Select the street names and intersection address
                   -- identifiers for addresses to match
                   --
                   SELECT DISTINCT
                      IntersectionAddressFk,
                      CompleteStreetName
                   FROM
                      IntersectionParsed
                 ) as foo
              GROUP BY
                 foo.IntersectionAddressFk
            ) as bar
```

```
                INNER JOIN IntersectionParsed a
                   ON bar.IntersectionAddressFk =
a.IntersectionAddressFk
        ) as bim
        ON bam.CountNodesFk = bim.CountIntersectionAddressFk
           AND
           bam.CompleteStreetName = bim.CompleteStreetName
   INNER JOIN IntersectionAddress a
      ON bim.IntersectionAddressFk = a.id
ORDER BY
   bim.IntersectionAddressFk,
   bim.CompleteStreetName
;
```

Query for anomalies

```
SELECT
   a.IntersectionAddressFk,
   a.CompleteStreetName,
   c.IntersectionAddress
FROM
   IntersectionParsed a
      LEFT JOIN IntersectionAddressMatch b
         ON a.IntersectionAddressFk = b.IntersectionAddressFk
            AND
            a.completestreetname = b.completestreetname
      INNER JOIN intersectionaddress c
         ON a.intersectionaddressfk = c.id
WHERE
   b.completestreetname IS NULL
;
```

4.5.19.7 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   COUNT( DISTINCT a.IntersectionAddressFk )
FROM
   IntersectionParsed a
      LEFT JOIN IntersectionAddressMatch b
         ON a.IntersectionAddressFk = b.IntersectionAddressFk
            AND
            a.completestreetname = b.completestreetname
      INNER JOIN intersectionaddress c
         ON a.intersectionaddressfk = c.id
WHERE
   b.completestreetname IS NULL
;
```

count_of_total_records

```
SELECT
```

```
      COUNT(*)
   FROM
      IntersectionAddress
```

4.5.19.8 Result Report Example
Tested Intersection Validity Measure at 75% conformance.

4.5.20 Left Right Odd Even Parity Measure

4.5.20.1 Measure Name
LeftRightOddEvenParityMeasure

4.5.20.2 Measure Description
This measure tests the association of odd and even values in each Two Number Address Range or Four Number Address Range with the left and right side of the thoroughfare.

4.5.20.3 Report
Logical Consistency

4.5.20.4 Evaluation Procedure
Check the odd/even status of the numeric value of each address number for consistency with the established local rule for associating address

4.5.20.5 Code Example: Testing Records
The query below assumes even addresses on the left and odd on the right side of the street.

QUERY: LOCAL RULE IS EVEN ON LEFT, ODD ON RIGHT

```
SELECT
   a.AddressID
from
   AddressPtCollection a
      INNER JOIN StCenterlineCollection b
         ON a.RelatedTransportationFeatureID =
b.TransportationFeatureID
WHERE
   a.AddressNumber BETWEEN b.Range.Low AND b.Range.High
   AND
   a.CompleteStreetName = b.CompleteStreetName
   AND
   ( ( a.AddressNumberParity = 'odd'
       AND
       a.AddressSideOfStreet = 'left'
     )
     OR
     ( a.AddressNumberParity = 'even'
       AND
       a.AddressSideOfStreet = 'right'
     )
```

```
   )
```

QUERY: LOCAL RULE IS ODD ON LEFT, EVEN ON RIGHT

```
SELECT
   a.AddressID
from
   AddressPtCollection a
      INNER JOIN StCenterlineCollection b
         ON a.RelatedTransportationFeatureID =
b.TransportationFeatureID
WHERE
   a.AddressNumber BETWEEN b.Range.Low AND b.Range.High
   AND
   a.CompleteStreetName = b.CompleteStreetName
   AND
   ( ( a.AddressNumberParity = 'even'
       AND
       a.AddressSideOfStreet = 'left'
     )
     OR
     ( a.AddressNumberParity = 'odd'
       AND
       a.AddressSideOfStreet = 'right'
     )
   )
```

4.5.20.6 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

local rule is even on left, odd on right

```
        SELECT
           COUNT( a.AddressID )
        from
           AddressPtCollection a
              INNER JOIN StCenterlineCollection b
                 ON a.RelatedTransportationFeatureID =
b.TransportationFeatureID
        WHERE
           a.AddressNumber BETWEEN b.Range.Low AND
b.Range.High
           AND
           a.CompleteStreetName = b.CompleteStreetName
           AND
           ( ( a.AddressNumberParity = 'odd'
               AND
               a.AddressSideOfStreet = 'left'
             )
             OR
             ( a.AddressNumberParity = 'even'
               AND
               a.AddressSideOfStreet = 'right'
```

```
                  )
               )
```

local rule is odd on left, even on right

```
            SELECT
               COUNT( a.AddressID )
            from
               AddressPtCollection a
                  INNER JOIN StCenterlineCollection b
                     ON a.RelatedTransportationFeatureID =
b.TransportationFeatureID
            WHERE
               a.AddressNumber BETWEEN b.Range.Low AND
b.Range.High
               AND
               a.CompleteStreetName = b.CompleteStreetName
               AND
               ( ( a.AddressNumberParity = 'even'
                   AND
                   a.AddressSideOfStreet = 'left'
                 )
                 OR
                 ( a.AddressNumberParity = 'odd'
                   AND
                   a.AddressSideOfStreet = 'right'
                 )
               )
```

count_of_total_records

```
      SELECT
         COUNT(*)
      FROM
         AddressPtCollection
```

4.5.20.7 Result Report Example

Tested Left Right Odd Even Parity Measure at 75% conformance.

4.5.21 Location Description Field Check Measure

4.5.21.1 Measure Name

LocationDescriptionFieldCheckMeasure

4.5.21.2 Measure Description

This measure describes checking the location description in the field.

4.5.21.3 Report

Attribute Accuracy

4.5.21.4 Evaluation Procedure
Use the Location Description to navigate to the address, checking for discrepancies between the description and ground conditions. It can note that additional information such as the date the Location Description was collected or last validated and/or the name of the people who collected or entered it. This information can reinforce the lineage of the address.

4.5.21.5 Spatial Data Required
No digital spatial data are required.

4.5.21.6 Result Report Example
Tested LocationDescriptionFieldCheckMeasure at 68% conformance.

4.5.22 Low High Address Sequence Measure

4.5.22.1 Measure Name
LowHighAddressSequenceMeasure

4.5.22.2 Measure Description
This measure confirms that the value of the low address is less than or equal to the high address in a range assigned to a street segment.

