Paper 317-2011

The Day the Server Said Tweet!

William W. Viergever, Viergever & Associates, California
Daniël Kuiper, SAS Institute B.V., The Netherlands
Koen Vyverman, SAS Institute B.V., The Netherlands

ABSTRACT

It was an idyllic summer afternoon at the SAS® Netherlands Technical Support call center. The pervasive hum and buzz of problem-solving activity was punctuated by the shrill cries of woodland birds drifting in through the open windows. Then, a dormant screen sprang to life, and panicky error messages and warnings started scrolling by! A distant siren wailed. Support staff gathered swiftly around the screen. It was the MIS server tweeting messages of impending doom! A SAS® Portal guru sat down at a console, logged on to the server, diagnosed, and fixed the problem. Not five minutes later, the screen went black again and relative serenity returned. The Managing Director had not even noticed the portal's hiccups. This paper shows how to use Twitter to let your mission-critical SAS systems tweet their warnings and errors straight to your local support team for immediate follow up, and thus helps to minimize downtime!

INTRODUCTION

System administrators typically have a variety of tools at their disposal to monitor the health of the systems and applications which they are support and maintain for their business users. Some of these monitoring solutions involve physically inspecting a reporting and diagnostics console application. Others are centered around a mechanism that sends diagnostic e-mail messages to the administrators. SAS Platform administrators in particular have access to the SAS 9.2 Enterprise Business Intelligence Audit and Performance Measurement (EBIAPM) utilities package, which monitors a number of aspects of a SAS Business Analytics server environment that are critical to the health of the system: is the SAS OLAP Server running? Is the Portal up? Is SAS Web Report Studio working?

Whatever the monitoring tool being used however, the bottom-line is always that an administrator first needs to notice that something is out of kilter. They will then begin to investigate what's going on, and attempt to fix things. While this will work in some cases, a full-fledged SAS Business Analytics environment can be a complex thing, and troubleshooting will take time, perhaps to eventually lead to the conclusion that SAS Technical Support needs to be involved. Valuable time is lost, the problem may become aggravated, and a business critical system goes from being merely unstable to being truly unavailable.

In this paper we present a mechanism to dramatically shorten the time between a problem occurring — or merely threatening to occur! — and your local Support team being on the case and potentially resolving the situation before it becomes a real issue, and hence reduce or avoid downtime. The idea is in essence a simple one: a server in the SAS environment (hereafter: the Server) tweets real-time diagnostic information to a specific Twitter account. The Support team receive these tweets on a dedicated screen and take action as needed! The main advantage of tweeting rather than sending diagnostic e-mails to the Support center is that the tweets are instantly visible whereas e-mails will sit in the incoming queue amidst a zillion other messages until they are read and processed by the operational dispatcher. Ideally, the two mechanisms complement one another. Bursts of tweets appearing on a monitor will attract Support staff attention, while an accompanying e-mail may contain more detail about the situation…

The first chapter of this paper explains what needs to be done on the Twitter side: setting up Twitter accounts for the Server and the Support center; ensuring that only the Support center is able to read the Server's tweets; an introduction to OAuth, the authentication mechanism used by Twitter, and the various OAuth implementations available on the web. In the following chapter we choose a specific OAuth for Twitter implementation using PHP, and explain how to set up the PHP environment and scripts.

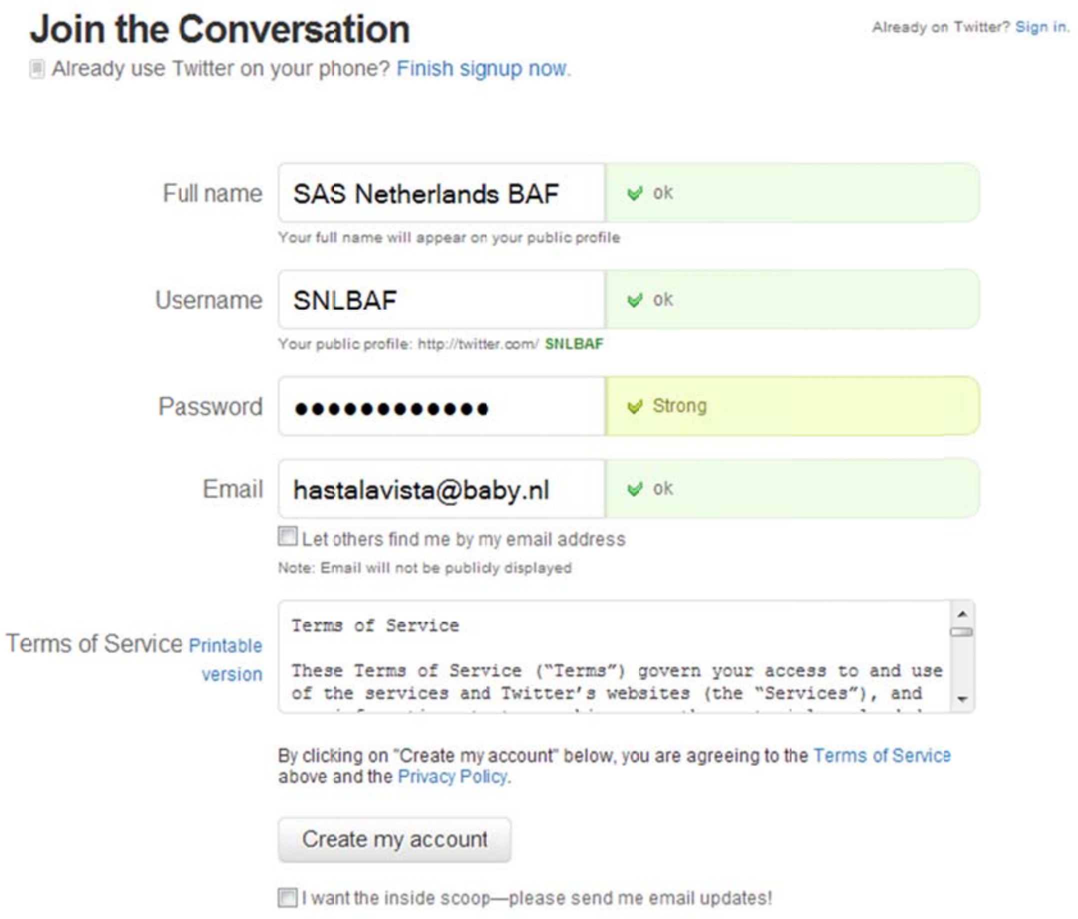
The next chapter explores three practical examples of Server-side SAS batch processes detecting certain worrisome events and tweeting about these: running out of disk space on the WORK volume; an important SAS server process going down; performance degradation by runaway user processes. We conclude the paper with some remarks about practicalities and suggestions for future development.

Throughout this paper we assume that both Server and client machines are running some Windows OS. On other operating systems, the necessary utilities will need to be replaced by suitable alternatives. The SAS code was developed and tested in a SAS 9.2 Maintenance 3 environment.

PREREQUISITES 1 – TWITTER

THE BASICS

Twitter is arguably the most popular micro-blogging service nowadays. Hundreds of millions of humans use it to send 140-character messages — 'tweets' — out into the aether. And, judging from a cursory examination of the daily Twitter message volume, there are also quite a few applications, bots, and assorted automatons contributing to the sonorous cacophony. We want our Server to tweet to an audience of one though, not to the entire world. On the Twitter site (www.twitter.com) we sign up for a new account. (Fig. 1)



Join the Conversation

Already on Twitter? Sign in.

☐ Already use Twitter on your phone? Finish signup now.

Full name: SAS Netherlands BAF ✓ ok

Your full name will appear on your public profile

Username: SNLBAF ✓ ok

Your public profile: http://twitter.com/ SNLBAF

Password: ●●●●●●●●●●●● ✓ Strong

Email: hastalavista@baby.nl ✓ ok

☐ Let others find me by my email address

Note: Email will not be publicly displayed

Terms of Service Printable version

Terms of Service

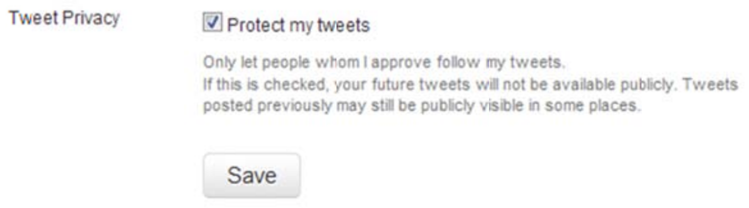
These Terms of Service ("Terms") govern your access to and use of the services and Twitter's websites (the "Services"), and

By clicking on "Create my account" below, you are agreeing to the Terms of Service above and the Privacy Policy.

☐ I want the inside scoop—please send me email updates!

Figure 1. Create the SNLBAF Twitter account for the Server.

