UUCP Invite-File And Transport Abstraction Specification

Purpose

Background

UUCP is a primitive but effective file and message transfer system that depends on a call model – a caller node reaches out to a receiver node, and the two parties transact messages and files during the call.

Nodes can be setup as both a caller and receiver role, and also forward to other nodes.

This scheme enables the creation of a working peer-to-peer asynchronous communications network.

UUCP requires that nodes know each other by name, and also have communication details defined, which can be unique or advanced. For example, modern use of UUCP might be done done over SSH pipes – meaning a caller will need a private key to talk to a receiver, know the SSH server address, and also an SSH username.

What This Specification Can Make Easier

UUCP operators typically manually update configuration files to enable another node to talk to them. However, consider the case where you are operating a UUCP receiver/hub, and wish to provide caller/"leaf" access to a user who barely knows what UUCP is. Or maybe you want to build a private UUCP peer-to-peer network and would like to onboard other peers quickly.

A scheme to define node and connection parameters in a object that can automatically processed by a script or tool would go a long way toward making this process easier.

This specification describes a text-based *invite* file format that could be used by such a tool. The format is designed to be easy to parse by normal UNIX tools without being too difficult to read.

This specification depends on a concept of an "transport" which is a virtual layer over the basic pieces of UUCP configuration. Details on this are below.

Again, the intent is that a script or tool processes the file and updates changes to UUCP configuration files according to the directives therein. Such tools should backup existing files before making changes and "be on the user's side" insofar as handling unexpected conditions.

General Structure

Assumptions

All directives in an invite file describe a single UUCP receiver, for the purpose of enabling a single UUCP caller to talk to it.

Elements

- A UUCP invite file is a plain text file that is expected to contain:
  - Configuration directives
  - # Unix style comments
  - Blank lines

Bad Invite Files

If an invite file parser encounters a condition is encountered that identifies the file as bad, the invite parser SHOULD stop parsing the file at the first encounter, issue/communicate an error, and not modify the existing UUCP configuration at all.

If any part of an invite file is bad, the entire file is considered bad and the file MUST be rejected in its entirety. Partial data that was valid before any encountered error MUST not be used to modify UUCP configuration or take any action.

Ignored Stuff

The following is ignored and not parsed by an invite interpreter:

Presence of any of the below does not indicate a bad invite file and SHOULD NOT raise an error condition.

- Blank lines
- Leading/trailing whitespace
- Lines beginning with # (after removing whitespace) are ignored.
- Unknown directives (see below)

No “Include” Mechanisms

A specification-compliant invite file is completely self-contained and independent. An invite file parser MUST never open another file to read in additional configuration directives.

Expectations - Unknown Directives

Later revisions of this specification may introduce additional directives, or deprecate existing ones.

If a directive is unknown to the invite parser, but otherwise syntactically valid, this does not mean the invite file is bad. In this case the invite parser SHOULD move on to the next line without taking action and should not raise an error condition.

This does not make its line exempt from line or file size limits.

Expectations - Line And File Sizes

Invite files are designed to be read completely before being processed. As such, line and file size limits are defined to prevent resource exhaustion.

The following conditions mean an invite file is bad and MUST be rejected entirely per the above:

- 256 or more lines in a file.
- 4096 or more characters occur between newlines anywhere in the file.
- Newlines can be CR, LF, or CR+LF.

Expectations - Syntax

Directives have a simple format – two items separated by a space.

```
[directive-name] [value]
```

1. The directive name and value is separated by at least 1 space.
2. Directives are not case sensitive. Values are, though.
    - Not all directives will take values.
3. If a value is specified for any directive that doesn't accept one, the invite file is bad.
4. Directives must start with a letter.
    - If any directive name doesn't start with a letter, the invite file is bad
5. A directive name may have the following characters:
    - a-z, A-Z, 0-9, hyphens "-", underscores "_"
    - If any directive name has any characters other than the above, the invite file is bad.

Values may have any characters except newlines, which complete the directive. What is a valid value for a given directive will depend on the specific directive.

Directives That Start And End Blocks

Some directives are part of a "block pair" - these will allow inline data between them.

One directive is a "block-starting" directive, the other is a "block-ending" directive.

