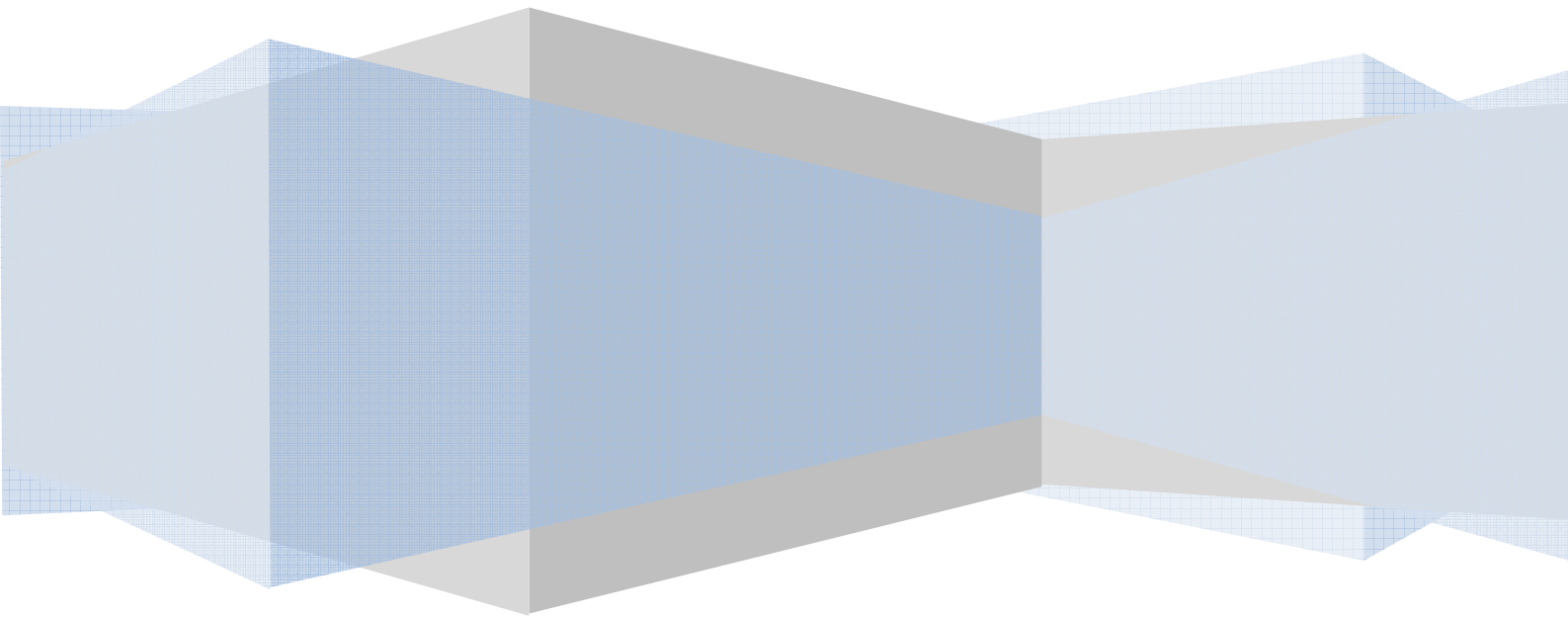


Jannie Specification - **DRAFT**

Maltego Local Transforms – January 9, 2009

RT

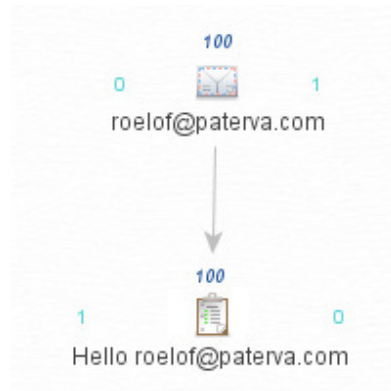


DON'T PANIC

No – really – this is mostly harmless...

