GPFS / Spectrum Scale Python API @ UG 2016

Jez Tucker
Head of Research & Product Development

17 May 2016

‘ArcaPix’: Comprises ArcaStream & Sister-Company Pixit Media

ArcaPix Software Stack

GUI

Plugins

Middleware

Filesystem API

ArcaPix Software Stack

ArcaPix GPFS Python API

Headline Feature Support

- Filesets
- Snapshots
- Quotas
- AFM
- Storage Pools
- List Processing and Policies
- Light Weight Events Compliant
- Callbacks
- QoS (0.7 June ‘16)
- Node / Nodeclasses

Currently unsupported:

- mmdelfs
- mmdelnsd
- mmdelcluster

Installed on over ~100 clusters

Advantages of a Python API

Allows rapid creation of toolsets and workflow using familiar Python methods:

- Integrate GPFS operations with Project creation
- Creating Filesets, assigning associated Quotas
- Create / manage Snapshot rotations
- Fine grained File Attr (0.7 June)
- Policy Control (Placement and Management)
- Search metadata and produce reports
- Analyse filesystem state

Link with 3rd party Python toolsets / APIs:

- Fine grained data management
- Creating snapshots prior to automated data removal
- Analytics

Advantages of a Python API

Pythonic use across all object types is fully supported.

```
for fset in filesets.values():
    if ('genomics-lab') in fset.name:
        # do something
```

All API interfaces operate in a consistent, standardised manner.

All objects can be utilised directly as well as through the Cluster object: useful for small single use case scripts. E.G:

```
Cluster().filesystems['mmfs1'].filesets.new('mynewfileset')
```

vs

```
Fileset('mynewfileset', 'mmfs1').create()
```

GPFS clusters can be … huge.
The API lazy loads where applicable.

Filesets

Real World Issue

Customer: I ran out of inodes, how do I increase the inodes on all filesets *easily*?

Filesets

```
from arcapix.fs.gpfs import Cluster, IndependentFileset

threshold_pct = 80        # watermark for inode increasing
incr_pct = 20             # increase by pct
max_inode_incr = 50000    # do not increase by more than max_inode_incr

for fset in Cluster().filesystems['mmfs1'].filesets.independent().values():

        # Check if the fileset has breached its inode watermark
        if fset.allocInodes >= (fset.maxInodes * threshold_pct / 100.):

                # Increase the inodes of the fileset
                new_inodes_num = int(fset.maxInodes * incr_pct / 100.)

                # Ensure that the additional increase does not exceed the maximum inode increase
                if new_inodes_num > max_inode_incr:
                        new_inodes_num = max_inode_incr

                # Add the new allocation on top of the current maximum allocation
                new_inodes_num = new_inodes_num + fset.maxInodes

                # Make the change
                fset.change(maxInodes=new_inodes_num)
```

AFM

Filesets also support AFM operations

```
from arcapix.fs.gpfs import CacheFileset
import uuid

# Create an AFM fileset (Using default NFS protocol and read-only cache approach)
myfileset1 = CacheFileset('mmfs1', 'cache-fileset1', '/mmfs1/projects/project1','gw1')
myfileset1.create()

# Change the number of read threads
myfileset1.change(afmNumReadThreads=4)

# Create another AFM fileset, using GPFS protocol
myfileset2 = CacheFileset('mmfs1', 'cache-fileset2', '/remote/mmfs1/projects/project2',protocol='gpfs')
```

Snapshots

Real World Issue

Customer: HELP. I’ve ran out of space on my cluster.

Support: You have 11362 huge snapshots.

Customer: Ah. We need to be able to write to the fast pool NOW.

How do I delete all my test snapshots in my fast pool *easily*?

