gt4atproto

Veit Heller (feenk)

What is atproto?

What is atproto?

“The **Authenticated Transfer Protocol**, aka **atproto**, is a federated protocol for large-scale distributed social applications.”

¿¿¿

DID, PDS, BGS, NSID, XRPC...

In brief: we built a client environment

So what?

```
{
  "lexicon": 1,
  "id": "com.atproto.identity.updateHandle",
  "defs": {
    "main": {
      "type": "procedure",
      "description": "Updates the handle of the account",
      "input": {
        "encoding": "application/json",
        "schema": {
          "type": "object",
          "required": [
            "handle"
          ],
          "properties": {
            "handle": {
              "type": "string",
              "format": "handle"
            }
          }
        }
      }
    }
  }
}
```

```
{
  "lexicon": 1,
  "id": "app.bsky.graph.follow",
  "defs": {
    "main": {
      "type": "record",
      "description": "A social follow.",
      "key": "tid",
      "record": {
        "type": "object",
        "required": [
          "subject",
          "createdAt"
        ],
        "properties": {
          "subject": {
            "type": "string",
            "format": "did"
          },
          "createdAt": {
            "type": "string",
            "format": "datetime"
          }
        }
      }
    }
  }
}
```

Class

AppBskyGraphFollowMain

Superclass: GtApModelEntity Package: Gt4AtProtoGeneratedCode Tag:

Methods | Comment | References | Definition references | Schema | Advice definitions | Magritte Descriptions

atpCreatedAt

```
<atpType:#String format:'datetime'>
^ atpCreatedAt
```

accessing instance

atpCreatedAt: accessing instance

atpSubject accessing instance

atpSubject: accessing instance

createdAtDescription

```
<magritteDescription>
^ MAStringDescription new
    label: 'CreatedAt';
    accessor: #atpCreatedAt;
    beRequired
```

magritte instance

subjectDescription magritte instance

atpDescription accessing class

atpFile accessing class

atpRecordName accessing class

atpSchema accessing class

fromRequest: serialization class

Class

GtApGeneratedClient

Superclass: GtApClient Package: Gt4AtProtoGeneratedCode Tag:

Methods | Comment | References | Advice definitions

```
comAtprotoIdentityUpdateHandleHandle: handle
    "Updates the handle of the account"

    <xrpcProcedure>
    ^ GtApRequestResolver
        from: {} asDictionary
        andRequest: (self
                procedureOn: 'com.atproto.identity.updateHandle'
                withBody: {'handle' -> handle} asDictionary)
```

*Gt4AtProtoGeneratedCode instance

comAtprotoLabelQueryLabelsUriPatterns:sources:

comAtprotoModerationCreateReportReasonType:reason:subject:

comAtprotoRepoApplyWritesRepo:validate:writes:swapCommit:

comAtprotoRepoCreateRecordRepo:collection:rkey:validate:record:swapCommit:

comAtprotoRepoDeleteRecordRepo:collection:rkey:swapRecord:swapCommit:

comAtprotoRepoDescribeRepoRepo:

comAtprotoRepoGetRecordRepo:collection:rkey:cid:

comAtprotoRepoListRecordsRepo:collection:rkeyStart:rkeyEnd:reverse:

comAtprotoRepoPutRecordRepo:collection:rkey:validate:record:swapRecord:swapCommit:

comAtprotoRepoRebaseRepoRepo:swapCommit:

comAtprotoRepoUploadBlobFile:

comAtprotoServerCreateAccountEmail:handle:did:inviteCode:password:recoveryKey:

comAtprotoServerCreateAppPasswordName:

comAtprotoServerCreateInviteCodeUseCount:forAccount:

comAtprotoServerCreateInviteCodesCodeCount:useCount:forAccounts:

comAtprotoServerCreateSessionIdentifier:password:

ToC Pages

Generating code from lexicons

First, import the lexicons from an `atproto` directory (see Importing lexicons from files ▸ for mo details):

```
lexiconsDirectory := 'atproto' asFileReference ▸ / 'lexicons'.
lexicons := GtApLexiconImporter ▸ new ▸
    importLexiconsFromDirectory: ▸ lexiconsDirectory
```

Lexicons describe the procedures, queries and record types. This means we can generate code ba on the imported model. The code generation can be parameterized in several ways through a `builder`:

```
builder := GtApModelBuilder ▸ new ▸
        lexicons: ▸ lexicons;
        useDefinitionsBuilderInPackage: #Gt4AtProtoGeneratedCod
        useDefinitionInstanceVarBuilderWithPrefix: ▸ #atp;
        useDefinitionAccessorBuilderWithPrefix: ▸ #atp;
        useDefinitionFromRequestBuilderWithPrefix: ▸ #atp;
        useDefinitionMagritteBuilderWithPrefix: ▸ #atp;
        useDocumentsBuilderInPackage: #Gt4AtProtoGeneratedCode
            inSubclass: ▸ #GtApGeneratedClient
```

