
Google™
10

Bulkloader, aka Data Migration with App Engine

Matthew Blain

May 19, 2010

Follow Along

View live notes and ask questions about this session on Google Wave

http://bit.ly/appengine4

Also find sample code in this talk at

http://bulkloadersample.appspot.com/

Agenda

- Background
- How the bulkloader works
- How to create configurations
- Advanced features
- Troubleshooting common issues
- Future directions
- Q&A

http://bulkloadersample.appspot.com/

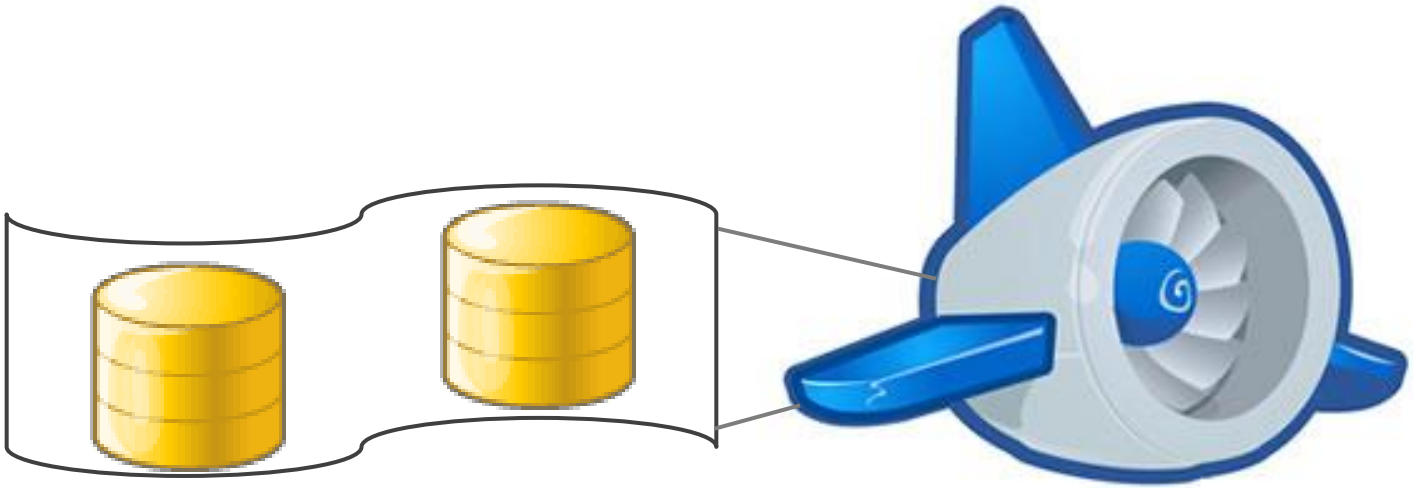
Agenda

- **Background**
- How the bulkloader works
- How to create configurations
- Advanced features
- Troubleshooting common issues
- Future directions
- Q&A

http://bulkloadersample.appspot.com/

Bulkloader

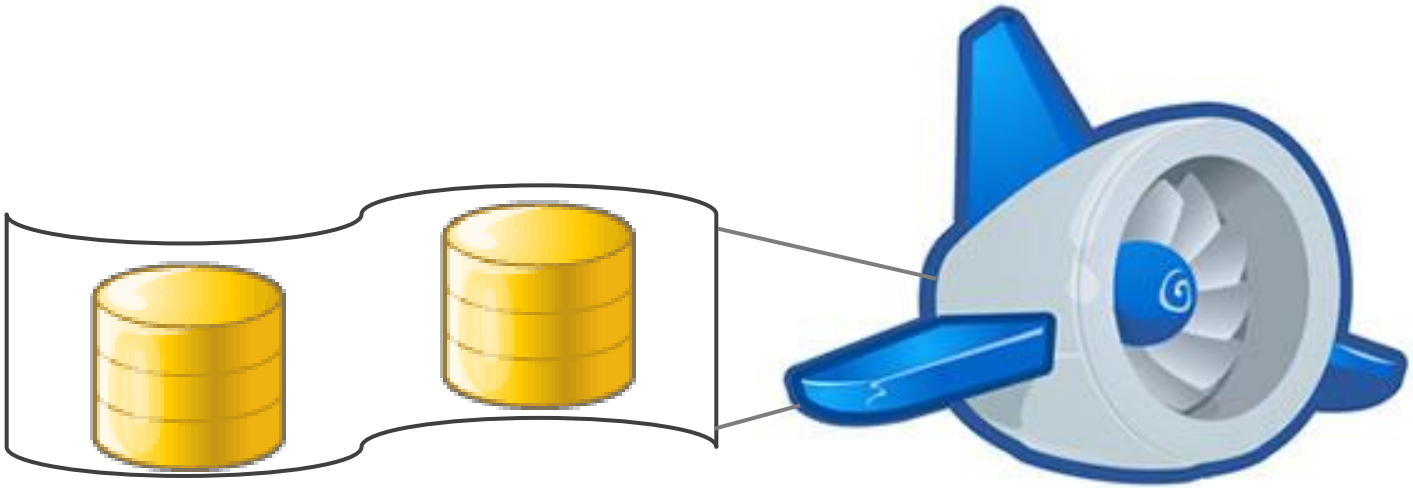
What is that?