Snapshots

The API Way

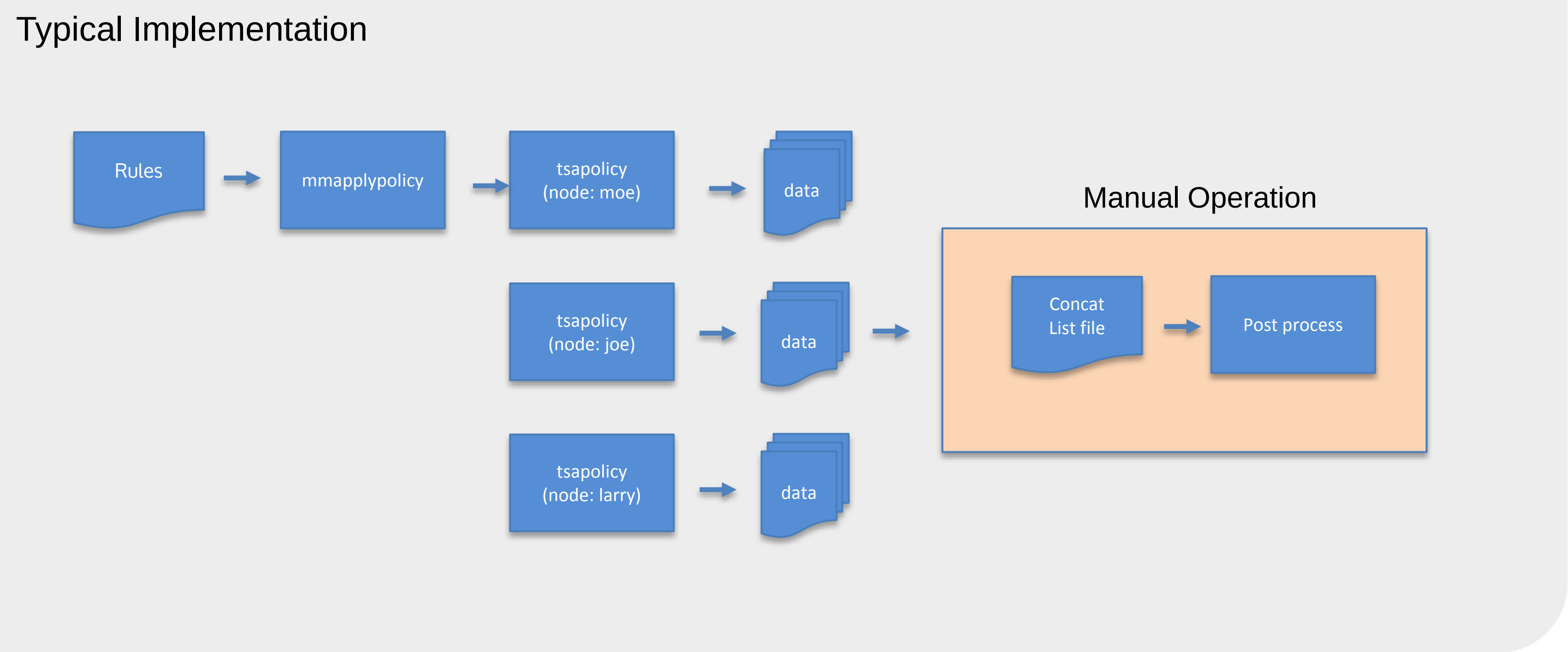
```
# Load the cluster
mycluster = Cluster()

# Delete all the test snapshots
for fset in mycluster.filesystems['mmfs1'].filesets.values():
    if fset.name.startswith('fast-'):
          for snap in fset.snapshots.values():
                    if 'test' in snap.name:
                              snap.delete()
```

Snapshots

Create snapshots compatible with SAMBA's vss_shadow_copy2

```
# Load the cluster
filesys = Cluster().filesystems['mmfs1']

# Iterate the filesystems filesets
for fset in filesys.filesets.independent().values():

        # Create a snapshot name compatible with SAMBA's vss_shadow_copy2
        today = datetime.datetime.today()
        s_name = datetime.datetime.strftime(today, '@GMT-%Y.%m.%d-%H.%M.%S')

        # Create the snapshot
        fset.snapshots.new(s_name)
```