4.5.22.3 Report
Logical Consistency

4.5.22.4 Evaluation Procedure
Check the values for each range.

4.5.22.5 Spatial Data Required
None. Attributes listed in StCenterlineCollection are included in the query.

4.5.22.5 Pseudocode Example: Testing Records
```
SELECT
   AddressTransportationFeatureID
FROM
   StCenterlineCollection
WHERE
   Range.Low > Range.High
```

4.5.22.6 Pseudocode Example: Testing the Conformance of a Data Set

FUNCTION
See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   COUNT(*)
FROM
   StCenterlineCollection
WHERE
   Range.Low > Range.High
```

count_of_total_records

```
SELECT
     COUNT(*)
FROM
   StCenterlineCollection
```

4.5.22.7 Result Report Example

Tested Low High Address Sequence Measure at 50% conformance.

4.5.23 Official Status Address Authority Consistency Measure

4.5.23.1 Measure Name

OfficialStatusAddressAuthorityConsistencyMeasure

4.5.23.2 Measure Description

This measure tests logical agreement of the Official Status with the Address Authority.

4.5.23.3 Report

Logical Consistency

4.5.23.4 Evaluation Procedure

Use TabularDomainMeasure to validate Official Status entries against the domain. Check logical agreement between the status values and the business process.

4.5.23.5 Spatial Data Required

None. Attributes listed in AddressPtCollection are included in the query.

4.5.23.6 Code Example: Testing Records

```
SELECT
   AddressID
FROM
   AddressPtCollection
WHERE
   ( AddressAuthority IS NULL
     AND
     (
       OfficialStatus = 'Official'
       OR
       OfficialStatus = 'Official Alternate or Alias'
       OR
```

```
       OfficialStatus = 'Alternate Established by an Official Renaming
Action of the Address Authority'
       OR
       OfficialStatus = 'Alternates Established by an Address
Authority'
     )
   )
   OR
   ( AddressAuthority IS NOT NULL
     AND
     (
       OfficialStatus = 'Unofficial Alternate or Alias'
       OR
       OfficialStatus = 'Alternate Established by Colloquial Use'
       OR
       OfficialStatus = 'Unofficial Alternate in Frequent Use'
       OR
       OfficialStatus = 'Unofficial Alternate Names In Use by an Agency
or Entity'
       OR
       OfficialStatus = 'Posted or Vanity Address'
       OR
       OfficialStatus = 'Verified Invalid'
     )
   )
```

4.5.23.7 Code Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
      SELECT
         AddressID
      FROM
         AddressPtCollection
      WHERE
         ( AddressAuthority IS NULL
          AND
         (
           OfficialStatus = 'Official'
           OR
           OfficialStatus = 'Official Alternate or Alias'
           OR
           OfficialStatus = 'Alternate Established by an Official
   Renaming Action of the Address Authority'
           OR
           OfficialStatus = 'Alternates Established by an Address
   Authority'
         )
       )
       OR
       ( AddressAuthority IS NOT NULL
         AND
```

```
          (
            OfficialStatus = 'Unofficial Alternate or Alias'
            OR
            OfficialStatus = 'Alternate Established by Colloquial Use'
            OR
            OfficialStatus = 'Unofficial Alternate in Frequent Use'
            OR
            OfficialStatus = 'Unofficial Alternate Names In Use by an
Agency or Entity'
            OR
            OfficialStatus = 'Posted or Vanity Address'
            OR
            OfficialStatus = 'Verified Invalid'
          )
        )
```

count_of_total_records

```
SELECT
   COUNT( * )
FROM
   AddressPtCollection
```

4.5.23.8 Result Report Example

Tested Official Status Address Authority Consistency Measure at 85% conformance.

4.5.24 Overlapping Ranges Measure

4.5.24.1 Measure Name

OverlappingRangesMeasure

4.5.24.2 Measure Description

This measure checks the sequence of numbers where one non-zero Two Number Address Range or Four Number Address Range meets another. The example shown describes the direction of the segment geometry going from the low Address Number to the high Address Number. Where the direction of the geometry varies, the query will have to be altered accordingly. In cases where segment directionality may vary it is extremely helpful to describe that directionality in the database.

4.5.24.3 Report

Logical Consistency

4.5.24.4 Evaluation Procedure

Check ranges on each side of a common point.

4.5.24.5 Spatial Data Required

StreetsNodes, StCenterlineCollection

4.5.24.6 Pseudocode Example: Testing Records

The query should be run for each street name in the database. The example uses Main Street for illustration. It may be helpful to use identifiers instead of text to identify street names.

```
SELECT
   a.Nodesfk,
   b.SegmentEnd,
   b.RelatedTransportationFeatureID,
   c.Range.Low
   c.Range.High
   d.SegmentEnd,
   d.RelatedTransportationFeatureID,
   e.Range.Low
   e.Range.High
FROM
   (
     SELECT
        Nodesfk
     FROM
        StreetsNodes
     WHERE
        CompleteStreetName = 'Main Street'
     GROUP BY
        Nodesfk
     HAVING
        COUNT( Nodesfk ) > 1
   ) as a
     INNER JOIN StreetsNodes b
        ON a.Nodesfk = b.Nodesfk
     INNER JOIN StCenterlineCollection c
        on b.RelatedTransportationFeatureID = c.TransportationFeatureID
     INNER JOIN StreetsNodes d
        ON a.Nodesfk = d.Nodesfk
     INNER JOIN StCenterlineCollection e
        on d.RelatedTransportationFeatureID = e.TransportationFeatureID
WHERE
   b.SegmentEnd = 'end'
   AND
   d.SegmentEnd = 'start'
   AND
   c.CompleteStreetName = 'Main Street'
   AND
   e.CompleteStreetName = 'Main Street'
   and
   c.Range.High > e.Range.Low
ORDER BY
   a.Nodesfk
;
```

4.5.24.7 Pseudocode Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   count( a.Nodesfk )
FROM
   (
     SELECT
        Nodesfk
     FROM
        StreetsNodes
     WHERE
        CompleteStreetName = 'Main Street'
     GROUP BY
        Nodesfk
     HAVING
        COUNT( Nodesfk ) > 1
   ) as a
     INNER JOIN StreetsNodes b
        ON a.Nodesfk = b.Nodesfk
     INNER JOIN StCenterlineCollection c
        on b.RelatedTransportationFeatureID =
c.TransportationFeatureID
     INNER JOIN StreetsNodes d
        ON a.Nodesfk = d.Nodesfk
     INNER JOIN StCenterlineCollection e
        on d.RelatedTransportationFeatureID =
e.TransportationFeatureID
WHERE
   b.SegmentEnd = 'end'
   AND
   d.SegmentEnd = 'start'
   AND
   c.CompleteStreetName = 'Main Street'
   AND
   e.CompleteStreetName = 'Main Street'
   and
   c.Range.High > e.Range.Low
ORDER BY
   a.Nodesfk
;
```

count_of_total_records

```
SELECT
   COUNT( * )
FROM
   Nodes
```

4.5.24.8 Result Report Example

Tested OverlappingRangesMeasure at 90% consistency.

4.5.25 Pattern Sequence Measure

4.5.25.1 Measure Name
PatternSequenceMeasure

4.5.25.2 Measure Description
This measure tests the sequence of values in each complex element for conformance to the pattern for the complex element. The query produces a list of complex elements in the address collection that do not match a sequence of simple elements. For those complex elements ordered by an Element Sequence Number please refer to ComplexElementSequenceNumberMeasure.

Complex elements called "Complete" lend themselves to normalized database tables, so that each simple element value is recorded only once. Once the text comprising a complex element has been split up into a number of tables, this test identifies database entries that have come to differ from the original data.

Some typical uses include:

- Checking a concatenated version of parsed Complete Street Name values against the original, unparsed data
- Checking a parsed Numbered Thoroughfare Address against the original, unparsed data
- Checking any concatenated Address Data Classification against original, unclassified data

4.5.25.3 Report
Logical Consistency

4.5.25.4 Evaluation Procedure
Check each complex element value against the original data for completeness.

4.5.25.5 Spatial Data Required
None.

4.5.25.6 Pseudocode Example: Testing Records
Due to the wide applicability of this measure the exact data sets are not specified, even as views.