Bulkloader

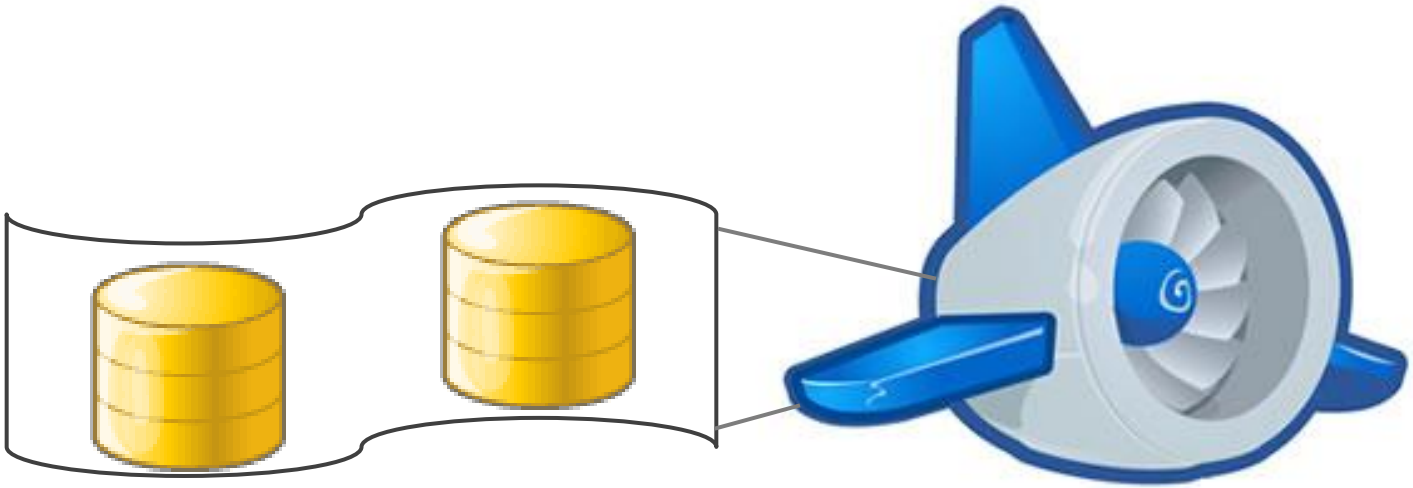
What is that?

- Datastore Import/Export



Bulkloader

This talk will show you how to configure and use the bulk loader, and explain some of the theory.



Background

The App Engine Data Store

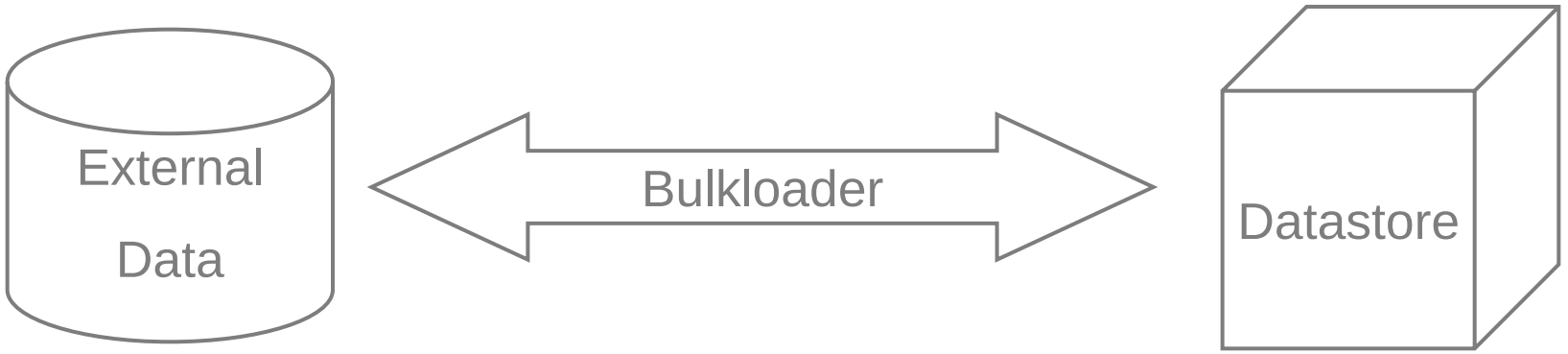
- Schemaless tables of entities
- Each table represents a single *Kind*
- Each entity has a single *key* but may have multiple indexes
- Keys are hierarchical: kinds and ids
- Properties are typed

- Python wraps these with *Model*s
- Java wraps these with multiple wrappers

Background

What the bulkloader does

- Converts external data to entities (and back)
- Copies them to/from the App Engine datastore via HTTP



Background: Requirements

All you need is…

- An app engine app—either dev_appserver or on appspot
- Server works in both Python and Java runtime environments
- Client requires Python 1.3.4 SDK
- Python 2.5 to run the SDK

Agenda

- Review of the existing system
- **How the bulkloader works**
- How to create configurations
- Advanced features
- Troubleshooting common issues
- Future directions
- Q&A

http://bulkloadersample.appspot.com/

Zero Configuration

Very common tasks: Backup, cross-app migration

- Dump data from an app, appspot or dev_appserver
- Restore data to an application
  - Same app or instance, or different

```
appcfg.py
  download_data
  --filename dump.sql3
  --url http://myapplication.appspot.com/remote_api
```

```
appcfg.py
  upload_data
  --filename dump.sql3
  --url http://myapplication.appspot.com/remote_api
```