List Processing the Archaic Way



AFM Cache Eviction via Conventional List Processing

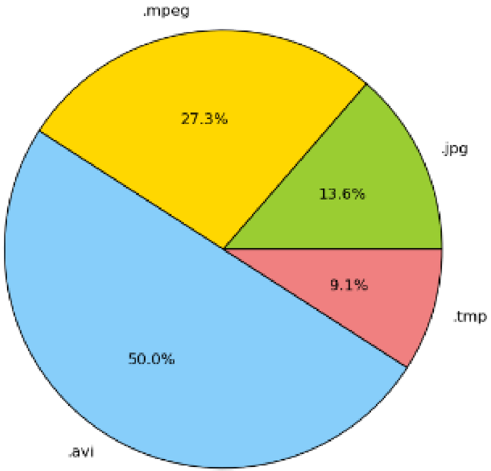
```
... (excerpt) ...

if getVersion() < '4.1.1':
   sys.exit(1)

mypolicy = ManagementPolicy()

extlist = mypolicy.rules.new(ExternalListRule, listname='all-files', script='mmafmctl evict --list-file')

polrules = mypolicy.rules.new(ListRule, listname=extlist.listname, sort='ACCESS_TIME')
polrules.change(show=Rule.show('KB_ALLOCATED', 'FILE_SIZE', 'MISC_ATTRIBUTES', 'FILESET_NAME',
      'ACCESS_TIME'))

polrules.criteria.new(Criteria.Regex('MISC_ATTRIBUTES','[u]'))
polrules.criteria.new(Criteria.like('FILESET_NAME', args.fileset))
polrules.criteria.new(Criteria.lt('access', args.lastaccess))

mypolicy.save('evict_afm_cache.pol', overwrite=True)
mypolicy.run(filesys, nodes='clusternode01')
```

Basic List Processing the API way

```
# Create a Management Policy
p = ManagementPolicy()

# Create a ListProcessing Rule
r = p.rules.new(ListProcessingRule, listname='temp_files_bytes', processor=lambda lst: sum(x.filesize for x in lst))

# Add criteria to specify filetype
r.criteria.new(Criteria.like('name', '*.tmp'))

# Run policy
print p.run('mmfs1')
{‘tmp_files_bytes': 208456737}
```

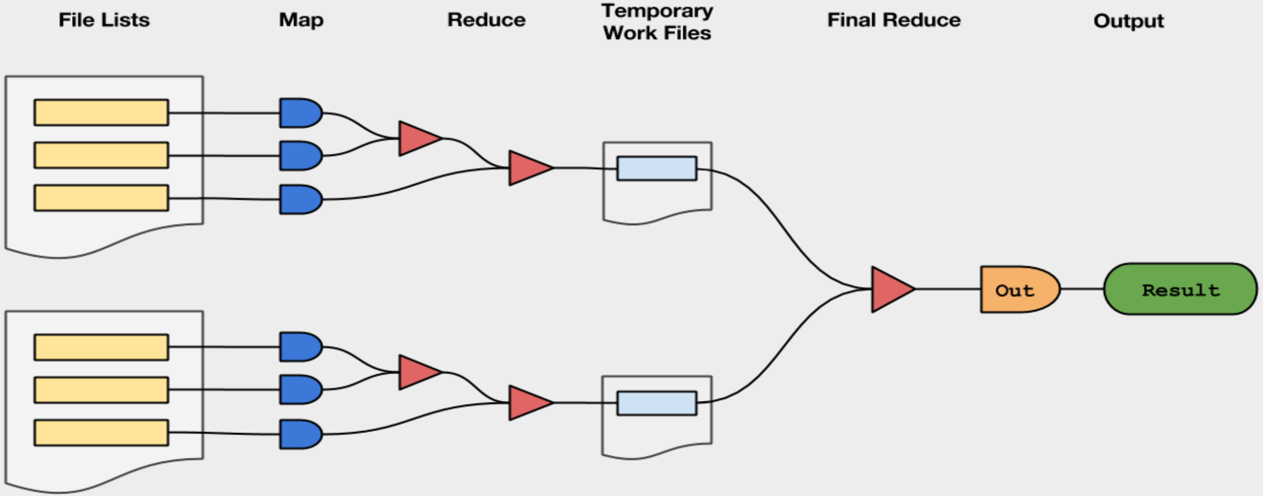
Advanced List Processing the API way

```
import matplotlib.pyplot as plt
from collections import Counter

p = ManagementPolicy()

def type_sizes(file_list):
  c = Counter()
  for f in file_list: c.update( { splitext(f.name) : f.filesize } )
  return c

r = p.rules.new(ListProcessingRule, 'types', type_sizes)
result = p.run('mmfs1')['types']

plt.pie(result.values(), labels=result.keys(),autopct='%1.1f%%')
plt.axis('equal')
plt.show()
```