```
SELECT
   a.ComplexElement As disagreeWithSequence
FROM
   AddressDatabase a
      LEFT JOIN OriginalData b
         ON a.ComplexElement = b.OriginalDataString
WHERE
```

```
b.OriginalDataString IS NULL
```

4.5.25.7 Pseudocode Example: Testing the Conformance of a Data Set

FUNCTION
See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   COUNT( a.ComplexElement )
FROM
   AddressDatabase a
      LEFT JOIN OriginalData b
         ON a.ComplexElement = b.OriginalDataString
WHERE
   b.OriginalDataString IS NULL
```

count_of_total_records

```
SELECT
   COUNT(*)
FROM
   AddressDatabase
```

4.5.25.8 Result Report Example

Tested [list address elements] against [original data title] using Pattern Sequence Measure at 88% conformance.

4.5.26 Range Domain Measure

4.5.26.1 Measure Name

RangeDomainMeasure

4.5.26.2 Measure Description

This measure tests each Address Number for agreement with ranges. Address Number Fishbones Measure is frequently used to establish the Related Transportation Feature ID value for AddressPtCollection.

4.5.26.3 Report

Logical Consistency

4.5.26.4 Evaluation Procedure

Validate Address Number values against low and high range values.

4.5.26.5 Spatial Data Required

None. Attribute values from AddressPtCollection and StCenterlineCollection are included in the query.

4.5.26.6 Pseudocode Example: Testing Records

```
SELECT
   a.AddressID,
   a.RelatedTransportationFeatureID,
   a.AddressNumber,
   b.Range.Low,
   b.Range.High
FROM
   AddressPtCollection a
      INNER JOIN StCenterlineCollection b
         ON a.RelatedTransportationFeatureID =
b.TransportationFeatureID
WHERE
   NOT( a.AddressNumber BETWEEN b.Range.Low AND b.Range.High )
```

4.5.26.7 Pseudocode Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
     SELECT
        COUNT( a.AddressID )
     FROM
        AddressPtCollection a
           INNER JOIN StCenterlineCollection b
              ON a.RelatedTransportationFeatureID =
   b.TransportationFeatureID
     WHERE
        NOT( a.AddressNumber BETWEEN b.Range.Low AND b.Range.High )
```

count_of_total_records

```
     SELECT
        COUNT(*)
     FROM
        AddressPtCollection
```

4.5.26.8 Result Report Example

Tested RangeDomainMeasure at 70% conformance.

4.5.27 Related Element Uniqueness Measure

4.5.27.1 Measure Name

RelatedElementUniquenessMeasure

4.5.27.2 Measure Description

This measure checks the uniqueness of the values related to a given element, in either the same table or a related table. For example, you might check the uniqueness of Complete Address Number values along a given Complete Street Name. The example query illustrates this use of the measure, checking the uniqueness of address numbers along Main Street. Customized versions of the same query can be used to check a wide range of data.

4.5.27.3 Report

Logical Consistency

4.5.27.4 Evaluation Procedure

Review records associated with inconsistent values in the related table.

4.5.27.5 Spatial Data Required

None

4.5.27.6 Code Example: Testing Records

The example code below tests a single primary value, in this case the Complete Street Name. It should be repeated for all the unique primary values in a data set to test the uniqueness of the related values. The query below can be restated as a function or stored procedure for convenience.

```
SELECT
   a.AddressID
   a.CompleteAddressNumber,
   a.CompleteStreetName
FROM
   AddressPtCollection a
      INNER JOIN
      (
        SELECT
            CompleteAddressNumber
        FROM
           ( SELECT DISTINCT
                CompleteStreetName,
                CompleteAddressNumber
             FROM
                AddressPtCollection
             WHERE
                CompleteStreetName = 'Main Street'
           ) AS foo
        GROUP BY
           CompleteAddressNumber
        HAVING
           COUNT( CompleteAddressNumber ) > 1
      ) AS bar
     ON
       a.CompleteAddressNumber = bar.CompleteAddressNumber
WHERE
    a.CompleteStreetName = 'Main Street'
;
```

4.5.27.7 Code Example: Testing the Conformance of a Data Set

FUNCTION
See Perc Conforming for the sample query.

FUNCTION PARAMETERS
The count of conforming records, like the testing query, should be run for all the primary values tested. It may be restated as a function or stored procedure for convenience.

count_of_conforming_records

```
SELECT
   COUNT(*)
FROM
   AddressPtCollection a
      INNER JOIN
      (
        SELECT
            CompleteAddressNumber
        FROM
           ( SELECT DISTINCT
                CompleteStreetName,
                CompleteAddressNumber
             FROM
                AddressPtCollection
             WHERE
                CompleteStreetName = 'Main Street'
           ) AS foo
        GROUP BY
           CompleteAddressNumber
        HAVING
           COUNT( CompleteAddressNumber ) > 1
      ) AS bar
    ON
       a.CompleteAddressNumber = bar.CompleteAddressNumber
WHERE
    a.CompleteStreetName = 'Main Street'
;
```

count_of_total_records

```
SELECT
   COUNT(*)
FROM
   AddressPtCollection
;
```

4.5.27.8 Result Report Query

Tested [table].[column] primary values to find unique related values in [table].[column] at 88% conformance.

4.5.28 Related Element Value Measure

4.5.28.1 Measure Name
Related Element Value Measure

4.5.28.2 Measure Description
This measure checks the logical consistency of data related to another part of the address. These may be values in a single table, or values referenced through a foreign key. There are two ways to use this concept:

- **Compare authoritative data values against those in use.** In this case, by definition, those values that do not conform to the authoritative data values are anomalies.

  Example
  Comparing ZIP code values published by the United States Postal Service against those in the AddressPtCollection and StCenterlineCollection

- **Compare two related values that each conform to the same domain and have a relationship that indicates that the values will be the same.** In this case, either value could be anomalous, or both could be correct.

  Example
  Comparing Complete Street Name values between AddressPtCollection and StCenterlineCollection where the address point is associated with the centerline, as in the example query given below. Where the values conflict either may be anomalous, or both may be correct. The latter case could result when a thoroughfare has a both state highway number and a local name. It may be customary to call the street by the state highway number, while some or all addresses may have been assigned using the local name: Highway 41 vs. Main Street.

- **Check related values where one is dependent on the other.**

  Example
  Comparing Street Name Pre Type and Street Name Post Type entries against functional classifications of the named road where specific types are associated with particular functional classes.

4.5.28.3 Report
Logical Consistency

4.5.28.4 Evaluation Procedure
Check for inconsistent values, appropriate to the nature of the specific query.

4.5.28.5 Code Example: Checking Related Element Values
The example query checks Complete Street Name values in addresses against the Complete Street Name values on the streets associated with those addresses. This is

simply an illustration. The measure is intended to check related values of any kind, either in the same table or a related table.

