3.3.2 Address Collection

An Address Collection is an aggregation of address data with its associated metadata, which can then be transferred from one party to another. The Address Collection conforms to the Feature Collection construct that is generally described in the FGDC Geographic Information Framework Data Content Standard, Base Part, section 7.8.1. The Address Collection is described in more detail in Part 4 of this standard.

4 Address Data Quality

4.1 Introduction

4.1.1 Purpose

The purpose of Part Four is to help users assess the quality of their address data. It provides ways to measure each element, attribute and classification. Some measures compare values to address assignment schemes 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 schemes, even and odd addresses on the same side of the street disrupt normal address usage. While the assignment of each address is important, patterns make the system work. The methods describe ways to discover anomalies and how to report the quality of the data.

4.1.2 Quality Definition

Measuring quality requires a definition for quality. There are definitions of quality in existing standards. Those describing addresses, however, describe 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 an 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. What is needed is a definition of the information required, a definition of 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. They have quality reporting requirements that describe the elements of address quality. These elements provide guidance for checking the various aspects of quality control, and a way of 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 Part 7.8: Appendix H.

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 five core elements:

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

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.

4.2 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 scheme systematically, or the scheme 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.

4.2.1 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.

4.3 Measuring Address Quality

4.3.1 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
- Pseudocode Script or Function: an example of the test, in SQL pseudocode.

  The pseudocode was written (except where noted otherwise) using standard ISO/IEC 9075-1:2008 SQL. Spatial predicates used in the pseudocode are described in OpenGIS Simple Features Specification for SQL.

- Parameters for calculating anomalies as a percentage of the data set

4.3.2 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.

4.3.3 Calculating Conforming Records as a Percentage of the Data Set

Perc Conforming measures the results of tests for anomalies and describes the percentage of data elements that conform, that are not anomalies. 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.

Pseudocode function: Calculating conforming records as % of data set	**Description** The function receives information directly from SQL statements including the standard COUNT aggregator and calculates percentages. **Function** Function Perc Conforming(count_of_nonconforming_records, count_of_total_records) RETURNS numeric BEGIN SELECT INTO calc_perc ROUND((CAST((count_of_total_records - count_of_nonconforming_records) AS NUMERIC) / (CAST(count_of_total_records AS NUMERIC)) * 100), 2) RETURN(calc_perc)

	END
Generalized pseudocode example: **Parameters in *italics***	**Query** SELECT Perc Conforming((*SELECT COUNT(*) FROM Address Collection element = nonconforming_condition*), (*SELECT COUNT(*) FROM Address Collection*)) as percent_conforming_to_condition **Successful result: 100% conforming** percent_conforming_to_condition --- 100.00 **Unsuccessful result: 30% conforming** percent_conforming_to_condition --- 30.00

4.3.4 Notation

The tests are described in pseudocode based on SQL. It uses SQL constructs and operators.

Operators used include:

Operator	Description	SQL Example Statement	Statement Result
||	concatenation	SELECT 'a' || 'b';	ab
%	modulo	SELECT 5 % 2;	1

4.4 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 Number Fishbones	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

		Measure	the location of the grid cell. Many Address 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."		

4.5 How to use the Measures in a Quality Control Program

4.5.1 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 scheme boundaries and components
- Address 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 Drive.

Address schemes 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.

4.5.2 Construction

Once all of the domains and rules have been assembled, use the guidance in Parts One and Two to assemble a list of measures for each aspect of your data. Informative Appendices E through H can be helpful in maintaining an overview of the process. Order the measures by their sequence: check simple elements first, then complex elements, then classifications. Where the conditions are entirely local, such as the "Courts" and "Boulevards" example given above, there may not be a standard measure available. In this case create, name and document your own test, taking case to chose 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.

4.5.3 Testing

Construct SQL statements specific to your system from the 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 measures 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 you may have a problem built into your address process.

4.5.4 Interpreting Results

The measures are written to produce sets of identifiers. In practice it's important to see the data you're examining in context. In a normalized relational database it is most often easiest to construct a view to display the data you want to see. For example, the table for Complete Street Name entries will most often be composed of foreign keys to other tables with domains of values for all the street name components. A view with concatenated components can help you see patterns of anomalies that you can handle in an organized, efficient way.

4.5.5 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. Both depend on the process used to assemble a given data set. 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 what you did. One simple thing to do is to record when each suite of tests was used. Users will question aspects of the data. Knowing the condition of the data over time simplifies your response, and increases both the reality and perception of the value of the data.

4.5.6 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 a process that produces data changes.

4.6 About Nodes for Quality Control

4.6.1 About Nodes

Nodes are the end points for each road segment. They are used throughout Address Data Quality in checking features at intersections. The pseudocode below shows 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. This particular example is given in PostgreSQL/PostGIS. 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.

4.6.2 StreetsNodes

The transaction below creates and fills the table.

```
begin;

create table StreetsNodes
(
  id serial primary key,
  nodesfk integer references nodes,
  transegfk integer references TranSeg,
  CompleteStreetName varchar(100),
  seg_end varchar(4)
)
;
```

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

insert into StreetsNodes( transegfk, CompleteStreetName, seg_end, geom )
(
  select
     id,
     CompleteStreetName,
     'from',
     st_startpoint( a.geom )

  from

     TranSeg
)
union
(
  select
     id,
     CompleteStreetName,
     'to',
     st_endpoint( a.geom )
  from
     TranSeg
)
;
end;
```

4.6.3 Nodes

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

The following transaction creates and fills the table

```
    begin;

    create table Nodes
```

```
(
  id serial primary key
)
;

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

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

end;
```

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;
```

4.7 Quality Measures

4.7.1 Address Completeness Measure

Measure Name	Address Completeness Measure
Measure Description	Completeness: a comparison of 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 a domain of exceptions.
Report	Completeness
Evaluation Procedure	Compare the number of addressable objects with the address information recorded.
Spatial Data Required	Geometry describing addressable objects attributed with Address ID, and polygon(s) describing Address Reference System extent.
Pseudocode Example: Testing records	**Query** SELECT Addressable Objects. Address ID FROM AddressReferenceSystemExtent, Addressable Objects WHERE INTERSECTS(Address Reference System Extent.Geometry, Addressable Objects.Geometry) AND Addressable Objects. Address ID is null **Result Without Anomalies** AdddressID ------ **Anomalies** ID ------ 23 97 186 ....

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(Addressable Objects.ID)* *FROM Scheme Extent, Addressable Objects* *WHERE* *INTERSECTS(Scheme Extent.Geometry, Addressable Objects.Geometry)* *AND* *Addressable Objects.Address ID isnull* count_of_total_records *SELECT COUNT(*)* *FROM Addressable Objects*
Result Report Example	Tested Address Completeness Measure at 98% conformance.

4.7.2 Address Elevation Measure

Measure Name	Address Elevation Measure
Measure Description	This measure checks elevations against polygons created from contours of elevation.
Report	Attribute (Thematic) Accuracy
Evaluation Procedure	Check each elevation identified by the measure as outside the range defined by the polygons.
Spatial Data Required	Point locations for each address with an elevation, polygons created from contours of elevation.
Pseudocode Example: Testing Records	**Function** create or replace function addr_check_elev(int) returns boolean as $BODY$ declare addr_elev_id alias for $1; within_poly_values boolean; begin

	select into within_poly_values case when a.AddressElevation between b.min_elev and b.max_elev then TRUE else FALSE end as "within_poly" from AddressElevations a, ContourPolygons b where a.id = addr_elev_id and intersects(a.Geometry, b.Geometry) ; return (within_poly_values); end $BODY$ language 'plpgsql'; **Query** select id, addr_check_elev(id) from AddressElevations
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 AddressElevationPoints* *WHERE addr_check_elev(id) = FALSE* count_of_total_records *SELECT COUNT(*)* *FROM AddressElevationPoints*
Result Report Example	Tested Address Elevation Measure at 94% conformance.

4.7.3 Address Left Right Measure

Measure Name	Address Left Right Measure
Measure Description	Determine the side of the street centerline, left or right, on which each address is located. Left and right attributes are frequently entered by hand, an error-prone process. It is important to confirm the actual locations of the addresses. 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. AddressLeftRightMeasure 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, you can 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.
Report	Logical Consistency
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.
Spatial Data Required	Street centerline and address point locations. The identifier for the street segment associated with each address must be recorded with the address points, and the Address Range Directionality must be recorded with each segment.

Pseudocode Example: Assembling Data	**Query to determine left and right** -- -- The query below describes finding the left-right status of -- address points, storing the results in a table for use in -- later quality control processes. Each subquery is commented. -- -- This query requires pre-existing information about the street -- centerline segment to which each address point is related. -- -- The **select** query may be used on its own, or the **insert** -- clause pre-pended to it in order to save the information in a table. -- This example assumes you want to save quite a lot of information -- ---------- insert into AddressLeftRight (Address ID, StreetCenterlineID, AddressCompleteStreetNameID, StreetCenterlineAddressRangeDirectionality, StreetCenterlineCompleteStreetNameID, Point2LineDistance, Side, AddressPointGeometry) -- -- Figure the angles to determine whether the point is to the left or right of -- the portion of the line segment closest to the access point. Select -- all the foreign keys, street name identifiers and distance between the -- the access point and the street segment along with it. The results show -- left-right parity, along with the information used to derive it. -- Use select distinct to account for the adjacent points for which you are -- generating measurement. When the address point is perfectly aligned with -- the two point in the segment it will generate two records, one for right and -- one for left. These can be discovered using the next query example, and

```
-- resolved individually in favor of the one that agrees with the
-- Address Reference System. There are usually very few of these in
-- any given jurisdiction.

select distinct
bar.AddressID
bar.StreetCenterlineID
bar.AddressCompleteStreetNameID,
bar.StreetCenterlineCompleteStreetNameID,
bar.StreetCenterlineAddressRangeDirectionality,
bar.Point2LineDistance,
case
when
(
degrees( azimuth( bar.Pt1, bar.AddressPointGeometry ) )
-
degrees( azimuth( bar.Pt1, bar.Pt2 ) )
) between 0 and 180 then 'right'
when
(
degrees( azimuth( bar.Pt1, bar.AddressPointGeometry ) )
-
degrees( azimuth( bar.Pt1, bar.Pt2 ) )
) between 180 and 360 then 'left'
when
(
degrees( azimuth( bar.Pt1, bar.AddressPointGeometry ) )
-
degrees( azimuth( bar.Pt1, bar.Pt2 ) )
) between -180 and 0 then 'left'
when
(
degrees( azimuth( bar.Pt1, bar.AddressPointGeometry ) )
-
degrees( azimuth( bar.Pt1, bar.Pt2 ) )
) between -360 and -180 then 'right'
end as "Side",
bar.AddressPointGeometry
from
--
-- Select all the fields from the inner query, with the
-- exception of the selected line geometry. Adjacent sets
-- of points describing the line geometry are generated
```

```
-- instead. These adjacent points allow measurement of the
-- angle to the closest part of the line segment.
--
( select
foo.AddressID,
foo.StreetCenterlineID,
foo.AddressCompleteStreetNameID,
foo.StreetCenterlineCompleteStreetNameID,
foo.Point2LineDistance,
foo.AddressPointGeometry,
foo.ClosestPointOnStreetCenterline,
foo.!StreetCenterlineAddressRangeDirectionality,
pointn( foo.st_geom, generate_series( 1, ( numpoints(
foo.StreetCenterlineGeometry ) - 1 ) ) ) as Pt1,
pointn( foo.st_geom, generate_series( 2, numpoints(
foo.StreetCenterlineGeometry ) ) ) as Pt2
from
--
-- Assemble the various identifiers and geometries for the
-- query.
--
( select
a.AddressID,
b.StreetCenterlineID
a.CompleteStreetNameID as AddressCompleteStreetNameID,
b.CompleteStreetNameID as
StreetCenterlineCompleteStreetNameID,
distance( a.AddressLocationGeometry, b.locationGeometry) as
Point2LineDistance,
line_interpolate_point( b.locationGeometry, ( line_locate_point(
b.locationGeometry, a.locationGeometry ) ) ) as
ClosestPointOnStreetCenterline,
a.AddressPointGeometry,
b.!StreetCenterlineAddressRangeDirectionality,
b.StreetCenterlineGeometry
from
--
-- Select the access point foreign key,
-- the primary address foreign key,
-- and the street name foreign key
-- from the access point layer,
-- the primary address layer
-- and the intersection table.
--
```

	(select Address ID, CompleteStreetNameID AddressPointGeometry from Address Collection) as a, -- -- Select the street centerline primary key, -- the street name foreign key, -- the Address Range Directionality, -- and the geometry as a linestring from a layer -- where the geometry is stored as multilinestrings. -- (select StreetCenterlineID, CompleteStreetNameID StreetCenterlineAddressRangeDirectionality, StreetCenterlineGeometry from StreetCenterline) as b where a.StreetCenterlineIDfk = b.StreetCenterlineID) as foo) as bar where expand(bar.ClosestPointOnStreetCenterline, 1) && makeline(bar.Pt1, bar.Pt2) ;
Pseudocode Example: Checking For Address Points with both Left and Right Records	-- This query should describe few, if any, records. They should be resolved individually -- 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 ;
Pseudocode Example: Checking existing attributes against data produced by the queries above	-- This query produces a list of Address ID values for which the left/right -- attribute can not be checked by the left/right information in the query results -- or where the left/right attribute conflicts with query results -- select a.AddressID from TableWithHandEnteredLeftRightAttributes a left join AddressLeftRight b on a.AddressID = b.AddressID where b.AddressID is null or (a.Side != b.Side and (b.StreetCenterlineAddressRangeDirectionality = 'With' or (a.Side = 'left' and b.StreetCenterlineAddressRangeDirectionality = 'With-Against') or (a.Side = 'right' and b.StreetCenterlineAddressRangeDirectionality = 'Against-With')) or (a.Side = b.Side and (b.StreetCenterlineAddressRangeDirectionality = 'Against' or (a.Side = 'left'

	and b.StreetCenterlineAddressRangeDirectionality = 'Against-With') or (a.Side = 'right' and b.StreetCenterlineAddressRangeDirectionality = 'With-Against')) ;
Pseudocode Example: Testing the conformance of a Data Set	**Function** See Perc Conforming for the sample query. **Function Parameters** count_of_nonconforming_records **Result Without Anomalies** Address ID --------- **Anomalies** Address ID --------- 37 52 96 ...
Pseudocode Example: Testing the Conformance of a Data Set	**Function** See Perc Conforming for the sample query. **Function Parameters** count_of_nonconforming_records count_of_total_records *SELECT COUNT(*)* *FROM Address Collection*
Result Report Example	Tested Address Left Right Measure at 85% conformance.