1. Directives that end blocks will always begin with "`end-`".
2. The following conditions mean an invite file is bad and MUST be rejected entirely.
    - Block-ending directive before block-starting directive.
    - Block-starting directive without block-ending directive.
3. The following conditions do not mean an invite file is bad:
    - No lines between block-ending and block-starting directive (that just means no data in the block)
    - After a block-starting directive is encountered, the parser should be in a "verbatim" mode where it is basically gathering and concatenating lines as they appear in the file. This includes blank lines, whitespace, lines that begin with or contain #, and lines that might look like other directives other than the matching block-ending directive.
4. The block-ending directive must appear on its own line to not be considered part of the block's inline data.
5. Inline data lines are not exempt from line or file size limits.

Expectations – Required Directives

If a required directive is missing, the invite file is bad and MUST be rejected entirely.

Transport Abstractions

UUCP is essentially two `uucico`'s speaking a serial protocol to each other – over *something, anything*. The `uucico`'s do not really care as long what that is, as long as data can move between them.

A node can, for example, make its `uucico` reachable through any of these methods:

- UNIX login over local serial port with `uucico` as a shell
- Talk directly to a `uucico` listening in on a local serial port
- The two above options, but where a modem is assumed to exist and a dialer is asked to dial a number first.
- Talk directly to a `uucico` listening in on a TCP socket
- Anything that will work with the pipe method, including `netcat` (maybe you want to do UUCP over UDP), `bondcat`, `ssh`, or anything else that works as a standard UNIX command over `stdin`/`stdout`.

And everything is fine as long as one side knows how to talk to the other side.

Each of those methods has specific parameters that needs to be known in order to work. Getting those methods to work in a UUCP configuration will touch multiple UUCP configuration files.

Two concrete, detailed examples:

- If you expose UUCP on a phone line, and want to invite someone to connect it, they will need to know:
    - The fact that they should talking to a modem over their local serial port,
    - That they should be using a dialer,
    - Which phone number to call.

- If you expose UUCP via a private-key-protected SSH login, they will need to know:
    - The fact that they should be running the SSH command to establish a pipe,
    - The SSH private key,
    - The name of the server to contact,
    - The SSH username.

So this invite specification supports an abstraction of UUCP *transport* – which will be group of standard parameters for a given "way" to connect to a UUCP node.

The `transport-type` directive is used to specify that transport.

```
transport-type {transport-type}
```

Then, depending on *which* transport is declared, additional directives will need to be specified – and which ones those are depend on that specific transport.

Currently Supported `transport-type`s

`None`	Requires no additional directives.
`KnownSystem`	Requires no additional directives.
`SSHTransport`	Requires additional directives to define SSH-related parameters.

Directives

Directive Definitions

The term “you” refers to the UUCP node processing the invite file.

The term “inviting system” refers to the system that provided you the invite file.

Directive List

Following is the list of currently defined directives, details on their intent/meaning, and implementation notes.

Directives – Global

These directives apply to any transport type.

`expected-caller-nodename`

This directive is *optional*.

```
expected-caller-nodename {nodename}
```

When `expected-caller-nodename` appears in the invite file, this means that the inviting system expects you to be named `nodename`.

UUCP calls require the caller to state the nodename at the beginning of the call--if it's not a name that is known to the receiver, the receiver will drop the call and communication will not happen.

Therefore making sure you and the inviting system agree on the nodename is important.

Implementation Notes

An implementation may choose to:

- Set the global `nodename` option in the UUCP config file to this value. This sets the default nodename reported to all receivers. Implementations should warn the user of the effect of that change.

- Add a `nodename` line among the options for this particular system in the UUCP sys file, which will make the caller state the specified nodename to that system only. This is likely the best implementation choice.

`forward-to-me`

This directive is *optional*.

```
forward-to-me [ {remote-system-name} {remote-system-name-2} ... ]
```

When `forward-to-me` appears in the invite file, this means the inviting system will accept files from the you that are intended for destinations other than the inviting system.

A whitespace-separated list of system names may follow the `forward-to-me` directive.

If this is the case, these are systems that the inviting system knows and can get files and/or messages to.

- The inviting system may be able to "forward to" more systems than specified.
- If a list of systems is not provided, the inviting system is expecting you to know these systems beforehand or obtain the list through an alternate method.