Demo: Import/Export from Guestbook

Demo using the new config format

Guestbook sample data and Python model

Some sample data and the Python model

content	date	author
hello	20090605T21:06	
this is cool	20100418T01:30	demo@mblain.com

```
class Greeting(db.Model):
  author = db.UserProperty()
  content = db.StringProperty(multiline=True)
  date = db.DateTimeProperty(auto_now_add=True)
```

Guestbook JDO model

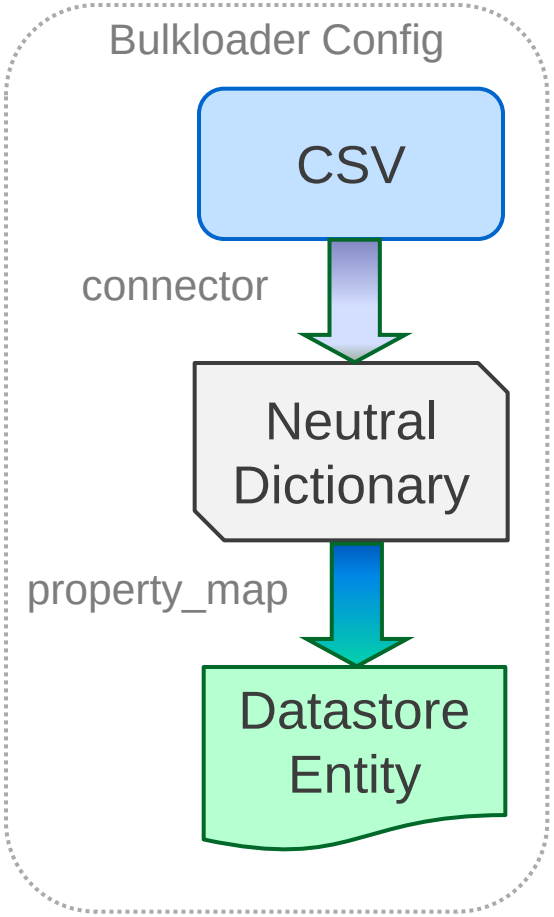
```
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class Greeting {
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Long id;

    @Persistent
    private User author;

    @Persistent
    private String content;

    @Persistent
    private Date date;

    ...
}
```

Demo 1: Import/Export from Guestbook

Key parts of the bulkloader.yaml

```
- kind: Greeting
  connector: csv
  property_map:
    - property: content
      external_name: content

    - property: author
      external_name: author
      import_transform: users.User

    - property: date
      external_name: date
      import_transform:
        transform.import_date_time('%Y-%m-%dT%H:%M')
      export_transform:
        transform.export_date_time('%Y-%m-%dT%H:%M')
```

How did that work? Data transform



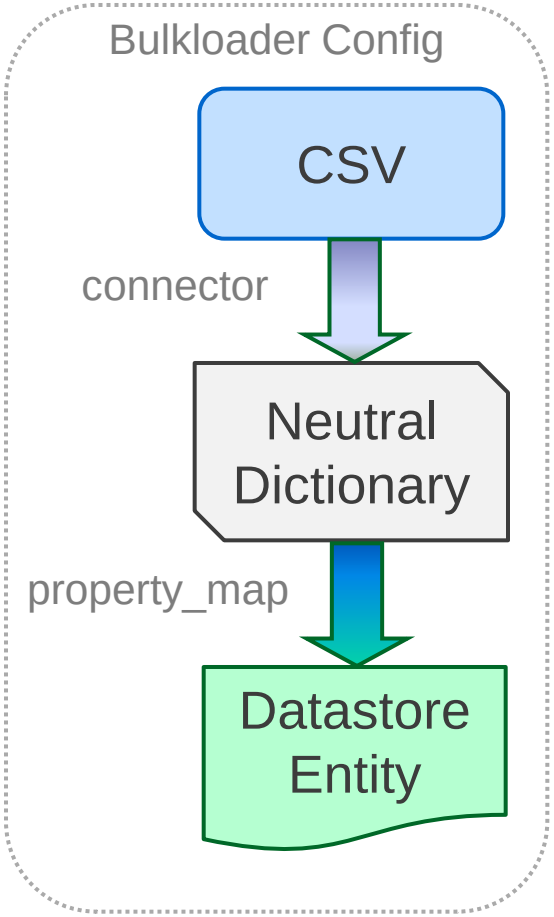

```
content      date              author
cool stuff, 20010418T01:30, demo@example.com
```

```
{'content': 'cool stuff',
 'date':    '20100418T01:30',
 'author':  'demo@example.com'}
```

```
entity['content'] = dictionary['content']
entity['date'] = transform.import_date_time
  ('%Y-%m-%dT%H:%M')(dictionary['date'])
entity['author'] =
  users.User(dictionary['author'])
```