So the Server's Twitter account is SNLBAF, as in SAS Netherlands Business Analytics Framework. Note that we have unchecked all unnecessary checkboxes. We log on to the Twitter site as SNLBAF and edit the account settings:



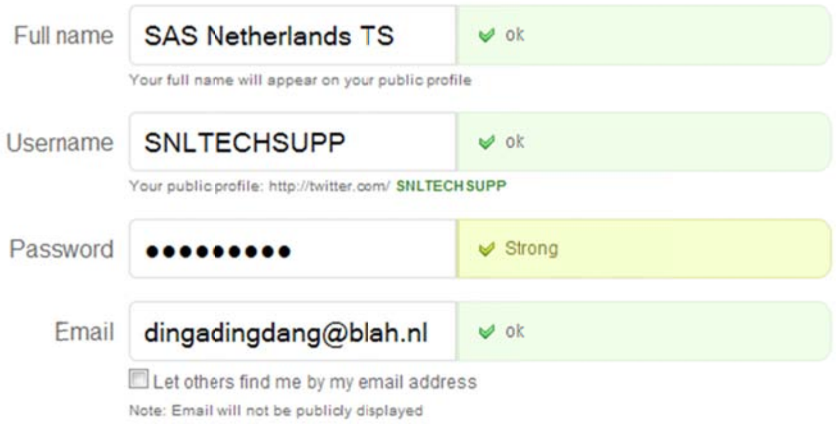
Tweet Privacy

☑ Protect my tweets

Only let people whom I approve follow my tweets.
If this is checked, your future tweets will not be available publicly. Tweets posted previously may still be publicly visible in some places.

Figure 2. SNLBAF Twitter account settings.

Tweet privacy is off by default, and the setting (Fig. 2) to protect the tweets is vital to keeping things private between the Server and the Support center! We proceed to log off from Twitter, and in similar vein sign up for another account to be used by the Support center: SNLTECHSUPP. (Fig. 3)



Full name: SAS Netherlands TS ✓ ok

Your full name will appear on your public profile

Username: SNLTECHSUPP ✓ ok

Your public profile: http://twitter.com/ SNLTECHSUPP

Password: ●●●●●●●●● ✓ Strong

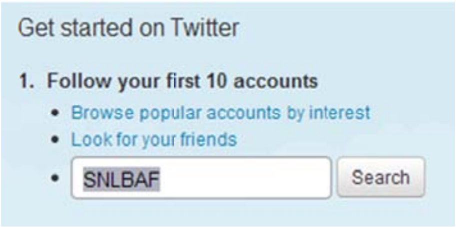
Email: dingadingdang@blah.nl ✓ ok

☐ Let others find me by my email address

Note: Email will not be publicly displayed

Figure 3. Create the SNLTECHSUPP Twitter account for the Support center.

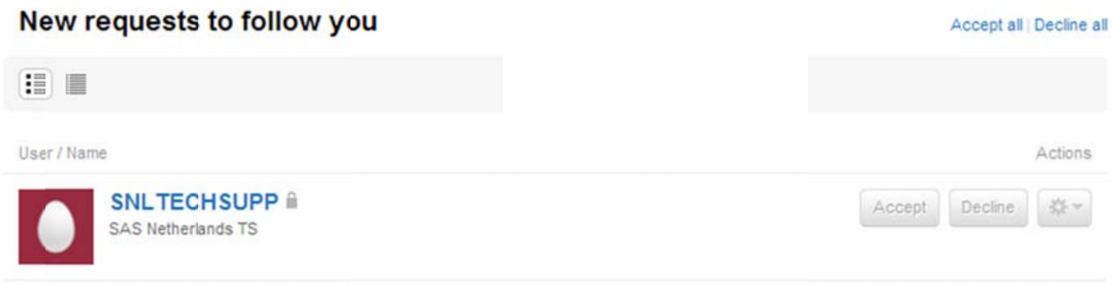
Just like we did for SNLBAF, we log on to the Twitter site as SNLTECHSUPP and activate Tweet Privacy. Note that while setting up the accounts, it is possible to define interests and import friends. No need for all that however, we just want a bare account. While logged on as SNLTECHSUPP, we search for SNLBAF whose tweets we want to follow, and send a request to follow SNLBAF. (Fig. 4)



Get started on Twitter

1. Follow your first 10 accounts
    - Browse popular accounts by interest
    - Look for your friends
    - SNLBAF Search

Figure 4. Send a follow-request to SNLBAF.



New requests to follow you

SNLTECHSUPP

SAS Netherlands TS

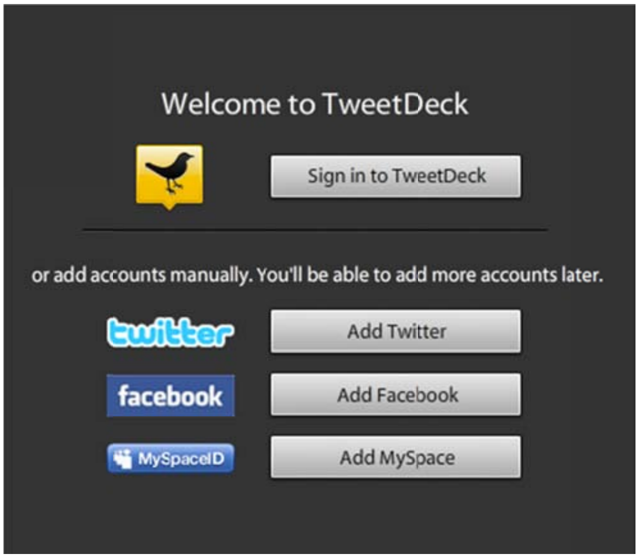
Figure 5. SNLBAF has a pending follow request from SNLTECHSUPP.

Logging on again as SNLBAF we see that there is a pending request. (Fig. 5) After accepting the request, SNLBAF now has 1 follower, viz. SNLTECHSUPP.

To display the Server's tweets at the Support center, we deployed a freeware application called TweetDeck. (www.tweetdeck.com) After installing TweetDeck on a Support center machine and launching it for the first time, the application offers to sign in with a TweetDeck account, or alternatively, to add a social network account. (Fig. 6)

There is no need to create a TweetDeck account. We simply click the 'Add Twitter' button and proceed to enter the SNLTECHSUPP Twitter account credentials. (Fig. 7)

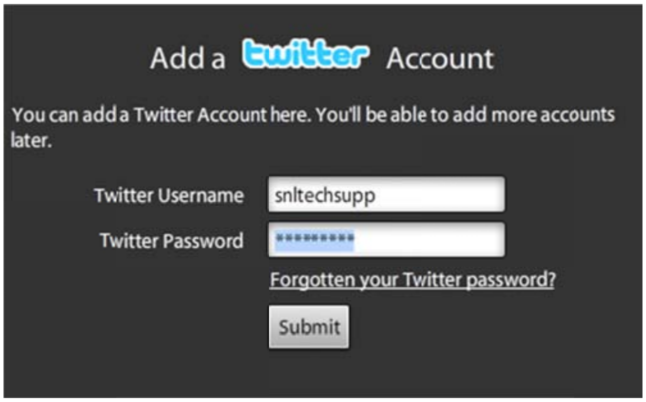
We then log on to the Twitter site as SNLBAF and send a test tweet. (Fig. 8)



Welcome to TweetDeck

or add accounts manually. You'll be able to add more accounts later.

Figure 6. Upon running TweetDeck.



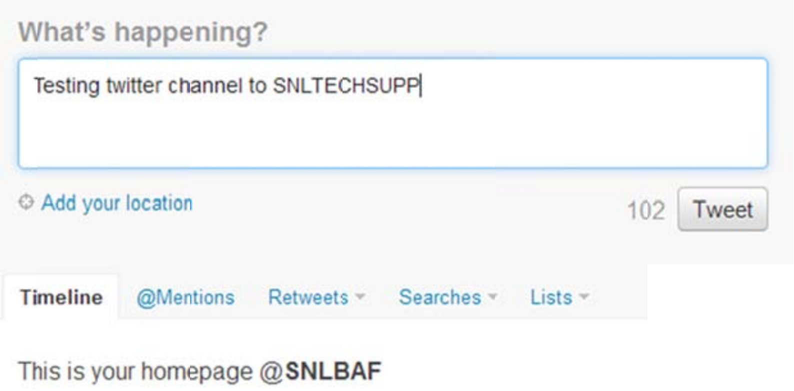
Add a twitter Account

You can add a Twitter Account here. You'll be able to add more accounts later.

Twitter Username: snltechsupp

Twitter Password: *********

Figure 7. Adding a Twitter account to TweetDeck.