4.7.4 Address Lifecycle Status Date Consistency Measure

Measure Name	Address Lifecycle Status Date Consistency Measure

Measure Description	Test the agreement of the Address Lifecycle Status with the development process. This query produces a list of Address I Ds for which the Address Lifecycle Status attributes do not logically agree with the corresponding stages in the process. This query is far more conceptual than many other queries 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.
Report	Temporal Accuracy/Logical Consistency
Evaluation Procedure	Use Simple Elements checks to validate Address Lifecycle Status entries against the domain. Check logical relationships between the status values and stages in the development process.
Spatial Data Required	None

Pseudocode Example: Testing records	**Query**
	--
	-- In this example, 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.
	--
	
	SELECT
	  Address ID
	FROM
	  Address Collection
	WHERE
	(Address Lifecycle Status = 'Potential'
	AND
	(BuildingPermit is not null
	OR Address Start Date is null
	OR Address End Date is not null
	)
	)
	OR
	(Address Lifecycle Status = 'Proposed'

	AND (BuildingPermit is not null OR Address Start Date is null OR Address End Date is not null)) OR (Address Lifecycle Status = 'Active' AND (Address Start Date is null or Address End Date is not null)) OR (Address Lifecycle Status = 'Retired' AND Address End Date is null)) ;
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 Address Collection* *WHERE* *(Address Lifecycle Status = 'Potential'* *AND* *(BuildingPermit is not null* *OR Address Start Date is null* *OR Address End Date is not null* *)* *)* *OR* *(Address Lifecycle Status = 'Proposed'* *AND* *(BuildingPermit is not null* *OR Address Start Date is null* *OR Address End Date is not null* *)* *)* *OR* *(Address Lifecycle Status = 'Active'* *AND (Address Start Date is null or Address End Date is not null* *)* *)*

	OR *(* Address Lifecycle Status *= 'Retired'* *AND* Address End Date *is null)* *)* count_of_total_records *SELECT COUNT(*)* *FROM Address Collection*
Result Report Example	Tested Address Lifecycle Status Date Consistency Measure at 100% conformance.

4.7.5 Address Number Fishbones Measure

Measure Name	Address Number Fishbones Measure
Measure Description	Generate 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. Where there are sequence conflicts lines between those points will cross. 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.
Report	Logical Consistency
Evaluation Procedure	Check for addresses having constructed lines that cross others.
Spatial Data Required	Addressed object location points attributed with Address ID, Address X Coordinate and Address Y Coordinate values; and points describing the same addresses geocoded on a street centerline with 0 offset distance attributed with corresponding, Address ID, TranSegId, Address X Coordinate and Address Y Coordinate values. Note that the query produces spatial data, and a table to collect those results is recommended.

Pseudocode Example: Testing records	The result table has to hold, at minimum, the Address ID, the TranSegId and the lines generated. This allows the fishbones to describe a relationship between the addressed feature and the street on which it is addressed in both geometry and attributes. It may be convenient to add additional information, such as the Address Number and/or Complete Street Name. In the latter case, both the query and the table should be altered to respectively create and hold the information. Note that the geomfromtext function, from the Open Geospatial Consortium's Simple Features for SQL Specification, requires an Address Coordinate Reference System ID. The example is the European Petroleum Survey Group (EPSG) identifier for "NAD83 / Colorado South (ftUS), listed in version 6.13 of the database. **Result Table** CREATE TABLE site2geocode (pkey SERIAL PRIMARY KEY, Address ID INTEGER NOT NULL REFERENCES Address Collection, TranSegID INTEGER NOT NULL REFERENCES TranSeg Collection, geom geometry) **Query** insert into site2geocode(Address ID, geom) select a.AddressID, b.TranSegId, st_makeline(st_makepoint(a.addr_x, a.addr_y), st_makepoint(b.geocode_x, b.geocode_y)) from Address Point a, Geocoded Point With 0 Offset b where a. Address ID = b. Address ID ;
Pseudocode Example: Testing the Conformance of a Data Set	**Function** See Perc Conforming for the query example. **Function Parameters**

	count_of_nonconforming_records *SELECT COUNT(selected_id)* *FROM site2geocode* *WHERE crosses_id isnull* count_of_total_records *SELECT COUNT(* Address Number *)* *FROM Address Collection*
Result Report Example	Tested Address Number Fishbones Measure at 90% conformance.
Attribution	Address Number Fishbones Measure is based on the "fishboning" test developed by Richard Allen of MAGIC GIS, and is included with his permission.

4.7.6 Address Number Parity Measure

Measure Name	Address Number Parity Measure
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 ((Address Number % 2) = 1 and Address Number Parity = 'even') or ((Address Number % 2) = 0 and Address Number Parity = 'odd');
Report	Logical Consistency
Evaluation Procedure	Compare the odd/even status of the numeric value of an address number with the Address Number Parity attribute.
Spatial Data Required	None.
Pseudocode Example: Testing records	**Query** SELECT Address ID, Address Number FROM Address Collection WHERE (Address Number - ((Address Number / 2) * 2)) = 0 and Address Number Parity = 'odd') or (Address Number - ((Address Number / 2) * 2)) = 1 and Address Number Parity = 'even')

	Result Without Anomalies Address Number Parity Measure --- **Anomalies** Address Number Parity Measure --- Address ID, Address Number 1 Address ID, Address Number 2 Address ID, Address Number 3 ....
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(Address ID)* *FROM Address Collection* *WHERE* *(Address Number - ((Address Number / 2) * 2)) = 0 and Address Number Parity = 'odd')* *or* *(Address Number - ((Address Number / 2) * 2)) = 1 and Address Number Parity = 'even')* count_of_total_records *SELECT COUNT(Address ID)* *FROM Address Collection*
Result Report Example	Tested Address Number Parity Measure at 82% conformance.

4.7.7 Address Number Range Completeness Measure

Measure Name	Address Number Range Completeness Measure
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.
Report	Logical Consistency
Evaluation Procedure	Check for a non-zero value for both low and high each range.
Spatial Data Required	None.
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, however, 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 range values where you see -- [Two Number Address Range.Low | Four Number Address Range.Side.Low] -- -- Two Number Address Range | Four Number Address Range.Side test -- SELECT Two Number Address Range | Four Number Address Range.Side FROM Address Collection WHERE (Two Number Address Range.Low | Four Number Address Range.Side.Low is null OR Two Number Address Range.Low | Four Number Address Range.Side.Low = 0) OR (Two Number Address Range.High | Four Number Address Range.Side.High is null OR Two Number Address Range.High | Four Number Address Range.Side.High = 0) -- -- Two Number Address Range | Four Number Address Range.Side test -- *SELECT*

	COUNT(Two Number Address Range | Four Number Address Range.*Side)* *FROM* *Address Collection* *WHERE* *(* Two Number Address Range.*Low* | Four Number Address Range.*Side.Low is null OR* Two Number Address Range.*Low* | Four Number Address Range.*Side.Low = 0)* *OR* *(* Two Number Address Range.*High* | Four Number Address Range.*Side.High isnull OR* Two Number Address Range.*High* | Four Number Address Range.*Side.High = 0)* count_of_total_records -- -- Two Number Address Range | Four Number Address Range.Side test -- *SELECT COUNT(* Two Number Address Range | Four Number Address Range.*Side)* *FROM Address Collection* --
Result Report Example	Tested Address Number Range Completeness Measure at 50% conformance.

4.7.8 Address Number Range Parity Consistency Measure

Measure Name	Address Number Range Parity Consistency Measure
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. Alternate WHERE clauses with a modula are listed below. Two Number Address Range | Four Number Address Range.Side test: WHERE (Two Number Address Range.Low | Four Number Address Range.Side.Low % 2) != (Two Number Address Range.High | Four Number Address Range.Side.High % 2)
Report	Logical Consistency
Evaluation Procedure	Compare the odd/even status of the numeric value of each address number.
Spatial Data Required	None.

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, however, 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 range values where you see -- [Two Number Address Range.Low | Four Number Address Range.Side.Low] -- Two Number Address Range | Four Number Address Range.Side test: SELECT Two Number Address Range | Four Number Address Range.Side FROM Address Collection WHERE (Two Number Address Range.Low | Four Number Address Range.Side.Low - (truncate(Two Number Address Range.Low | Four Number Address Range.Side.Low / 2) * 2)) != (Two Number Address Range.High | Four Number Address Range.Side.High - (truncate(Two Number Address Range.High | Four Number Address Range.Side.High/ 2) * 2)) **Result Without Anomalies** Address Number Range Parity Consistency Measure ---------- **Anomalies** Two Number Address Range | Four Number Address Range.Side test: Address Number Range Parity Consistency Measure ---------- Two Number Address Range | Four Number Address Range.Side 1 Two Number Address Range | Four Number Address Range.Side 2 Two Number Address Range | Four Number Address

	Range.Side 3 ....
Pseudocode Example: Testing the Conformance of a Data Set	**Function** See Perc Conforming for the sample query. **Function Parameters** Two Number Address Range | Four Number Address Range.Side test: count_of_nonconforming_records *SELECT COUNT(* Two Number Address Range | Four Number Address Range.*Side)* *FROM Address Collection* *WHERE (* Address Number.*Low - ((* Address Number.*Low / 2) *))* *!= (* Address Number.*High - ((* Address Number.*Low / 2) *))* count_of_total_records Two Number Address Range | Four Number Address Range.Side test: *SELECT COUNT(* Two Number Address Range | Four Number Address Range.*Side)* *FROM Address Collection*
Result Report Example	Tested Address Number Range Parity Consistency Measure at 90% consistency.

4.7.9 Address Number Range Sequence Measure

Measure Name	Address Number Range Sequence Measure
Measure Description	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.
Report	Logical Consistency
Evaluation Procedure	Check ranges on each side of a common point.

Spatial Data Required	Street centerline.
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, however, 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 range values where you see -- [Two Number Address Range.Low | Four Number Address Range.Side.Low] -- -- Two Number Address Range | Four Number Address Range.Side test -- SELECT a.CompleteStreetName, a. Two Number Address Range.Low | Four Number Address Range.Side.Low, a. Two Number Address Range.High | Four Number Address Range.Side.High, b. Two Number Address Range.Low | Four Number Address Range.Side.Low, b. Two Number Address Range.High | Four Number Address Range.Side.High FROM Address Collection as a, (SELECT Complete Street Name, Two Number Address Range.Low | Four Number Address Range.Side.Low, Two Number Address Range.High | Four Number Address Range.Side.High FROM Address Collection) as b WHERE a.SegmentDirection = 'low to high' AND b.SegmentDirection = 'low to high' AND st_equals(st_endpoint(a.Geometry), st_startpoint(b.Geometry)) AND a. Two Number Address Range.Low | Four Number Address

	Range.Side.Low >= b. Two Number Address Range.High | Four Number Address Range.Side.High AND a.CompleteStreetName = b.CompleteStreetName GROUP BY a.CompleteStreetName, a. Two Number Address Range.Low | Four Number Address Range.Side.Low, a. Two Number Address Range.High | Four Number Address Range.Side.High, b. Two Number Address Range.Low | Four Number Address Range.Side.Low, b. Two Number Address Range.High | Four Number Address Range.Side.High **Result Without Anomalies** street | end_toleft | end_toright | start_fromleft | start_fromright ----------+---------------+----------------+--------------+----------- - **Result With Anomalies** street | end_toleft | end_toright | start_fromleft | start_fromright ----------+---------------+----------------+--------------+----------- - BIRD DRIVE | 1048 | 1049 | 1048 | 1049 BIRD DRIVE | 1248 | 1249 | 1148 | 1149
Pseudocode Example: Testing the Conformance of a Data Set	**Function** See Perc Conforming for the sample query. **Function Parameters** count_of_nonconforming_records -- -- Two Number Address Range | Four Number Address Range.Side test -- *SELECT COUNT(*)* *FROM Address Collection as a,*

	(SELECT Complete Street Name*,* Four Number Address Range*.Side.Low,* Four Number Address Range*.Side.High* *FROM Address Collection)* *as b* *WHERE* *a.SegmentDirection = 'low to high'* *AND* *b.SegmentDirection = 'low to high'* *AND* *st_equals(st_endpoint(a.Geometry), st_startpoint(b.Geometry))* *AND* *a.FourNumberAddressRange.Side.Low >=* *b.FourNumberAddressRange.Side.High* *AND* *a.CompleteStreetName = b.CompleteStreetName* *GROUP BY a.CompleteStreetName,* *a.FourNumberAddressRange.Side.Low,* *a.FourNumberAddressRange.Side.High,* *b.FourNumberAddressRange.Side.Low,* *b.FourNumberAddressRange.Side.High* *count_of_total_records --* -- Two Number Address Range | Four Number Address Range*.Side test* -- *_SELECT COUNT(* Two Number Address Range | Four Number Address Range*.Side)* *FROM Address Collection* --
Result Report Example	Tested Address Number Range Sequence Measure at 85% conformance.