List Processing the API way

Optimal List Processing is Achieved via MapReduce



Advanced List Processing with the API: “SNAPDIFF”

```
from arcapix.fs.gpfs import ManagementPolicy, MapReduceRule

# List processing helper function
def helper(fobj):
   # used to compare files by name / path name / creation time
   return [(fobj.name, fobj.pathname.split('/', 4)[-1], fobj.creation)]

# Create rule to find and process files (can use same rule for both searches)
p = ManagementPolicy()
p.rules.new(MapReduceRule, 'snap', mapfn=helper, output=set)

# Interrogate the snapshots
out_old = p.run('/mmfs1/.snapshots/snap1/testdata')['snap']
out_new = p.run('/mmfs1/.snapshots/snap2/testdata')['snap']

# Diff
deleted = sorted(i[0] for i in (out_old - out_new))
created = sorted(i[0] for i in (out_new - out_old))
unchanged = sorted(i[0] for i in (out_old & out_new))
```

List Processing Demo: “SNAPDIFF”

Basic Callbacks

Real-World Issue Refresh

Customer: I ran out of inodes, how do I increase the inodes as required on all filesets *easily*?

Qu: How do I make sure this *never* happens again?

Ans: Utilise a Callback

```
Cluster().callbacks.new(callbackId='increase_inodes', command='increase_inodes.py',
events='noDiskSpace', parms=['%filesetName'])
```

```
root@elmo:~$ mmlscallback
increase_inodes
    command       = /opt/arcapix/callbacks/increase_inodes.py
    event         = noDiskSpace
    parms         = %filesetName
```

A somewhat effective method, but only when we’ve actually run out of space.
Let’s improve on that...

Advanced Callbacks (..putting it all together..)

```
...
soft_quota_pct = threshold_pct   # the per-fileset SoftQuota inode threshold, sensibly the same as the threshold_pct
hard_quota_pct = 95              # the per-fileset HardQuota inode threshold.

# This function will be called when the callback fires
def increase_max_inodes(fsName, filesetName):

        filesys = Filesystem(fsName)
        fset = filesys.filesets[filesetName]

        if fset.allocInodes >= (fset.maxInodes * threshold_pct / 100.):
                new_inodes_num = int(fset.maxInodes * incr_pct / 100.)

                if new_inodes_num > max_inode_incr:
                        new_inodes_num = max_inode_incr

                new_inodes_num = new_inodes_num + fset.maxInodes

                new_soft_inode_quota = int(new_inodes_num * soft_quota_pct / 100.)
                new_hard_inode_quota = int(new_inodes_num * hard_quota_pct / 100.)

                try:
                        fset.change(maxInodes=new_inodes_num, filesSoftLimit=new_soft_inode_quota,
                                filesHardLimit=new_hard_inode_quota)

                except GPFSExecuteException:
                        pass #alert here via your method of choice
```

Advanced Callbacks (.. putting it all together..)

Installing the Callback

```
...

if isinstance(fset, IndependentFileset):

        # Set an initial Quota
        soft = int(fset.maxInodes * soft_quota_pct/100.)
        hard = int(fset.maxInodes * hard_quota_pct/100.)
        fset.quotas.fileset.new(filesSoftLimit=soft, filesHardLimit=hard)

        # Set the Callback on the Fileset
        fset.onSoftQuotaExceeded.new(callbackId='increase_max_inodes-%s' % fset.name,
command=increase_max_inodes)

else:
        print 'Auto-increases via quota triggers can only be set on Independent Filesets.'
        sys.exit(1)

# Exit nicely
sys.exit(0)
```

Advanced Callbacks (.. putting it all together..)