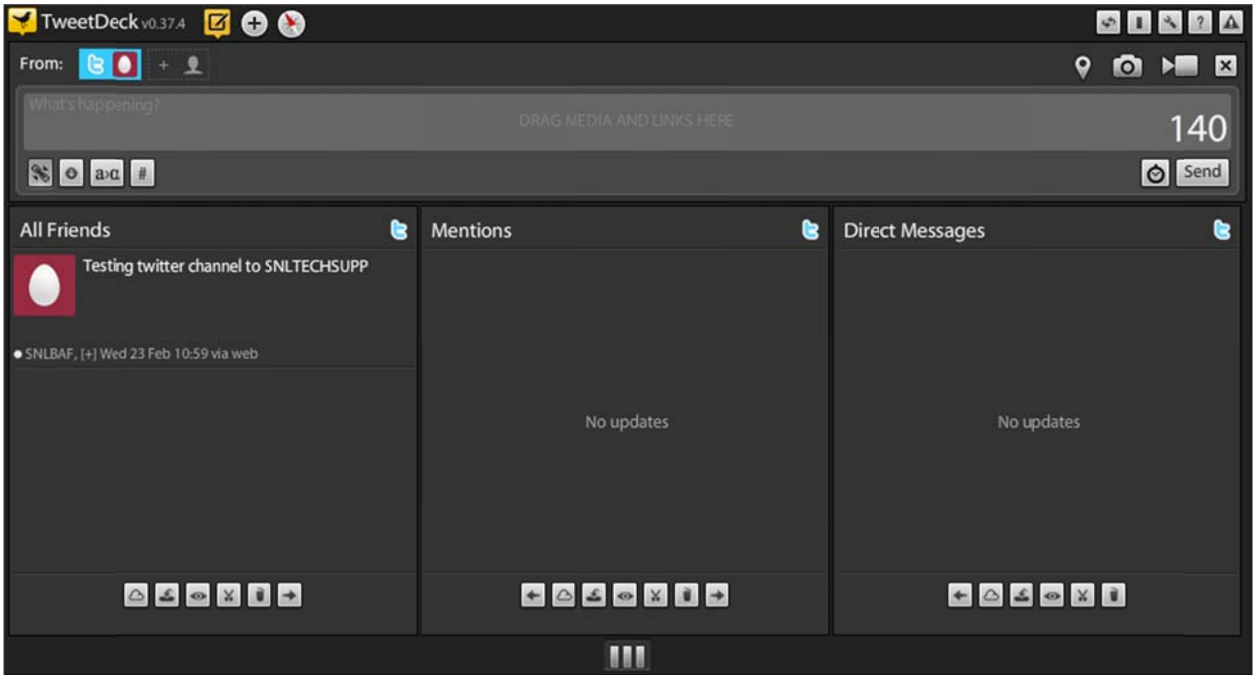
What's happening?

Testing twitter channel to SNLTECHSUPP

This is your homepage @SNLBAF

Figure 8. A test tweet from SNLBAF.

The link is made! (Fig. 9) Whatever SNLBAF tweets is instantly displayed in SNLTECHSUPP's TweetDeck. So far, so good. But remember that we need SNLBAF to generate its tweets through code, and not by typing stuff into the Twitter site. Which leads us to the matter of how Twitter handles authentication…



Testing twitter channel to SNLTECHSUPP

SNLBAF, [+] Wed 23 Feb 10:59 via web

No updates

No updates

Figure 9. SNLTECHSUPP can see the test tweet from SNLBAF in TweetDeck.

OAUTH

Until the middle of 2010, Twitter used a very basic authentication mechanism (Basic Auth) and life was easy. It also wasn't very secure, but at least it was easy to send a tweet from a piece of SAS code simply by means of PROC HTTP. Since then however, Twitter has ditched Basic Auth in favour of the open authorization standard OAuth (www.oauth.net) which is a different beast altogether.

For those keen on understanding OAuth in its full glory, we refer to the Beginner's Guide to OAuth which is accessible from the aforementioned site. Suffice it to say that it is an authentication mechanism that allows users to permit applications to act on their behalf without actually providing the user's credentials. Instead, tokens and token-secrets are being exchanged.

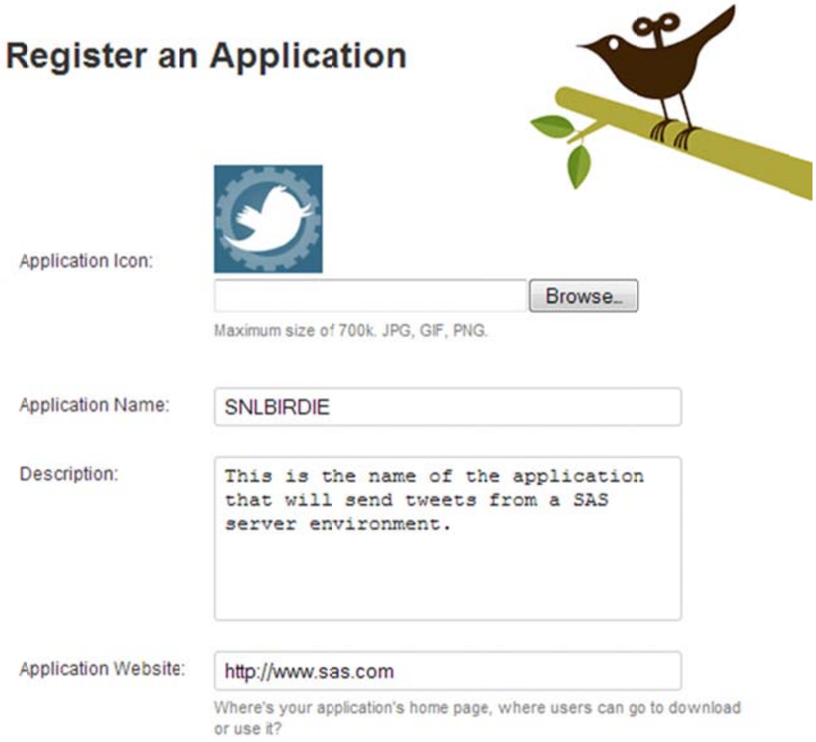
Somewhat oversimplified perhaps, but what happens is essentially this: we need a SAS program (in OAuth speak: the Application) to be able to send tweets. That is, the SAS program must use the Twitter service on behalf of the Twitter user SNLBAF. In order to do that, the SNLBAF Twitter user tells Twitter that it is OK for the application (which we've named SNLBIRDIE) to pretend that it is SNLBAF. Twitter then gives the Application a bunch of stuff to authenticate itself: an OAuth token, an OAuth token secret, and a PIN code. Whenever the Application needs to tweet it will use these, and there's no need for the Application to be aware of the actual SNLBAF credentials.

Clever, huh? There is more to it of course. The Twitter developers site (see Recommended Reading) has a great OAuth flow diagram describing the seven steps involved in authentication. Furthermore, the OAuth protocol relies on the HMAC-SHA1 message authentication code to guarantee the integrity and authenticity of transmitted messages. SHA1 is a secure hash algorithm well-known in cryptography. While we encourage the reader to explore these concepts further, any attempt at explaining them is way out of scope for this paper…

Luckily we don't need to code SHA-1 encryption and HMAC message authentication from scratch, because others have done this before although not in SAS. The Twitter Developers site lists a slew of publicly available OAuth libraries in a variety of languages: C/C++, JAVA, Perl, PHP, Python, Ruby, you name it. We investigated a few of these, keeping in mind that the authentication in itself is not one of the major points of interest for this paper, and that the chosen variety should integrate with as little fuss as possible with our applications written in SAS code. We settled on a PHP-based solution and a couple of really easy PHP-scripts by one Joe Chung.

But before we get to the PHP part of our set-up, we need to register our tweet-sending application as a Twitter OAuth client via http://twitter.com/oauth_clients/new (Fig. 10) This is where we tell Twitter that an application named SNLBIRDIE is going to be given permission to act on the behalf of the Twitter user SNLBAF (which we are presently signed in with). The fact that the application itself is at this point nonexistent is irrelevant. We merely register it by its name SNLBIRDIE, and make some choices.

Twitter recognizes two application types, client and browser (Fig. 11). Our SAS processes are not going to run from a browser so we pick 'client' as the application type. An odd choice perhaps in view of the fact that they'll merely be bits of code running in batch on some server without any front-end, but the right choice nonetheless. Even though we choose 'client', the registration form insists on an application website, so we enter a popular one just to comply. We also set the default access type to ‘Read & Write’, which will enable SNLBIRDIE to use both HTTP GET and HTTP POST request methods in its interactions with the Twitter API.



Register an Application

Application Icon:

Maximum size of 700k. JPG, GIF, PNG.