4.7.10 Address Range Directionality Measure

Measure Name	Address Range Directionality Measure
Measure Description	This measure derives Address Range Directionality values, allowing update to and/or checks of values stored in the database. It requires that Address Side Of Street be known, and checked with Address Left Right Measure.

Report	Logical Consistency
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.
Spatial Data Required	Street centerlines, address point locations, and fishbones (see Address Number Fishbones Measure).
Pseudocode Example	-- -- The example below determines the AddressRangeDirectionality value for a single segment. -- Fill in the TranSegId value for the segment where "[*fill in TranSegId*]" appears in the query. -- select boom.TranSegId, case when boom.directionality_left = boom.directionality_right then boom.directionality_left when boom.directionality_left is not null and boom.directionality_right is null then boom.directionality_left when boom.directionality_left is null and boom.directionality_right is not null then boom.directionality_right when boom.directionality_left is not null and boom.directionality_right is not null and boom.directionality_left = boom.directionality_right then boom.directionality_left || '-' || boom.directionality_right end as "AddressRangeDirectionality" from (

```
select
a.TranSegId,
bim.lf_id,
bim.lf_number,
bim.lt_id,
bim.lt_number,
case
when
bim.lf_number is not null
and
bim.lt_number is not null
and
bim.lf_number = bim.lt_number
and
( st_distance( st_startpoint( a.Geometry ), bim.lf.Geometry )
  <
  st_distance( st_endpoint( a.Geometry ), bim.lt.Geometry )
)
then
'with'
when
bim.lf_number is not null
and
bim.lt_number is not null
and
bim.lf_number = bim.lt_number
and
( st_distance( st_startpoint( a.Geometry ), bim.lf.Geometry )
   >
   st_distance( st_endpoint( a.Geometry ), bim.lt.Geometry )
)
then
'against'
end as "directionality_left",
bam.rf_id,
bam.rf_number,
bam.rt_id,
bam.rt_number,
case
when
bam.rf_number is not null
and
bam.rt_number is not null
and
```

```
bam.rf_number = bam.rt_number
T and
( st_distance( st_startpoint( a.Geometry ), bam.rf.Geometry )
  <
  st_distance( st_endpoint( a.Geometry ), bam.rt.Geometry )
)
then
'with'
when
bam.rf_number is not null
and
bam.rt_number is not null
and
bam.rf_number = bam.rt_number
and
( st_distance( st_startpoint( a.Geometry ), bam.rf.Geometry )
  >
  st_distance( st_endpoint( a.Geometry ), bam.rt.Geometry )
)
then
'against'
end as "directionality_right"
from
TranSegCollection a
left join
( select
foo.TranSegId,
b.AddressID as lf_id,
b.AddressNumber as lf_number,
b.Geometry as lf_geom,
d.AddressID as lt_id,
d.AddressNumber as lt_number,
d.Geometry as lt.Geometry
from
AddressNumberFishbones a,
AddressCollection b,
AddressNumberFishbones c,
AddressCollection d,
(
select
[ fill in TranSegId ] as TranSegId,
min( c.AddressNumber ),
max( c.AddressNumber )
from
```

TransSegCollection a,
AddressNumberFishbones b,
AddressCollection c,
Address Side Of Street d
where
a.id = [*fill in TranSegId*]
and
a.id = b.TranSegId
and
b.AddressID = c.AddressID
and
c.AddressID = d.AddressID
and
d.AddressSideOfStreet = 'left'
) as foo
where
a.TranSegId = foo.TranSegId
and
a.AddressID = b.AddressID
and
b.AddressNumber = foo.min
and
c.TranSegId = foo.TranSegId
and
c.AddressID = d.AddressID
and
d.AddressNumber = foo.max
) as bim
on a.TranSegId = bim.TranSegId
left join
(select
foo.TranSegId,
b.AddressID as rf_id,
b.AddressNumber as rf_number,
b.Geometry as rf_geom,
d.geodb_oid as rt_id,
d.AddressNumber as rt_number,
d.Geometry as rt.Geometry
from
AddressNumberFishbones a,
AddressCollection b,
AddressNumberFishbones c,
AddressCollection d,
(

	select [*fill in TranSegId*]::integer as TranSegId, min(c.AddressNumber), max(c.AddressNumber) from TranSegCollection a, AddressNumberFishbones b, AddressCollection c, AddressLeftRight d where a.geodb_oid = [*fill in TranSegId*] and a.geodb_oid = b.TranSegId and b.AddressID = c.geodb_oid and c.geodb_oid = d.AddressID and d.side = 'right') as foo where a.TranSegId = foo.TranSegId and a.AddressID = b.geodb_oid and b.AddressNumber = foo.min and c.TranSegId = foo.TranSegId and c.AddressID = d.geodb_oid and d.AddressNumber = foo.max) as bam on a.geodb_oid = bam.TranSegId where a.TranSegId = [*fill in TranSegId*]) as boom ;
Pseudocode Example: Checking existing attributes against data produced by the query above	-- -- Run the query above for each of the segments within the Address Reference System Extent --

	select a.TranSegId, a.AddressRangeDirectionality, b.AddressRangeDirectionality from Address Range Directionality Measure Results a left join Database AddressRangeDirectionality Values b on a.TranSegId = b.TranSegId where b.AddressRangeDirectionality is null or a.AddressRangeDirectionality = b.AddressRangeDirectionality ;
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(a.TranSegId)* *from* Address Range Directionality Measure *Results a* *left join Database AddressRangeDirectionality Values b* *on a.TranSegId = b.TranSegId* *where* *b.AddressRangeDirectionality is null* *or* *a.AddressRangeDirectionality = b.AddressRangeDirectionality* *;* count_of_total_records *SELECT COUNT(TranSegId)* *FROM TranSegIdCollection*
Result Report Example	Tested Address Range Directionality Measure at 94% conformance.

4.7.11 Address Reference System Axes Point Of Beginning Measure

Measure Name	Address Reference System Axes Point Of Beginning Measure
Measure Description	Check the location of the Address Reference System Axis Point Of Beginning against the intersection of the Address Scheme X Axis
Report	Logical Consistency

Evaluation Procedure	Make sure the axes actually intersect. If that intersection is also the location of the Address Reference System Axis Point Of Beginning, compare the point locations.
Spatial Data Required	Address Reference System Axis Point Of Beginning, Address Reference System Axis
Pseudocode Example: Locating Address Reference System Axis Point Of Beginning in use	**Description** The query assumes that the data are stored topologically, so each axis is split at the intersection. It makes, however, no assumptions about the directionality of the axis lines themselves. The query should return a single TRUE result. 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. **Query** select equals(foo.x_origin, foo.y_origin) 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(geom) as start, st_endpoint(geom) as end from axes where axis = 'north') as north, (select st_startpoint(geom) as start, st_endpoint(geom) as end from axes where axis = 'south') as south, (select st_startpoint(geom) as start, st_endpoint(geom) as end from axes where axis = 'east') as east, (select st_startpoint(geom) as start, st_endpoint(geom) as end from axes where axis = 'west') as west) as foo ;
Pseudocode Example: Checking Address Reference System Axis Point Of Beginning **in use against the** Address Reference System Axis Point Of Beginning **of record**	**Description** The query assumes that the previous query has been run successfully, describing the Address Reference System Axis Point Of Beginning in use. The query should return a single TRUE result. **Query**

	select equals(foo.x_origin, a.AddressReferenceSystemAxisPointOfBeginning) from Address Reference System Axis Point Of Beginning a, (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(geom) as start, st_endpoint(geom) as end from axes where axis = 'north') as north, (select st_startpoint(geom) as start, st_endpoint(geom) as end from axes where

	axis = 'south') as south, (select st_startpoint(geom) as start, st_endpoint(geom) as end from axes where axis = 'east') as east, (select st_startpoint(geom) as start, st_endpoint(geom) as end from axes where axis = 'west') as west) as foo ;
Pseudocode Example: Testing the Conformance of a Data Set	**Function** See Perc Conforming for the sample query. **Function Parameters** count_of_nonconforming_records N/A. This measure produces a result that conforms 100% or 0%. count_of_total_records N/A. This measure produces a result that conforms 100% or 0%.
Result Report Example	Tested Address Reference System Axes Point Of Beginning Measure at 100% conformance.

2997 ### 4.7.12 Address Reference System Rules Measure

Measure Name	Address Reference System Rules Measure
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.
Report	Logical Consistency. Given the variability involved in testing it will be important to report the queries actually used along with the results.
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.
Spatial Data Required	For the example, street centerline 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.
Pseudocode Example: Testing Records	select a.CompleteStreetName, a.TranSegId, b.GridCellId, a.[Two Number Address Range.Low | Four Number Address Range.Side.Low] b.EastWestLowRangeNumber, b.NorthSouthLowRangeNumber, case when ((a.[Two Number Address Range.Low | Four Number Address Range.Side.Low] = b.EastWestLowRangeNumber and a.GeometryDirection = 'east-west') or (a.[Two Number Address Range.Low | Four Number Address Range.Side.Low] = b.NorthSouthLowRangeNumber and a.GeometryDirection = 'north-south')) then 'ok' else 'anomaly' end as "RangeAnomaly" from TranSegCollection a.

	Address Reference System b, where intersects(st_startpoint(a.Geometry), b.Geometry) order by a.[Two Number Address Range.Low | Four Number Address Range.Side.Low] limit 1 ;
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(TranSegId)* *FROM TranSegCollection* *WHERE RangeAnomaly = 'anomaly'* count_of_total_records *SELECT COUNT(TranSegId) FROM TranSegCollection*
Result Report Example	Tested Address Reference System Rules at 95% conformance.

4.7.13 Check Attached Pairs Measure

Measure Name	Check Attached Pairs Measure
Measure Description	This measure describes how to check Attached Element attributes set to "attached" for matching values describing adjacent street name components. By definition, if a component is "attached", one of the adjacent components must also be "attached".
Report	Logical Consistency
Evaluation	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. The value of the street name as a whole will be checked in the Complete Street Name Tabular Domain Measure.
Spatial Data Required	None
Pseudocode Example	SELECT Address ID

	FROM Street name component attributes WHERE attached = TRUE and ((PreviousStreetNameComponent.attached = FALSE or PreviousStreetNameComponent.attached is null) and (SucceedingStreetNameComponent.attached = FALSE or SucceedingStreetNameComponent.attached is null))
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(Address ID) FROM Street name component attributes WHERE attached = TRUE and ((PreviousStreetNameComponent.attached = FALSE or PreviousStreetNameComponent.attached is null) and (SucceedingStreetNameComponent.attached = FALSE or SucceedingStreetNameComponent.attached is null)) ; count_of_total_records SELECT COUNT(Address ID) FROM Street name component attributes ;
Result Report Example	Tested Check Attached Pairs Measure at 92% conformance.