The Callback is stored in GPFS itself, not on the disk.

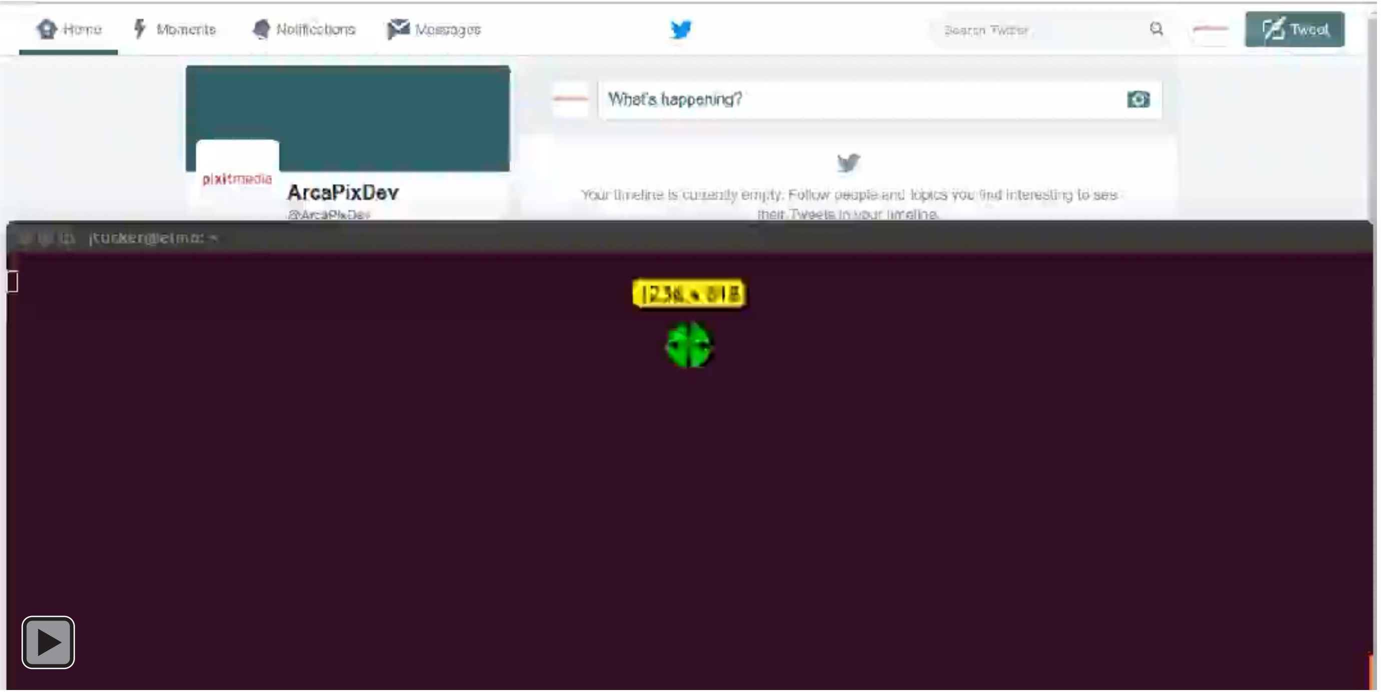
```
/usr/lpp/mmfs/bin/mmlscallback
lowspace_increase_inodes
    command = /usr/lib/python2.7/site-packages/arcapix/fs/gpfs/callbackdriver.py
    event = softQuotaExceeded
    node = elmo
    parms = Z0mefUs4K4to4Y0yPcit6qZZnnfTaaFvIkUQ3l4ZJgPbnQzs35zaukkz78kehCNzukhenn5denQer...
```

Upon de-serialising the values of variables in the Callback are *identical* to the moment prior to serialising.
Utilise this functionality for checking state before and when the Callback fires.

