

Transactional Atlas
as used by WorldForge games
version 0.1 DRAFT

Anders Petersson

November 2003

Contents

1 States	2
1.1 Connected	2
1.1.1 Serverinfo	2
1.1.2 Account information	2
1.2 Logged in	4
1.2.1 Logout	4
1.2.2 OOG	4
1.2.3 Transition to in-game	4
1.3 In-Game	4
1.3.1 Entity information	4
1.4 How to find out where an op is aimed	4
2 General	5
2.1 Concurrent operations	5
2.1.1 Serialno/Refno	5
2.2 Multicontrol	5
2.3 Atlas hirearchy discovery	5
2.4 Errors	5
2.5 Administrator accounts	5
2.6 Media	5
2.7 Caching	5
A References	6
A.0.1 Basic Atlas	6
A.0.2 Atlas Attributes	6
A.0.3 Codec specifications	6
A.0.4 Acorn Whitepaper	6

Chapter 1

States

The following states are stacked, that is most ops used in the various states are still viable during the following states. The most obvious exceptions being the one-shot ops used to change states such as login, the state discovering ops are always good such as the oog look discovery.

1.1 Connected

This is the state you enter directly after atlas codec negotiation, a client doesn't expect any ops to be sent and the server doesn't send any until the client takes some action.

1.1.1 Serverinfo

The serverinfo op is used to discover information about the server commonly used by metaserver clients.

This op is structured as an anonymous get, that is a get without from nor to being set.

spec: parents = (get), !to, !from

The returned op is an info op with a server argument.

spec: parents = (info)

spec arg: parents = (server), clients : int, name : string, ruleset : string, server : string, uptime : float "seconds", version : string, ?builddate : string

1.1.2 Account information

spec arg: parents = (account) id : string "used as from when account is doing anything" *character_types : string list "ad-hoc way of telling what types of PCs can be created, will probably be deprecated in the future" characters : string list "the list of characters currently active in the account, this is the basis of what ids can be used to move in-game" name : string "the name displayed when the account is taking a visual action, mainly OOG chat"

Login

The login op is a concurrent (do *not* rely on serialno/refno, look in 2.1.1) state transitional op, it is only allowed in the connected state, the op can be reversed by sending a logout (see 1.2.1) op when logged in.

The structure of a login op is as follows, currently there is no way of discovering password scrambling supported by the server.

spec: parents = (login)

spec arg: parents = (account), username : string, password : string

On successful login an info op with the account is sent, possibly with a type derived from account (commonly player or admin, see 2.3 for information on type discovery). It has lots of useful information used in the later states.

spec: parents = (info)

spec arg: See 1.1.2 with the exception of password probably not being set.

If the login was unsuccessful an error op (see 2.4 for details) will be sent with a reason and the info op as an arg.

spec: parents = (error)

spec arg: parents = (login)

Create Account

The login op is a concurrent (do *not* rely on serialno/refno, look in 2.1.1) state transitional op, it is only allowed in the connected state, the op can be reversed by sending a logout (see 1.2.1) op when logged in.

The create-account is the way of logging in if no account has been previously created. Be careful as the server can at will override any arguments you send, rely on the response and not on what you sent. (Clients can with great success use the same callback to handle both the response of both login ops and create account ops.)

The op is structured as a create op with an account argument.

spec: parents = (create)

spec arg: See 1.1.2. But note that id will most likely be overridden (many servers will probably keep an username, id equivalence). Also character_types and characters will be ignored (unless you connect to a very strange server).

The response on a successful create account is an info op with the account argument (exactly the same you would expect from a successful login).

spec: parents = (info)

spec arg: See 1.1.2 with the exception of password probably not being set.

In case of an error (could be anything ranging from username collision to requiring some kind of authorization) you will be sent an error op see 2.4 for details.

spec: parents = (error)

spec arg: parents = (create)

Changing information on an existing account

It is possible to change information on an existing account. It is done by sending set ops to the account id with the arguments to set as the argument.

spec: parents = (set), to = *id

spec arg: See 1.1.2. Usually you can set any attributes not derived by the server state (ie not id, characters, character_types and such).

1.2 Logged in

This is the state the connection enters when a successful login or create account operation has been performed. It can be reversed by a logout (1.2.1) operation.

1.2.1 Logout

The logout operation is used when a client wants to login using another account or wants to end the connection in a clean way (dropping the connection will always work of course).

If the logout op has no argument then the account to be logged out is the value of the from attribute. If it does have arguments then the id attribute of the arguments is used. *However* servers which do not support the logging out of other accounts might not bother to check the arguments and use the from directly (cypheis-c++ most notably).

```
spec: parents = (logout) from : string
spec arg: id : string
```

1.2.2 OOG

1.2.3 Transition to in-game

Character creation

Character activation

1.3 In-Game

1.3.1 Entity information

1.4 How to find out where an op is aimed

Chapter 2

General

2.1 Concurrent operations

2.1.1 Serialno/Refno

Don't depend on them!

2.2 Multicontrol

Multiple logins and characters using the same connection are allowed (if supported by the server) but require some care in the client library for successful decoding. More specifically from/to become exorbitantly important.

2.3 Atlas hierarchy discovery

2.4 Errors

spec: parents = (error), XXXreason : string XXX should we have some machine discoverable way of determining the cause of the error? Perhaps a shorter string which is well specified?

2.5 Administrator accounts

2.6 Media

2.7 Caching

Appendix A

References

- A.0.1 Basic Atlas
- A.0.2 Atlas Attributes
- A.0.3 Codec specifications
- A.0.4 Acorn Whitepaper