```
SELECT
   a.AddressID,
   a.CompleteStreetName,
   b.CompleteStreetName
FROM
   AddressPtCollection a
       left join StCenterlineCollection b
          on a.RelatedTransportationFeatureID =
b.AddressTransportationFeatureID
WHERE
     a.CompleteStreetName != b.CompleteStreetName
```

4.5.28.6 Code Example: Testing the Conformance of a Data Set

FUNCTION
See Perc Conforming for the sample query

FUNCTION PARAMETERS

count_of_nonconforming_records

```
SELECT
   a.AddressID,
   a.CompleteStreetName,
   b.CompleteStreetName
FROM
   AddressPtCollection a
       left join StCenterlineCollection b
          on a.RelatedTransportationFeatureID =
b.AddressTransportationFeatureID
WHERE
     a.CompleteStreetName != b.CompleteStreetName
```

count_of_total_records

```
SELECT
      count(*)
FROM
      AddressPtCollection
;
```

4.5.28.7 Result Report Example

Tested [Table.Column] against [Table.Column] using Related Element Value Measure at 72% conformance.

4.5.29 Related Not Null Measure

4.5.29.1 Measure Name

RelatedNotNullMeasure

4.5.29.2 Measure Description

This measure checks the completeness of data related to another part of the address. These may be values in a single table, or values referenced through a foreign key. For example:

- Addresses in a given jurisdiction that require Street Name Post Directional quadrants
- Elements of Numbered Thoroughfare Address table stored in related tables. Street Name Post Type values stored in a related table, for instance.

4.5.29.3 Report

Completeness

4.5.29.4 Evaluation Procedure

Check for invalid null values.

4.5.29.5 Spatial Data Required

None

4.5.29.6 Pseudocode Example: Testing Records

```
SELECT
   a.AddressID,
   a.RelatedDataIdentifier
FROM
   AddressDatabaseTable a
      LEFT JOIN RelatedTable b
         ON a.RelatedDataIdentifier = b.Identifier
WHERE
   b.Identifier is null
```

4.5.29.7 Pseudocode Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   COUNT( AddressID )
FROM
   AddressDatabaseTable a
      LEFT JOIN RelatedTable b
         ON a.RelatedDataIdentifier = b.Identifier
WHERE
   b.Identifier is null
```

count_of_total_records

```
SELECT
   COUNT( * )
FROM
   AddressDatabaseTable
```

4.5.29.8 Result Report Example

Tested [AddressDatabaseTable.Column] against [RelatedTable.Column] using Related Not Null Measure at 90% conformance.

4.5.30 Segment Directionality Consistency Measure

4.5.30.1 Measure Name

SegmentDirectionalityConsistencyMeasure

4.5.30.2 Measure Description

Check consistency of street segment directionality, which affects the use of Two Number Address Range and Four Number Address Range values. The test checks for segments with the same street name where more than one "from" or "to" ends meet at the same node.

4.5.30.3 Report

Logical Consistency

4.5.30.4 Evaluation Procedure

Examine segments where the measure indicates inconsistent directionality and take appropriate action. Depending on the use, that may mean reversing the directionality of inconsistent segments or making note in the database.

4.5.30.5 Spatial Data Required

Nodes and Streets Nodes as described in About Nodes For Quality Control

4.5.30.6 Code Example: Testing Records

```
SELECT
   a.NodesFk,
   a.CompleteStreetName,
   a.SegmentEnd,
   a.RelatedTransportationFeatureID,
   b.RelatedTransportationFeatureID
FROM
   StreetsNodes a
      INNER JOIN StreetsNodes b
         ON a.CompleteStreetName = b.CompleteStreetName
            AND
            a.NodesFk = b.NodesFk
WHERE
   a.RelatedTransportationFeatureId < b.RelatedTransportationFeatureId
   AND
   a.SegmentEnd = b.SegmentEnd
ORDER BY
```

```
   a.Nodesfk
;
```

4.5.30.7 Code Example: Testing the Conformance of a Data Set

FUNCTION
See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   COUNT( a.NodesFk )
FROM
   StreetsNodes a
      INNER JOIN StreetsNodes b
         ON a.CompleteStreetName = b.CompleteStreetName
            AND
            a.NodesFk = b.NodesFk
WHERE
   a.RelatedTransportationFeatureId <
b.RelatedTransportationFeatureId
   AND
   a.SegmentEnd = b.SegmentEnd
```

count_of_total_records

```
SELECT
   COUNT( RelatedTransportationFeatureID )
FROM
   StCenterlineCollection
```

4.5.30.8 Result Report Example

Tested SegmentDirectionalityConsistencyMeasure at 50% conformance.

4.5.31 Spatial Domain Measure

4.5.31.1 Measure Name

SpatialDomainMeasure

4.5.31.2 Measure Description

This measure tests values of some simple elements constrained by domains based on spatial domains: ZIP Codes, PLSS descriptions, etc. This is limited to domains that are identified by the simple element alone. Address numbers, for example, cannot be tested against centerline ranges because the street name is only identified in a complex element. The query produces a list of simple elements in the address collection that do not conform to a spatial domain.

4.5.31.3 Report

Positional Accuracy

4.5.31.4 Evaluation Procedure

Check addresses outside the spatial domain.

4.5.31.5 Spatial Data Required

Address Pt Collection or St Centerline Collection, spatial domain geometry

4.5.31.6 Pseudocode Example: Testing Records

Note that the example uses AddressPtCollection. It can be altered to check on elements and attributes of StCenterlineCollection also.

```
SELECT
   a.AddressID
FROM
   AddressPtCollection a
      LEFT JOIN SpatialDomain b
         ON a.[column to test] = b.[corresponding column]
WHERE
   NOT( INTERSECTS( a.AddressPtGeometry, b.SpatialDomainGeometry ) )
```

4.5.31.7 Pseudocode Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   COUNT(*)
FROM
   AddressPtCollection a
      LEFT JOIN SpatialDomain b
         ON a.[column to test] = b.[corresponding column]
WHERE
   NOT( INTERSECTS( a.AddressPtGeometry,
b.SpatialDomainGeometry ) )
```

count_of_total_records

```
SELECT
   COUNT( * )
FROM
   AddressPtCollection
```

4.5.31.8 Result Report Example

Tested [field name] in [AddressPtCollection or StCenterlineCollection] using Spatial Domain Measure at 87% conformance.