2999

4.7.14 Complete Street Name Tabular Domain Measure

Measure Name	Complete Street Name Tabular Domain Measure
Measure Description	Test the agreement of each Complete Street Name with the domain.
Report	Attribute (Thematic) Accuracy
Evaluation Procedure	Check the Complete Street Name against a domain of values. Creating a list of Complete Street Name values requires taking account of Attached Element attributes. Both the components and the Attached Element attributes may be selected to suit the needs of a particular locality. The function below is one example of a typical function for assembling components into Complete Street Name values. It checks for Attached Element values before putting a space between components. These attributes may describe any of the street name components but most often describe the relationship between a Separator Element and a Street Name. The example function given here illustrates that condition. The query following the function allows you to check a list of street names compiled without parsing against those assembled using the function. If you maintain the street names only as parsed components, assemble the street names using the function and ask at least two people who did not enter them to check the list by hand.
Spatial Data Required	None
Pseudocode Example: Testing records	**Function** create or replace function concat_csn(varchar, varchar, varchar, varchar, boolean, varchar, boolean, varchar, varchar, varchar) returns varchar as $$ declare predir alias for $1; premod alias for $2; pretype alias for $3; separator alias for $4 separator_attached alias for $5; stname alias for $6; stname_attached alias for $7; postmod alias for $8; posttype alias for $9; postdir alias for $10; csn varchar;

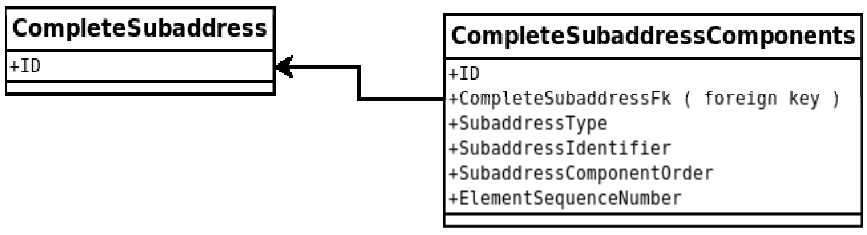
```
begin

csn = '';

-- predir
csn = csn || coalesce( predir, '' );
if predir is not null
then csn = csn || ' ' ;
end if;

-- premod
csn = csn || coalesce( premod, '' );
if premod is not null
then csn = csn || ' ' ;

-- pretype
csn = csn || coalesce( pretype, '' );
if pretype is not null
then csn = csn || ' ' ;
end if;

-- separator
csn = csn || coalesce( separator, '' );
if separator is not null
and
(separator_attached isnull or separator_attached = FALSE )
then csn = csn || ' ' ;
end if;

-- stname
csn = csn || stname;
if posttype is not null or postdir is not null
then csn = csn || ' ' ;
end if;

-- postmod
csn = csn || coalesce( postmod, '' );
if postmod is not null
and
postdir is not null
then csn = csn || ' ' ;
end if;

-- posttype
```

	csn = csn || coalesce(posttype, ");
	if posttype is not null
	and
	(postdir is not null)
	then csn = csn || ' ' ;
	end if;
	
	-- postdir
	csn = csn || coalesce(postdir, ");
	
	return trim(both csn);
	
	end;
	
	$$ language plpgsql;
	
	Query
	
	SELECT
	CompleteStreetName As disagreeWithCompleteStreetNameDomain
	FROM
	Address Collection
	LEFT OUTER JOIN Complete Street Name Domain ON
	CompleteStreetName = Complete Street Name Domain.Value
	WHERE
	CompleteStreetName Domain.Value isnull
	;
	
	Result Without Anomalies
	disagreeWithCompleteStreetNameDomain
	---------- 	
	
	Anomalies
	disagreeWithCompleteStreetNameDomain
	---------- 	
	Complete Street Name 1
	Complete Street Name 2
	Complete Street Name 3
	....

Pseudocode Example: Testing the Conformance of a	**Function** See Perc Conforming for the sample query. **Function Parameters**

Data Set	count_of_nonconforming_records *SELECT COUNT(* Complete Street Name *)* *FROM Address Collection* *LEFT OUTER JOIN* Complete Street Name *Domain* *ON* Complete Street Name = Complete Street Name *Domain Value* *WHERE* Complete Street Name *Domain Value isnull* count_of_total_records *SELECT COUNT(Complete Street Name) FROM Address Collection*
Result Report Example	Tested Complete Street Name Tabular Domain Measure at 83% conformance.

3001 **4.7.15 Complex Element Sequence Number Measure**

Measure Name	Complex Element Sequence Number Measure
Measure Description	This measure describes how to assemble a complex element in order by Element Sequence Number, and test it for completeness of the elements. 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.
Report	Attribute (Thematic) Accuracy
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.
Spatial Data Required	None
Pseudocode Example	**Table Structure** The table structure below describes the structure for the example function and query. It is an example only: tables for any given jurisdiction will almost certainly differ. 

Function

```
create or replace function AssembleSubaddressString( int ) returns varchar
as
$BODY$
declare
id alias for $1;
csa CompleteSubaddressComponents%rowtype;
Complete Subaddress varchar;
begin
FOR csa in SELECT * FROM CompleteSubaddressComponents where
CompleteSubaddressFk = id order by element_seq
LOOP
IF Element Sequence Number = 1
AND
( Subaddress Component Order = 1 or Subaddress Component Order = 3 )
THEN Complete Subaddress := Subaddress Type || ' ' || Subaddress
Identifier;

ELSIF Element Sequence Number = 1
AND
Subaddress Component Order = 2
THEN Complete Subaddress := Subaddress Identifier || ' ' || Subaddress
Type;

ELSIF Element Sequence Number > 1
AND
( Subaddress Component Order = 1 OR Subaddress Component Order = 3
)
THEN Complete Subaddress := Complete Subaddress || ', ' || Subaddress
Type || ' ' || Subaddress Identifier ;

ELSIF Element Sequence Number > 1
AND
Subaddress Component Order = 2
THEN Complete Subaddress := Complete Subaddress || ', ' || Subaddress
Identifier || ' ' || Subaddress Type ;

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
	;

Pseudocode Example: Testing the Conformance of a Data Set	**Function**
	See Perc Conforming for the sample query.
	
	Function Parameters
	count_of_nonconforming_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
	  AssembleSubaddressString(b.CompleteSubaddressFk)
	;
	count_of_total_records SELECT COUNT(*)
	FROM Complete Subaddress
	;

Result Report Example	Tested Complex Element Sequence Number Measure at 93% conformance.

4.7.16 Data Type Measure

Measure Name	Data Type Measure
Measure Description	This test 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 it the file. In this case the data are frequently loaded to a staging table with all the fields defined as character varying (VARCHAR). 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 they 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.
Report	Logical consistency
Evaluation Procedure	Test each value in the address collection for its data type. Any elements that do not agree with the specified data type are anomalies.
Spatial Data Required	None
Pseudocode Example: Testing Records	**Query** SELECT COUNT(value_type), value_type FROM (SELECT CASE WHEN coalesce(trim(value::varchar)) ~ '^[0-9]*$' then 'integer' WHEN coalesce(trim(value::varchar)) ~ '^[0-9]*.[0-9]{1,}$' then 'numeric' ELSE 'other'

	END AS value_type FROM table WHERE value::varchar ~ '[A-Za-z0-9]') AS foo GROUP BY value_type ; **Successful result example: integer required, integer found** Value Type --- integer **Unsuccessful result example: integer required, not found** Value Type --- other
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(Domain.Values)* *FROM Domain* *LEFT OUTER JOIN Source ON Domain.Value = Source.Value* *WHERE Source.Value isnull* count_of_total_records *SELECT COUNT(Domain.Values)* *FROM Domain*
Result Report Example	Tested Data Type Measure at 94% conformance.

4.7.17 Delivery Address Type Subaddress Measure

Measure Name	Delivery Address Type Subaddress Measure
Measure Description	Check for null values where the Delivery Address Type indicates an associated Complete Subaddress, and values present where the Delivery

	Address Type indicates otherwise.
Report	Logical consistency
Evaluation Procedure	Check results for inconsistencies.
Spatial Data Required	None
Pseudocode Example: Testing records	SELECT AddressID, DeliveryAddressType, CompleteSubaddressForeign Key FROM Address Collection WHERE (Delivery Address Type = 'Subaddress Included' and Complete Subaddress.Foreign Key is null) or (Delivery Address Type = 'Subaddress Excluded' and Complete Subaddress.Foreign Key is not null) ;
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(Address ID) FROM Address Collection WHERE (Delivery Address Type = 'Subaddress Included' and Complete Subaddress.Foreign Key is null) or (Delivery Address Type = 'Subaddress Excluded' and Complete Subaddress.Foreign Key is not null) ; count_of_total_records SELECT COUNT(Address ID) FROM Address Collection ;
Result Report Example	Tested Delivery Address Type Subaddress Measure at 92% conformance.

4.7.18 Duplicate Street Name Measure

Measure Name	Duplicate Street Name Measure
Measure Description	In many Address Reference Systems distantly disconnected street segments with the same names constitute an anomaly. This query returns TranSegId 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. The query may be customized with a local distance-based test or the results may be examined individually by Complete Street Name to create a final list of duplicate street names.
Report	Logical Consistency
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.
Spatial Data Required	Street centerlines, Address Reference System Extent and nodes and StreetsNodes as described in About Nodes For Quality Control.
Pseudocode Example: Testing records	-- -- The following function and the query using it, as written, assume that there is an integer primary key -- (CompleteStreetNameID) for Complete Street Name values. -- **Function** create or replace function too_many_ends(integer) returns boolean as $BODY$ declare stid alias for $1; chk_dup boolean; begin select into chk_dup case when count(bim.CompleteStreetNameID) > 2 then TRUER% else FALSE end as "check_for_duplicate_names" from (select distinct bar.nodesfk, a.CompleteStreetNameID, a.TranSegId from

```
StreetsNodes a,
TranSegCollection b,
Address Reference System Extent c,
( select
foo.nodesfk
  from
( select
nodesfk
  from
StreetsNodes
  where
CompleteStreetNameID = stid
) as foo
  group by
foo.nodesfk
  having
count( nodesfk ) = 1
) as bar
where
a.nodesfk = bar.nodesfk
and
CompleteStreetNameID = stid
and
a.TranSegId = b.TranSegId
and
( c.AddressReferenceSystemName = 'This Jurisdiction'
  and
  intersects( b.Geometry, c.Geometry )
)
) as bim
group by
bim.CompleteStreetNameID
;

return chk_dup;
end
$BODY$
language 'plpgsql';
```

Query

```
--
-- This query is likely to require considerable hardware resources as written.
-- Depending on your individual situation, it may be better to run it
```

	individually by Complete Street Name -- To do this, add a " **and Complete Street Name = 'Street Name'** " qualifier to the outer where clause. -- -- select distinct a.CompleteStreetNameID, a.TranSegId from StreetsNodes a, TranSegCollection b, (select foo.nodesfk from (select nodesfk from StreetsNodes where too_many_ends(CompleteStreetNameID) = TRUE) as foo group by foo.nodesfk having count(nodesfk) = 1) as bar where a.nodesfk = bar.nodesfk and a.TranSegId = b.geodb_oid ;
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(distinct a.TranSegId)* *from* *StreetsNodes a,* *TranSegCollection b,* *(* *select* *foo.nodesfk*

	from *(select* *nodesfk* *from* *StreetsNodes* *where* *too_many_ends(CompleteStreetNameID) = TRUE* *) as foo* *group by* *foo.nodesfk* *having* *count(nodesfk) = 1* *) as bar* *where* *a.nodesfk = bar.nodesfk* *and* *a.TranSegId = b.geodb_oid* *count_of_total_records* *_SELECT COUNT(TranSegId) FROM TranSegCollection*
Result Report Example	Tested Duplicate Street Name Measure at 95% conformance.

4.7.19 Element Sequence Number Measure

Measure Name	Element Sequence Number Measure
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.
Report	Attribute (Thematic) Accuracy
Evaluation Procedure	Examine Element Sequence Number values for sequences identified by the query.
Spatial Data Required	None
Pseudocode Example: Testing records	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 specific to PostgreSQL, and similar to the **nextval** used in Oracle. The SQL standard uses **NEXT VALUE FOR**. **Function**

```
create function test_element_sequence_numbers( int ) returns int as
$BODY$
declare
subaddr_id alias for $1;
bum_seq int;
begin
create temporary sequence temp_seq;

select into anomaly_seq
foo.CompleteSubaddressFk
from
( select
CompleteSubaddressFk,
nextval( 'temp_seq' ) as TestSequenceNumber,
Element Sequence Number
 from
CompleteSubaddressComponents
 where
CompleteSubaddressFk = subaddr_id
) as foo
where
foo.ElementSequenceNumber != TestSequenceNumber
;

drop sequence test_seq;

return ( anomaly_seq );

end
$BODY$
language 'plpgsql';
```

Query

```
select
test_element_sequence_numbers( CompleteSubaddressFk ),
Element Sequence Number
from
CompleteSubaddressComponents
where
test_element_sequence_numbers( CompleteSubaddressFk ) is not null
order by
test_element_sequence_numbers( CompleteSubaddressFk ),
Element Sequence Number
```

	;
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(distinct CompleteSubaddressFk)* *from* *CompleteSubaddressComponents* *where* *test_element_sequence_numbers(CompleteSubaddressFk) is not null* *order by* *test_element_sequence_numbers(CompleteSubaddressFk),* Element Sequence Number ; count_of_total_records SELECT COUNT(*) FROM Complete Subaddress ;
Result Report Example	Tested Element Sequence Number Measure at 93% conformance.

3006

3007 ### 4.7.20 Future Date Measure

Measure Name	Future Date Measure
Measure Description	This measure produces a list of dates that are in the future.
Report	Temporal Accuracy
Evaluation Procedure	Check dates.
Spatial Data Required	None
Pseudocode Example: Testing records	**Query** SELECT AddressID, Address Start Date, Address End Date FROM Address Collection WHERE

	Address Start Date > now or Address End Date > now **Result Without Anomalies** Address ID | Address Start Date | Address End Date ----------+----------+---------- **Anomalies** Address ID | Address Start Date | Address End Date ----------+----------+---------- ID 1 | Start Date 1 | End Date 1 ID 2 | Start Date 2 | End Date 2 ID 3 | Start Date 3 | End Date 3 ....
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(Address ID)* *FROM* *Address Collection* *WHERE* *Address Start Date > now() or Address End Date > now()* *count_of_total_records* *_SELECT COUNT(Address ID)* *FROM Address Collection*
Result Report Example	Tested Future Date Measure at 87% conformance

4.7.21 Intersection Validity Measure

Measure Name	Intersection Validity Measure
Measure Description	Check intersection addresses for streets that do not intersect in geometry.
Report	Logical Consistency
Evaluation Procedure	Check for the intersection of the geometry
Spatial Data Required	street centerline

Pseudocode Example: Preparing data	Prepare intersection geometry as described in About Nodes For Quality Control. 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. This simple example uses PostgreSQL. 1. Create a staging table with a primary key (ID) and a field for the intersection strings. It will look something like this: create table intersection_address (id serial primary key, intersection_address varchar(255)); 2. Fill the table with your strings. Let the primary key increment automatically to create the intersection indentifiers. **id | intersection_address** 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 3. 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 intersection_parsed (id serial primary key, intersection_addressfk integer references intersection_address, completestreetname varchar(80)); 4. Fill the new table insert into intersection_parsed(intersection_addressfk, completestreetname) select id, trim(both regexp_split_to_table(intersection_address , '.| and | && | &

	| y ')) from intersection_address ; 5. Check your results. They should look something like this: id intersection_addressfk 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 (17 rows)
Pseudocode Example: Testing records	-- -- The example is shown for a single intersection -- Methods of repetition are left to the user -- **Query 1: List the streets at the intersection** select Complete Street Name from intersection_parsed where intersection_addressfk = 2559 ; **Result:** Complete Street Name Elm Street Southwest Oak Road Northwest Oak Road Southwest **Query 2: Identify the corresponding intersections in the geometry** select a.nodesfk, a.CompleteStreetName

