3PE Lookup Service

Parts of this document highlighted in purple, like this, are left for historical interest, but are not intended to be the current focus of development. They may be removed later.

This lookup service provides a bidirectional mapping between third-party network (*3PN*) identifiers in whatever forms they arrive, and Matrix IDs suitable for identifying the Matrix-side ghost users and portal rooms representing those 3PN entities (*3PE*s).

The following examples give details of the proposed lookup service for the client to ask the Homeserver (*HS*), and HS to ask application services (*AS*es) to provide a mapping from third-party network locations (*3PL*s) and users (*3PU*s) into native Matrix IDs. This is designed primarily to support the UX designed in First Class UX For Third Parties. The initial information required from the user is some natural form of identifying the 3PE in whatever form is naive to that network. This would be a textual (ideally Unicode) identifier, but does not generally have any particular form. Some networks may use URIs to identify some entities, most will not. Some types of network may even require multiple distinct components to uniquely identify an entity - for example a generic IRC bridge will need to know both the network name and the name of the entity within that network.

The reader should also be familiar with the general bridging concepts outlined in 3PN Bridging.

This document expands on the initial proposal in the Matrix spec found at

> http://matrix.org/docs/spec/r0.0.1/application_service.html#user-ids

Client Mapping 3PN Entities To Matrix IDs

This operation is the typical initial use-case, where the user of a client wishes to discover the underlying Matrix room alias or user ID that represents a given entity on a 3PN. In order to answer it, the HS might perform further lookups on its individual ASes, or talk to an identity server, or any other behaviour it finds appropriate.

Query Metadata

The following queries that originate from a client performing a request on behalf of the user may need to be aware of 3PN-specific details such as field names. A mechanism is needed that lets clients discover what information is required for such a request.

```
GET /_matrix/client/:ver/thirdparty/protocols
```

This query fetches the overall metadata about protocols supported by the HS. The format of the return value is still very much a subject under heavy development; the following example suggests what might be returned.

```
{
  "irc": {
    "user_fields": ["network", "nickname"],
    "location_fields": ["network", "channel"],
    "icon": "mxc://example.org/aBcDeFgH",
    "field_types": {
      "network": {
        "regexp": "([a-z0-9-]+\.)*[a-z0-9-]+",
        "placeholder": "irc.example.org"
      },
      "nickname": {
        "regexp": "[^\s#]+",
        "placeholder": "username"
      },
      "channel": {
        "regexp": "#[^\s]+",
        "placeholder": "#foobar"
      }
    },
    "instances": [ ... (see below) ],
  },
  "gitter": {
    "user_fields": ["username"],
    "location_fields": ["room"],
    "field_types": { ... },
    "instances": [ ... ],
  },
  "slack": { ... }
}
```

Here the toplevel object contains keys named after the protocols supported by the HS. The value of each is an object containing metadata relating to that protocol. The contained keys are:

- `user_fields`: A list of strings, giving the names of the fields required to identify a user; these are the fields given to the `/thirdparty/user` query.
- `location_fields`: A list of strings, giving the names of the fields required to identify a location; these are the fields given to the `/thirdparty/location` query.
- `icon`: An `mxc://` URL for a small image to illustrate this protocol type in a user interface.
- `field_types`: An object that gives information about the accepted values for each of the named identification fields.

- `instances`: A list of objects representing logically independent sets of configuration.

The `user_fields` and `location_fields` lists should return a list of key names that match the names of keys used in the `fields` object of user or location queries. These keys should be ordered to suggest the way that entities might be grouped, with higher groupings ordered first. For example, the name of a network or server should come before the name of a user or location within that server. Clients may make use of this sorting order to display a hierarchical breakdown of mapped entities.