4.5.32 Start End Date Order Measure

4.5.32.1 Measure Name
StartEndDateOrderMeasure

4.5.32.2 Measure Description
Test the logical ordering of the start and end dates.

4.5.32.3 Report
Temporal Accuracy, Attribute (Thematic) Accuracy

4.5.32.4 Evaluation Procedure
Check dates for records where the Address Start Date and Address End Date are out of order.

4.5.32.5 Spatial Data Required
None.

4.5.32.6 Code Example: Testing Records
```
SELECT
   AddressStartDate,
   AddressEndDate
FROM
   AddressPtCollection
WHERE
   AddressEndDate IS NOT NULL
   AND
   ( AddressStartDate > AddressEndDate
     OR
     AddressStartDate IS NULL
   )
```

4.5.32.7 Code Example: Testing the Conformance of a Data Set

FUNCTION
See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   AddressStartDate,
   AddressEndDate
FROM
   AddressPtCollection
WHERE
   AddressEndDate IS NOT NULL
   AND
   ( AddressStartDate > AddressEndDate
     OR
```

```
            AddressStartDate IS NULL
          )
```

count_of_total_records

```
SELECT
   COUNT(*)
FROM
   AddressPtCollection
```

4.5.32.8 Result Report Example

Tested Start End Date Order Measure at 100% conformance.

4.5.33 Subaddress Component Order Measure

4.5.33.1 Measure Name

SubaddressComponentOrderMeasure

4.5.33.2 Measure Description

This measure tests Subaddress Elements against the component parts in the order specified by the Subaddress Component Order element.

4.5.33.3 Report

Attribute (Thematic) Accuracy

4.5.33.4 Evaluation Procedure

Check complex element against concatenated simple elements for anomalies.

4.5.33.5 Spatial Data Required

None

4.5.33.6 Pseudocode Example: Testing Records

```
SELECT
    SubaddressElement,
    SubaddressType,
    SubaddressIdentifier,
    SubaddressComponentOrder
FROM
    Subaddress Collection
WHERE
    (
          ( SubaddressElement = SubaddressType || ' ' ||
SubaddressIdentifier
          or
          ( SubaddressElement = SubaddressIdentifier and SubaddressType
is null )
          )
       and
```

```
      SubaddressComponentOrder = 2
   )
   or
   ( SubaddressElement = SubaddressIdentifier || ' ' || SubaddressType
      and
      SubaddressComponentOrder = 1
   )
;
```

4.5.33.7 Pseudocode Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_nonconforming_records

```
     SELECT
         Count(*)
     FROM
         Subaddress Collection
     WHERE
         (
            ( SubaddressElement = SubaddressType || ' ' ||
SubaddressIdentifier
               or
               ( SubaddressElement = SubaddressIdentifier
                  and
                  SubaddressType is null
                )
            )
          and
          SubaddressComponentOrder = 2
         )
         or
         ( SubaddressElement = SubaddressIdentifier || ' ' ||
SubaddressType
            and
            SubaddressComponentOrder = 1
          )
     ;
```

count_of_total_records

```
      SELECT
            COUNT(*)
      FROM
            Subaddress Collection
      ;
```

4.5.33.8 Result Report Example

Tested SubaddressComponentOrderMeasure at 96% conformance.

4.5.34 Tabular Domain Measure

4.5.34.1 Measure Name
TabularDomainMeasure

4.5.34.2 Measure Description
This measure tests each value for a simple element for agreement with the corresponding tabular domain. The query produces a list of simple elements in the address collection that do not conform to a domain.

4.5.34.3 Report
Attribute (Thematic) Accuracy

4.5.34.4 Evaluation Procedure
Investigate values that do not match the domain. They may include aliases, new values for the domain and/or simple mistakes.

4.5.34.5 Spatial Data Required
None.

4.5.34.6 Pseudocode Example: Testing Records
```
SELECT
   a.SimpleElement As disagreeWithDomain
FROM
   AddressPtCollection a
      LEFT JOIN Domain b
         ON a.SimpleElement = b.DomainValue
WHERE
   b.DomainValue IS NULL
;
```

4.5.34.7 Pseudocode Example: Testing the Conformance of a Data Set

FUNCTION
See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   a.SimpleElement As disagreeWithDomain
FROM
   Address Collection a
      LEFT JOIN Domain b
         ON a.SimpleElement = b.DomainValue
WHERE
   b.DomainValue IS NULL
;
```

count_of_total_records

```
SELECT
   COUNT( a.SimpleElement )
FROM
   Address Collection
```

4.5.34.8 Result Report Example

Test [table name].[column name] using TabularDomainMeasure at 80% conformance.

4.5.35 Uniqueness Measure

4.5.35.1 Measure Name

UniquenessMeasure

4.5.35.2 Measure Description

This measure tests the uniqueness of a simple or complex value.

4.5.35.3 Report

Attribute (Thematic) Accuracy

4.5.35.4 Evaluation Procedure

Investigate cases where a two or more values exist where a single value is expected. This is often used to check the "Domain" tables before they are used in TabularDomainMeasure: tables with unique values for individual street name components, for example.

4.5.35.5 Spatial Data Required

None.

4.5.35.6 Pseudocode Example: Testing Records

```
SELECT
   COUNT(Element), Element
FROM
   Address Collection
GROUP BY
   Element
HAVING
   COUNT(Element) > 1
```

4.5.35.7 Pseudocode Example: Testing the Conformance of a Data Set

FUNCTION

See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_non_conforming_records

```
SELECT
   SUM( foo.NumberPerElement )
FROM
   (
     SELECT
        COUNT( Element ) as NumberPerElement
     FROM
        Address Collection
     GROUP BY
        Element
     HAVING
        COUNT( Element ) > 1
   ) as foo
```

count_of_total_records

```
SELECT
   COUNT( Element )
FROM
   Address Collection
```

4.5.35.8 Result Report Example

Tested [table name].[column name] using UniquenessMeasure at 100% conformance.

4.5.36 USNG Coordinate Spatial Measure

4.5.36.1 Measure Name

USNGCoordinateSpatialMeasure

4.5.36.2 Measure Description

This measure tests the agreement between the location of the addressed object and the area described by the USNational Grid Coordinate. This test derives the USNG for a point geometry and compares it to the USNG coordinate.

4.5.36.3 Report

Positional accuracy

4.5.36.4 Spatial Data Required

If the derived USNG matches the recorded USNG the comparison is successful. The coord2usng function is an example. Exact code may vary across systems. An inverse function, converting USNG to UTM coordinates, is provided for convenience in an **Addendum** section.

