IPTOINT Function

Computes an integer value for a four-octet internet protocol (IP) address. Source value must be a valid IP address or a column reference to IP addresses.

IP addresses must be in the following format:

```
aaa.bbb.ccc.ddd
```

where `aaa`, `bbb`, `ccc`, and `ddd`, are integers 0 - 255, inclusive.

> **NOTE:** IPv6 addresses are not supported.

The formula used to compute the integer equivalent of the above IP address is the following:

$(aaa * 256^3) + (bbb * 256^2) + (ccc * 256) + (ddd)$

As a result, each valid IP address has a unique integer equivalent.

Basic Usage

Numeric literal example:

```
derive type:single value: IPTOINT('1.2.3.4' ) as:'myAddress'
```

Output: Generates a column containing the integer value 16909060.

Column reference example:

```
derive type:single value: IPTOINT(IpAddr) as: 'ip_as_int'
```

Output: Generates the new `ip_as_int` column containing the value of the `IpAddr` column converted to an integer value.

Syntax and Arguments

```
derive type:single value: IPTOINT(column_ipaddr)
```

Argument	Required?	Data Type	Description
column_ipaddr	Y	string	Column name or string literal identifying the IP address to convert to an integer value

For more information on syntax standards, see *Language Documentation Syntax Notes*.

column_ipaddr

Name of the column or IP address literal whose values are used to compute the equivalent integer value.

- Missing input values generate missing results.
- Multiple columns and wildcards are not supported.

Usage Notes:

Required?	Data Type	Example Value
Yes	String literal or column reference (IP address)	`4.3.2.1`

Examples

> **Tip:** For additional examples, see *Common Tasks*.

Example - Convert IP addresses to integers

This examples illustrates how you can convert IP addresses to numeric values for purposes of comparison and sorting. This example illustrates the following functions:

- `IPTOINT` - converts an IP address to an integer value according to a formula. See *IPTOINT Function*.
- `IPFROMINT` - converts an integer value back to an IP address according to formula. See *IPFROMINT Function*.

Source:

Your dataset includes the following values for IP addresses:

IpAddr
192.0.0.1
10.10.10.10
1.2.3.4
1.2.3
http://12.13.14.15
https://16.17.18.19

Transform:

When the above data is imported, the application initially types the column as URL values, due to the presence of the `http://` and `https://` protocol identifiers. Select the IP Address data type for the column. The last three values are listed as mismatched values. You can fix the issues with the last two entries by applying the following transform, which matches on both `http://` and `https://` strings:

```
replace col:IpAddr with:'' on:`http%?://`
```

> **NOTE:** The %? Trifacta® pattern matches zero or one time on any character, which enables the matching on both variants of the protocol identifier.

Now, only the `1.2.3` value is mismatched. Perhaps you know that there is a missing zero at the end of it. To add it back, you can do the following:

```
replace col: IpAddr on: `1.2.3{end}` with: '1.2.3.0' global: true
```

All values in the column should be valid for the IP Address data type. To convert these values to their integer equivalents:

```
derive type:single value:IPTOINT(IpAddr) as:'ip_as_int'
```

You can now manipulate the data based on this numeric key. To convert the integer values back to IP addresses for checking purposes, use the following:

```
derive type:single value:IPFROMINT(ip_as_int) as:'ip_check'
```

Results:

X	ip_as_int	ip_check
192.0.0.1	3221225473	192.0.0.1
10.10.10.10	168430090	10.10.10.10
1.2.3.4	16909060	1.2.3.4
1.2.3.0	16909056	1.2.3.0
12.13.14.15	202182159	12.13.14.15
16.17.18.19	269554195	16.17.18.19