“It has to be very, very easy to use”. This was the main consideration we had when we wrote the specification. At the same time we wanted something that would expose all of Maltego’s features to the developer without restrictions. We also wanted something that someone with relatively little development skills could get working within minutes – at the same time we wanted to create something that allows for growth.

Enough already...let’s look at an example. We want to create a transform that will take an email address as input and creates a Phrase entity that simply says “Hello XXX” where XXX was the input:



The PERL script for this looks like this:

```
#!/usr/bin/perl

# let's get the input entity value
my $entityValue = $ARGV[0];

# output block
my $output=<<EOT;
<MaltegoMessage>
<MaltegoTransformResponseMessage>
<Entities>
  <Entity Type='Phrase'><Value>Hello $entityValue</Value></Entity>
</Entities>
</MaltegoTransformResponseMessage>
</MaltegoMessage>
EOT
;
```

```
# send to STDOUT
print $output;
```

When running the script the output looks as follows:

```
% ./myscript.pl roelof@paterva.com

<MaltegoMessage>
<MaltegoTransformResponseMessage>
<Entities>
  <Entity Type='Phrase'><Value>Hello roelof@paterva.com</Value></Entity>
</Entities>
</MaltegoTransformResponseMessage>
</MaltegoMessage>
```

Now, that wasn't too painful was it?

Let's look at something **slightly** more complex – returning a list of entities. Code looks like this:

```
#!/usr/bin/perl
my $entityValue = $ARGV[0];

#XML header
my $header=<<<EOT;
<MaltegoMessage>
<MaltegoTransformResponseMessage>
<Entities>
EOT
;

#XML footer
my $footer=<<<EOT;
</Entities>
</MaltegoTransformResponseMessage>
</MaltegoMessage>
EOT
;

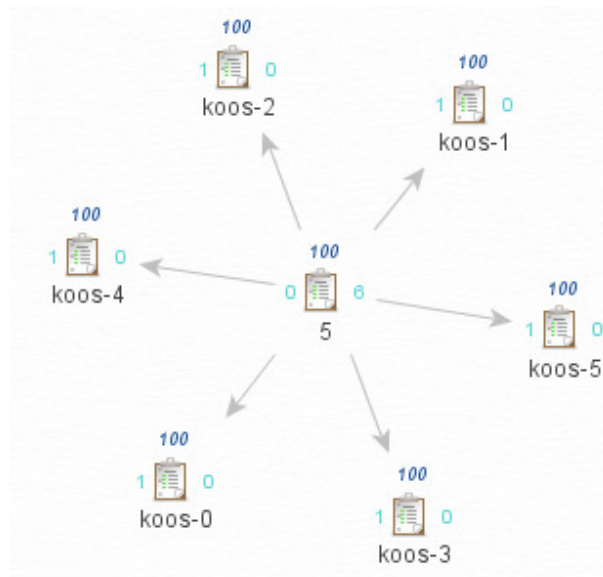
print $header;
for (0..$entityValue){
  print "<Entity Type='Phrase'><Value>koos-$_</Value></Entity>\n";
}
print $footer;
```

The output when running this:

```
$ ./koos.pl 5
```

```
<MaltegoMessage>
<MaltegoTransformResponseMessage>
<Entities>
<Entity Type='Phrase'><Value>koos-0</Value></Entity>
<Entity Type='Phrase'><Value>koos-1</Value></Entity>
<Entity Type='Phrase'><Value>koos-2</Value></Entity>
<Entity Type='Phrase'><Value>koos-3</Value></Entity>
<Entity Type='Phrase'><Value>koos-4</Value></Entity>
<Entity Type='Phrase'><Value>koos-5</Value></Entity>
</Entities>
</MaltegoTransformResponseMessage>
</MaltegoMessage>
```

When using a phrase entity with the value “5” this is what the output looks like (organic layout):



That’s how easy this is. Now – let’s get to work...