4.5.36.5 Code Example: Testing Records

FUNCTION

```
create or replace function coord2usng( numeric, numeric, numeric,
numeric, integer )
returns varchar as '
```

```
declare
  utm_x alias for $1;
  utm_y alias for $2;
  dd_long alias for $3;
  dd_lat alias for $4;
  precision alias for $5;
  utm_zone integer;
  gzd_alpha char(1);
  set integer;
  e100k_grp1 varchar[8];
  e100k_grp2 varchar[8];
  e100k_grp3 varchar[8];
  n100k_grp1 varchar[20];
  n100k_grp2 varchar[20];
  e_100k integer;
  n_100k integer;
  x_alpha_gsz char(1);
  y_alpha_gsz char(1);
  usng varchar;
  x_grid_coord varchar;
  y_grid_coord varchar;
  num integer;
begin

--find utm zone
select into utm_zone
  case
    when dd_long between -180 and -174 then 1
    when dd_long between -174 and -168 then 2
    when dd_long between -168 and -162 then 3
    when dd_long between -162 and -156 then 4
    when dd_long between -156 and -150 then 5
    when dd_long between -150 and -144 then 6
    when dd_long between -144 and -138 then 7
    when dd_long between -138 and -132 then 8
    when dd_long between -132 and -126 then 9
    when dd_long between -126 and -120 then 10
    when dd_long between -120 and -114 then 11
    when dd_long between -114 and -108 then 12
    when dd_long between -108 and -102 then 13
    when dd_long between -102 and -96 then 14
    when dd_long between -96 and -90 then 15
    when dd_long between -90 and -84 then 16
    when dd_long between -84 and -78 then 17
    when dd_long between -78 and -72 then 18
    when dd_long between -72 and -66 then 19
    when dd_long between -66 and -60 then 20
    when dd_long between -60 and -54 then 21
    when dd_long between -54 and -48 then 22
    when dd_long between -48 and -42 then 23
    when dd_long between -42 and -36 then 24
    when dd_long between -36 and -30 then 25
    when dd_long between -30 and -24 then 26
    when dd_long between -24 and -18 then 27
    when dd_long between -18 and -12 then 28
    when dd_long between -12 and -6 then 29
```

```
    when dd_long between -6 and 0 then 30
    when dd_long between 0 and 6 then 31
    when dd_long between 6 and 12 then 32
    when dd_long between 12 and 18 then 33
    when dd_long between 18 and 24 then 34
    when dd_long between 24 and 30 then 35
    when dd_long between 30 and 36 then 36
    when dd_long between 36 and 42 then 37
    when dd_long between 42 and 48 then 38
    when dd_long between 48 and 54 then 39
    when dd_long between 54 and 60 then 40
    when dd_long between 60 and 66 then 41
    when dd_long between 66 and 72 then 42
    when dd_long between 72 and 77 then 43
    when dd_long between 78 and 84 then 44
    when dd_long between 84 and 90 then 45
    when dd_long between 90 and 96 then 46
    when dd_long between 96 and 102 then 47
    when dd_long between 102 and 108 then 48
    when dd_long between 108 and 114 then 49
    when dd_long between 114 and 120 then 50
    when dd_long between 120 and 126 then 51
    when dd_long between 126 and 132 then 52
    when dd_long between 132 and 138 then 53
    when dd_long between 138 and 144 then 54
    when dd_long between 144 and 150 then 55
    when dd_long between 150 and 156 then 56
    when dd_long between 156 and 162 then 57
    when dd_long between 162 and 168 then 58
    when dd_long between 168 and 174 then 59
    when dd_long between 174 and 180 then 60
  end;

-- find grid zone character

select into gzd_alpha
  case
    when dd_lat between -80 and -72 then ''C''
    when dd_lat between -72 and -64 then ''D''
    when dd_lat between -64 and -56 then ''E''
    when dd_lat between -56 and -48 then ''F''
    when dd_lat between -48 and -40 then ''G''
    when dd_lat between -40 and -32 then ''H''
    when dd_lat between -32 and -24 then ''J''
    when dd_lat between -24 and -16 then ''K''
    when dd_lat between -16 and -8 then ''L''
    when dd_lat between -8 and 0 then ''M''
    when dd_lat between 0 and 8 then ''N''
    when dd_lat between 8 and 16 then ''P''
    when dd_lat between 16 and 24 then ''Q''
    when dd_lat between 24 and 32 then ''R''
    when dd_lat between 32 and 40 then ''S''
    when dd_lat between 40 and 48 then ''T''
    when dd_lat between 48 and 56 then ''U''
    when dd_lat between 56 and 64 then ''V''
    when dd_lat between 64 and 72 then ''W''
    when dd_lat between 72 and 84 then ''X''
```

```
  end;

  -- derive set
    if ( utm_zone <= 6 ) then
        set := utm_zone;
    else
        if ( utm_zone % 6 = 0 ) then
           set := 6;
        else
           set := utm_zone % 6;
        end if;
     end if;

  -- construct arrays describing grid zone squares
  select into e100k_grp1
array[''A'',''B'',''C'',''D'',''E'',''F'',''G'',''H''];
  select into e100k_grp2
array[''J'',''K'',''L'',''M'',''N'',''P'',''Q'',''R''];
  select into e100k_grp3
array[''S'',''T'',''U'',''V'',''W'',''X'',''Y'',''Z''];
  select into n100k_grp1
array[''A'',''B'',''C'',''D'',''E'',''F'',''G'',''H'',''J'',''K'',''L''
,''M'',''N'',''P'',''Q'',''R'',''S'',''T'',''U'',''V''];
  select into n100k_grp2
array[''F'',''G'',''H'',''J'',''K'',''L'',''M'',''N'',''P'',''Q'',''R''
,''S'',''T'',''U'',''V'',''A'',''B'',''C'',''D'',''E''];

 -- get the digit for the 100K places ( easting and northing )

    select into e_100k
    substring( utm_x::text from ( length( trunc( utm_x )::text ) - 5 )
for 1 );

    n_100k = ( floor( utm_y / 100000 ) % 20 ) + 1;

 -- get the grid

   select into x_alpha_gsz
     case
       when ( set = 1 or set = 4 ) then e100k_grp1[e_100k]
       when ( set = 2 or set = 5 ) then e100k_grp2[e_100k]
       when ( set = 3 or set = 6 ) then e100k_grp3[e_100k]
     end;

   select into y_alpha_gsz
     case
       when ( set = 1 or set = 3 or set = 5 ) then n100k_grp1[n_100k]
       when ( set = 2 or set = 4 or set = 6 ) then n100k_grp2[n_100k]
     end;

-- get coordinates

   select into x_grid_coord
   case
      when ( precision = 10000 ) then
         substring( utm_x::text from ( length( trunc( utm_x )::text ) -
4 ) for 1 )
```

```
      when ( precision = 1000 ) then
         substring( utm_x::text from ( length( trunc( utm_x )::text ) -
4 ) for 2 )
      when ( precision = 100 ) then
         substring( utm_x::text from ( length( trunc( utm_x )::text ) -
4 ) for 3 )
      when ( precision = 10 ) then
         substring( utm_x::text from ( length( trunc( utm_x )::text ) -
4 ) for 4 )
      when ( precision = 1 ) then
         substring( utm_x::text from ( length( trunc( utm_x )::text ) -
4 ) for 5 )
   end;

   select into y_grid_coord
   case
      when ( precision = 10000 ) then
         substring( utm_y::text from ( length( trunc( utm_y )::text ) -
4 ) for 1 )
      when ( precision = 1000 ) then
         substring( utm_y::text from ( length( trunc( utm_y )::text ) -
4 ) for 2 )
      when ( precision = 100 ) then
         substring( utm_y::text from ( length( trunc( utm_y )::text ) -
4 ) for 3 )
      when ( precision = 10 ) then
         substring( utm_y::text from ( length( trunc( utm_y )::text ) -
4 ) for 4 )
      when ( precision = 1 ) then
         substring( utm_y::text from ( length( trunc( utm_y )::text ) -
4 ) for 5 )
   end;

   -- assemble the USNG value

   usng := utm_zone || gzd_alpha || x_alpha_gsz || y_alpha_gsz ||
x_grid_coord || y_grid_coord;

   return( usng );

end;
' language 'plpgsql';
```