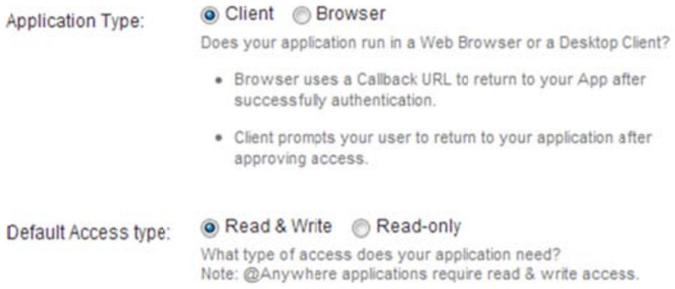
Application Name: SNLBIRDIE

Description: This is the name of the application that will send tweets from a SAS server environment.

Application Website: http://www.sas.com

Where's your application's home page, where users can go to download or use it?

Figure 10. The Twitter OAuth Application Registration Form.



Application Type: ● Client ○ Browser

Does your application run in a Web Browser or a Desktop Client?

- Browser uses a Callback URL to return to your App after successfuly authentication.
- Client prompts your user to return to your application after approving access.

Default Access type: ● Read & Write ○ Read-only

What type of access does your application need?
Note: @Anywhere applications require read & write access.

Figure 11. Additional choices to make at Application Registration

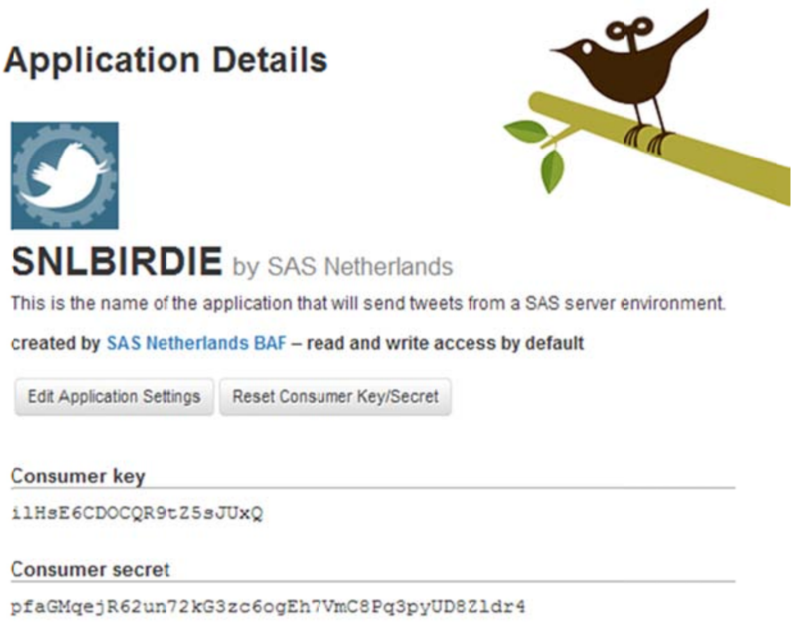
With these choices we get a page (Fig. 12) confirming that the application SNLBIRDIE was created by SNLBAF with read and write access. The confirmation page also gives us two long alphanumeric strings, which are apparently associated with the SNLBIRDIE application:

Consumer Key: ilHsE6CDOCQR9tZ5sJUxQ

Consumer Secret: pfaGMqejR62un72kG3zc6ogEh7VmC8Pq3pyUD8Zldr4

We take note of these since we will need them in the PHP scripts. In OAuth speak, a Consumer is an Application interacting with a Service Provider on behalf of a User. A User is the individual who has an account with the Service Provider. So the Consumer Key and the Consumer Secret are really the credentials that the application SNLBIRDIE must give to Twitter to prove its identity. For more details on OAuth terminology and definitions we refer to the RFC 5849 OAuth specification (see Recommended Reading).

Note (Fig. 12) that it is possible to reset the Consumer Key and Consumer Secret associated with the SNLBIRDIE application. This amounts to asking Twitter for a new application ID and password, and therefore it is safe to publish key and secret in this paper for didactical purposes since we will have reset everything by the time the paper hits the presses…



Application Details

SNLBIRDIE by SAS Netherlands

This is the name of the application that will send tweets from a SAS server environment.

created by SAS Netherlands BAF – read and write access by default

Consumer key

ilHsE6CDOCQR9tZ5sJUxQ

Consumer secret

pfaGMqejR62un72kG3zc6ogEh7VmC8Pq3pyUD8Zldr4

Figure 12. Twitter Application Registration Confirmation with OAuth Consumer Key and Consumer Secret.

PREREQUISITES 2 – PHP

THE SOFTWARE

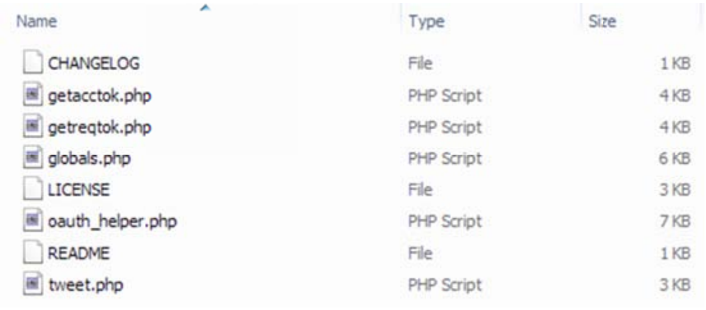
As explained in the previous chapter, we plan to use an extant set of PHP scripts to handle the OAuth authentication between our SAS code and Twitter. So we need a PHP interpreter on the machine where the SAS code will be running. The downloads section of the PHP project site (www.php.net) offers a number of compiled binaries for Windows. We downloaded the php-5.3.5-nts-Win32-VC9-x86.msi installer and so installed the PHP 5.3.5 release on our SAS server environment. We then added the location of the php.exe to the Windows PATH system environment variable.

The PHP project site only offers 32-bit compiled Windows binaries. For those needing a 64-bit binary: there are some available on the web, see e.g. www.anindya.com

To install PHP, simply click through the installer and when presented with a web server choice, choose ‘Do not set up a web server’. The rest of the installation is straightforward. A word of warning however: do not perform a full install but go with the suggested extensions. Adding more extensions may introduce dependencies on 3rd party software — e.g. the Oracle extension needs certain Oracle client software to be present — and PHP will refuse to even start up. For the purpose of this paper nothing beyond the default component selection is needed.

JOE CHUNG’S SCRIPTS

While searching for a suitable OAuth to Twitter implementation, we came across the following marvelously lucid tutorial written by Joe Chung: http://nullinfo.wordpress.com/oauth-twitter/ Joe also provides a set of self-contained bare-bones PHP scripts to go with his tutorial. These can be downloaded from the GitHub open source code sharing community at https://github.com/joechung/oauth_twitter After unzipping the archive to a suitable PHP scripts directory we get five scripts and a few accompanying files. (Fig. 13)



Name	Type	Size
CHANGELOG	File	1 KB
getacctok.php	PHP Script	4 KB
getreqtok.php	PHP Script	4 KB
globals.php	PHP Script	6 KB
LICENSE	File	3 KB
oauth_helper.php	PHP Script	7 KB
README	File	1 KB
tweet.php	PHP Script	3 KB

Figure 13. Joe Chung’s OAuth Twitter scripts.

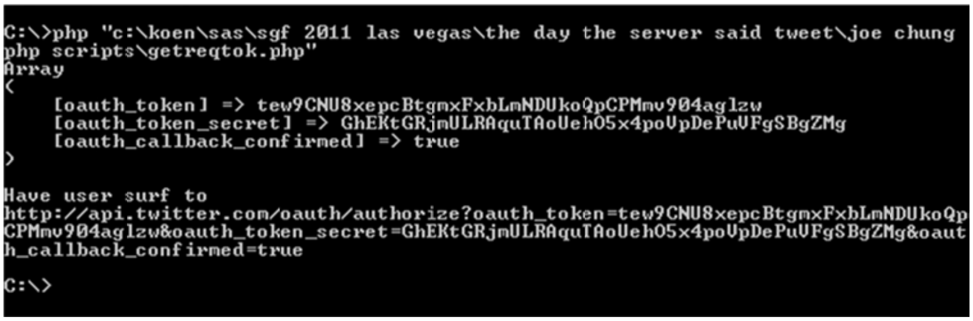
The scripts need some minimal editing. We first edit the globals.php script, which contains a number of definitions used in the other scripts. Near the top of the script we provide our Consumer Key and Consumer Secret as obtained in the previous section:

```
// Fill in the next two constants
define('OAUTH_CONSUMER_KEY', 'ilHsE6CDOCQR9tZ5sJUxQ');
define('OAUTH_CONSUMER_SECRET', 'pfaGMqejR62un72kG3zc6ogEh7VmC8Pq3pyUD8Zldr4');
```

Since we are behind a web proxy we also need to add a few Curl options to allow connecting to the outside world. The reader should provide their own proxy name, port, and user–password, both in the definition of the DO_GET and of the DO_POST functions:

```
CURLOPT_PROXY => 'your.proxy.here'
CURLOPT_PROXYPORT => 80
CURLOPT_PROXYUSERPWD => 'proxy_uid:proxy_pwd'
```

In order to authorize our SNLBIRDIE application to access the SNLBAF Twitter account we need to do some more work. The Get Request Token script (getreqtok.php) can be submitted without edits. We open a command window and feed getreqtok.php to the PHP executable. (Fig. 14)



```
C:\>php "c:\koen\sas\sgf 2011 las vegas\the day the server said tweet\joe chung
php scripts\getreqtok.php"
Array
(
    [oauth_token] => tew9CNU8xepcBtgmxFxbLmNDUkoQpCPMmv904aglzw
    [oauth_token_secret] => GhEKtGRjmULRAquTAoUehO5x4poVpDePuVFgSBgZMg
    [oauth_callback_confirmed] => true
)

Have user surf to
http://api.twitter.com/oauth/authorize?oauth_token=tew9CNU8xepcBtgmxFxbLmNDUkoQp
CPMmv904aglzw&oauth_token_secret=GhEKtGRjmULRAquTAoUehO5x4poVpDePuVFgSBgZMg&oaut
h_callback_confirmed=true

C:\>
```

Figure 14. Upon running the Get Request Token script.