The `field_types` object contains type information about the sorts of accepted values in each of the fields used to identify a user or location, which may be helpful to a client to assist it in building a UI to ask the user for their values. The keys of the map are the possible field names for identifying users and locations, and the values of each are themselves a sub-object giving validation information about values of that field. The object should contain two keys:

- `regexp`: A string containing a regular expression pattern that valid values should match. This may be a fairly coarse filter; it is not required that the pattern reject all possible invalid strings, as the underlying application service can apply its own better logic anyway. It may simply serve as a reasonable first-cut filter on the client end, perhaps to avoid server roundtrips, or to indicate validation in realtime as the user types a value.
- `placeholder`: A string containing an example value to suggest to the user the sort of thing they might want to enter here. It ought to be valid according to the validation regular expression.

The `instances` list contains objects that represent each individual set of configuration that is known for the protocol. Typically this is used to represent individual networks or domains of a partitioned 3PN type. Usually a unified network would have just one. This list is in no particular order. Servers are not obliged to return results in a consistent order, and clients should take care to sort results consistently somehow, perhaps by description, if they are displayed in a list to the user.

Each object should look like:

```
{
  "desc": "Freenode",
  "icon": "mxc://......",
  "fields": {
    "network": "freenode.net",
  },
}
```

The contained keys are:

- `desc`: A human-readable description string to identify this particular instance configuration to a user, for display on a user interface

- `icon`: An optional `mxc://` URL for a small image to illustrate this configuration of the protocol. This will override a generic one provided at the protocol top-level object.
- `fields`: An object that provides preset values for certain lookup fields. This allows a client to offer a choice of values to the user, or perhaps for other intermediate code to validate the values of fields in queries.
-

Other fields that might need specifying in future could include:

- Typing or enumeration value hints for other identification fields - validation regexps for usernames, placeholder/example values, etc...

For efficiency, the client may also request information about a particular protocol

```
GET /_matrix/client/:ver/thirdparty/protocol/:protocol
```

This will return an object containing the metadata of the requested protocol (i.e. a single value from the object that the `/protocols` request would have returned).

Rooms (aka "Locations")

The endpoint path encodes the protocol name; other fields that identify the 3PL are encoded as query parameters:

```
GET /_matrix/client/:ver/thirdparty/location/:protocol
  ?field1=$value1&field2=...
```

(*Note*: this URL is wrapped for presentation purposes in this document but would appear in a single line over the wire)

A successful mapping results in a JSON list containing objects to represent the Matrix room or rooms that represent a portal to this 3PN. Each has the Matrix room alias string, an identifier for the particular 3PN protocol, and an object containing the network-specific fields that comprise this identifier. It should attempt to canonicalise the identifier as much as reasonably possible given the network type.

```
[{
  "alias": $matrix_room_alias,
  "protocol": $3pn_network_type,
  "fields": {
    "field1": $value1, ...
  }
},
...]
```

The toplevel value is a list because in general the HS may have more than one match for a given 3PL, as it may be aware of multiple different bridges hosted in different places that route to the same 3PN. If multiple matches are returned, the order is generally unspecified;

the client should not infer any particular meaning from it. It may decide to offer the choice on to the user.

An otherwise well-formed client request that happens to result in no matches yields a successful response containing an empty list. Errors in the request (such as missing fields, fields containing invalid values for that protocol) returns a 400-series response.

Note that this API generally simply performs a translation on the textual representation of a potential 3PN entity from one form into another; it does not actually contact the 3PN in question to verify that such an entity exists. A successful mapping does not mean that the requesting user will actually be able to use the returned Matrix identifier, it simply means that this is the identifier it should use to attempt it.

For example to find the `#Matrix` channel on the IRC network `freenode`:

```
GET /_matrix/client/:ver/thirdparty/location/irc
  ?network=freenode&channel=%23Matrix

[{
  "alias": "#freenode_#matrix:matrix.org",
  "protocol": "irc",
  "fields": {
    "network": "freenode",
    "channel": "#matrix"
  }
}]
```

In particular in this example, note that the 3PN AS has case-folded the channel name. It knows that in this case, IRC is not case sensitive for channel names, so it has converted the capital `M` of `#Matrix` into lower-case.