Table of Contents

No – really – this is mostly harmless.....	2
Introduction	7
Input (Maltego GUI -> external program).....	7
Output (External program -> Maltego GUI).....	7
Message Wrapper	8
Entity definition.....	8
<DisplayInformation>	8
<Additional fields>.....	9
<IconURL>	10
<UIMessages>	11
Handling Exceptions	12
Messages to STDERR (progress, debug):	12
Timeouts	13
Appendix A: Sample XML output	14
Simple:	14
More advanced:	14
Appendix B: Maltego built-in style sheet.....	15
Appendix C: Maltego predefined entities.....	17
Domain.....	17
DNS Name	17
MX record	17
NS record	17
IP Address	18
Netblock or network.....	18
AS number.....	18
Web site	19
URL.....	19
Phrase	19
Document	20
Person.....	20
Email address	20

Affiliations 21
Location 21
Phone number..... 22

Introduction

The Jannie specification describes how the Maltego GUI speaks with external programs or scripts. These scripts are typically located on the same machine as the GUI, running locally. The spec is similar to the GUI client / TAS server spec.

Input (Maltego GUI -> external program)

Transforms receive their input via command line parameters. The first (mandatory) parameter contains the selected entity value, the second (optional) parameter contains the additional field values.

```
program EntityValue [Field1=field1value#Field2=field2value]
```

For example:

```
C:\Perl\perl.exe c:\myscripts\jannie.pl kaas.paterva.com
```

```
C:\Python\python.exe jannie.py 'Andrew Mac' UID=23123
```

```
/usr/bin/python /usr/scripts/jannie.py 'Roelof Temmingh' 'country=ZA#additional=moo'
```

If a hash (#) needs to be passed as input it needs to be encoded with a backslash (\). Thus # becomes \#.

Input is specifically not in XML to make parsing easier. Pseudo code for reading input and fields could be:

```
entity_value = param(1);

// if script needs to also read fields it needs to be read into a hash table- %fields_hash
@fields = param(2).split('#')
foreach (field in @fields)
    %fields_hash{ field.split('=')[0] } = field.split('=')[1];
```

Output (External program -> Maltego GUI)

There are two streams involved:

Standard error (**STDERR**): Progress and debug messages are sent to standard error.

Standard out (**STDOUT**): Output is sent to standard out (STDOUT). The transform will run until it terminates or timeout is reached. See also section on timeout. The following sections deals with the output on STDOUT:

Message Wrapper

The following XML defines the wrapper around returned entities:

```
<MaltegoMessage>
  <MaltegoTransformResponseMessage>
    <Entities>
      <Entity>
      <Entity>
      <...>
    </Entities>
    <UIMessages>
  </MaltegoTransformResponseMessage>
</MaltegoMessage>
```

The different components of this message will be explained in upcoming sections – starting with the entity itself:

Entity definition

The following is the XML that defines an entity:

```
<Entity Type="EntityType">
  <Value>How entity will be displayed in GUI</Value>
  <Weight>Relative weight of entity</Weight>
  <DisplayInformation>
  <AdditionalFields>
  <IconURL>
</Entity>
```