Google I/O 10

How did that work? Data transform



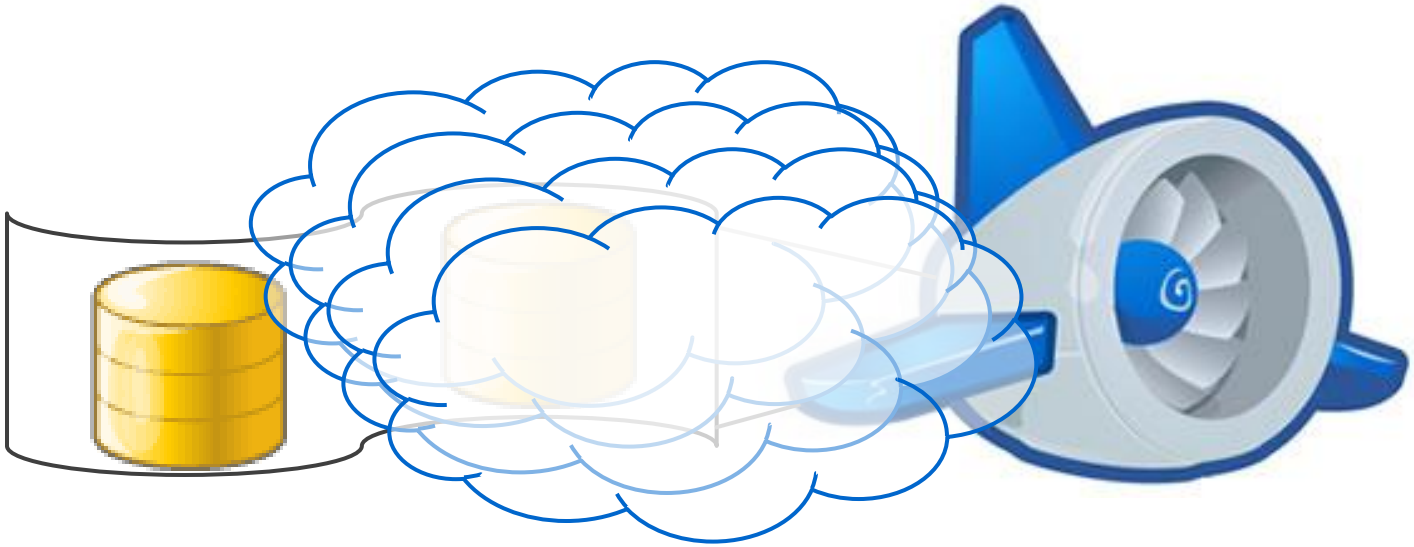

```
content      date                author
cool stuff,  20010418T01:30,     demo@example.com
```

```
{'content': 'cool stuff',
 'date':    '20100418T01:30',
 'author':  'demo@example.com'}
```

```
{'content': 'cool stuff',
 'date':    datetime.datetime(2010,4,18,1,30),
 'author':  users.User(email='demo@example.com')}
```

How did that work? Data transfer

Remote_API



Demo 1: Import/Export from Guestbook

Enable remote_api (Python)

```
application: guestbook
version: 1
runtime: python
api_version: 1
handlers:
- url: /remote_api
  script: $PYTHON_LIB/google/appengine/ext/remote_api/handler.py
  login: admin
- url: .*
  script: guestbook.py
```

Demo 1: Import/Export from Guestbook

Enable remote_api (Java)

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee“
         version="2.5">
  <servlet>
    <servlet-name>remoteapi</servlet-name>
    <servlet-class>
      com.google.apphosting.utils.remoteapi.RemoteApiServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>remoteapi</servlet-name>
    <url-pattern>/remote_api</url-pattern>
  </servlet-mapping>
...
</web-app>
```

Agenda

- Review of the existing system
- How the bulkloader works
- **How to create configurations**
- Advanced features
- Troubleshooting common issues
- Future directions
- Q&A

http://bulkloadersample.appspot.com/

Making configuration simple

- The quickest way to create your own bulkloader is to use the new create_bulkloader_config action.
- How does it know what to create?

Making configuration simple

- The quickest way to create your own bulkloader is to use the new create_bulkloader_config action
- How does it know what to create?
- Datastore Statistics

Making configuration simple

- The quickest way to create your own bulkloader is to use the new create_bulkloader_config action
- How does it know what to create?
- Datastore Statistics
- Which are themselves stored in the datastore
- create_bulkloader_config is itself a bulkloader config

Making configuration simple

Demo

```
appcfg.py create_bulkloader_config
  --filename bulkloader.yaml
  --url http://myapplication.appspot.com/remote_api
```

What to edit after the wizard is done?

Connector and settings

```
# Autogenerated bulkloader.yaml file.
# You're likely to have to do various edits to this file:
# At a minimum address the items marked with TODO:

Kind: Customer
connector: CSV # TODO: Choose a connector here
connector_options:
    # TODO: Add connector options here--specific to connector.
    encoding: windows-1252
```

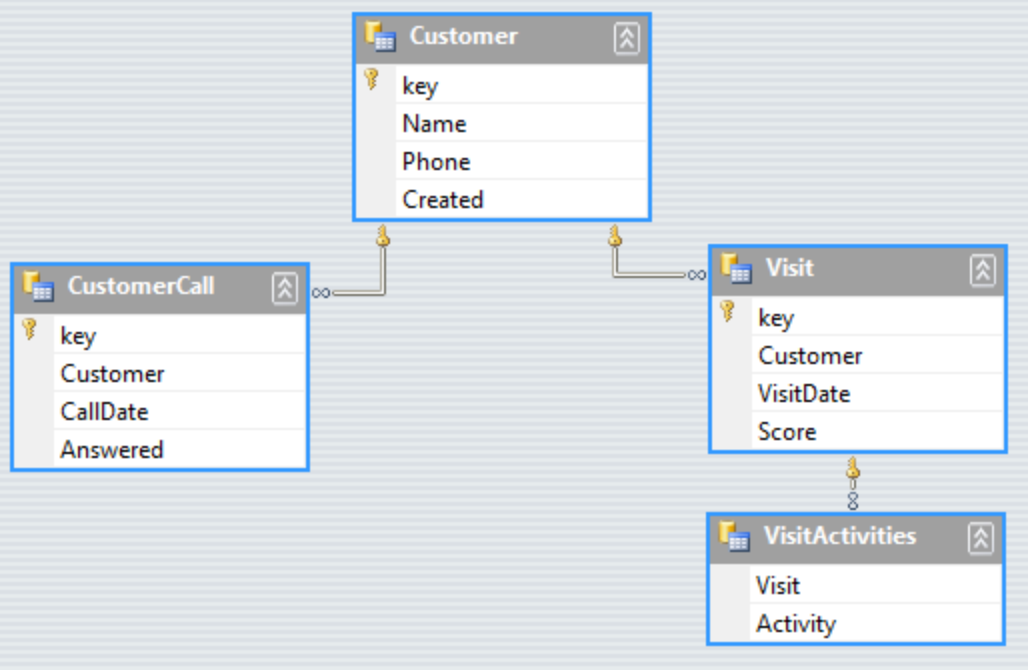
What to edit after the wizard is done?

Review property map

- The 'external_name' is the name of your CSV column, XML tag, etc.
- Check __key__, dupes, types, etc.

```
- property: __key__
  external_name: key
  export_transform: transform.key_id_or_name_as_string
- property: answered
  external_name: answered
  # Type: Boolean Stats: 3 properties of this type in this kind.
  import_transform: transform.regexp_bool('true', re.IGNORECASE)
```

Keys: Primary and Foreign



Keys: Entity and Reference

- Entity Key (~Primary Key)
- Reference Key (~Foreign Key)
- Entity Groups or Owned Relationships
- Several styles
  - Autogenerated (numeric) keys
  - Developer named keys
  - Parent (owned) relationships

- These are typically the key to creating relations

Keys: Entity and Reference

- A common case is a named key with no parent.
- Entity (primary) key: `property: __key__`
- Import
  - A string on import will be used as the key name for __key__
  - `transform.create_foreign_key` for a reference property
- Export
  - `transform.key_id_or_name_as_string` : safe export
  - works on __key__ and reference properties in entities

Keys: Entity and Reference

```
class Visit(db.Model):
  customer = db.ReferenceProperty(Customer)

- kind: models.Visit
  property_map:
    - property: __key__
      external_name: visitid
      export_transform: datastore.Key.name
    - property: customer
      external_name: customer
      import_transform:
        transform.create_foreign_key('Customer')
      export_transform: datastore.Key.name
```

Keys with Parent Keys

For Owned Relationships, Entity Groups

```
property_map:
- property: __key__
  external_name: Activity
  import_transform:
    transform.create_deep_key(('Visit', ‘Visit'),
      ('VisitActivity', ‘Activity’))
  export:
  - external_name: Visit
    export_transform: transformhelper.extract_deep_key(1)
  - external_name: Activity
    export_transform: transformhelper.extract_deep_key(2)
```

List Properties

```
class Visit(db.Model):
  ...
  activities = db.ListProperty(str)
```

Adding more options

List Properties

- Lists show up in auto-generated configurations with far more properties than non-lists.

```
- kind: Visit
  property_map:
    - property: __key__
    - property: activities
      # Type: String Stats: 90 properties of this type in this kind.
    - property: customer
      # Type: Key Stats: 42 properties of this type in this kind.
    - property: score
      # Type: Float Stats: 42 properties of this type in this kind.
    - property: visit_date
      # Type: Date/Time Stats: 42 properties of this type in this kind.
```

Adding more options

List Properties

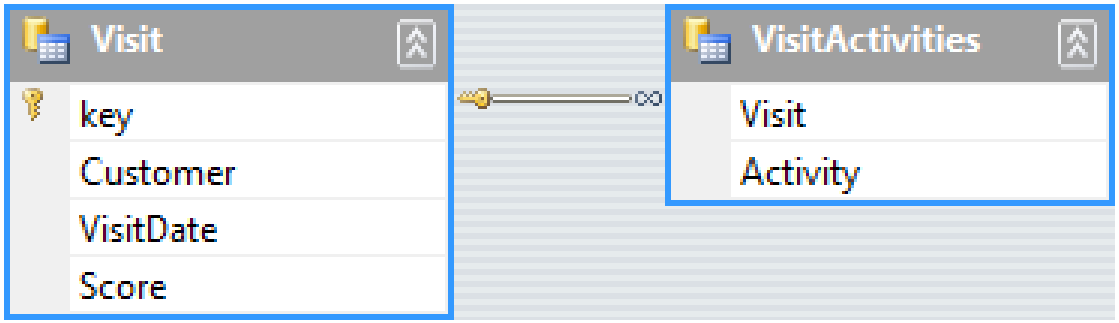
```
class Visit(db.Model):
  ...
  activities = db.ListProperty(str)