```
from
StreetsNodes a,
( select
a.nodesfk
from
( select nodesfk from StreetsNodes where Complete Street Name =
'Elm Street Southwest' ) as a,
( select nodesfk from StreetsNodes where Complete Street Name =
'Oak Road Northwest' ) as b,
( select nodesfk from StreetsNodes where Complete Street Name =
'Oak Road Southwest' ) as c
where
a.nodesfk = b.nodesfk
and
a.nodesfk = c.nodesfk
) as foo
where

foo.nodesfk = a.nodesfk
order by
Complete Street Name
;
```

Result:

Nodesfk | CompleteStreetName

31617 | Elm Street Southwest
31617 | Oak Road Northwest
31617 | Oak Road Southwest

Query 3: Test the Intersection Address

```
select
foo.CompleteStreetName
from
( select
Complete Street Name
from
intersection_parsed
where
intersection_addressfk = 2559
) as foo
left join
select
a.nodesfk,
a.CompleteStreetName
from
```

	StreetsNodes a, (select a.nodesfk from (select nodesfk from StreetsNodes where Complete Street Name = 'Elm Street Southwest') as a, (select nodesfk from StreetsNodes where Complete Street Name = 'Oak Road Northwest') as b, (select nodesfk from StreetsNodes where Complete Street Name = 'Oak Road Southwest') as c where a.nodesfk = b.nodesfk and a.nodesfk = c.nodesfk) as bar on foo.CompleteStreetName = bar.CompleteStreetName where bar.CompleteStreetName is null ; **Result Without Anomalies** Intersection Validity Measure --- **Anomalies** Intersection Validity Measure --- Complete Street Name 1 Complete Street Name 2 Complete Street Name 3 ....
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(foo.CompleteStreetName)* *from* *(select* Complete Street Name *from* *intersection_parsed*

	where *intersection_addressfk = 2559* *) as foo* *left join* *select* *a.nodesfk,* *a.CompleteStreetName* *from* *StreetsNodes a,* *(select* *a.nodesfk* *from* *(select nodesfk from StreetsNodes where* Complete Street Name *= 'Elm Street Southwest') as a,* *(select nodesfk from StreetsNodes where* Complete Street Name *= 'Oak Road Northwest') as b,* *(select nodesfk from StreetsNodes where* Complete Street Name *= 'Oak Road Southwest') as c* *where* *a.nodesfk = b.nodesfk* *and* *a.nodesfk = c.nodesfk* *) as bar* *on foo.CompleteStreetName = bar.CompleteStreetName* *where* *bar.CompleteStreetName is null* *;* *count_of_total_records* *_SELECT COUNT(Intersection Address)* *FROM Address Collection*
Result Report Example	Tested Intersection Validity Measure at 96% conformance.

4.7.22 Left Right Odd Even Parity Measure

Measure Name	Left Right Odd Even Parity Measure
Measure Description	Test association of odd and even values in each Block Face Range with the left and right side of the thoroughfare.
Report	Logical Consistency
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

	parity with the right or left side of the street when traveling away from the address axis.
Spatial Data Required	Street centerline and spatially derived left and right attributes for each address. Depending on local rules, it may be necessary to derive the overall cardinal direction of the road as well. For example, a local parity rule may require odds to the north and west. In practice this translates to odd numbers to the left where roads run east-west, with the lowest address numbers on the west end, increasing to the east, and so on. Pseudocode listed below describes evaluating parity once the spatial information is in place. Establishing the left-right and directional attributes varies by system.
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, however, 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 range values where you see -- [TwoNumberAddressRange.Low | Four Number Address Range.Side.Low] -- SELECT Address ID FROM Address Collection WHERE Complete Address Number BETWEEN [TwoNumberAddressRange.Low | Four Number Address Range.Side.Low] AND [Two Number Address Range.High | Four Number Address Range.Side.High] AND Delivery Address. Complete Street Name = [TwoNumberAddressRange | Four Number Address Range]. Complete Street Name AND ((Address Number Parity = 'odd' AND

	Address Number Parity.localRule = 'odd addresses on right' AND ((AddressLeftRightMeasure = 'right' AND (Address Range Directionality = 'with' OR AddressRangeDirectionality = 'against-with')) OR (AddressLeftRightMeasure = 'left' AND (Address Range Directionality = 'against' OR AddressRangeDirectionality = 'against-with')))) OR (Address Number Parity = 'even' AND Address Number Parity.localRule = 'even addresses on right' AND ((AddressLeftRightMeasure = 'right' AND (Address Range Directionality = 'with' OR AddressRangeDirectionality = 'against-with')) OR (AddressLeftRightMeasure = 'left' AND (Address Range Directionality = 'against' OR AddressRangeDirectionality = 'against-with')))) OR (Address Number Parity = 'odd' AND Address Number Parity.localRule = 'odd addresses on left'

AND
((AddressLeftRightMeasure = 'left'
AND
(Address Range Directionality = 'with'
OR
AddressRangeDirectionality = 'with-against'
)
)
OR
(AddressLeftRightMeasure = 'right'
AND
(Address Range Directionality = 'against'
OR
AddressRangeDirectionality = 'with-against'
)
))
) OR
(Address Number Parity = 'even'
AND
Address Number Parity.localRule = 'even addresses on left'
AND
((AddressLeftRightMeasure = 'left'
AND
(Address Range Directionality = 'with'
OR
AddressRangeDirectionality = 'with-against'
)
)
OR
(AddressLeftRightMeasure = 'right'
AND
(Address Range Directionality = 'against'
OR
AddressRangeDirectionality = 'with-against'
)
))
)
)

Result Without Anomalies

Address ID

	Anomalies Address ID --------- 37 52 96 ...
Pseudocode Example: Testing the Conformance of a Data Set	**Function** See Perc Conforming for the sample query. **Function Parameters** count_of_nonconforming_records -- The query below is identical for features using either Two Number Address Range or Four Number Address Range. -- One range type must, however, 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 range values where you see -- [TwoNumberAddressRange.Low | Four Number Address Range.Side.Low] -- *SELECT COUNT(* Address ID *)* *FROM Address Collection* *WHERE* Complete Address Number *BETWEEN* *[TwoNumberAddressRange.Low* | Four Number Address Range*.Side.Low] AND [* Two Number Address Range*.High* | Four Number Address Range*.Side.High]* *AND* Delivery Address. Complete Street Name = *[TwoNumberAddressRange* | Four Number Address Range *]*. Complete Street Name *AND* *(* *(* Address Number Parity *= 'odd'* *AND* Address Number Parity*.localRule = 'odd addresses on right'* *AND*

	((AddressLeftRightMeasure = 'right' *AND* *(Address Range Directionality = 'with'* *OR* *AddressRangeDirectionality = 'against-with'* *)* *)* *OR* *(AddressLeftRightMeasure = 'left'* *AND* *(Address Range Directionality = 'against'* *OR* *AddressRangeDirectionality = 'against-with'* *)* *))* *)* *OR* *(* *Address Number Parity = 'even'* *AND* *Address Number Parity.localRule = 'even addresses on right'* *AND* *((AddressLeftRightMeasure = 'right'* *AND* *(Address Range Directionality = 'with'* *OR* *AddressRangeDirectionality = 'against-with'* *)* *)* *OR* *(AddressLeftRightMeasure = 'left'* *AND* *(Address Range Directionality = 'against'* *OR* *AddressRangeDirectionality = 'against-with'* *)* *))* *)* *OR* *(Address Number Parity = 'odd'* *AND* *Address Number Parity.localRule = 'odd addresses on left'* *AND* *((AddressLeftRightMeasure = 'left'*

	AND *(Address Range Directionality = 'with'* *OR* *AddressRangeDirectionality = 'with-against'* *)* *)* *OR* *(AddressLeftRightMeasure = 'right'* *AND* *(Address Range Directionality = 'against'* *OR* *AddressRangeDirectionality = 'with-against'* *)* *))* *) OR* *(Address Number Parity = 'even'* *AND* *Address Number Parity.localRule = 'even addresses on left'* *AND* *((AddressLeftRightMeasure = 'left'* *AND* *(Address Range Directionality = 'with'* *OR* *AddressRangeDirectionality = 'with-against'* *)* *)* *OR* *(AddressLeftRightMeasure = 'right'* *AND* *(Address Range Directionality = 'against'* *OR* *AddressRangeDirectionality = 'with-against'* *)* *))* *)* *)* count_of_total_records *SELECT COUNT(*)* *FROM Address Collection*
Example: Determining left-right attributes	See Address Left Right Measure
Example:	See attached query --

Determining road direction	-- The query below describes finding the overall direction of a road. -- Like the left-right query, it describes finding the directional status -- of a single road within a jurisdiction. The query is run in a loop -- to capture the status of each road. The query itself is complicated, -- so for this purpose the loop is not shown. Each subquery is commented. -- -- Also like the left-right query, the query is complicated by a -- relationship between the primary address and a separately recorded -- access point. In this rural area the access point is essential to -- emergency services. -- -- As presented, the query determines direction for a street name with -- a foreign key of 23. -- -- The query requires: -- * a street centerline layer -- * the foreign key for the street name (CompleteStreetNameID) -- * the foreign key for the segment along which the lowest -- address number is assigned (from_segfk) -- * the foreign key for the segment along which the highest -- address number is assigned (to_segfk) -- -- The query assumes that the street centerline segments are consistently -- named, and that the street names are stored in a table, with a foreign -- key in the street centerline spatial data table. -- -- Further, the query assumes that the street centerline segments are -- directionally consistent with the addresses. -- ---------- -- -- Store the results for use in the parity check -- insert into CompleteStreetNameDirections(CompleteStreetNameID, from_addressfk, from_segfk, to_addressfk, to_segfk, from_address, from_x,

```
from_y,
to_address,
to_x,
to_y,
degrees,
direction
)
--
-- Begin with the foreign key for the street name for which you want
-- to determine the road direction. In this case it's 23.
--
-- Select foreign keys for the location of the lowest and highest
-- address number for the given street name foreign key, and the
-- foreign keys for the street segments along which those addresses are
-- located.
--
-- Select the start point for the segment corresponding to the lowest
-- address point, the end point for the segment corresponding to the
-- highest address point and figure the azimuth. Convert to degrees and
-- calculate the cardinal direction from the result.
--
select
23::integer as CompleteStreetNameID,
bar.pt_id as from_addressfk,
bar.seg_id as from_segfk,
bim.pt_id as to_addressfk,
bim.seg_id as to_segfk,
bar.address as from_address,
round( ( x( bar.geom ) )::numeric, 2 ) as from_x,
round( ( y( bar.geom ) )::numeric, 2 ) as from_y,
bim.address as to_address,
round( ( x( bim.geom ) )::numeric, 2 ) as to_x,
round( ( y( bim.geom ) )::numeric, 2 ) as to_y,
degrees( azimuth( bar.geom, bim.geom ) ),
case
when
( degrees( azimuth( bar.geom, bim.geom ) ) between 0 and 44.99
or
degrees( azimuth( bar.geom, bim.geom ) ) between 134.99 and 224.99
or
degrees( azimuth( bar.geom, bim.geom ) ) between 314.99 and 359.99
)
and
y( bar.geom ) < y( bim.geom )
```