<DisplayInformation>

The <DisplayInformation> section of each entity is defined as follows:

```
<DisplayInformation>
  <Label Name="Title of section" Type="MIME type">
    Values
  </Label>

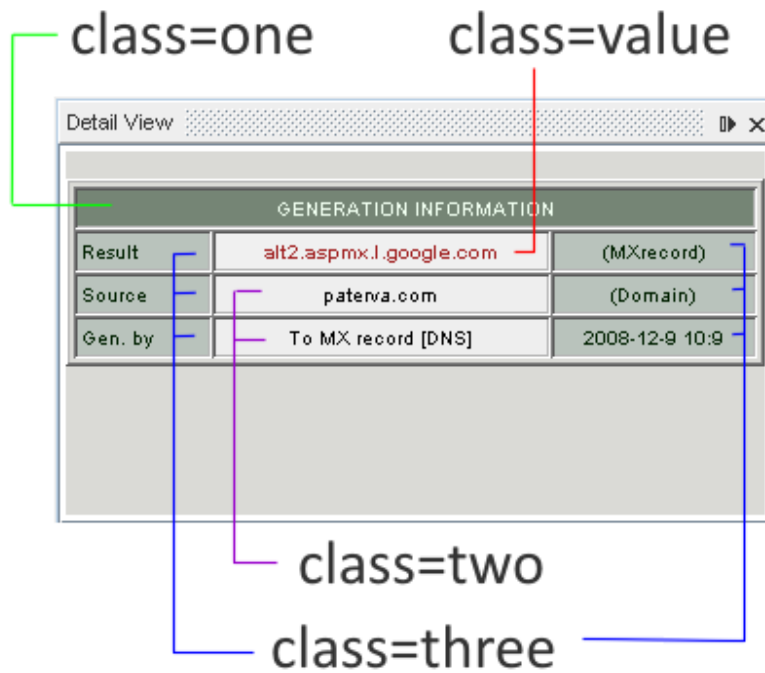
  <Label ...
  ...
  </Label>
</DisplayInformation>
```

Initially MIME types of "text/html" and "text/plain" will be supported, but it is envisaged that other types could be useful in the future. The field content is dependent on the MIME type used. For the currently implemented MIME types, the content must be wrapped in CDATA sections, for example:

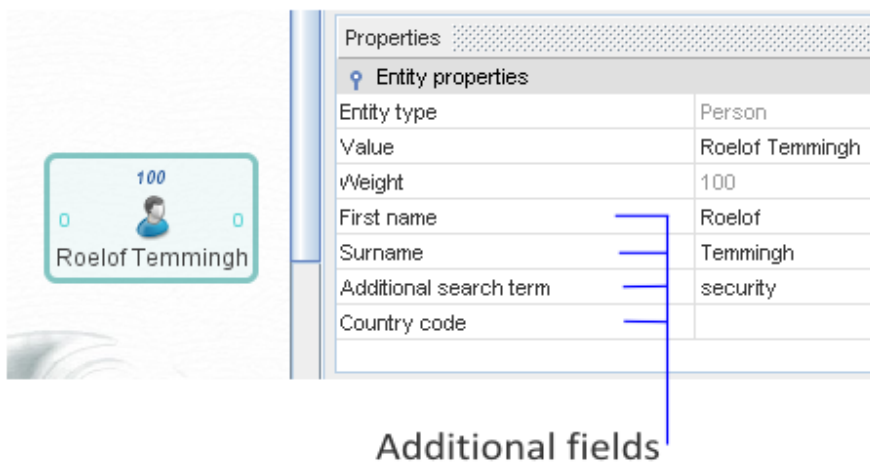
```
<Label Name="Thumbnail image" Type="text/html">
```

```
<![CDATA[<img src=http://www.myimages.com/image1.png]]>
</Label>
```

Keep in mind that this information is for display purposes only. As such it will only be seen in a response from a transform - it will never be passed from the client as input to another transform. The transform writer can feel free to display any information within this section that might be interesting to the user, and can get creative with HTML. The following style sheet can be referenced (in detail in Appendix B):



<Additional fields>



When extending the properties of the entity using the <AdditionalFields> section the transform writer needs to ask him/herself if it is really useful to do so - e.g. if another transform writer will have benefit from the extended field as input to his/her transform. Where the transform writer wants to simply display additional information to the user the <DisplayInformation> section should be used. Another consideration is if the information in additional fields should not rather be an entity itself.

Fields within the <AdditionalFields> section will be editable by the user (and will be passed on to subsequent transforms) while fields in the <DisplayInformation> are read-only, typically HTML and for display purposes only (and will not be passed along to other transforms). To ensure that the entity does not become 'overweight' with additional parameters, transforms writers should not abuse the <AdditionalFields> section. The section looks as follows:

```
<AdditionalFields>
  <Field Name="TheFieldName" DisplayName="HowToDisplayTheFieldInUI" MatchingRule="strict"> Value </Field>
  <Field Name....>
</AdditionalFields>
```

The GUI displays these fields in a property grid and values can be edited. Only the “Name” attribute is mandatory. When the entity is passed to a subsequent transform all of these fields will be passed along.