Once the builder exist, we can use it to generate the code:

```
builder build
```

Or to remove the already generated code:

```
builder remove
```

▸ 5 explicit references

ToC Pages

- Glamorous Toolkit for AT Protocol [[About Glamorous Toolkit for AT Prot...
  - About Glamorous Toolkit for AT Protocol This is an environment for [[AT Proto...
  - Using the environment The environment can be used both for ...
    - Posting to Bluesky from your knowledge base This page is written in [[Lepiter]], ...
    - Browsing a BlueSky user from a dedicated snippet A user's timeline can be inspected af...
    - Inspecting a user through an API query First, build the client:pdsUrl := 'ht...
    - Working with the API through the generated client The [[XRPC]] client offers us a basic...
    - Editing lexicon files The environment also offers support f...
  - How the environment works {{gtTodo:label=TODO}}
    - Basic API requests using XRPC The [[AT Protocol]] relies on [[XRPC]]...
    - Importing lexicons from files To work with [[Bluesky]] we need the ...
    - Generating code from lexicons First, import the lexicons from an ` a...
    - Browsing lexicons generated code The generated code is already availab...
    - Handling extensions for the generated code through traits Generated code should never be manual...
    - Generating lexicons from code We are able to create lexicons from a...
    - Documenting API endpoints The class {{gtClass: GtApRestCall}} i...
    - ESUG 2023 slideshow {GtPresenterSlideShow create: GtApEsu...
  - Glossary These are different terms used in thi...
    - AT Protocol [](https://atproto.com)
    - Bluesky A specific social media app based on ...
    - feenk Our mission is to make the inside of ...
    - Glamorous Toolkit Glamorous Toolkit is the [[Moldable D...
    - Lepiter Lepiter is the knowledge management s...
    - Lexicon Lexicon is a schema document format u...
    - Moldable Development Moldable Development is a way of prog...
    - XRPC [](https://atproto.com/specs/xrpc)

Posting to Bluesky from your knowledge base

This page is written in Lepiter ▸ , the knowledge management component from Glamorous Tool
▸ . Lepiter is made out of snippets, each of which can define its own language and visual appeara

gt4atproto adds a Bluesky post snippet. Like the one below.

@weaknsmall.bsky.social at 2023-05-24 20:25

This post was authored and posted from Glamorous Toolkit.

Through this snippet, you can create and organize social posts from within your personal knowle
base.

▸ 2 explicit references

Let’s search for things!

Server: https://bsky.social

Username:

Password:

Now that we know, we do.

0/300

weaknsmall.bsky.social

What’s next?

ToC Pages

Generating lexicons from code

We are able to create lexicons from a set of classes, not unlike an inverse of Generating code from lexicons ▸ .

To do this, we first have to build a lexicon generator:

```
generator := GtApLexiconGenerator ▸ forClasses: ▸ LeContent ▸ with

generator
    id: 'com.gtoolkit.lepiter';
    description: 'Lepiter record types'.
```

The generator above will create lexicons for all Lepiter content types, such as pages and snippets. will also generate all the subtypes that these types depend on automatically.

After building the generator we can now generate the model:

```
generator generateModel
```

We can also generate JSON for those models:

```
generator generateJson
```

Because that JSON conforms to the lexicon definition outlined by the AT protocol, we can even fe back into our importer.

```
directory := 'lepiter-example-lexicon' asFileReference ▸ ensureCr

directory / (generator id , '.json')
    writeStreamDo: [ :aStream | aStream nextPutAll: generator gen

GtApLexiconImporter ▸ new ▸ importLexiconsFromDirectory: ▸ directo
```