QUERY

```
SELECT
    USNationalGridCoordinate
FROM
    AddressPtCollection
WHERE
   coord2usng
   ( st_x( st_transform( c.geom, 26916) )::numeric,
     st_y( st_transform( c.geom, 26916) )::numeric,
     st_x( st_transform( c.geom, 4269 ) )::numeric,
     st_y( st_transform( c.geom, 4269 ) )::numeric,
     1)
      != USNationalGridCoordinate
;
```

4.5.36.6 Code Example: Testing the Conformance of a Data Set

FUNCTION
See Perc Conforming for the sample query.

FUNCTION PARAMETERS

count_of_nonconforming_records

```
SELECT
    COUNT(*)
FROM
    AddressPtCollection
WHERE
   coord2usng
   ( st_x( st_transform( c.geom, 26916) )::numeric,
     st_y( st_transform( c.geom, 26916) )::numeric,
     st_x( st_transform( c.geom, 4269 ) )::numeric,
     st_y( st_transform( c.geom, 4269 ) )::numeric,
     1)
      != USNationalGridCoordinate
;
```

count_of_total_records

```
SELECT
   COUNT(*)
FROM
    AddressPtCollection
```

4.5.36.7 Result Report Example

Tested USNGCoordinateMeasure at 96% conformance.

4.5.36.8 Addendum

NOTE
This function returns a pair of coordinates at the center of the area described by the precision of the USNG grid reference.

FUNCTION

```
create or replace function usng2coord( text )
returns text as $$

declare
  usng alias for $1;
  zone integer;
  grid_zone text;
  set integer;
  offset_north numeric;
  x_alpha_gsz text;
  y_alpha_gsz text;
  e_coord integer;
  n_coord integer;
  e100k integer;
  n100k integer;
```

```
  e100k_grp1 text[];
  e100k_grp2 text[];
  e100k_grp3 text[];
  n100k_grp1 text[];
  n100k_grp2 text[];
  e_gsz integer;
  n_gsz integer;
  grid numeric;
  precision numeric;
  e_grid integer;
  n_grid integer;
  usng_coords varchar;
  xmin numeric;
  ymin numeric;

begin


  -- parse UTM zone
select into zone cast( ( substring( usng from '^[[:digit:]]*') ) as
integer );

  -- derive set
  if ( zone <= 6 ) then
    set := zone;
  else
    if ( zone % 6 = 0 ) then
      set := 6;
    else
      set := zone % 6;
    end if;
  end if;

  --- parse grid zone

  select into grid_zone substring( usng from ( length(zone::text ) + 1
) for 1 );

  -- parse grid zone squares
  select into x_alpha_gsz substring( usng from ( length( zone::text ) +
2 ) for 1 );
  select into y_alpha_gsz substring( usng from ( length( zone::text ) +
3 ) for 1 );

  -- calculate offset_north

  select into offset_north
    case
      when ( grid_zone = 'N' or grid_zone = 'P' )
         then 0
      when ( grid_zone = 'Q' and ( set % 2 ) = 1 and y_alpha_gsz <= 'K'
)
         then 2000000
      when ( grid_zone = 'Q' and ( set % 2 ) = 1 and y_alpha_gsz >= 'L'
)
         then 0
```

```
      when ( grid_zone = 'Q' and ( set % 2 ) = 0 and y_alpha_gsz >= 'F'
and y_alpha_gsz <= 'Q')
           then 2000000
      when ( grid_zone = 'Q' and ( set % 2 ) = 0 and ( y_alpha_gsz <=
'E' or y_alpha_gsz >= 'R') )
         then 0
      when ( grid_zone = 'R' )
         then 2000000
      when ( grid_zone = 'S' and ( set % 2 ) = 1 and y_alpha_gsz <= 'K'
)
         then 4000000
      when ( grid_zone = 'S' and ( set % 2 ) = 1 and y_alpha_gsz >= 'L'
)
         then 2000000
      when ( grid_zone = 'S' and ( set % 2 ) = 0 and y_alpha_gsz >= 'F'
and y_alpha_gsz <= 'Q')
         then 4000000
      when ( grid_zone = 'S' and ( set % 2 ) = 0 and ( y_alpha_gsz <=
'E' or y_alpha_gsz >= 'R') )
         then 2000000
      when ( grid_zone = 'T' )
         then 4000000
      when ( grid_zone = 'U' and ( set % 2 ) = 1 and y_alpha_gsz <= 'C'
)
         then 6000000
      when ( grid_zone = 'U' and ( set % 2 ) = 1 and y_alpha_gsz >= 'D'
)
         then 4000000
      when ( grid_zone = 'U' and ( set % 2 ) = 0 and y_alpha_gsz >= 'F'
and y_alpha_gsz <= 'H' )
         then 6000000
      when ( grid_zone = 'U' and ( set % 2 ) = 0 and ( y_alpha_gsz <=
'E' or y_alpha_gsz >= 'J' ) )

         then 4000000
      when ( grid_zone = 'V' or grid_zone = 'W' )
         then 6000000
      when ( grid_zone = 'X' and ( set % 2 ) = 1 and y_alpha_gsz = 'V'
)
         then 6000000
      when ( grid_zone = 'X' and ( set % 2 ) = 1 and y_alpha_gsz != 'V'
)
         then 8000000
      when ( grid_zone = 'X' and ( set % 2 ) = 0 and y_alpha_gsz = 'E'
)
         then 6000000
      when ( grid_zone = 'X' and ( set % 2 ) = 0 and y_alpha_gsz != 'E'
)
         then 8000000
    end;

  -- construct arrays describing grid zone squares
  select into e100k_grp1 array['A','B','C','D','E','F','G','H'];
  select into e100k_grp2 array['J','K','L','M','N','P','Q','R'];
  select into e100k_grp3 array['S','T','U','V','W','X','Y','Z'];
```

```
  select into n100k_grp1
array['A','B','C','D','E','F','G','H','J','K','L','M','N','P','Q','R','
S','T','U','V'];
  select into n100k_grp2
array['F','G','H','J','K','L','M','N','P','Q','R','S','T','U','V','A','
B','C','D','E'];

 -- derive X coordinate for grid zone square

   for e_gsz in 1 .. 8 loop
      if ( set = 1 or set = 4 ) then
          if ( x_alpha_gsz = e100k_grp1[e_gsz] ) then
            e_coord := 100000 * e_gsz;
            exit;
          end if;
       elsif ( set = 2 or set = 5 ) then
          if ( x_alpha_gsz = e100k_grp2[e_gsz] ) then
            e_coord := 100000 * e_gsz;
            exit;
          end if;
       else
          if ( x_alpha_gsz = e100k_grp3[e_gsz] ) then
            e_coord := 100000 * e_gsz;
            exit;
          end if;
      end if;
   end loop;

-- derive Y coordinate for grid zone square

  for n_gsz in 1 .. 20 loop
    if ( set = 1 or set = 3 or set = 5 ) then
       if ( y_alpha_gsz = n100k_grp1[n_gsz] ) then
         n_coord = 100000 * ( n_gsz - 1);
       end if;
    elsif( set = 2 or set = 4 or set = 6 ) then
       if ( y_alpha_gsz = n100k_grp2[n_gsz] ) then
         n_coord = 100000 * ( n_gsz - 1);
       end if;
    end if;
  end loop;

-- derive grid coordinates and precision

   grid = substring( usng, '[[:digit:]]*$' );

   select into e_grid
     case
       when length( grid::text ) = 2
         then ( cast( substring( grid::text from 1 for 1 ) as integer )
) * 10000
       when length( grid::text ) = 4
         then ( cast( substring( grid::text from 1 for 2 ) as integer )
) * 1000
       when length( grid::text ) = 6
         then ( cast( substring( grid::text from 1 for 3 ) as integer )
) * 100
```

```
       when length( grid::text ) = 8
         then ( cast( substring( grid::text from 1 for 4 ) as integer )
) * 10
       when length( grid::text ) = 10
         then cast( substring( grid::text from 1 for 5 ) as integer )
     end;

   select into n_grid
     case
       when length( grid::text ) = 2
         then ( cast( substring( grid::text from 2 for 1 ) as integer )
) * 10000
       when length( grid::text ) = 4
         then ( cast( substring( grid::text from 3 for 2 ) as integer )
) * 1000
       when length( grid::text ) = 6
         then ( cast( substring( grid::text from 4 for 3 ) as integer )
) * 100
       when length( grid::text ) = 8
         then ( cast( substring( grid::text from 5 for 4 ) as integer )
) * 10
       when length( grid::text ) = 10
         then cast( substring( grid::text from 6 for 5 ) as integer )
     end;

  select into precision
     case
        when length( grid::text ) = 2
           then 10000
        when length( grid::text ) = 4
           then 1000
        when length( grid::text ) = 6
           then 100
        when length( grid::text ) = 8
           then 10
        when length( grid::text ) = 10
           then 1
     end;

-- create usng coords

xmin = round( ( e_coord + e_grid + ( precision / 2 ) ), 1 );
ymin = round( ( offset_north + n_coord + n_grid + ( precision / 2 ) ),
1 );

usng_coords = xmin || ' ' || ymin ;

return ( usng_coords );

end;

$$ language 'plpgsql';
```