These fields are not strongly typed and it is up the transform writer to ensure that his/her transform handles these fields correctly.

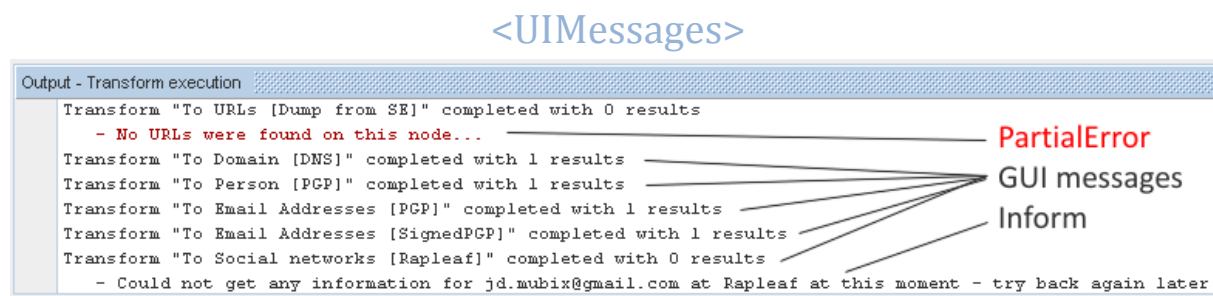
Finally note the *MatchingRule* attribute – this tells the UI if the additional field should be used to match entities. As an example, imagine two persons, mr Andrew Nus with an additional field called “Age” set to 34 and another person also called Andrew Nus but with Age set to 88. If the GUI only considered the entity's value field it will assume that these are the same person (when in fact they are not). By setting the *MatchingRule* attribute to *strict* the transform writer can tell the GUI that it should only match persons when the value AND the Age field match. In the current spec the *MatchingRule* attribute value can only be set to “strict” or “loose” – if the attribute is omitted it is assumed that no matching will happen on the field.

<IconURL>



This section is used for the Icon that will be used in the GUI for this entity. It will be fetched remotely at the URL marked with <IconURL>. Icons are 24x24 pixel images with transparent backgrounds. GIF, JPEG and PNG make good candidates. The GUI will try to resize pictures that are larger than 24x24.

This feature is useful for obvious reasons.



The XML that defines messages that are displayed in the GUI looks like this:

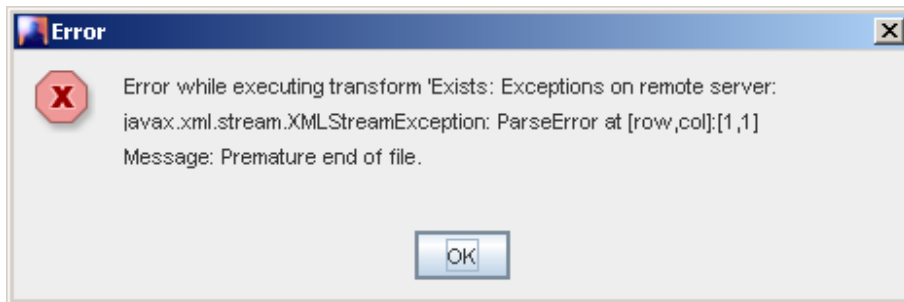
```
<UIMessages>
  <UIMessage MessageType="Inform">Could only enumerate 20 entries</UIMessage>
</UIMessages>
```

The <UIMessages> section contains messages that are sent to the client application to make the user aware of something that occurred. The message could be an error message or just an informational message. The types of messages supported are:

- FatalError – when the transform could not run at all - e.g. no results.
- PartialError – when the transform could not complete, but some results are available.
- Inform – informational message to the user about the transform results.
- Debug – information used for debugging the transform. Messages of this type are currently ignored by the Maltego UI.