The API also supports *per-Fileset* Callbacks

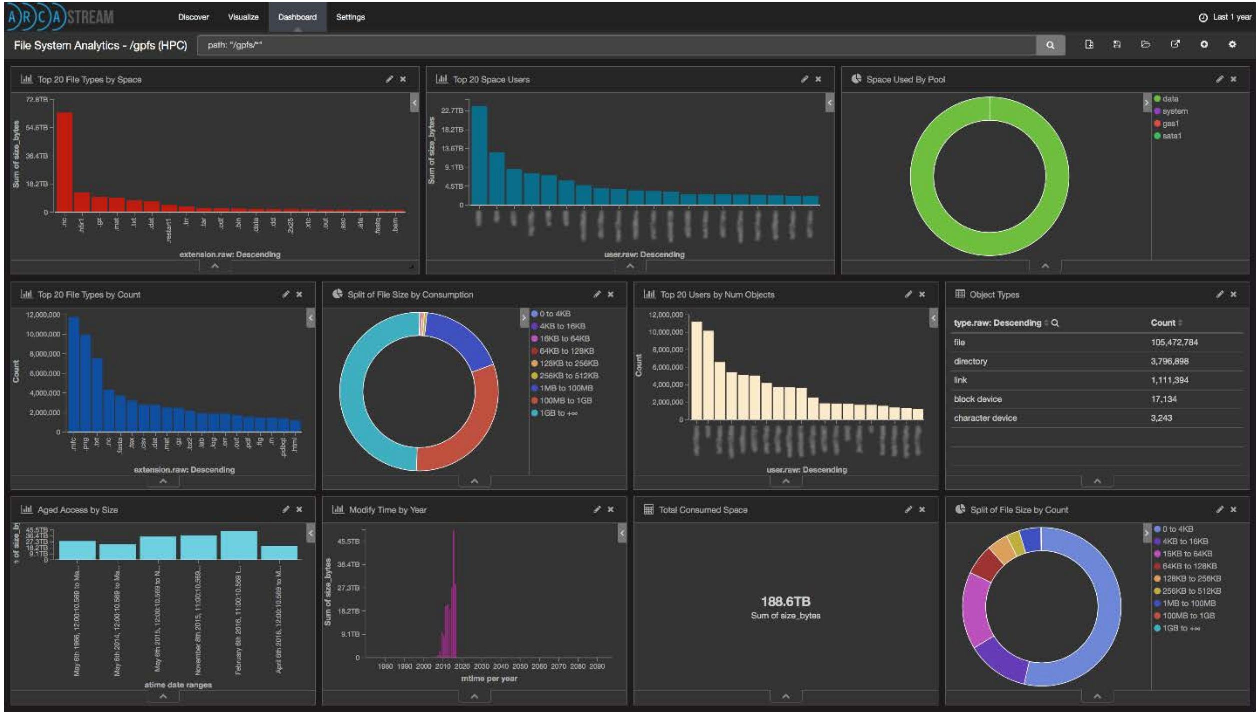
```
/usr/lpp/mmfs/bin/mmlscallback
lowspace_increase_inodes
    command = /usr/lib/python2.7/site-packages/arcapix/fs/gpfs/callbackdriver.py
    event = softQuotaExceeded
    node = elmo
    parms = RESTRICT myfilesetname Z0mefUs4K4to4Y0yPcit6qZZnnfTaaFvIkUQ3l4ZJgPbnQzs35z...
```

Callbacks Demo



Your timeline is currently empty. Follow people and topics you find interesting to see their Tweets in your timeline.