4.5.37 XYCoordinate Completeness Measure

4.5.37.1 Measure Name
XYCoordinateCompletenessMeasure

4.5.37.2 Measure Description
This measure checks for coordinate pairs with one member missing. The query produces a list of Address ID and coordinate values where one of the coordinates is null.

4.5.37.3 Report
Logical consistency

4.5.37.4 Evaluation Procedure
Check for null values.

4.5.37.5 Spatial Data Required
AddressPtCollection

4.5.37.6 Code Example: Testing Records
```
SELECT
   AddressID,
   AddressXCoordinate,
   AddressYCoordinate
FROM
   AddressPtCollection
WHERE
   AddressXCoordinate isnull OR AddressYCoordinate isnull
```

4.5.37.7 Code Example: Testing the Conformance of a Data Set

FUNCTION
See Perc Conforming for the query example.

FUNCTION PARAMETERS

count_of_nonconforming_records

```
SELECT
   COUNT(*)
FROM
   AddressPtCollection
WHERE
   AddressXCoordinate isnull OR AddressYCoordinate isnull
```

count_of_total_records

```
SELECT
   COUNT(*)
FROM
   AddressPtCollection
```

4.5.37.8 Result Report Example
Tested XYCoordinateCompletenessMeasure at 93% conformance.

4.5.38 XYCoordinate Spatial Measure

4.5.38.1 Measure Name
XYCoordinateSpatialMeasure

4.5.38.2 Measure Description
This measure compares the coordinate location of the addressed object with the coordinate attributes. The measure applies to both types of coordinate pairs listed in Part One: Address XCoordinate, Address YCoordinate and Address Longitude, Address Latitude. The query produces a list of Address ID and coordinate values in the address collection that do not conform to a spatial domain.

4.5.38.3 Report
Positional accuracy

4.5.38.4 Evaluation Procedure
Check point locations where the geometry does not match coordinate attributes.

4.5.38.5 Spatial Data Required
AddressPtCollection

4.5.38.6 Code Example: Testing Records
It may be important to round the products of **ST_X** or **ST_Y** functions and the Address XCoordinate and Address YCoordinate values to get an accurate match.

QUERY
```
SELECT
     AddressID,
     AddressXCoordinate,
     AddressYCoordinate
FROM
     AddressPtCollection
WHERE
      ST_X( AddressPtGeometry  ) != AddressXCoordinate
      or
       ST_Y( AddressPtGeometry ) != AddressYCoordinate
;
```

4.5.38.7 Code Example: Testing the Conformance of a Data Set

FUNCTION
See Perc Conforming for the query example.

FUNCTION PARAMETERS

count_of_nonconforming_records

```
SELECT
     AddressID,
     AddressXCoordinate,
     AddressYCoordinate
FROM
     AddressPtCollection
WHERE
      ST_X( AddressPtGeometry  ) != AddressXCoordinate
      OR
       ST_Y( AddressPtGeometry ) != AddressYCoordinate
;
```

count_of_total_records

```
SELECT
     COUNT(*)
FROM
      [[AddressPtCollectionMeasureView][AddressPtCollection]]
 ;
```

4.5.38.8 Result Report Example

Tested XYCoordinateSpatialMeasure at 90% conformance.