A transform that completely fails to return results will only contain this section without any entities. Note that these messages are not relevant to an individual entity, but to the complete response.

Handling Exceptions



Additionally to the <UIMessages> section the transform writer could choose to report exceptions that occurred before entity creation could take place. The format of this message is as follows:

```
<MaltegoMessage>
  <MaltegoTransformExceptionMessage>
    <Exceptions>
      <Exception>Protocol error</Exception>
      ...
    </Exceptions>
  </MaltegoTransformExceptionMessage>
</MaltegoMessage>
```

Messages to STDERR (progress, debug):



There are three types of messages that can go to standard error. These messages are interpreted by the GUI in real time – e.g. they can be shown and acted upon while the transform is running and the complete XML message has not been received.

Progress: Telling the GUI how much processing has been completed.

Debug: These messages are useful for real time debugging of scripts.

The format of the messages is as follows:

- Progress:
 - % *XX*
- Debug:
 - *D:message*

Each message should be followed by a newline character – e.g. each message should appear on a new line. Valid output could look like this:

D:Accessing data store
D:Ready to processing 100 records
%20
%40
%60
%80
D:Crunching numbers – this might take a while
D:Done!
%100

In version 2.0.2 of Maltego only the Debug messages are implemented (and they are not displayed in real time). Real time debugging and progress messages will be supported by a subsequent release.

Timeouts

Timeouts are set per transform within the GUI. After the given timeout has been exceeded, the transform process will be forcibly terminated.

Appendix A: Sample XML output

Simple:

```
<MaltegoMessage>
<MaltegoTransformResponseMessage>
  <Entities>
    <Entity Type="Domain"><Value>paterva.com</Value></Entity>
    <Entity Type="Domain"><Value>pinkmatter.com</Value></Entity>
    <Entity Type="Domain"><Value>everythingsaninterview.com</Value></Entity>
  </Entities>
</MaltegoTransformResponseMessage>
</MaltegoMessage>
```

More advanced:

```
<MaltegoMessage>
<MaltegoTransformResponseMessage>
  <Entities>
    <Entity Type="AffiliationBebo">
      <Value>Bennie van der Broekwurm</Value>
      <Weight>100</Weight>
      <DisplayInformation>
        <Label Name="Details" Type="text/html">
          <![CDATA[<html>MYHTML HERE</html>]]>
        </Label>
      </DisplayInformation>
      <AdditionalFields>
        <Field Name="uid" DisplayName="Unique identifier" MatchingRule="strict">334234110</Field>
        <Field Name="network" DisplayName="Network">Schnoep</Field>
      </AdditionalFields>
      <IconURL> http://network.com/people/334234110.png</IconURL>
    </Entity>
    <Entity>
      another one..
    </Entity>
  </Entities>
  <UIMessages>
    <UIMessage MessageType="Inform">Could only enumerate 20 entries</UIMessage>
  </UIMessages>
</MaltegoTransformResponseMessage>
</MaltegoMessage>
```

Appendix B: Maltego built-in style sheet

This is the built-in style sheet that Maltego uses. It is listed here so that you can test your output in a browser.

```
<style>
  BODY
  {
    background-color: #dadad6;
    color: #000000;
    font-family: Verdana;
    font-size: 10pt;
  }
  table
  {
    width: 100%;
    font-family: Verdana font-size:10pt;
    padding: 1;
    border: 0;
  }
  td
  {
    background-color: #ffffff;
    color: #000000;
  }
  td.one
  {
    background-color: #758575;
    color: #f0ff0;
    text-align: center;
  }
  td.two
  {
    background-color: #eeeeee;
    color: #000000;
    text-align: left;
  }
  td.three
  {
    background-color: #B9C3BC;
    color: #002000;
    text-align: left;
  }
  td.value
  {
    background-color: #f0f0f0;
    color: #a02020;
    font-weight: bold;
    text-align: left;
  }
  A
```

```

{
  color: #2222b2;
  text-align: center;
  text-decoration: ==none;
}
</style>