One more thing…



type.raw: Descending	Count
file	105,472,784
directory	3,796,898
link	1,111,394
block device	17,134
character device	3,243

- Approximately 1 hour to create an end-to-end Analytics pipeline via the API
- Utilises MapReduce API methods
- Data is parsed and mutated on-the-fly

One more thing…

We also support the low-level GPFS C Library via a Python API

Inode Stat

```
from arcapix.fs.gpfsclib import stat
stat('/mmfs1/.policytmp/testclone1')
gpfsclib.stat(blocks=256, ctime=1461061390, blocksize=4194304, rdev=0, dev=61334, nlink=2, gid=0,
mode=33279, mtime=1456850352, uid=0, atime=1461946936, inode=26368, size=1394)
```

Inode Scanning

```
from arcapix.fs.gpfsclib import inode_scan
      for i in inode_scan('mmfs1'):
            print "%010d/%010d\t%010u\t%010u\n" % (i.inode, i.generation, i.mtime, i.ctime)

0000000003/0000000001     1462870041   1462870041
0000000042/0000065538     1448989426   1448989426
0000000043/0000065538     1448989427   1448989427
0000000044/0000065538     1448989427   1448989427
0000004038/0779965892     1459948505   1459948505
```

- Translates C methods (open_inodescan, next_inode) into Python idioms (iterator)
- Implements ~50 lines of GPFS' samples 'tstimes.c' example script in 3 lines of Python

One more thing…

Middleware

REST capable interface provides rapid and consistent methods to:

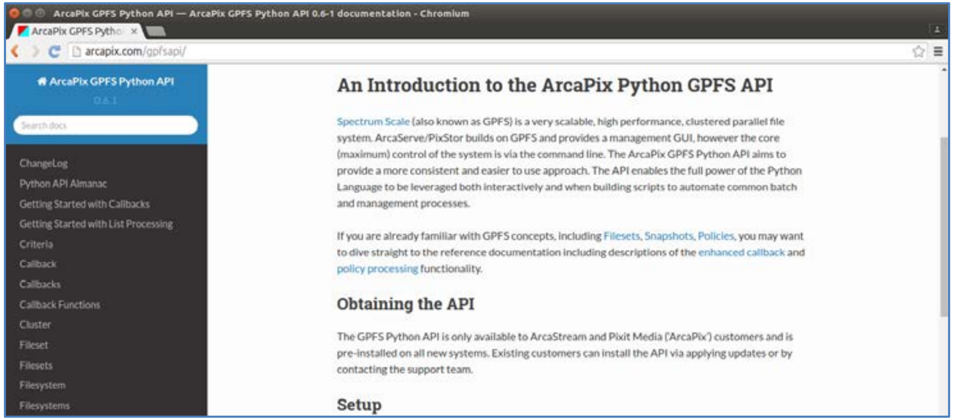
- Query data
- Create / Delete / Modify via the underlying API(s)
- Build your own custom interfaces
- Provide workflow enhancements to current toolsets

API Examples & Documentation

All the examples from this presentation are available at:

http://github.com/arcapix/gpfsapi-examples

API Documentation is available at: http://arcapix.com/gpfsapi



ArcaPix GPFS Python API — ArcaPix GPFS Python API 0.6-1 documentation - Chromium
ArcaPix GPFS Pytho ×
arcapix.com/gpfsapi/
ArcaPix GPFS Python API
0.6.1
Search docs

- ChangeLog
- Python API Almanac
- Getting Started with Callbacks
- Getting Started with List Processing
- Criteria
- Callback
- Callbacks
- Callback Functions
- Cluster
- Fileset
- Filesets
- Filesystem
- Filesystems

An Introduction to the ArcaPix Python GPFS API

Spectrum Scale (also known as GPFS) is a very scalable, high performance, clustered parallel file system. ArcaServe/PixStor builds on GPFS and provides a management GUI, however the core (maximum) control of the system is via the command line. The ArcaPix GPFS Python API aims to provide a more consistent and easier to use approach. The API enables the full power of the Python Language to be leveraged both interactively and when building scripts to automate common batch and management processes.

If you are already familiar with GPFS concepts, including Filesets, Snapshots, Policies, you may want to dive straight to the reference documentation including descriptions of the enhanced callback and policy processing functionality.

Obtaining the API

The GPFS Python API is only available to ArcaStream and Pixit Media ('ArcaPix') customers and is pre-installed on all new systems. Existing customers can install the API via applying updates or by contacting the support team.

Setup

www.arcastream.com
www.pixitmedia.com

arcapix.com/gpfsapi