▸ 1 explicit reference

ToC Pages

- Glamorous Toolkit for AT Protocol [[About Glamorous Toolkit for AT Prot...
  - About Glamorous Toolkit for AT Protocol This is an environment for [[AT Proto...
  - Using the environment The environment can be used both for ...
    - Posting to Bluesky from your knowledge base This page is written in [[Lepiter]], ...
    - Browsing a BlueSky user from a dedicated snippet A user's timeline can be inspected af...
    - Inspecting a user through an API query First, build the client:pdsUrl := 'ht...
    - Working with the API through the generated client The [[XRPC]] client offers us a basic...
    - Editing lexicon files The environment also offers support f...
  - How the environment works {{gtTodo:label=TODO}}
    - Basic API requests using XRPC The [[AT Protocol]] relies on [[XRPC]]...
    - Importing lexicons from files To work with [[Bluesky]] we need the ...
    - Generating code from lexicons First, import the lexicons from an ` a...
    - Browsing lexicons generated code The generated code is already availab...
    - Handling extensions for the generated code through traits Generated code should never be manual...
    - Generating lexicons from code We are able to create lexicons from a...
    - Documenting API endpoints The class {{gtClass: GtApRestCall}} i...
    - ESUG 2023 slideshow {GtPresenterSlideShow create: GtApEsu...
  - Glossary These are different terms used in thi...
    - AT Protocol [](https://atproto.com)
    - Bluesky A specific social media app based on ...
    - feenk Our mission is to make the inside of ...
    - Glamorous Toolkit Glamorous Toolkit is the [[Moldable D...
    - Lepiter Lepiter is the knowledge management s...
    - Lexicon Lexicon is a schema document format u...
    - Moldable Development Moldable Development is a way of prog...
    - XRPC [](https://atproto.com/specs/xrpc)

Editing lexicon files

The environment also offers support for editing lexicons that includes styling and completion.

To play with, first clone `atproto`:

```
atprotoDirectory := 'atproto' asFileReference
```

```
atprotoDirectory ensureDeleteAll.
IceRepositoryCreator
        fromUrl: 'git@github.com:bluesky-social/atproto.git'
        to: atprotoDirectory.
atprotoDirectory
```

And then inspect a document file:

```
GtApLexiconDirectoryModel new directory: atprotoDirectory
```

2 explicit references

ToC Pages

- Glamorous Toolkit for AT Protocol [[About Glamorous Toolkit for AT Prot...
  - About Glamorous Toolkit for AT Protocol This is an environment for [[AT Proto...
  - Using the environment The environment can be used both for ...
    - Posting to Bluesky from your knowledge base This page is written in [[Lepiter]], ...
    - Browsing a BlueSky user from a dedicated snippet A user's timeline can be inspected af...
    - Inspecting a user through an API query First, build the client:pdsUrl := 'ht...
    - Working with the API through the generated client The [[XRPC]] client offers us a basic...
    - Editing lexicon files The environment also offers support f...
  - How the environment works {{gtTodo:label=TODO}}
    - Basic API requests using XRPC The [[AT Protocol]] relies on [[XRPC]]...
    - Importing lexicons from files To work with [[Bluesky]] we need the ...
    - Generating code from lexicons First, import the lexicons from an ` a...
    - Browsing lexicons generated code The generated code is already availab...
    - Handling extensions for the generated code through traits Generated code should never be manual...
    - Generating lexicons from code We are able to create lexicons from a...
    - Documenting API endpoints The class {{gtClass: GtApRestCall}} i...
    - ESUG 2023 slideshow {GtPresenterSlideShow create: GtApEsu...
  - Glossary These are different terms used in thi...
    - AT Protocol [](https://atproto.com)
    - Bluesky A specific social media app based on ...
    - feenk Our mission is to make the inside of ...
    - Glamorous Toolkit Glamorous Toolkit is the [[Moldable D...
    - Lepiter Lepiter is the knowledge management s...
    - Lexicon Lexicon is a schema document format u...
    - Moldable Development Moldable Development is a way of prog...
    - XRPC [](https://atproto.com/specs/xrpc)

Documenting API endpoints

The class `GtApRestCall` ▶ is a subclass of `ZnJSONRestCall` ▶ that adds documentation and validation through autogenerated lexicons.

To document all of the enpoints, you may use `GtApRestCallLexiconGenerator` ▶ .

```
GtApRestCallLexiconGenerator ▶ new ▶
    id: ▶ 'com.gtoolkit.example';
    description: ▶ 'My server definitions';
    restCalls: ▶ {GtApExampleRestCall ▶ };
    generateJson ▶
```

To enable that, what do we need to provide?

There are a few messages that need to be implemented, namely `GtApRestCall class>>#allowedMethods` ▶ , `GtApRestCall class>>#endpointName` ▶ (the name of the endpoint in the documentation), `GtApRestCall class>>#errors` ▶ (if your view m return status codes other than 200), `GtApRestCall class>>#modelClass` ▶ (which is the objec return from the system), and `GtApRestCall class>>#parameters` ▶ (which is the list of param the REST call takes. You can find an example in `GtApExampleRestCall` ▶ .

Setting the parameters will also ensure that the REST body is validated when the request is receiv before the class is activated.

The return object needs to be serializable through `LeJsonV4` ▶ . You can check `UUID` ▶ as a simp example for a non-Lepiter class that is serializable by `LeJsonV4` ▶ .

Network-UUID > UUID class
leJsonV4MappingFor: ▲

Network-UUID > UUID class
leJsonV4Name ▲

Thank you!

Questions?