```

The HTML to generate a typical table looks like this:

```

<table width=100% border=1 rules=cols frame=box cellpadding=2>
<tr><td class=one colspan=3 align=center>GENERATION INFORMATION</td></tr>

<tr>
  <td class=three width=20%>Result</td>
  <td class=value align=center>Value</td>
  <td class=three width=30% align=center>(RType)</td> </tr>

<tr>
  <td class=three width=20%>Source</td>
  <td class=two align=center>Source</td>
  <td class=three width=30% align=center>(LType)</td> </tr>

<tr>
  <td class=three width=20%>Gen. by</td>
  <td class=two align=center>Name </td>
  <td class=three width=30% align=center>Date</td> </tr>
</table>

```

Appendix C: Maltego predefined entities

Domain



Entity name: **Domain**

Value: the domain name –e.g. *foo.com*

Additional fields:

- **whois**: a text field containing the whois information from various registrars
-

DNS Name



Entity name: **DNSName**

Value: the DNS name – e.g. *hidden.foo.com*

Additional fields: None

MX record



Entity name: **MXrecord**

Value: the DNS name of the MX record -e.g *mx.foo.com*

Additional fields: None

NS record



Entity name: **NSrecord**

Value: the DNS name of the NS record -e.g *ns.foo.com*

Additional fields: None

IP Address



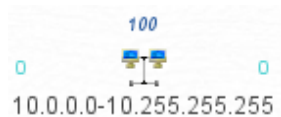
Entity name: **IPAddress**

Value: the IP address – e.g. *10.0.0.1*

Additional fields:

- **whois**: a text field containing the whois information from various registrars
-

Netblock or network



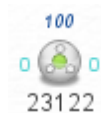
Entity: **Netblock**

Value: the netblock formatted as follows: firstIP-lastIP –e.g. *10.0.0.0-10.255.255.255*

Additional fields:

- **ASnumber**: The AS number corresponding to the netblock –e.g. *4331*
 - **startIP**: The first IP of the netblock – e.g. *10.0.0.1*
 - **endIP**: The last IP of the netblock – e.g. *10.255.255.255*
 - **country**: The shortcode of the country where the netblock is located in –e.g. *CA*
 - **description**: A free format text field that can hold info such as registrar, owner etc.
-

AS number



Entity name: **ASNumber**

Value: the AS number -e.g *23122*

Additional fields: None

Web site



Entity name: **Website**

Value: the name of the website – omit `http://` or `https://` -e.g *www.foo.com*

Additional fields:

- **http:** ports that respond to the HTTP protocol, comma separated –e.g. *80,8080*
 - **https:** ports that respond to the HTTPS protocol, comma separated – e.g. *443,1433*
 - **servertype:** the server banner –e.g *Apache 2.0.99*
 - **URLS:** A list of URLs (when the web site has been collected from a search engine). Format of URLs are: `URL1 Title1\nURL2 Title2\nURLn Titlen`
-

URL



Entity name: **URL**

Value: the title of the URL. Where no title is available, the first 60 character of the URL followed by ... - e.g. *http://www.foo.com/doc/mydocs/..* OR *The best webpage ever*

Additional fields:

- **theurl:** the full URL –e.g. *http://www.foo.com/doc/mydocs/more/there/page.html*
 - **Note:** when exporting data – this field is exported
 - **fulltitle:** the full title of the web page –e.g. *This is a very very long title and we don't to display it all*
-

Phrase



Entity name: **Phrase**

Value: the phrase – this is a powerful entity as its value can contain any free text which is useful when using with search engines–e.g. *maltego site:za filetype: pdf*

Additional fields: None

Document



Entity name: **Document**

Value: the title of the document – e.g. *How to make friends*

Additional fields:

- **link**: URL of the document –e.g. *http://www.foo.com/my.doc*
 - **Note**: when exporting data – this field is exported
 - **metainfo**: meta data of the document – in EXIF format
-