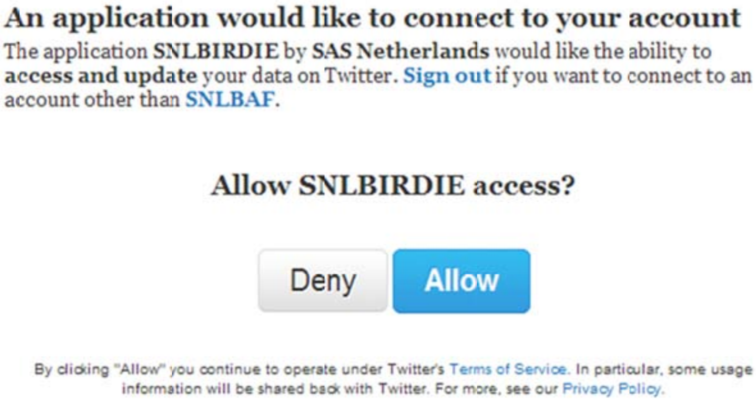
The Get Request Token script returns three items, a Request Token (oauth_token), a Request Token Secret (oauth_token_secret), and a URL:

Request Token = tew9CNU8xepcBtgmxFxbLmNDUkoQpCPMmv904aglzw

Request Token Secret = GhEKtGRjmULRAquTAoUehO5x4poVpDePuVFgSBgZMg

http://api.twitter.com/oauth/authorize?oauth_token=tew9CNU8xepcBtgmxFxbLmNDUkoQpCPMmv904aglzw
&oauth_token_secret=GhEKtGRjmULRAquTAoUehO5x4poVpDePuVFgSBgZMg
&oauth_callback_confirmed=true

Note that the URL contains the Request Token and the Request Token Secret, and feeds these to the Twitter Authorization API. Upon copying and pasting the URL in a browser we are asked to confirm access. (Fig. 15)



An application would like to connect to your account

The application **SNLBIRDIE** by **SAS Netherlands** would like the ability to **access and update** your data on Twitter. Sign out if you want to connect to an account other than SNLBAF.

Allow SNLBIRDIE access?

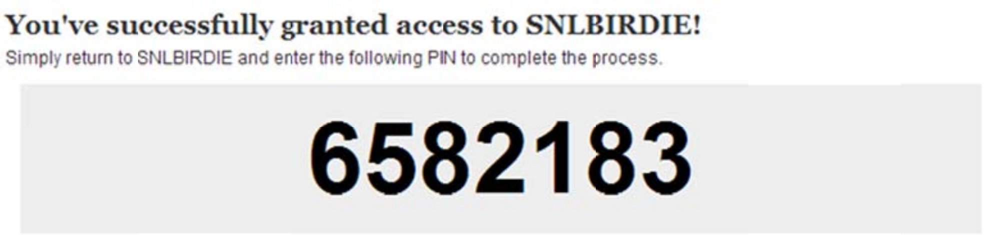
By clicking "Allow" you continue to operate under Twitter's Terms of Service. In particular, some usage information will be shared back with Twitter. For more, see our Privacy Policy.

Figure 15. Talking to the Twitter Authorization API.

Clicking the ‘allow’ button, we are rewarded with a PIN-code (Fig. 16):

PIN = 6582183

We ignore the instruction to return to the SNLBIRDIE application to enter the PIN, because there is nowhere to enter the PIN… We will need the PIN though in the next and final step of setting up our OAuth connectivity.



You've successfully granted access to SNLBIRDIE!

Simply return to SNLBIRDIE and enter the following PIN to complete the process.

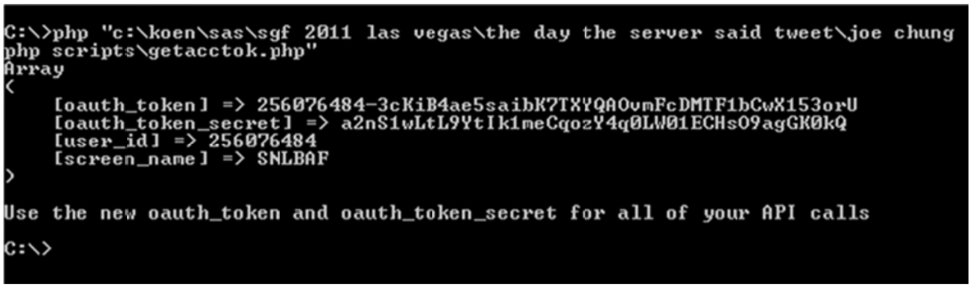
6582183

Figure 16. The SNLBIRDIE application is now authorized.

The last step in preparing our Twitter OAuth connectivity is to request an Access Token and Access Token Secret. We edit the Get Access Token script (getacctok.php) by inserting our newly obtained Request Token, Request Token Secret, and PIN code:

```
// Fill in the next 3 variables.
$request_token='tew9CNU8xepcBtgmxFxbLmNDUkoQpCPMmv904aglzw';
$request_token_secret='GhEKtGRjmULRAquTAoUehO5x4poVpDePuVFgSBgZMg';
$oauth_verifier= 6582183;
```

We proceed to feed the script to PHP. (Fig. 17)



```
C:\>php "c:\koen\sas\sgf 2011 las vegas\the day the server said tweet\joe chung
php scripts\getacctok.php"
Array
(
    [oauth_token] => 256076484-3cKiB4ae5saibK7TXYQAOvmFcDMTF1bCwX153orU
    [oauth_token_secret] => a2nS1wLtL9YtIk1meCqozY4q0LW01ECHsO9agGK0kQ
    [user_id] => 256076484
    [screen_name] => SNLBAF
)

Use the new oauth_token and oauth_token_secret for all of your API calls

C:\>
```

Figure 17. Upon running the Get Access Token script.

The Get Access Token script returns the two essential items we'll need to start tweeting: an Access Token (oauth_token), and an Access Token Secret (oauth_token_secret):