- kind: Visit
  property_map:
    ...
  - property: activities
    external_name: activities
    import_transform: transformhelper.split_string(';')
    export_transform: transformhelper.join_list(';')
```

Agenda

- Review of the existing system
- How the bulkloader works
- How to create configurations
- **Advanced features**
  - **Multiple Tables**
  - Writing your own code
  - More tips
- Troubleshooting common issues
- Future directions
- Q&A

http://bulkloadersample.appspot.com/

Multiple tables




```
class Visit(db.Model):
  customer = db.ReferenceProperty(Customer)
  visit_date = db.DateTimeProperty()
  score = db.FloatProperty()
  activities = db.ListProperty(str)
```

Multiple Tables

Using XML

```
<Visits>
 <Visit>
  <key>3</key>
  <Customer>jm</Customer>
  <Score>3.5</Score>
  <VisitDate>05/05/2010</VisitDate>
  <VisitActivities>
   <Activity>eat</Activity>
   <Activity>drink</Activity>
   <Activity>play</Activity>
   <Activity>sleep</Activity>
  </VisitActivities>
 </Visit>
</Visits>
```

Multiple Tables Using XML

```
connector: simplexml
connector_options:
  xpath_to_nodes: /Visits/Visit
  style: element_centric
```

```
<Visits>
 <Visit>
  <key>3</key>
  <Customer>jm</Customer>
  <Score>3.5</Score>
  <VisitDate>05/05/2010</VisitDate>
  <VisitActivities>
   <Activity>eat</Activity>
   <Activity>drink</Activity>
   <Activity>play</Activity>
   <Activity>sleep</Activity>
  </VisitActivities>
 </Visit>
</Visits>
```

Multiple Tables Using XML

```
connector: simplexml
connector_options:
  xpath_to_nodes: /Visits/Visit
  style: element_centric

property_map:
  . . .
- property: activities
  external_name: VisitActivities
  import_transform:
    transformhelper.list_from_xml_node('VisitActivities/Activity')
  export_transform:
    transformhelper.child_node_from_list('Activity‘)
```

```
<Visits><Visit>
  ...
  <VisitActivities>
   <Activity>eat</Activity>
   <Activity>drink</Activity>
   <Activity>play</Activity>
   <Activity>sleep</Activity>
  </VisitActivities>
</Visit></Visits>
```

Agenda

- Review of the existing system
- How the bulkloader works
- How to create configurations
- **Advanced features**
  - Multiple Tables
  - **Writing your own code: transforms and connectors**
  - More tips
- Troubleshooting common issues
- Future directions
- Q&A

http://bulkloadersample.appspot.com/

Custom Transforms

Inline or in a module

```
import_transform: lambda value: int(value) * 2
```

```
def double(value):
  return int(value) * 2

import_transform: mymodule.double
```

```
def multiply(multiple):
  def multiply_lambda(value):
    return int(value) * multiple
  return multiply_lambda

import_transform: mymodule.multiply(3)
```

Custom Transforms

Bulkload_state in transform functions

- Lets you see into the bigger picture:
  - current_dictionary
  - current_instance / current_entity
  - loader_opts
  - exporter_opts
  - filename

Custom Transforms: Using bulkload_state

- Simplified version of transform.create_deep_key

```
def create_key_from_multiple_columns(kind_columns):
  def transform_function(value, bulkload_state):
     path = []
     for kind, column in kind_columns:
       path.append(kind);
       path.append(bulkload_state.current_dictionary[column])
     return Key.from_path(*path)
  return transform_function


kind_columns =
  (('rootkind', 'rootcolumn'), ('childkind', 'childcolumn'))

result = Key.from_path(
  'rootkind', d['rootcolumn'], 'childkind', d['childcolumn'])
```

Post-processing

post_import_function/post_export_function

- Arbitrary transform of data
- Filtering (e.g. on export)
- Creating additional entities on import, perhaps a second kind

Writing your own connector

CSV is not the whole world

- Other file formats
- Database connections
- Other web sites

- Implement connector_interface.ConnectorInterface

- (This interface may change)

Writing your own connector

Implement connector_interface.ConnectorInterface

```
class JsonConnector(connector_interface.ConnectorInterface):
  def __init__(self, options, name='unknown'):
    self.style = options.get('style')

  def generate_import_record(self, filename, bulkload_state):
    file_handle = open(filename)
    for line in file_handle.readlines():
      yield simplejson.loads(line)

  def initialize_export(self, filename, bulkload_state):
    self.filename = filename
    self.file_handle = open(filename, 'w')

  def write_dict(self, dictionary):
    self.file_handle.write(simplejson.dumps(dictionary) + '\n')