Implementation Notes

An implementation would likely:

- at least ensure the UUCP `forward-to` option appears in the `sys` file, with the inviting systems name specified, if it doesn't already exist,
- add the list of systems in the directive to this option, ensuring there are no duplicates, and
- create "known system" `system` entries (without `port` options) in the `sys` file for systems that don't already appear in `sys`.

`please-forward-from-me`

This directive is *optional*.

```
please-forward-from-me [ {remote-system-name} {remote-system-name-2}
... ]
```

When `please-forward-from-me` appears in the invite file, this means the inviting system wants you to accept files that are intended for destinations other than you.

A whitespace-separated list of system names may follow the `please-forward-from-me` directive.

If this is the case, these are systems that the inviting system knows and could send you files to hold and relay to.

- The inviting system may ask you to "forward from" more systems than specified (though this would be especially rude).
- If a list of systems is not provided, the inviting system is expecting you to know these systems beforehand or obtain the list through an alternate method.

Implementation Notes

An implementation would likely:

- at least ensure the UUCP `forward-from` option appears in the `sys` file, with the inviting system's name specified, if it doesn't already exist,
- add the list of systems in the directive to this option, ensuring there are no duplicates, and
- create "known system" `system` entries (without `port` options) in the `sys` file for systems that don't already appear in `sys`.

`system-name`

This directive is *required*.

```
system-name {remote-system-name}
```

The `system-name` directive tells you the nodename of the inviting system.

Implementation Notes

Generally this name will have to appear in your UUCP `sys` file for you to begin make a successful connection to the inviting system.

Implementations will very likely use this name in a new `system` block in the `sys` file.

Conflicts With Existing Systems

This specification does not yet define any expectations, requirements, or notes on invite parser behavior if the system is already defined in your UUCP `sys` file.

`transport-type`

This directive is *required*.

`transport-type {transport-type}`

The `transport-type` directive tells you the desired transport abstraction the inviting system wishes to use in order to provide connection parameters.

Additional directives may/need to be specified depending on the transport type specified with this directive.

Page 5 has a list of the currently defined transport types.

Implementation Notes

Two transport abstractions can be specified that aren't really a "transport" *per se*, and don't have any transport-specific directives: `KnownSystem` and `None`.

`KnownSystem`

An implementation would likely handle a `KnownSystem` by type creating a `system` entry, without a `port` option, in the `sys` file—if the inviting system didn't already exist in `sys`. Typical uses for this arrangement are A) allowing you to receive calls from the system if you're setup to do that, and B) allowing you to recognize the system in a forwarding-to or forwarding-for transaction.

`None`

`None` is intended to communicate that the inviting system doesn't exist, and expects you not to call it or try to forward to/from it. Implementations may choose to remove the system from any list selectable by the end user, or otherwise inactivate it.

Directives – "`SSHTransport`" Transport Type

These directives apply to the "`SSHTransport`" transport type.

SSHTransport Abstraction

UUCP runs great over SSH – and details on how to do that are available here: [craziness/UUCP in 2022 at main · lawrencecandilas/craziness · GitHub]

In order to provide context for the environment that `SSHTransport` directives are assumed to operate under, two excerpts from that document are reproduced below. Caller and receiver setup details are provided even though an implementation may be caller-only or receiver-only. None of the below are specification requirements, the excerpts are intended to aid understanding.

The excerpts explain in detail how the receiver and caller roles can be setup manually. It is expected that an implementation consuming an invite file would do something similar for the `SSHTransport` transport type.

[Excerpt]
SSH RECEIVER role setup step-by-step

1. Create A Local User Just For Incoming UUCP Requests

On my system I called that user `uucp-external`.

Also: I like to put the home directories of "system" users in `/etc/local/servicehome` but that is just my preference, and you can set that to be wherever (default `/home` is fine)

```
# adduser --ingroup uucp --home /etc/local/servicehome/uucp-external --uid 400
--disabled-password uucp-external
```

2. Create SSH Key Files

A Note On Key Hygiene

If you read the excerpt at the beginning of this guide, you should recall that it said to **never store the private key on your server**. You really shouldn't—you will be giving it out to receivers but it definitely should not stay in this spot once you are in "production."