Access Token = 256076484-3cKiB4ae5saibK7TXYQAOvmFcDMTF1bCwX153orU

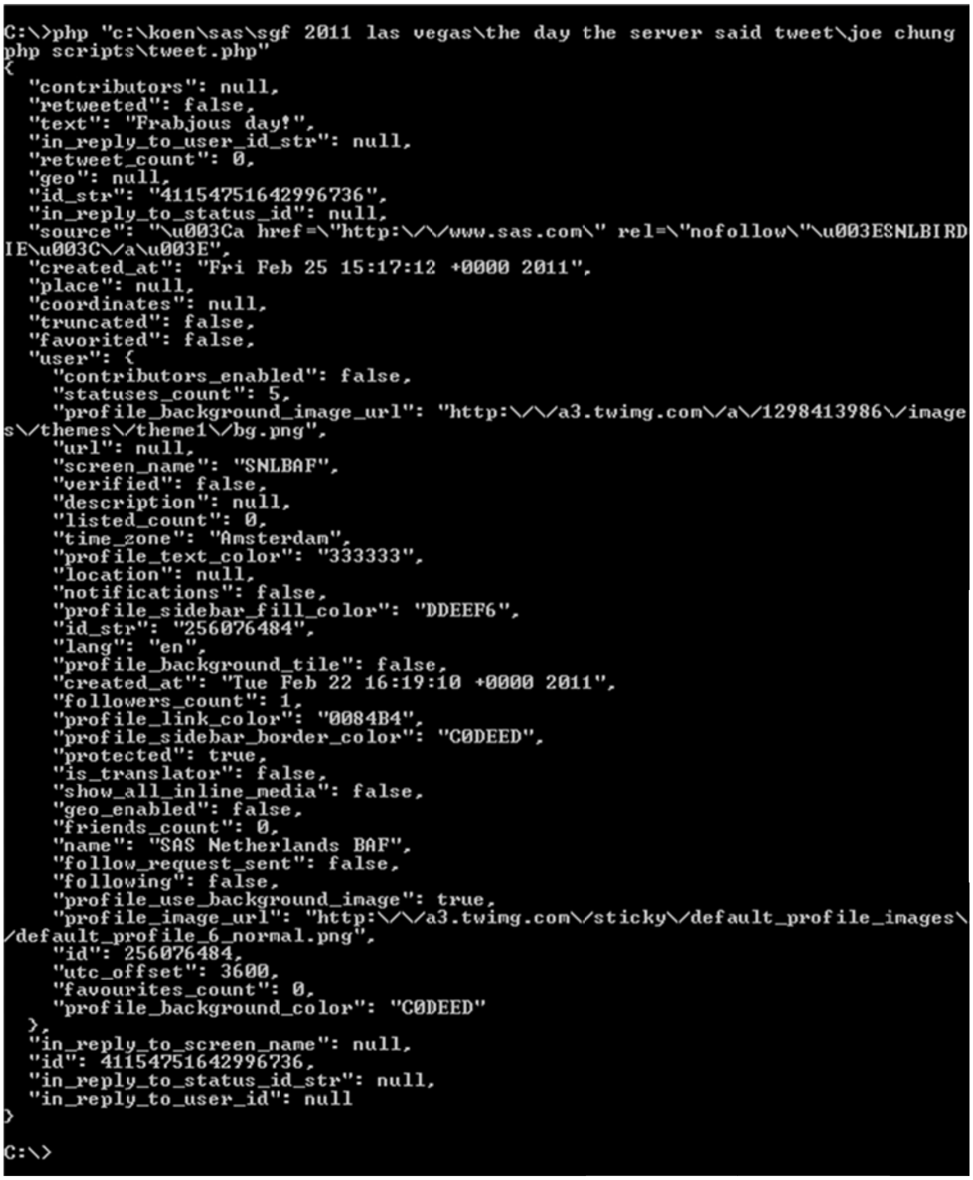
Access Token Secret = a2nS1wLtL9YtIk1meCqozY4q0LW01ECHsO9agGK0kQ

Note that the Access Token starts with a user ID. The phrase "Use the new oauth_token and oauth_token_secret for all of your API calls" indicates the end of all the manual steps which we had to go through. And indeed, Twitter does not currently expire Access Tokens and Access Token Secrets. The ones which we just obtained should remain good indefinitely. Or at least until Twitter changes its policy…

So let's put everything we did so far to the test and see whether we can now use Joe Chung's Tweet script (tweet.php). We edit the script to contain our SNLBAF Access Token and Access Token Secret, and provide the message to be tweeted.:

```
// Fill in the next 2 variables.
$access_token='256076484-3cKiB4ae5saibK7TXYQAOvmFcDMTF1bCwX153orU';
$access_token_secret='a2nS1wLtL9YtIk1meCqozY4q0LW01ECHsO9agGK0kQ';
$tweet = 'Frabjous day!';
```

Upon feeding the tweet.php script to PHP we get a load of details back from the Twitter API. (Fig. 18)

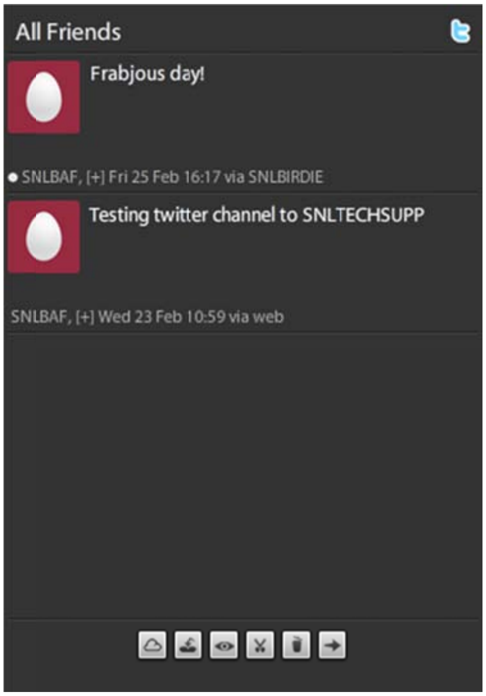


```
C:\>php "c:\koen\sas\sgf 2011 las vegas\the day the server said tweet\joe chung
php scripts\tweet.php"
{
  "contributors": null,
  "retweeted": false,
  "text": "Frabjous day!",
  "in_reply_to_user_id_str": null,
  "retweet_count": 0,
  "geo": null,
  "id_str": "41154751642996736",
  "in_reply_to_status_id": null,
  "source": "\u003Ca href=\"http:\/\/www.sas.com\" rel=\"nofollow\"\u003ESNLBIRD
IE\u003C\/a\u003E",
  "created_at": "Fri Feb 25 15:17:12 +0000 2011",
  "place": null,
  "coordinates": null,
  "truncated": false,
  "favorited": false,
  "user": {
    "contributors_enabled": false,
    "statuses_count": 5,
    "profile_background_image_url": "http:\/\/a3.twimg.com\/a\/1298413986\/image
s\/themes\/theme1\/bg.png",
    "url": null,
    "screen_name": "SNLBAF",
    "verified": false,
    "description": null,
    "listed_count": 0,
    "time_zone": "Amsterdam",
    "profile_text_color": "333333",
    "location": null,
    "notifications": false,
    "profile_sidebar_fill_color": "DDEEF6",
    "id_str": "256076484",
    "lang": "en",
    "profile_background_tile": false,
    "created_at": "Tue Feb 22 16:19:10 +0000 2011",
    "followers_count": 1,
    "profile_link_color": "0084B4",
    "profile_sidebar_border_color": "C0DEED",
    "protected": true,
    "is_translator": false,
    "show_all_inline_media": false,
    "geo_enabled": false,
    "friends_count": 0,
    "name": "SAS Netherlands BAF",
    "follow_request_sent": false,
    "following": false,
    "profile_use_background_image": true,
    "profile_image_url": "http:\/\/a3.twimg.com\/sticky\/default_profile_images\
/default_profile_6_normal.png",
    "id": 256076484,
    "utc_offset": 3600,
    "favourites_count": 0,
    "profile_background_color": "C0DEED"
  },
  "in_reply_to_screen_name": null,
  "id": 41154751642996736,
  "in_reply_to_status_id_str": null,
  "in_reply_to_user_id": null
}

C:\>
```

Figure 18. Upon running the Tweet script.

And indeed, over at the Support center, the TweetDeck shows the Server's first script-generated tweet! (Fig. 19) Note the 'via SNLBIRDIE' footnote, indicating that an application SNLBIRDIE sent the tweet on behalf of SNLBAF.



Frabjous day!

SNLBAF, [+] Fri 25 Feb 16:17 via SNLBIRDIE

Testing twitter channel to SNLTECHSUPP

SNLBAF, [+] Wed 23 Feb 10:59 via web

Figure 19. SNLTECHSUPP can see the script-generated test tweet from SNLBAF in TweetDeck.

APPLICATIONS

Now that we can send tweets by calling a script, the possibilities for sending real-time diagnostics are endless. The following three examples are derived from frequently reported SAS Technical Support issues. We will use the tweet.php script to send custom messages in each case, and for reasons of code readability we rename the script to tweet_template.php. The reader should also keep in mind that the code presented here has not been optimized for elegance or performance, but rather for conceptual simplicity. That is, it'll do the job but it may not comply with your coding standards, nor with ours for that matter…

TOO MUCH WORK!

In our first example we try to detect the WORK library running out of disk space, e.g. because a user is trying to sort a gigantic data set by a zillion variables at once. By defining a suitable threshold, say the WORK volume running over 90% full, the Support staff can hopefully take action before the entire environment grinds to a halt on a 'Disk Full' error. The SAS batch job to keep an eye on WORK-space could look like this:

```
options noxwait; ❶

%let max_bytes     = 250048507904; ❷
%let threshold_pct = 90;
%let scripts_dir   = c:\wherever\the\php\scripts\are;
%let dir_out       = c:\temp\dir_out.txt;
%let worklib       = %sysfunc(getoption(work));

filename dirout   "&dir_out";
filename tweettmp "&scripts_dir\tweet_template.php";
filename tweetmod "&scripts_dir\tweet.php";

data _null_; ❸
  length
    cmd $ 100
    work_volume $ 3;
  work_volume=substr(symget("worklib"),1,3);
  cmd="dir"||" "||work_volume||" > &dir_out.";
  call system(cmd);
run;
```

```
data foo; ❹
  infile dirout truncover;
  input line $60.;
run;

data _null_; ❺
  length from to positions bytes_free pct_full 8;
  set foo end=last;
  if last then do;
    from=find(line,')')+2;
    to=find(line,'bytes');
    positions=to-from;
    bytes_free=input(compress(substr(line,from,positions),','),20.);
    pct_full=round(100-bytes_free/"&max_bytes"*100);
    call symput('pct_full',trim(left(put(pct_full,2.))));
    end;
run;

data _null_;
  length
    bytes 8
    tweet_msg $ 140;
  retain tweet_msg;
  if _n_=1 then tweet_msg="$tweet = 'the WORK volume is "||"&pct_full"
                          ||"% full - %sysfunc(datetime(),datetime19.)';"; ❻
  infile tweettmp;
  file tweetmod;
  input;
  if not (_infile_=:"$tweet") then put _infile_; ❼
  else put tweet_msg;
  if &pct_full < &threshold_pct then call symput('tweetflag','0'); ❽
  else call symput('tweetflag','1');
run;

data _null_; ❾
  if &tweetflag=1 then call system('php'||" "||"&scripts_dir"||'\tweet.php');
run;
```

A few comments about the code:

❶ We will interact with the OS through a DOS command window. Setting the NOXWAIT option allows the SAS code to continue running instead of waiting for a human to close the command window.