Person



Entity name: **Person**

Value: the person's name and surname –e.g. *Bennie van der Broekwurm*

Additional fields

- **firstname**: the person's firstname – e.g. *Bennie*
 - **lastname**: the person's lastname / surname – e.g. *van die Boekwurm*
 - **additional**: additional information (free text) about the person (this is normally passed along to the search engine) –e.g. *librarian TV*
 - **countrysc**: the country short code (this is normally passed along to the search engine) –e.g. *za*
-

Email address



Entity name: **EmailAddress**

Value: the email address –e.g. *roo@foo.com*

Additional fields

- **URLS:** A list of URLs (when the email address has been collected from a search engine). Format of URLs are: URL1 Title1\nURL2 Title2\nURLn Titlen
-

Affiliations



There are numerous affiliations that have the exact same structure. The structure is as follows:

Entity name: one of the following:

- **AffiliationBebo**
- **AffiliationFacebook**
- **AffiliationFlickr**
- **AffiliationLinkedin**
- **AffiliationMySpace**
- **AffiliationOrkut**
- **AffiliationSpock**
- **AffiliationTwitter**
- **AffiliationZoominfo**
- **AffiliationWikiEdit**

Value: A person's name or nickname –e.g. *Bennie van der Broekwurm* or *SuperDukeFooBar*

Additional fields:

- **uid:** the unique identifier for person/group on the network – e.g. *2343523* or *Broekwurm*
- **network:** where applicable, the network or group that the person belong to – e.g. *Schnoep*
- **profile_url:** the URL where the person's full profile can be viewed –e.g. *http://network.com/users?uid=2343523*

The iconURL field is very useful for the affiliation entity type.

Location



Entity name: **Location**

Value: the location formatted as follows: Area,City,Country – e.g. *Richmond, London, Britain*

Additional fields:

- **long**: Longitude in degrees (note that it can be negative) – e.g. *-18.32*
 - **lat**: Latitude in degrees (note that it can be negative) – e.g. *32.31*
 - **country**: Country name –e.g. *Angola*
 - **city**: City name –e.g. *Brisbane*
 - **area**: Area or suburb name –e.g. *Richmond*
 - **countrySC**: country short code – e.g. *uk*
-

Phone number



Entity name: **PhoneNumber**

Value: the telephone number in the format +XXX XXX XXX XXXX –e.g. *+44 212 334 8034*

Additional fields:

- **countrycode**: the country dial code – e.g. *+44*
- **citycode**: the city's dial code – e.g. *212*
- **areacode**: the area code – e.g. *334*
- **lastnumbers** – the remaining numbers – it could be more or less than 4 –e.g. *8034*
- **additional**: additional information (free text) about the person (this is normally passed along to the search engine) –e.g. *librarian TV*
- **countrysc**: the country short code (this is normally passed along to the search engine) –e.g. *za*
- **type**: type of phone – mobile or landline –e.g. *mobile*
- **URLS**: A list of URLs (when the phone number has been collected from a search engine). Format of URLs are: URL1 Title1\nURL2 Title2\nURLn Titlen

Banner



Entity Name: **Banner**

Value: a banner received from a service e.g. *Apache*

Additional Fields: None

Port

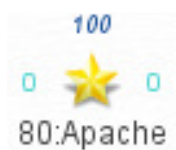


Entity Name: **Port**

Value: Port number, e.g. *80*

Additional Fields: None

Service



Entity Name: **Service**

Value: port and service separated by a colon (':') e.g. *80:Apache*

Additional Fields: None

Vuln



Entity Name: **Vuln**

Value: A vulnerability found, generally a .nasl but is subject to change depending on the operations performed on it. e.g. *ftp_anonymous.nasl*

Additional Fields: None

Webdir



Entity Name: **Webdir**

Value: The relative path to the web directory, e.g. */admin*

Additional Fields: None

Webtitle



Entity Name: **Webtitle**

Value: The title of the website found, e.g. *Welcome - please log in*

Additional Fields: None