Users

The endpoint path encodes the protocol name; other fields that identify the 3PL are encoded as query parameters:

```
GET /_matrix/client/:ver/thirdparty/user/:protocol
  ?field1=$value1&field2=...
```

A successful mapping results in a JSON list containing structures to represent the Matrix user ID for the ghost representing this 3PU, an identifier for the particular 3PN type, and an object containing the network-specific fields that comprise this identifier. As with room aliases, in general this list may contain more than one result; both because of multiple bridges, and in case the HS has decided to contact an Identity Server, which it trusts to provide a reference from the 3PID to a native Matrix account.

```
[{
  "userid": $matrix_user_alias,
  "protocol": $3pn_network_type,
```

```
    "fields": {
      "field1": $value1, ...
    }
  },
  ...]
```

As with the room mapping API, this is simply a translation of the textual representation of the name; it does not imply that the user actually exists.

For example to find the `@jim` user on Gitter:

```
GET /_matrix/client/:ver/thirdparty/user/gitter?user=%40jim

[{
  "userid": "@gitter_jim:matrix.org",
  "protocol": "gitter",
  "fields": {
    "user": "jim"
  }
}]
```

In particular in this example, note that the 3PN AS has stripped the leading `@` symbol from the user name, because it is not a significant part of the username on the gitter network.

Client Mapping Matrix IDs to 3PN Entities

Using these APIs to perform "reverse" lookups, turning Matrix native identifier forms into 3PN identifiers is just as simple, by supplying other forms of query parameters. In these cases, the path does not contain a component for the protocol type, because the requesting client may not yet know what that is.

```
GET /_matrix/client/:ver/thirdparty/location?alias=$mxalias

GET /_matrix/client/:ver/thirdparty/user?userid=$mxid
```

In both cases, the response can return the same shaped structure as it does in the forward mapping cases, because the response yields both the broken-down 3PN-specific fields, and the Matrix ID.

AppServ Mapping 3PN Entities To Matrix IDs

These APIs are called on the AS by the HS, either by request of a client, or merely to collect information on an AS that a client may want later.

Query Metadata

```
GET /_matrix/app/:ver/thirdparty/protocol/:protocol
```

This endpoint invoked by the HS returns a JSON object containing information for a single thirdparty protocol, in the form described above.

The following operations begin with a client requesting a mapping for an entity from the 3PN identifier form into a Matrix room alias or user ID that could be used to represent this entity within Matrix. The HS then picks any ASes it has configured that may be able to provide an answer.

Rooms

```
GET /_matrix/app/:ver/thirdparty/location/:protocol
  ?field1=$value1&field2=...
```

Example:

```
GET /_matrix/app/unstable/thirdparty/location/irc
  ?network=freenode&channel=#matrix...
```

The response to this request takes the same form as the response to the client-API request (such that the HS might just proxy the value through to the requesting client directly).

Users

```
GET /_matrix/app/:ver/thirdparty/user/:protocol
  ?field1=$value1&field2=...
```

Example:

```
GET /_matrix/app/unstable/thirdparty/user/irc
  ?network=freenode&nickname=mrrobot...
```

The response to this request takes the same form as the response to the client-API request (such that the HS might just proxy the value through to the requesting client directly).

AppServ Mapping Matrix IDs to 3PN Entities

Using these APIs to perform "reverse" lookups, turning Matrix native identifier forms into 3PN identifiers is just as simple, by supplying other forms of query parameters. In these cases, the path does not contain a component for the protocol type, because the requesting HS may not yet know what that is.

```
GET /_matrix/app/:ver/thirdparty/location?alias=$mxalias
```

```
GET /_matrix/app/:ver/thirdparty/user?userid=$mxid
```

In both cases, the response can return the same shaped structure as it does in the forward mapping cases, because the response yields both the broken-down 3PN-specific fields, and the Matrix ID.