  def finalize_export(self):
    self.file_handle.close()
```

Agenda

- Review of the existing system
- How the bulkloader works
- How to create configurations
- **Advanced features**
  - Multiple Tables
  - Writing your own code
  - **More tips**
- Troubleshooting common issues
- Future directions
- Q&A

http://bulkloadersample.appspot.com/

Bulkloader Tricks

Resuming Downloads

- --db_filename
- --result_db_filename

- Also useful to re-run export if you change/fix the transform

Bulkloader Tricks

Tuning

- The actual client<->server process is parallel
- The download is run in parallel, then the data exported in serial.
- The upload is run in parallel including conversion.

Bulkloader Tricks

Tuning

- --bandwidth_limit
- --batch_size
- --num_threads

- It’s often useful to do --batch_size 1 and --num_threads 40
- On dev_appserver, use --num_threads 1

Agenda

- Review of the existing system
- How the bulkloader works
- How to create configurations
- Advanced features
- **Troubleshooting common issues**
- Future directions
- Q&A

http://bulkloadersample.appspot.com/

Troubleshooting

Common connection issues

- Test the remote_api url. It should say “This request did not contain a necessary header.” Not anything else.
- Make sure you’re using the remote_api url, not the app. (Appcfg will manage this for you.)

Troubleshooting

Debugging the config file

- The new syntax should print more useful errors.
- Use `--dry_run` to test that all your data can be converted.
- Upload a small set of data to the dev_appserver.
- Check it with http://localhost:8080/_ah/admin

Agenda

- Review of the existing system
- How the bulkloader works
- How to create configurations
- Advanced features
- Troubleshooting common issues
- **Future directions**
- Q&A

http://bulkloadersample.appspot.com/

The Future

Addressing the pain

- More transform helper methods
- More connectors
- Server side processing! The system will need only small adjustments to run on the server as a ‘map’ operation

Agenda

- Review of the existing system
- How the bulkloader works
- How to create configurations
- Advanced features
- Troubleshooting common issues
- Future directions
- **Summary and Q&A**

http://bulkloadersample.appspot.com/

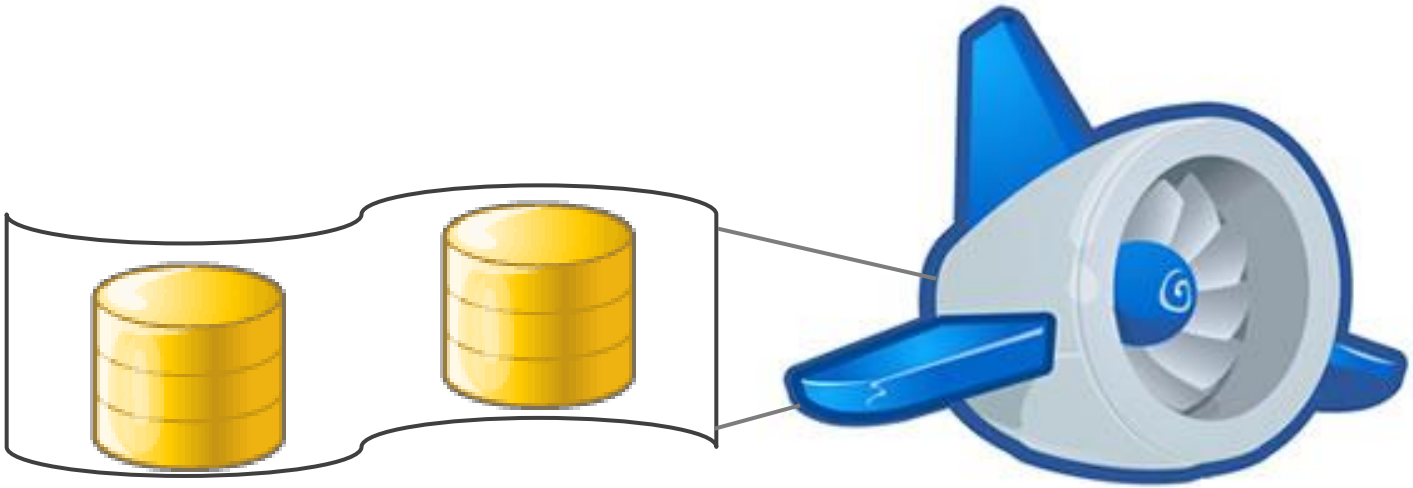
Bulkloader Review

- Runs on your workstation and moves data to the server
- No-config dump/restore everything, even across apps
- Auto create-config based on your datastore stats
- Lots of helpers in transform
- Keys are key

http://bulkloadersample.appspot.com/

Q&A

Moderator in Wave: http://bit.ly/appengine4



Google I/O 10

http://bulkloadersample.appspot.com/


Google™
10

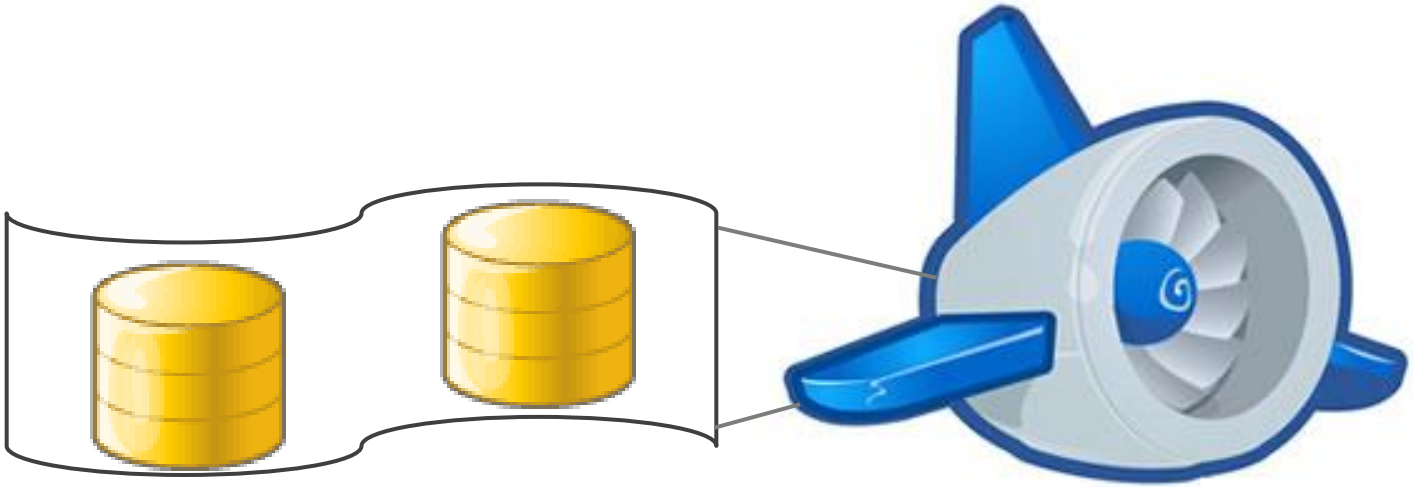
Appendix

http://bulkloadersample.appspot.com/

Bulkloader

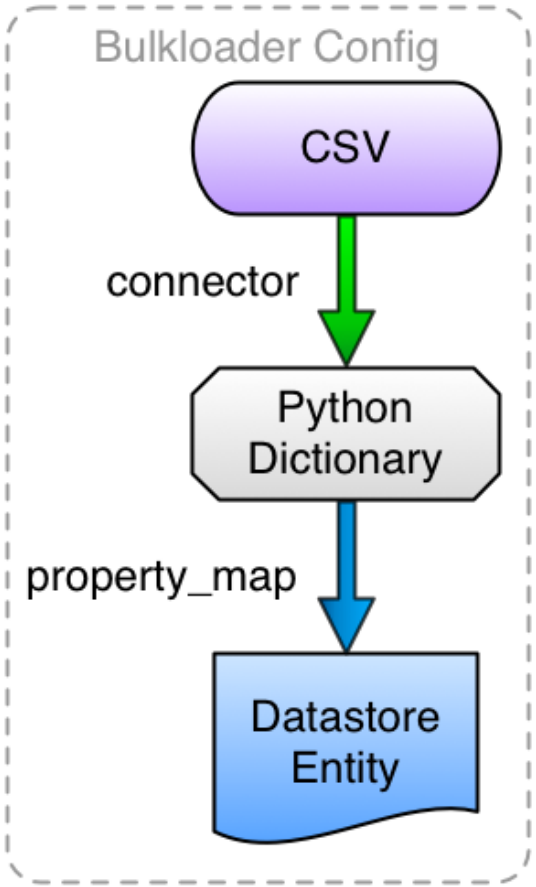
What's new?

- Non-Python-specific configuration
- Plug in multiple data formats



Details: Import/Export from Guestbook

How did that work?