❷ Some global variables and assorted definitions:
Max_bytes is the total storage capacity of the volume where the WORK library lives, expressed in bytes;
Threshold_pct is the volume usage percentage above which we want to send a warning tweet;
Scripts_dir is the directory where Joe Chung's PHP scripts were saved;
Dir_out points to a txt-file which will capture the output of a DIR command;
Worklib is the directory path to the WORK library;
Tweettmp points to the tweet_template.php script;
Tweetmod points to the tweet.php script, which will be the modified copy of the template to be executed.

❸ We extract the volume name from the WORK library path, and using CALL SYSTEM we run a dir command on it while piping the output thereof to the text-file dirout.

❹ Read the output of the dir command and store as text strings in a temporary data set.

❺ The last line of dir command output contains the number of free bytes on the volume, so we parse the last observation to pry the number (bytes_free) out of the text, and calculate the percentage full for the volume (pct_full).

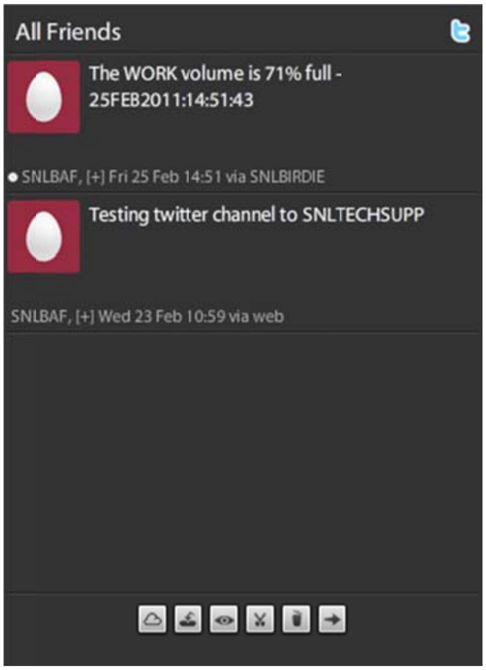
❻ Using pct_full we construct the actual tweet-message (tweet_msg), or rather the entire line in the tweet_template.php script that needs to be replaced to send a custom tweet. Note that we include a datetime stamp within the body of our tweet. We did this while developing and testing the code because Twitter imposes restrictions on tweeting the same message multiple times. We found it rather practical however since it shows the tweet-time quite visibly in the TweetDeck app, so we decided to leave it in.

❼ We then read the tweet_template.php script line by line. Unless the line is the one where the body of the tweet is defined, we simply output the _INFILE_ buffer again. For the line commencing '$tweet' we substitute the tweet_msg we constructed earlier. The result is a copy of the tweet_template.php script but with our own message substituting the default one.

❽ If the percentage full for the volume is below the threshold we set a tweetflag to zero. If it exceeds the threshold then the tweetflag is set to 1. For the purpose of this demo we have temporarily lowered the threshold to 70% otherwise nothing shows.

❾ If the tweetflag is set, we call PHP and feed it our customized tweet.php script.

When the threshold is passed, a tweet is sent and it shows up on the Support center's TweetDeck. (Fig. 20)



All Friends

The WORK volume is 71% full - 25FEB2011:14:51:43

● SNLBAF, [+] Fri 25 Feb 14:51 via SNLBIRDIE

Testing twitter channel to SNLTECHSUPP

SNLBAF, [+] Wed 23 Feb 10:59 via web

Figure 20. The SNLTECHSUPP TweetDeck shows the warning tweet from the SNLBAF server.

ANYBODY HOME?

In this example we use data generated by the SAS 9.2 Enterprise Business Intelligence Audit and Performance Measurement (EBIAPM) package as referenced in the introduction. When a SAS administrator installs EBIAPM, it can be configured to run regular checks on a whole lot of things. Like whether any of the vital SAS Servers are down. This is done by scheduling the runcheckserver.bat (which comes with EBIAPM) job to run periodically. The batch job generates the sas92_bi_health data set, which we inspect in the following code:

```
options noxwait;

%let scripts_dir = c:\wherever\the\php\scripts\are;
%let ebiapm_home = c:\sas\ebiedieg\lev1\ebiapm92; ❶
%let xcmds       = c:\temp\xcmds.bat;

libname  apmdata  "&ebiapm_home\data\status";
filename xcmds    "&xcmds";
filename tweettmp "&scripts_dir\tweet_template.php";
filename tweetmod "&scripts_dir\tweet.php";

data _null_; ❷
  file xcmds;
  put "cd &ebiapm";
  put 'runcheckserver.bat';
run;

x &xcmds;
```

```
data servers_down; ❸
  set apmdata.sas92_bi_health;
  if server_status=0 then output;
run;

data _null_;
  dsid=open("servers_down");
  nrobs=attrn(dsid,"nobs");
  rc=close(dsid);
  if nrobs=0 then call symput('tweetflag','0'); ❹
  else do;
    infile tweettmp end=last;
    file tweetmod;
    input;
    if not (_infile_=:"$tweet") then put _infile_;
    else put "$tweet = 'One or more SAS Servers are down -
                      %sysfunc(datetime(),datetime19.)';";
    if last then call symput('tweetflag','1');
    end;
run;

data _null_;
  if &tweetflag=1 then call system('php'||" "||"&scripts_dir"||'\tweet.php'); ❺
run;
```

The structure of the code is similar to the previous example. Items that were already explained or commented upon will not be reiterated here:

❶ Some global variables and assorted definitions:
Ebiapm_home is the home directory of our EBIAPM installation;
Xcmds points to a text-file which we'll use to run a few X commands in one go;
Apmdata points to the location where the EBIAPM runcheckserver.bat job writes its data sets;

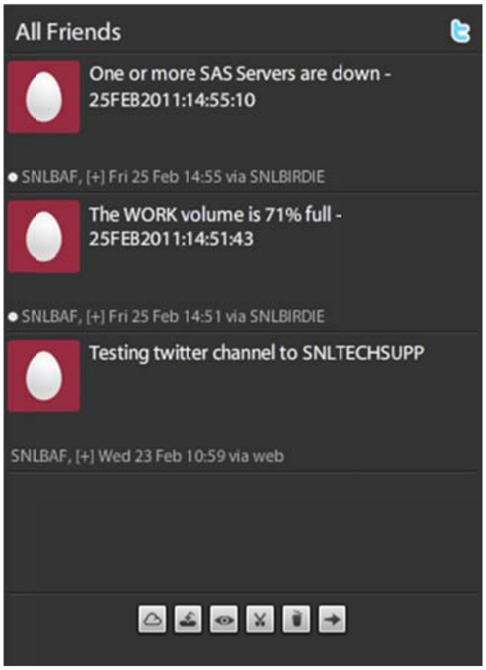
❷ The OS commands to run the EBIAPM runcheckserver.bat job are written to a text-file and subsequently executed by means of an X statement. Note that ordinarily the frequency of running this particular job should be kept as low as possible due to the potential impact on system performance.

❸ We then inspect the output of the runcheckserver.bat process — the data set sas92_bi_health — and retain only those records where a server down condition was detected.

❹ If any servers are found to be down we create a custom tweet.php from the tweet_template.php script by inserting an appropriate tweet message in the right place.

❺ And run the script if necessary…

An obvious improvement to the above would be to send more specific tweets based on the contents of the servers_down data set, explicitly mentioning in the tweet body which servers are impacted. For now, figure 21 shows the generic warning popping up in the Support center's TweetDeck.