```
then
'north'
when
( degrees( azimuth( bar.geom, bim.geom ) ) between 0 and 44.99
or
degrees( azimuth( bar.geom, bim.geom ) ) between 134.99 and 224.99
or
degrees( azimuth( bar.geom, bim.geom ) ) between 314.99 and 359.99
)
and
y( bar.geom ) > y( bim.geom )
then
'south'
when
( degrees( azimuth( bar.geom, bim.geom ) ) between 45 and 135
or
degrees( azimuth( bar.geom, bim.geom ) ) between 225 and 315
)
and
x( bar.geom ) < x( bim.geom )
then
'west'
when
( degrees( azimuth( bar.geom, bim.geom ) ) between 45 and 135
or
degrees( azimuth( bar.geom, bim.geom ) ) between 225 and 315
)
and
x( bar.geom ) > x( bim.geom )
then
'east'
end as "direction"
from
--
-- Select the lowest address number for the selected street,
-- the foreign key for the primary address point, the
-- foreign key for the street segment, and the geometry
-- for the start point for the street segment.
--
( select
a.address,
a.geodb_oid as pt_id,
d.geodb_oid as seg_id,
startpoint( d.geom ) as geom
```

```
from
primary_address a,
primary_address_has_access b,
axess c,
street_segments d,
--
-- Select the minimum address number for the
-- specified street name
--
( select
min( address )
from
primary_address
where
streetnames_id = 23
) as foo
where
a.streetnames_id = 23
and
a.address = foo.min
and
a.uniqueid = b.primaryaddressuniqueid
and
b.accessuniqueid = c.uniqueid
order by
distance( c.geom, d.geom )
limit 1
) as bar,
--
-- Select the highest address number for the selected street,
-- the foreign key for the primary address point, the
-- foreign key for the street segment, and the geometry
-- for the end point for the street segment.
--
( select
a.address,
a.geodb_oid as pt_id,
d.geodb_oid as seg_id,
endpoint( d.geom ) as geom
from
primary_address a,
primary_address_has_access b,
axess c,
street_segments d,
```

	-- -- Select the maximum address number for the -- specified street name -- (select max(address) from primary_address where streetnames_id = 23) as foo where a.streetnames_id = 23 and a.address = foo.max and a.uniqueid = b.primaryaddressuniqueid and b.accessuniqueid = c.uniqueid order by distance(c.geom, d.geom) limit 1) as bim ;
Result Report Example	Tested Left Right Odd Even Parity Measure at 85% conformance.

3010

3011 ### 4.7.23 Location Description Field Check Measure

Measure Name	Location Description Field Check Measure
Measure Description	Field check the location description
Report	Positional Accuracy/Lineage
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 can reinforce the lineage of the address.
Spatial Data	None

Required	

4.7.24 Low High Address Sequence Measure

Measure Name	Low High Address Sequence Measure
Measure Description	Confirm that the value of the low address is less than or equal to the high address.
Report	Logical consistency
Evaluation Procedure	Check the values for each range.
Spatial Data Required	None
Pseudocode Example: Testing records	-- Two Number Address Range test -- SELECT Two Number Address Range FROM Address Collection WHERE Two Number Address Range.Low < Two Number Address Range.High -- -- -- Four Number Address Range test -- Note that each side must be tested separately. -- Four Number Address Range.Side means either right or left. -- The same side should be used consistently throughout the query. -- SELECT Four Number Address Range.Side FROM Address Collection WHERE Four Number Address Range.Side.Low < Four Number Address Range.Side.High
Pseudocode Example: Testing the Conformance of a Data Set	**Function** See Perc Conforming for the sample query. **Function Parameters** count_of_nonconforming_records -- -- Two Number Address Range test -- *SELECT* Two Number Address Range *FROM Address Collection* *WHERE* Two Number Address Range.*Low* < Two Number Address Range.*High*

	-- -- -- Four Number Address Range test -- Note that each side must be tested separately. -- Four Number Address Range.Side means either right or left. -- The same side should be used consistently throughout the query. -- *SELECT* Four Number Address Range.*Side* *FROM Address Collection* *WHERE* Four Number Address Range.*Side.Low* < Four Number Address Range.*Side.High* count_of_total_records -- -- Two Number Address Range test -- *SELECT COUNT(* Two Number Address Range *)* *FROM Address Collection* -- -- -- Four Number Address Range test -- Note that each side must be tested separately. -- Four Number Address Range.Side means either right or left. -- The same side should be used consistently throughout the query. -- *SELECT COUNT(* Four Number Address Range.*Side)* *FROM Address Collection*
Result Report Example	Test Low High Address Sequence Measure at 91% conformance.

4.7.25 Official Status Address Authority Consistency Measure

Measure Name	Official Status Address Authority Consistency Measure
Measure Description	Test logical agreement of the Official Status with the Address Authority.
Report	Logical Consistency
Evaluation Procedure	Use Simple Elements checks to validate Official Status entries against the domain. Check logical agreement between the status values and the Address Authority.
Spatial Data Required	None

Pseudocode Example: Testing records	**Description** This query produces a list with Official Status attributes requiring an Address Authority that lack a corresponding entry. **Query** SELECT AddressID FROM Address Collection WHERE (Address Authority is null AND (Official Status = 'Official' OR Official Status = 'Official Alternate or Alias' OR Official Status = 'Official Renaming Action of the Address Authority' OR Official Status = 'Alternates Established by an Address Authority')) OR (Address Authority is not null AND (Official Status = 'Unofficial Alternate or Alias' OR Official Status = 'Alternate Names Established by Colloquial Use in a Community' OR Official Status = 'Unofficial Alternate Names Frequently Encountered' OR Official Status = 'Unofficial Alternate Names In Use by an Agency or Entity' OR Official Status = 'Posted or Vanity Address' OR Official Status = 'Verified Invalid') **Result Without Anomalies** addressID

	--------- **Anomalies** addressID --------- 98 387 598 ....
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 Address Collection* *WHERE* *(* *Address Authority is null* *AND (* Official Status = *'Official'* *OR* Official Status = *'Official Alternate or Alias'* *OR* Official Status = *'Official Renaming Action of the* Address Authority*'* *OR* Official Status = *'Alternates Established by an* Address Authority*'* *)* *)* *OR (* *Address Authority is not null* *AND (* Official Status = *'Unofficial Alternate or Alias'* *OR* Official Status = *'Alternate Names Established by Colloquial Use in a Community'* *OR* Official Status = *'Unofficial Alternate Names Frequently Encountered'* *OR* Official Status = *'Unofficial Alternate Names In Use by an Agency or Entity'* *OR* Official Status = *'Posted or Vanity Address'*

	OR Official Status = *'Verified Invalid'* *) _* *count_of_total_records* *_SELECT COUNT(*)* *FROM Address Collection*
Result Report Example	Tested Official Status Address Authority Consistency Measure at 81% conformance.

4.7.26 Overlapping Ranges Measure

Measure Name	Overlapping Ranges Measure
Measure Description	Check for overlapping ranges with the same Complete Street Name. Two Number Address Ranges and Four Number Address Ranges are most often used as part of the Address Reference System. Overlapping ranges create multiple potential locations for individual addresses, creating ambiguity during the assignment process. Anomalies discovered by this measure should, therefore, be resolved.
Report	Logical Consistency
Evaluation Procedure	Select block ranges or block face ranges with the same Complete Street Name.
Spatial Data Required	Street centerlines with Two Number Address Range or Four Number Address Range values. Nodes and StreetsNodes as described in About Nodes For Quality Control.
Pseudocode Example: Testing records	**Function** create or replace function overlapping_range_measure(integer, integer, integer, integer) returns boolean as $BODY$ declare low1 alias for $1; high1 alias for $2; low2 alias for $3; high2 alias for $4; overlap boolean; begin select into overlap case when low1 between low2 and high2

```
or
high1 between low2 and high2
then
TRUE
else
FALSE
end as "overlap_case";

return overlap;
end
$BODY$
;
```

Query

```
-- The query below is identical for features using either Two Number Address Range or Four Number Address Range.
-- One range type must, however, 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 range values where you see
-- [TwoNumberAddressRange.Low | Four Number Address Range.Side.Low ]
--

--
-- Use the overlapping_range_measure function to identify adjoining segments with overlapping ranges
--
select
a.NodePointGeometryIdentifier,
a.CompleteStreetName,
a.LinestringGeometryIdentifier,
[b.TwoNumberAddressRange.Low |
b.FourNumberAddressRange.Side.Low ]
[b.TwoNumberAddressRange.High |
b.FourNumberAddressRange.Side.High]
c.LinestringGeometryIdentifier
[d.TwoNumberAddressRange.Low |
d.FourNumberAddressRange.Side.Low ]
[d.TwoNumberAddressRange.High |
```