- If you want to be hardcore, you can have a USB drive or other external storage connected, cd to a directory that lives on that storage, and generate the files there.
- If you want to be better but less than hardcore, generate these keys and move the private key away from a location accessible by `uucp-external`.

Login as the user temporarily (easiest way)

```
# su uucp-external
```

Then:

```
$ ssh-keygen -C "uucp-external@YOUR_NODE_NAME" -f ~uucp-
external/YOUR_NODE_NAME.key
```

Then, press Enter twice (no passphrase). It should make two files, `YOUR_NODE_NAME.key` and `YOUR_NODE_NAME.key.pub`.

3. Putting The Key In The Right Spot For *sshd*

Run all these commands. Doing something wrong here will make SSH act like the key file doesn't exist and want a password.

```
$ cd ~

$ mkdir .ssh
```

```
$ chmod 700 .ssh

$ cd .ssh

$ touch authorized_keys

$ chmod 600 authorized_keys
```

4. Setting Up .ssh/authorized_keys

Open the `YOUR_NODE_NAME.key.pub` file (not the plain YOUR_NODE_NAME.key) file in a text editor and copy all of it.

Then open the `authorized_keys` file and paste it in there.

Next, add the following around that key you pasted to make your `.ssh/authorized_keys` file look like this, then save it.

```
no-port-forwarding,no-X11-forwarding,no-agent-forwarding,command="/usr/sbin/uucico
-l" ssh-rsa AAAB3BIG_STRING_OF_RANDOM_KEY_TEXT uucp-external@YOUR_EXTERNAL_DOMAIN
```

If you want to delete the public key file you can at this point.

5. Disable Password Authentication For Your UUCP Dedicated User

You want to disable password authentication for the account you have dedicated to UUCP.

Do that by adding the following to the end of your `/etc/ssh/sshd_config`:

```
Match User uucp-external

      PasswordAuthentication No
```

6. Define Some UUCP Users And Tell UUCP Your Chosen `nodename`

Add a username and password to `/etc/uucp/passwd` - it does not have to match any existing Linux usernames.

Callers will need to use these usernames and passwords, otherwise your `uucico` will give them the cold shoulder.

Reminders:

- `uucico` is the executable that uses this file.
- Keep in mind that `uucico` runs as the system user `uucp` and not root, so anything `uucico` does can be blocked with Linux permissions fairly easily, and by default `uucico` does not have free reign of your system.

- Don’t get scared just because this file has `passwd` in the name.

Then edit `/etc/uucp/config` and change the `nodename` to `YOUR_NODE_NAME`.

7. Add Systems That Will Call You To `/etc/uucp/sys`

[Excerpt]
SSH CALLER role setup step-by-step

1. Get That Private Key File From The Receiver.
Do that.

1a. Ask The Receiver To Add Your `nodename` To His/Her `/etc/uucp/sys` File
If your nodename is `awesome-node` (defined in `/etc/uucp/config`), then the receiver needs to append the following to his/her `/etc/uucp/sys` file.

```
system awesome-node

protocol i
```

That takes care of the receiver "knowing" the caller, which is required.

The receiver doesn't need further information unless it plans to start calls with you. If it does, it should be following these steps vice versa, with your information.

2. Put The Receiver's Private Key File In A Place *`uucico`* Can See And Read It
Recommended location is `/etc/uucp`.

If the sender followed the above steps, you should have a file like this:

`/etc/uucp/RECEIVER_NODE_NAME.key`

Now do these two things...

```
$ chmod 600 "/etc/uucp/RECEIVER_NODE_NAME.key"

$ chown uucp:uucp "/etc/uucp/RECEIVER_NODE_NAME.key"
```

...to that file.

SSH will check the permissions, and the `uucp` user needs to be able to read it.

- This does mean that anyone in the `uucp` group could get these private key files.
  - That's why the receiver connects the private key to a dedicated public user with limited capability.
  - That's also why you only put people in the `uucp` group that you trust.

3. Edit Some UUCP Configuration Files
`/etc/uucp/sys`

Append the below to `/etc/uucp/sys`. You will have to be root to make changes to the file (the `uucp` user can read it but not alter it).

```
call-login *

call-password *

time any

chat "" \d\d\r\c ogin: \d\L word: \P

chat-timeout 30

protocol i

port SSH-YOUR_NODE_NAME
```

/etc/uucp/call

Add a UUCP username and password from the receiver system to this file. You'll also need to specify the node name of the receiver system.