One or more SAS Servers are down - 25FEB2011:14:55:10

SNLBAF, [+] Fri 25 Feb 14:55 via SNLBIRDIE

The WORK volume is 71% full - 25FEB2011:14:51:43

SNLBAF, [+] Fri 25 Feb 14:51 via SNLBIRDIE

Testing twitter channel to SNLTECHSUPP

SNLBAF, [+] Wed 23 Feb 10:59 via web

Figure 21. Another ominous warning shows up in the Support center's TweetDeck.

BOGGED DOWN

As a final example we attempt to detect system performance degradation by hanging user Workspace Server processes gobbling up CPU cycles and bogging down the entire system. Depending on how exactly the SAS environment is configured and used it may be perfectly legitimate to have long-running user processes. In our reference system however, it is not. So how do we detect the relevant SAS.EXE processes among the mass of SAS sessions that are typically active in a SAS environment?

The standard set-up on Windows is that the Metadata Server, OLAP Server, Connect Server, and Share Server all run as the 'NT AUTHORITY\SYSTEM' user. Furthermore, everything running on a Stored Process Server and a Pooled Workspace Server is running as the SAS General Server User (SASSRV). So if we filter those out of the output of a DOS tasklist command, any SAS.exe processes left are Workspace Server processes running under the individual users' accounts.

In order to monitor how long processes have been running we use the PsList utility which is part of the PsTools suite, previously by SysInternals, nowadays to be found on the MS site: http://technet.microsoft.com/en-us/sysinternals/bb896649

The following code detects user Workspace Server processes running for more than 2 days:

```
options noxwait;

%let scripts_dir = c:\wherever\the\php\scripts\are;
%let pstools_home= c:\pstools; ❶
%let xcmds       = c:\temp\xcmds.bat;

filename pslist   'c:\temp\pslist.txt';
filename tsklist  'c:\temp\tasklist.txt';
filename tweettmp "&scripts_dir\tweet_template.php";
filename tweetmod "&scripts_dir\tweet.php";
filename xcmds    "&xcmds";

data _null_; ❷
  file xcmds;
  put "cd &pstools_home";
  put "pslist sas > c:\temp\pslist.txt";
  put "tasklist /V /FI ""imagename eq sas.exe"" > c:\temp\tasklist.txt";
run;

x &xcmds;
```

```
data pslist; ❸
  infile pslist firstobs=4;
  input name $ pid pri thd phd priv @49 cpu_time time11. @65 elapsed_time time11.;
run;

data tsklist;
  infile tsklist firstobs=4;
  input name $ pid system $ session memory comma10. kilobyte $ status $ user $40.;
run;

proc sql; ❹
  create table processes as
  select
    tsklist.user,
    pslist.elapsed_time
  from
    pslist,
    tsklist
  where
    pslist.pid=tsklist.pid
  ;
quit;

data user_processes;
  set processes;
  if upcase(user) in ("NT AUTHORITY\SYSTEM","FOO\SASSRV") then delete; ❺
run;

proc sort data=user_processes;
  by descending elapsed_time; ❻
run;

data _null_;
  dsid=open("user_processes");
  nrobs=attrn(dsid,"nobs");
  rc=close(dsid);
  if nrobs=0 then abort return; ❼
run;

data _null_;
  set user_processes(obs=1);
  if elapsed_time<172800 then abort return; ❽
run;

data _null_;
  infile tweettmp end=last;
  file tweetmod;
  input;
  if not (_infile_ =: "$tweet") then put _infile_;
  else put "$tweet='One or more Workspace Server user processes is running longer
                    than 2 days! - %sysfunc(datetime(),datetime19.) ';"; ❾
run;

data _null_;
  call system('php'||" "||"&scripts_dir"||'\tweet.php');
run;
```

Some additional notes:

❶ Some global variables and assorted definitions:
Pstools_home is the home directory of our PsTools installation;

Xcmds points to a text-file which we'll use to run a few X commands in one go;
Pslist points to a temporary text-file wherein we capture the output of a PsList command;
Tsklist points to a temporary text-file for capturing the output of a Tasklist command.

❷ In the xcmds.bat file we first move to our PsTools home directory, then call the PsList tool for SAS processes and redirect the output to the pslist.txt file. We also call the tasklist command with options for verbosity and filtering for sas.exe processes, and redirect the output to the tasklist.txt file.

❸ After running these commands in batch with an X statement, we proceed to read the newly generated text files into a pair of data sets. The pslist data set will essentially give us the elapsed time for each sas.exe process ID (PID), whereas the tsklist data set will link PIDs to users. For didactical purposes we also read the other fields available in the text-files, it's interesting to see what other process-related information is in there.

❹ We then join the user name to the elapsed process time on PID.

❺ And filter out anything that is not running under an individual user account.

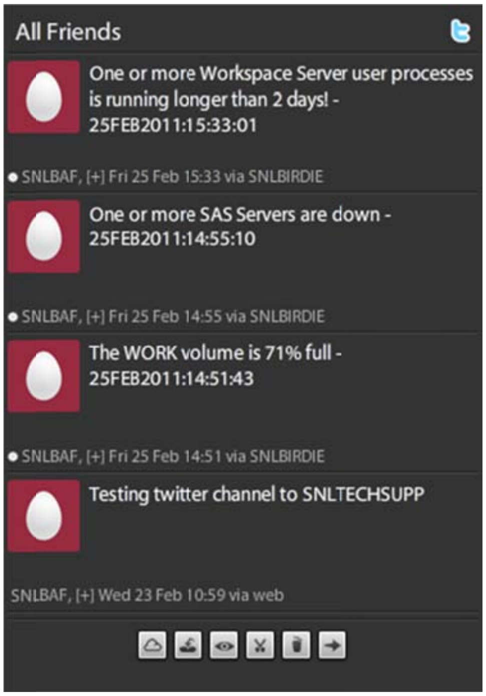
❻ Then sort by descending elapsed time, so that the longest running user process is represented by the first observation.

❼ If the resulting data set is empty we don't need to do anything else and ABORT RETURN kills the current SAS session, preventing the rest of the code being executed.

❽ If the longest running user process has an elapsed time of less than 2 days (expressed here in seconds) then we also bail out.

❾ We only get to this part of the code if there is at least one user process with an elapsed time of more than two days. We modify the tweet message as in the previous examples, and call PHP to send it.

Figure 22 shows the code-generated tweet in the Support center's TweetDeck.



One or more Workspace Server user processes is running longer than 2 days! - 25FEB2011:15:33:01

• SNLBAF, [+] Fri 25 Feb 15:33 via SNLBIRDIE

One or more SAS Servers are down - 25FEB2011:14:55:10

• SNLBAF, [+] Fri 25 Feb 14:55 via SNLBIRDIE

The WORK volume is 71% full - 25FEB2011:14:51:43

• SNLBAF, [+] Fri 25 Feb 14:51 via SNLBIRDIE

Testing twitter channel to SNLTECHSUPP

SNLBAF, [+] Wed 23 Feb 10:59 via web

Figure 22. Another warning is tweeted to the Support center's TweetDeck for follow-up.

CONCLUSION

We have shown how to set up a mechanism allowing SAS processes running in a SAS Business Analytics server environment to tweet diagnostic messages to a support center. Business critical systems should of course be adequately protected against downtime by solid failover mechanisms at all levels: database, hardware, applications. However, in case something goes awry, it is always good to involve the experts as quickly as possible. The proposed mechanism should provide some additional peace of mind for SAS Platform administrators. We took a shortcut to Twitter by using Joe Chung's PHP scripts, but it should be interesting to see whether the entire OAuth authentication can be done through SAS code. To make things interesting, this would require coding SHA1 encryption in Base SAS. A fun exercise!

RECOMMENDED READING

- SAS 9.2 Enterprise Business Intelligence Audit and Performance Measurement for Windows Environments
  http://support.sas.com/rnd/emi/EbiApm92/sas92.ebiapm.win.pdf

  The installation and configuration guide for the EBIAPM package gives a good idea of the variety of SAS environment aspects that can be monitored by the utility.
- The Twitter Developers Authentication page
  http://dev.twitter.com/pages/auth

  For those so inclined, we recommend having a look at the details of the OAuth authentication flow as explained on this site.
- RFC 5849 – The OAuth 1.0 Protocol
  http://tools.ietf.org/html/rfc5849

  The complete RFC document describing version 1 of the OAuth protocol. This is hardcore.

ACKNOWLEDGEMENTS

The authors should like to thank SAS Global Forum section chair Pete Lund for his interest in this somewhat maverick application of Social Media abuse in a SAS systems administration context, and the inimitable Joe Chung for making his excellent scripts publicly available and taking the bite out of OAuth!

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

William W. Viergever
Viergever & Associates
Health Data Analysis / Systems Design & Development
2920 Arden Way Suite N
Sacramento, CA 95825
USA
william@viergever.net
www.viergever.net
+1 (916) 483-8398

Daniël Kuiper / Koen Vyverman
SAS Institute B.V.
Flevolaan 69
1272PC Huizen
The Netherlands
support@snl.sas.com