```
d.FourNumberAddressRange.Side.High]
from
StreetsNodes a,
StreetCenterlines b,
StreetsNodes c,
StreetCenterlines d,
--
-- Select only those nodes where two segments with the given street name meet
--
(
select
foo.NodePointGeometryIdentifier
from
--
-- Select all the nodes on either end of segments with a given street name
--
( select
NodePointGeometryIdentifier
from
StreetsNodes
where
a.CompleteStreetName = 'Elm Street'
) as foo
group by
foo.NodePointGeometryIdentifier
having count( foo.NodePointGeometryIdentifier ) > 1
) as bar
where
a.NodePointGeometryIdentifier = bar.NodePointGeometryIdentifier
and
a.CompleteStreetName = 'Elm Street'
and
a.LinestringGeometryIdentifier = b.LinestringGeometryIdentifier
and
a.NodePointGeometryIdentifier = c.NodePointGeometryIdentifier
and
c.CompleteStreetName = 'Elm Street'
and
c.LinestringGeometryIdentifier = d.LinestringGeometryIdentifier
and
b.LinestringGeometryIdentifier = d.LinestringGeometryIdentifier
and
```

b.Geometry.AddressDirectionality = d.Geometry.AddressDirectionality
and
[b.TwoNumberAddressRange.Low |
b.FourNumberAddressRange.Side.Low]
<
[d.TwoNumberAddressRange.Low |
d.FourNumberAddressRange.Side.Low]
and
overlapping_range_measure(
[b.TwoNumberAddressRange.Low |
b.FourNumberAddressRange.Side.Low],
[b.TwoNumberAddressRange.High |
b.FourNumberAddressRange.Side.High],
[d.TwoNumberAddressRange.Low |
d.FourNumberAddressRange.Side.Low],
[d.TwoNumberAddressRange.High |
d.FourNumberAddressRange.Side.High]
) = TRUE
;

Result Without Anomalies
a.NodePointGeometryIdentifier
a.CompleteStreetName
a.LinestringGeometryIdentifier
[b.TwoNumberAddressRange.Low |
b.FourNumberAddressRange.Side.Low]
[b.TwoNumberAddressRange.High |
b.FourNumberAddressRange.Side.High]
c.LinestringGeometryIdentifier
[d.TwoNumberAddressRange.Low |
d.FourNumberAddressRange.Side.Low]
[d.TwoNumberAddressRange.High |
d.FourNumberAddressRange.Side.High]

Anomalies
a.NodePointGeometryIdentifier
a.CompleteStreetName
a.LinestringGeometryIdentifier
[b.TwoNumberAddressRange.Low |
b.FourNumberAddressRange.Side.Low]
[b.TwoNumberAddressRange.High |
b.FourNumberAddressRange.Side.High]
c.LinestringGeometryIdentifier

	[d.TwoNumberAddressRange.Low | d.FourNumberAddressRange.Side.Low] [d.TwoNumberAddressRange.High | d.FourNumberAddressRange.Side.High] 58104 Elm Street 34004 1 1309 29652 5 1309
Pseudocode Example: Testing the Conformance of a Data Set	Note that the function below calculates the percentage of conformance on the total set of records. The method of repetitively running the test is left to the user. **Function** See Perc Conforming for the sample query. **Function Parameters** count_of_nonconforming_records -- This query counts the non-conforming records by street name. It should be run for each street name in the data set, and all of the -- results summed, to get the total number of non-conforming records. *SELECT* *COUNT(a.*)* *from* *StreetsNodes a,* *StreetCenterlines b,* *StreetsNodes c,* *StreetCenterlines d,* -- *-- Select only those nodes where two segments with the given street name meet* -- *(* *select* *foo.NodePointGeometryIdentifier* *from* -- *-- Select all the nodes on either end of segments with a given street name* -- *(select*

	NodePointGeometryIdentifier *from* *StreetsNodes* *where* *a.CompleteStreetName = 'Elm Street'* *) as foo* *group by* *foo.NodePointGeometryIdentifier* *having count(foo.NodePointGeometryIdentifier) > 1* *) as bar* *where* *a.NodePointGeometryIdentifier = bar.NodePointGeometryIdentifier* *and* *a.CompleteStreetName = 'Elm Street'* *and* *a.LinestringGeometryIdentifier = b.LinestringGeometryIdentifier* *and* *a.NodePointGeometryIdentifier = c.NodePointGeometryIdentifier* *and* *c.CompleteStreetName = 'Elm Street'* *and* *c.LinestringGeometryIdentifier = d.LinestringGeometryIdentifier* *and* *b.LinestringGeometryIdentifier = d.LinestringGeometryIdentifier* *and* *b.Geometry.AddressDirectionality = d.Geometry.AddressDirectionality* *and* *[b.TwoNumberAddressRange.Low |* *b.FourNumberAddressRange.Side.Low]* *<* *[d.TwoNumberAddressRange.Low |* *d.FourNumberAddressRange.Side.Low]* *and* **overlapping_range_measure(** **[b.TwoNumberAddressRange.Low |** **b.FourNumberAddressRange.Side.Low],** **[b.TwoNumberAddressRange.High |** **b.FourNumberAddressRange.Side.High],** **[d.TwoNumberAddressRange.Low |** **d.FourNumberAddressRange.Side.Low],** **[d.TwoNumberAddressRange.High |** **d.FourNumberAddressRange.Side.High]** **) = TRUE_**

	count_of_total_records *_SELECT COUNT([* Two Number Address Range | Four Number Address Range*.Side]) FROM Address Collection*
Result Report Example	Tested Overlapping Ranges Measure at 73% conformance.

4.7.27 Pattern Sequence Measure

Measure Name	Pattern Sequence Measure
Measure Description	Test 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 Complex Element Sequence Number Measure
Report	Logical Consistency
Evaluation Procedure	Check each complex element value against the pattern defining it.
Spatial Data Required	None

Pseudocode Example: Testing records	**Query**
	
	SELECT
	  Complex Element As disagreeWithSequence
	FROM
	  Address Collection
	WHERE
	  (Simple Element || Simple Element ...) != Complex Element Pattern
	
	Result Without Anomalies
	
	disagreeWithSequence
	---------- 	
	
	Anomalies
	
	disagreeWithSequence
	---------- 	
	complex element 1
	complex element 2

	complex element 3 ....
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(Complex Element)* *FROM Address Collection* *WHERE (Simple Element || Simple Element ...) != Complex Element Pattern* count_of_total_records *SELECT COUNT(Complex Element) FROM Address Collection*
Result Report Example	Tested Pattern Sequence Measure at 95% conformance.

4.7.28 Range Domain Measure

Measure Name	Range Domain Measure
Measure Description	Test each domain for agreement with source ranges.
Report	Attribute (Thematic) Accuracy
Evaluation Procedure	Validate range domain values against low and high range values.
Spatial Data Required	None

Pseudocode Example: Testing records	**Description**
	
	This query produces a list of simple elements in the address collection that do not conform to a range domain.
	
	Query
	
	SELECT
	  AddressNumber As disagreeWithSource
	FROM
	  Address Collection
	WHERE
	  NOT(Address Number BETWEEN [Two Number Address Range.Low | Four Number Address Range.Side.Low] AND [Two Number Address Range.High | Four Number Address Range.Side.High])

	Result Without Anomalies disagreeWithSource ---------- **Anomalies** disagreeWithSource ---------- simple element 1 simple element 2 simple element 3 ....
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(* Address Number *)* *FROM* *Address Collection* *WHERE* *NOT(* Address Number *BETWEEN [* Two Number Address Range*.Low* | Four Number Address Range*.Side.Low] AND [* Two Number Address Range*.High* | Four Number Address Range*.Side.High])* *count_of_total_records* *_SELECT COUNT(* Address Number *)* *FROM Address Collection*
Result Report Example	Tested Range Domain Measure at 86% conformance.

4.7.29 Related Not Null Measure

Measure Name	Related Not Null Measure
Measure Description	Check to make sure a data element referenced through a foreign key exists.
Report	Logical consistency
Evaluation Procedure	Check for references that don't actually exist.

Spatial Data Required	None
Pseudocode Example: Testing records	SELECT AddressID, Related Data Identifier FROM Address Collection LEFT JOIN Related Data on Address Collection.Related Data Identifier = Related Data.Identifier WHERE Related Data.Identifier is null ;
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(Address ID) FROM Address Collection LEFT JOIN Related Data on Address Collection.Related Data Identifier = Related Data.Identifier WHERE Related Data.Identifier is null ; count_of_total_records SELECT COUNT(Address ID) FROM Address Collection ;
Result Report Example	Tested Related Not Null Measure at 83% conformance.

4.7.30 Related Element Uniqueness Measure

Measure Name	Related Element Uniqueness Measure
Measure Description	Check the uniqueness of the values related to a given element
Report	Attribute (Thematic) Accuracy

Evaluation Procedure	Check any duplicate values found.
Spatial Data Required	None

Pseudocode Example: Testing Records	**Function**
	-- The function for this example describes checking for unique Element Sequence Number values
	-- using the table structure diagrammed in Complex Element Sequence Number Measure. The same
	-- pattern can be followed for any related element.
	--
	create or replace function related_element_uniq(int) returns integer as
	$BODY$
	declare
	ele_id alias for $1;
	dup_element_seq int;
	begin
	select into dup_element_seq
	Element Sequence Number
	from
	Complete Subaddress a,
	CompleteSubaddressComponents b
	where
	a.id = ele_id
	and
	a.id = b.CompleteSubaddressFk
	group by
	Element Sequence Number
	having
	count(Element Sequence Number) > 1
	order by
	Element Sequence Number
	;
	
	return(dup_element_seq);
	end
	$BODY$
	language 'plpgsql';
	
	Query
	select
	id,
	related_element_uniq(id)

	from Complete Subaddress where related_element_uniq(id) is not null ; **Results without Anomalies** id | related_element_uniq ---------- **Results with Anomalies** id | related_element_uniq ---------- 2 | 1
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(Element)* *FROM Address Collection* *WHERE related_element_uniq(ElementID) is not null* count_of_total_records *SELECT COUNT(Element) FROM Address Collection*
Result Report Example	Tested Related Element Uniqueness Measure at 95% conformance.

4.7.31 Repeated Element Uniqueness Measure

Measure Name	Repeated Element Uniqueness Measure
Measure	Check complex elements with repeated elements for the uniqueness of those elements within the complete element.
Report	Attribute (Thematic) Accuracy
Evaluation Procedure	Check for a repeated element that appears more than once.
Spatial Data Required	None
Pseudocode	select

Example: Testing records	count(subaddress_type || ' ' || subaddress_identifier), complete_subaddress_fk, subaddress_type || ' ' || subaddress_identifier as subaddress_element from subaddress group by complete_subaddress_fk, subaddress_type || ' ' || subaddress_identifier having count(subaddress_type || ' ' || subaddress_identifier) > 1 order by complete_subaddress_fk ;
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(subaddress_type || ' ' || subaddress_identifier), complete_subaddress_fk, subaddress_type || ' ' || subaddress_identifier as subaddress_element from subaddress group by complete_subaddress_fk, subaddress_type || ' ' || subaddress_identifier having count(subaddress_type || ' ' || subaddress_identifier) > 1 order by complete_subaddress_fk ; count_of_total_records select count(subaddress_type || ' ' || subaddress_identifier), complete_subaddress_fk, subaddress_type || ' ' || subaddress_identifier as subaddress_element from subaddress ;
Result Report Example	Tested Repeated Element Uniqueness Measure at 94% conformance.

3020

4.7.32 Segment Directionality Consistency Measure

Measure Name	Segment Directionality Consistency Measure
Measure Description	Check consistency of street segment directionality. This is important to do before assigning or using Two Number Address Range or Four Number Address Range values, especially when no locations for addresses along those segments yet exist. The test checks for segments with the same street name where more than one "from" or "to" ends meet at the same node. The directionality of the inconsistent segment may be reversed, the Two Number Address Range or Four Number Address Range reversed, or the directionality noted in the database so that the data may be handled consistently.
Report	Logical Consistency
Evaluation Procedure	Examine segments where the measure indicates inconsistent directionality and take appropriate action
Spatial Data Required	Nodes and StreetsNodes as described in About Nodes For Quality Control.
Pseudocode Example: Testing Records	-- -- Pseudocode below describes checking a single Complete Street Name. Methods of repetition left to the user. -- In this example the query is repeated, listing the TranSegId from each side of a given node on separate lines. -- The result, therefore, has a single column listing anomalous TranSegId values. This simplifies the task of producing -- a SELECT DISTINCT list of TranSegId values to check, and for calculating -- the conformance of the dataset. -- select foo.id, foo.CompleteStreetName, foo.TranSegId, foo.TranSegFromTo, foo.NodesFk from ((select a.id, a.CompleteStreetName,

	b.TranSegId, b.TranSegFromTo, b.NodesFk from Complete Street Name a, StreetsNodes b, StreetsNodes c where a.CompleteStreetName = [*fill in* Complete Street Name *value*] and a.id = b.streetnames_staging_allfk and a.id = c.streetnames_staging_allfk and b.NodesFk = c.NodesFk and b.TranSegId < c.TranSegId and b.TranSegFromTo = c.TranSegFromTo) union (select a.CompleteStreetName, c.TranSegId, c.TranSegFromTo, b.NodesFk from streetnames_staging.streetnames_staging_all_streetname a, nodes.streets_nodes b, nodes.streets_nodes c where a.CompleteStreetName = [*fill in* Complete Street Name *value*] and a.CompleteStreetName = b.CompleteStreetName and a.CompleteStreetName = c.CompleteStreetName and b.NodesFk = c.NodesFk and b.TranSegId < c.TranSegId and b.TranSegFromTo = c.TranSegFromTo)

	) as foo order by foo.NodesFk, foo.TranSegId ;
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(distinct TranSegId) from SegmentDirectionalityConsistencyMeasure Result* ; count_of_total_records *SELECT COUNT(*) FROM TranSeg Collection* ;
Result Report Example	Tested Segment Directionality Consistency Measure at 55% conformance.

3022

3023 ### 4.7.33 Spatial Domain Measure

Measure Name	Spatial Domain Measure
Measure Description	Test 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.
Report	Positional Accuracy
Evaluation Procedure	Intersect the addressed spatial object with the corresponding location identified by the codeset.
Spatial Data Required	address collection geometry, spatial domain geometry
Pseudocode Example: Testing records	**Query** SELECT

	Simple Element As notWithinSpatialDomain FROM Address Collection, Spatial Domain WHERE NOT(INTERSECTS(Address Collection.Geometry, Spatial Domain.Geometry)) **Result Without Anomalies** notWithinSpatialDomain ---------- **Anomalies** notWithinSpatialDomain ---------- simple element 1 simple element 2 simple element 3 ....
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(Simple Element)* *FROM* *Address Collection* *WHERE* *NOT(INTERSECTS(Address Collection.Geometry, Spatial Domain.Geometry))* count_of_total_records *SELECT COUNT(Simple Element)* *FROM Address Collection*
Result Report Example	Tested Spatial Domain Measure at 87% conformance

4.7.34 Start End Date Order Measure

Measure Name	Start End Date Order Measure
Measure	Test the logical ordering of the start and end dates.

Report	Temporal Accuracy/Attribute (Thematic) Accuracy
Evaluation Procedure	Check the date order for all entries.
Spatial Data Needed	None

Pseudocode Example: Testing records	**Description**
	This query makes a list of the date values where the Address Start Date is after the Address End Date.
	
	Query
	
	SELECT
	  Address Start Date, Address End Date
	FROM
	  Address Collection
	WHERE
	  Address End Date is not null
	  and
	  Address Start Date > Address End Date
	
	Result Without Anomalies
	
	Address Start Date | Address End Date
	----------+----------
	
	Anomalies
	
	  Address Start Date | Address End Date
	----------+----------
	2004-03-25 | 2004-03-24
	2004-06-30 | 2004-05-30
	2005-08-14 | 2005-06-25
	... | ...

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 Address Collection* *WHERE Address Start Date >= Address End Date* count_of_total_records *SELECT COUNT(*)* *FROM Address Collection*

Result Report Example	Tested Start End Date Order Measure at 98% conformance.

4.7.35 Subaddress Component Order Measure

Measure Name	Subaddress Component Order Measure
Measure Description	Test Subaddress Elements against the component parts in the order specified by the Subaddress Component Order Measure.
Report	Attribute (Thematic) Accuracy
Evaluation Procedure	Check complex element against concatenated simple elements for anomalies.
Spatial Data Required	None
Pseudocode Example: Testing records	SELECT Subaddress Element, Subaddress Type, Subaddress Identifier, Subaddress Component Order FROM Subaddress Collection WHERE ((Subaddress Element = Subaddress Type || ' ' || Subaddress Identifier or (Subaddress Element = Subaddress Identifier and Subaddress Type is null)) and Subaddress Component Order = 2) or (Subaddress Element = Subaddress Identifier || ' ' || Subaddress Type and Subaddress Component Order = 1) ;
Pseudocode Example: Testing the	**Function** See Perc Conforming for the sample query.

Conformance of a Data Set	**Function Parameters** count_of_nonconforming_records SELECT COUNT(*) FROM Subaddress Collection WHERE ((Subaddress Element = Subaddress Type || ' ' || Subaddress Identifier or (Subaddress Element = Subaddress Identifier and Subaddress Type is null)) and Subaddress Component Order = 2) or (Subaddress Element = Subaddress Identifier || ' ' || Subaddress Type and Subaddress Component Order = 1) ; count_of_total_records SELECT COUNT(*) FROM Subaddress Collection ;
Result Report Example	Tested Subaddress Component Order Measure at 96% conformance.

4.7.36 Subaddress Element Z Level Measure

Measure Name	Subaddress Element Z Level Measure
Measure	Check the consistency of the association between Subaddress Element entries associated with a given address. Create a table called **subaddr_zlevel_anom** to hold the anomaly records.
Report	Attribute (Thematic) Accuracy
Evaluation Procedure	Check for multiple Subaddress Element entries associated with a single Address Z Level and Address ID

SQL Example: Testing records	Function
	create or replace function subaddr_zlevel() returns varchar as $$
	
	declare
	addr record;
	cnt integer;
	str varchar;
	
	begin
	
	cnt := 0;
	
	-- drop any pre-existing subaddr_zlevel_anom table
	
	drop table subaddr_zlevel_anom;
	
	FOR addr in select * from complete_subaddress LOOP
	
	IF cnt = 0 THEN
	
	-- Create the table with the first set of anomalies.
	-- Include the full set of subaddress information to
	-- make it easier to resolve the anomalies.
	
	create table subaddr_zlevel_anom as
	select
	bar.addressfk,
	a.complete_subaddress_fk,
	a.element_sequence_number,
	a.subaddress_type,
	a.subaddress_identifier,
	bar.z_level
	from
	subaddress a,
	complete_subaddress b,
	(
	-- Identify addresses where the zlevels conflict:
	-- either a given zlevel is assigned to more than one
	-- subaddress element or more than one zlevel
	-- is assigned to a given subaddress element.
	select
	foo.addressfk,
	foo.z_level
	from
	(select

```
        a.addressfk,
        b.complete_subaddress_fk,
        b.z_level
      from
        complete_subaddress a,
        subaddress b
      where
        a.addressfk = addr.addressfk
        and
        b.z_level is not null
      group by
        a.addressfk,
        b.complete_subaddress_fk,
        b.subaddress_type || ' ' || b.subaddress_identifier,
        b.z_level
      order by
        a.addressfk,
        b.complete_subaddress_fk,
        b.z_level
      ) as foo
      group by
        foo.addressfk,
         foo.z_level
      having
        count( z_level ) > 1
    ) as bar
  where
    b.addressfk = bar.addressfk
    and
    b.pkey = a.complete_subaddress_fk
    and
    a.z_level = bar.z_level
  ;

-- Insert additional anomaly records into the existing table.
-- Use the same query as above.