```
content          date                 author
cool stuff,      20100418T01:30,      demo@mblain.com
```

```
{'content': 'cool stuff',
 'date':    '20100418T01:30',
 'author':  'demo@mblain.com'}
```

```
{'content': 'cool stuff',
 'date': datetime.datetime(2010,4,18,1,30,0,0),
 'author': users.User(email='demo@mblain.com')}
```

Google I/O 10

Adding more options

Binary Data

- Binary data can be stored as a ByteStringProperty or BlobProperty.

```
- property: photo_name
  external_name: photo_name
- property: photo_data
  external_name: photo_filename
  export_transform:
    transform.blob_to_file('photo_name', 'photos')
```

- blobproperty_from_base64: “Cast” to the right type

```
- property: filecontents
  external_name: filecontents
  export_transform: base64.b64encode
  import_transform: transform.blobproperty_from_base64
```

Post-processing

post_import_function/post_export_function

- Filtering (e.g. on export)

```
def post_export (instance, dictionary, bulkload_state):
    if dictionary['access'] == 'secret':
      return None
    else:
      return dictionary
```

- Duplicate data, perhaps to a second kind.

```
def post_import(input_dict, instance, bulkload_state):
  child_instance = models.ChildKind(
    parent=instance,
    key_name=input_dict['childkey'],
    value=input_dict['childvalue'])
  return (instance, child_instance)
```

Advanced Configuration

Schemaless data

- Handling missing items is the most common case
  - `transform.none_if_empty` (*type*) (on import)
  - `transform.empty_if_none` (*type*) (on export)

- Other examples will need custom code

Bulkloader Tricks

Crazy data

- You can read the sqlite database of dumped entities
- You can even do this from within the export
  - The database has 3 columns: entity key, sort key, and the encoded entity
  - So you can load a few --dump'd tables in a later pass, use the sort key as an index, and create multi-table output
- You can also use remote_api
  - Datastore calls work, including model reference properties
  - At very high performance cost.

Advanced Configuration

Validation: Using a model

- Python models allow for validation and computation

```
python_preamble:
 - import: mymodels
transformers:
 - model: mymodels.Customer
```