/etc/uucp/Port

You'll need to add a port to the UUCP config. Here's where the receiver's private key is referenced.

You will also need to specify the receiver's actual DNS domain name or IP where it says `RECEIVER_DOMAIN_NAME_OR_IP`.

So, append the following to `/etc/uucp/Port`:

```
port SSH-RECEIVER_NODE_NAME

type pipe

command /usr/bin/ssh -a -x -q -i /etc/uucp/RECEIVER_NODE_NAME.key
-l uucp-external RECEIVER_DOMAIN_NAME_OR_IP

reliable true

protocol etyig
```

4. Log Into The Receiver Once Manually With The SSH Command In The `Port` File

Just copy and paste the `ssh` command and run it once.

```
$ /usr/bin/ssh -a -x -q -i /etc/uucp/RECEIVER_NODE_NAME.key -l uucp-external
RECEIVER_DOMAIN_NAME_OR_IP
```

You'll see the usual prompt about a previously unknown system, say Yes. You should then connect and get a `login`: prompt.

Aren't you already logged in through the private key? Yes, but that was just the private Uber/Lyft to the front door. Now you're talking to the "receptionist" who wants to know who you are – the receiver's `uucico` is talking to you now, and it wants a UUCP username and password.

That comes from that `/etc/uucp/call` file.

Since we are just testing, you can disconnect (or enter the UUCP username and password if you really want to keep going).

At this point, great job! The link is validated working, and uucp group members should be able to call the receiver at your convenience.

5. The Receiver Doesn't Need Something In `/etc/uucp/Port` For My System, As A Caller?

No. ***Ports are for callers***. Receiver end of access is taken care of through the SSH mechanism using this scheme and defining a port on the receiver end is not needed.

`ssh-server`

This directive is *required* if the transport type is SSHTransport.

```
ssh-server {domain-name-or-ip[:port]}
```

This directive identifies the domain name or IP address, and optionally the port, where the SSH server of the inviting system can be reached.

If the port is not specified, the SSH standard port of 22 is assumed.

`ssh-username`

This directive is *required* if the transport type is SSHTransport.

```
ssh-username {username}
```

This directive identifies the username the SSH server of the inviting system is expecting for incoming UUCP connections.

Implementation Notes

There is no password directive. The supported authentication method at this time is through a public/private key pair (see `private-key`).

An implementation will likely use this information to:

- create a new `port` entry in the UUCP `Port` file, and that entry will contain a `command` line that actually contains the `ssh` command and its parameters built from these directives.
- it will also likely create a new `system` entry in `sys`, pointing to the new `port`.

`private-key`

This directive is *required* if the transport type is SSHTransport.

This directive is a *block-starting directive* and has `end-private-key` as it's block-ending companion.

```
private-key
-----BEGIN OPENSSH PRIVATE_KEY-----
{line-of-key-text}
{line-of-key-text}
{line-of-key-text}
.
.
.
-----END OPENSSH PRIVATE KEY-----
end-private-key
```

This directive contains a private key intended to be used by SSH to authenticate to the server identified in `ssh-server`.

Implementation Notes

An implementation is likely to save everything between the directives as a text file in a location accessible to local UUCP binaries, and refer to the file when it constructs the `command` line for the system's port in the `Port` file.

The name of the key file is up to the implementation, this specification does not define a way for the inviting system to choose the filename or specify any method of storage for the key.

`uucico-call-password`

This directive is *required* if the transport type is SSHTransport.

```
uucico-call-password {password}
```

This directive identifies the username that will be communicated to what should be a remote `uucico`, after the SSH connection is established.

`uucico-call-username`

This directive is *required* if the transport type is SSHTransport.

```
uucico-call-username {username}
```

This directive identifies the password that will be communicated to what should be a remote `uucico`, after the SSH connection is established and the username is provided in response to a "Login:" prompt.

Implementation Notes

SSH gets a secure pipe between you and the inviting system – basically "past the front door." The next step is to identify yourself to what should be a `uucico` process asking for a separate username and password.

An implementation will likely add this information straight into your UUCP `call` file.