ELSE
  insert into subaddr_zlevel_anom
  select
    bar.addressfk,
    a.complete_subaddress_fk,
    a.element_sequence_number,
    a.subaddress_type,
    a.subaddress_identifier,
```

```
    bar.z_level
  from
    subaddress a,
    complete_subaddress b,
    (
     select
       foo.addressfk,
    foo.z_level
     from
       ( select
       a.addressfk,
       b.complete_subaddress_fk,
       b.z_level
     from
       complete_subaddress a,
       subaddress b
     where
       a.addressfk = addr.addressfk
       and
       b.z_level is not null
     group by
       a.addressfk,
       b.complete_subaddress_fk,
       b.subaddress_type || ' ' || b.subaddress_identifier,
       b.z_level
     order by
       a.addressfk,
       b.complete_subaddress_fk,
       b.z_level
    ) as foo
      group by
        foo.addressfk,
     foo.z_level
      having
        count( z_level ) > 1
    ) as bar
  where
    b.addressfk = bar.addressfk
    and
    b.pkey = a.complete_subaddress_fk
    and
    a.z_level = bar.z_level
  ;
END IF;
```

	-- keep a count of the anomaly records to report to the user.
	
	cnt = cnt + 1;
	
	END LOOP;
	
	-- report results to the user.
	
	str = cnt::varchar || ' ' || 'anomaly records in table subaddr_zlevel_anom.';
	
	return(str);
	
	end;
	
	$$ language plpgsql;
	
	Queries
	
	select subaddr_zlevel();
	select * from subaddr_zlevel_anom;

Notes	Due to the complexity of the function, it is listed in its entirety as a complete example rather than as pseudocode.

4.7.37 Tabular Domain Measure

Measure Name	Tabular Domain Measure
Measure Description	Test 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.
Report	Attribute (Thematic) Accuracy
Evaluation Procedure	Check the value of each simple element against the tabular domain by which it is constrained.
Spatial Data Required	None
Pseudocode Example: Testing records	**Query** SELECT Simple Element As disagreeWithDomain

	FROM Address Collection LEFT OUTER JOIN Domain WHERE Domain isnull ; **Result Without Anomalies** disagreeWithDomain ---------- **Result With Anomalies** disagreeWithDomain ---------- simple element 1 simple element 2 simple element 3 ....
Pseudocode Example: Testing the Conformance of a gData Set	**Function** See Perc Conforming for the query example. **Function Parameters** count_of_nonconforming_records *SELECT COUNT(Simple Element)* *FROM Address Collection* *LEFT OUTER JOIN Domain ON Simple Element.Field = Domain.Field* *WHERE Domain.Field isnull* count_of_total_records *SELECT COUNT(Simple Element) FROM Address Collection*
Result Report Example	Tested Tabular Domain Measure at 100% conformance.

4.7.38 Uniqueness Measure

Measure Name	Uniqueness Measure
Measure	Test uniqueness of a simple or complex value.
Report	Attribute (Thematic) Accuracy
Evaluation Procedure	Identify unique and duplicate values.
Spatial Data Required	None
Pseudocode Example:	**Description**

Testing records	This test returns duplicate elements, and the number of occurrences.
	
	Query
	
	SELECT
	  COUNT(Element), Element
	FROM
	  Address Collection
	GROUP BY
	  Element
	HAVING
	  COUNT(Element) > 1
	;
	
	Result Without Anomalies
	
	Count | Element
	------+-----------
	
	Anomalies
	
	Count | Element
	------+-----------
	  2 | Element 1
	  2 | Element 2
	  5 | Element 3
	  .... | ....

Pseudocode Example: Testing the Conformance of a Data Set	**Function** See Perc Conforming for the query example. **Function Parameters** count_of_nonconforming_records *SELECT COUNT(Element)* *FROM Address Collection* *WHERE Element* *IN (* *SELECT Element* *FROM Address Collection* *GROUP BY Element* *HAVING COUNT(Element) > 1* *)* count_of_total_records *SELECT COUNT(*) FROM Address Collection*

Result Report Example	Tested Uniqueness Measure at 94% conformance.

4.7.39 USNG Coordinate Spatial Measure

Measure Name	USNG Coordinate Spatial Measure
Measure Description	Test agreement between the location of the addressed object and the area described by the US National Grid Coordinate. This test derives the USNG for a point geometry and compares it to the USNG coordinate
Report	Positional accuracy
Evaluation Procedure	If the derived USNG matches the recorded USNG the comparison is successful. The coord2usng function is an example written for PostgreSQL. Code for other systems may vary. An inverse function, converting USNG to UTM coordinates, is provided for convenience in an **Addendum** section.
Spatial Data Required	Address points, USNG coordinates
Pseudocode Example: Testing records	**Function** coord2usng 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 <br /> 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 from ( length( trunc( utm_x ) ) - 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 from ( length( trunc( utm_x ) ) - 4 ) for 1 )
when ( precision = 1000 ) then
substring( utm_x from ( length( trunc( utm_x ) ) - 4 ) for 2 )
when ( precision = 100 ) then
substring( utm_x from ( length( trunc( utm_x ) ) - 4 ) for 3 )
when ( precision = 10 ) then
substring( utm_x from ( length( trunc( utm_x ) ) - 4 ) for 4 )
when ( precision = 1 ) then
substring( utm_x from ( length( trunc( utm_x ) ) - 4 ) for 5 )
end;

select into y_grid_coord
case
when ( precision = 10000 ) then
substring( utm_y from ( length( trunc( utm_y ) ) - 4 ) for 1 )
when ( precision = 1000 ) then
substring( utm_y from ( length( trunc( utm_y ) ) - 4 ) for 2 )
when ( precision = 100 ) then
substring( utm_y from ( length( trunc( utm_y ) ) - 4 ) for 3 )
when ( precision = 10 ) then
substring( utm_y from ( length( trunc( utm_y ) ) - 4 ) for 4 )
when ( precision = 1 ) then
substring( utm_y from ( length( trunc( utm_y ) ) - 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
	  Recorded USNG
	FROM
	  Address Collection
	WHERE
	  coord2usng(
	  easting,
	  northing,
	  longitude,
	  latitude,
	  precision
	  ) != Recorded USNG
	;
	Result Without Anomalies
	
	usng
	---- 	
	
	Result with Anomalies
	
	usng
	---------- 	
	18SUJ2348306479

Pseudocode Example: Testing the Conformance of a Data Set	Note that the function below calculates the percentage of conformance on the total set of records. The method of repetitively running the test is left to the user. **Function** See Perc Conforming for the query example. count_of_nonconforming_records *SELECT COUNT(result)* count_of_total_records *SELECT COUNT(usng) from Address Collection*
Result Report	Tested USNG Coordinate Spatial Measure at 96% conformance.

Example	

Addendum	**Function**
	usng2coord
	
	Note:
	This function returns a pair of coordinates at the center of the area described by the precision of the USNG grid reference.
	
	create or replace function usng2coord(varchar)
	returns varchar as '
	
	declare
	usng alias for $1;
	zone integer;
	grid_zone char(1);
	set integer;
	offset_north numeric;
	x_alpha_gsz char(1);
	y_alpha_gsz char(1);
	e_coord integer;
	n_coord integer;
	e100k integer;
	n100k integer;
	e100k_grp1 varchar[8];
	e100k_grp2 varchar[8];
	e100k_grp3 varchar[8];
	n100k_grp1 varchar[20];
	n100k_grp2 varchar[20];
	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 <br /> 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) + 1 ) for 1 );

-- parse grid zone squares
select into x_alpha_gsz substring( usng from ( length( zone) + 2 ) for
1 );
select into y_alpha_gsz substring( usng from ( length( zone ) + 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" )
<br /> 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") <br /> 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" )
<br /> 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") <br /> 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" )
<br /> 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" ) <br /> 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 ) = 2
then ( cast( substring( grid from 1 for 1 ) as integer ) ) * 10000
when length( grid ) = 4
then ( cast( substring( grid from 1 for 2 ) as integer ) ) * 1000
when length( grid ) = 6
then ( cast( substring( grid from 1 for 3 ) as integer ) ) * 100
when length( grid ) = 8
then ( cast( substring( grid from 1 for 4 ) as integer ) ) * 10
when length( grid ) = 10
then cast( substring( grid from 1 for 5 ) as integer )
end;
```

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

select into precision
case
when length( grid ) = 2
then 10000
when length( grid ) = 4
then 1000
when length( grid ) = 6
then 100
when length( grid ) = 8
then 10
when length( grid ) = 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';
```

4.7.40 XY Coordinate Completeness Measure

Measure Name	XY Coordinate Completeness Measure
Measure Description	Checks for coordinates pairs with one member missing. The query produces a list of incomplete coordinates.
Report	Logical consistency
Evaluation Procedure	Query for null values.
Spatial Data Required	Address X Coordinate, Address Y Coordinate
Pseudocode Example: Testing records	**Query** SELECT Address X Coordinate, Address Y Coordinate FROM Address Collection WHERE Address X Coordinate isnull OR Address Y Coordinate isnull **Result Without Anomalies** Address X Coordinate | Address Y Coordinate ----------+---------- **Anomalies** Address X Coordinate | Address Y Coordinate ----------+---------- x1 | x2 | | y3 .... | ....
Pseudocode 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 Address Collection* *WHERE Address X Coordinate isnull OR Address Y Coordinate isnull* count_of_total_records *SELECT COUNT(*)*

	FROM Address Collection
Result Report Example	Tested XY Coordinate Completeness Measure at 93% conformance.

4.7.41 XY Coordinate Spatial Measure

Measure Name	XY Coordinate Spatial Measure
Measure Description	Compare 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 X Coordinate, Address Y Coordinate and Address Longitude, Address Latitude. The query produces a list of coordinate values in the address collection that do not conform to a spatial domain.
Report	Positional accuracy
Evaluation Procedure	Intersect the addressed spatial object with the coordinate attributes. Note that the spatial referencing system of the addressed spatial objects must match that of the coordinate attributes.
Spatial Data Required	Address X Coordinate, Address Y Coordinate

Pseudocode Example: Testing records	**Query**
	
	SELECT
	  Address X Coordinate, Address Y Coordinate
	FROM
	  Address Collection
	WHERE
	  NOT(EQUALS(Addressed Object.Geometry, MakePoint(Address X Coordinate, Address Y Coordinate, Address Z Coordinate)))
	
	Result Without Anomalies
	
	Address X Coordinate | Address Y Coordinate
	----------+----------
	
	Anomalies
	
	Address X Coordinate | Address Y Coordinate
	----------+----------
	x1 | y1

	x2 | y2 x3 | y3 .... | ....
Pseudocode 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 Address Collection* *WHERE NOT(EQUALS(Addressed Object.Geometry, Point(Address X Coordinate, Address Y Coordinate)))* count_of_total_records *SELECT COUNT(*)* *FROM Address Collection*
Result Report Example	Tested XY Coordinate Spatial Measure at 90% conformance.

5 Address Data Exchange

5.1 Introduction

The purpose of this section is three-fold: to provide a template for the XML documents and metadata that will move addresses from place to place, to provide information on preparing address data to be packaged, and to provide information on unpackaging address data that has been received.

Historically, the data format aspect of data exchange has impeded the flow of information. By providing a single and flexible data structure for exchanging street address data, the Address Standard will simplify the implementation of data exchanges, making them more reliable and less likely to need small changes, especially over time. Local data processing systems and applications change over time and frequently data exchange programs and reports must be