Salesforce Canvas Adapter for Tableau

Embedding Tableau dashboards in Salesforce.com

Version 1.04

Last update: 31 August 2016

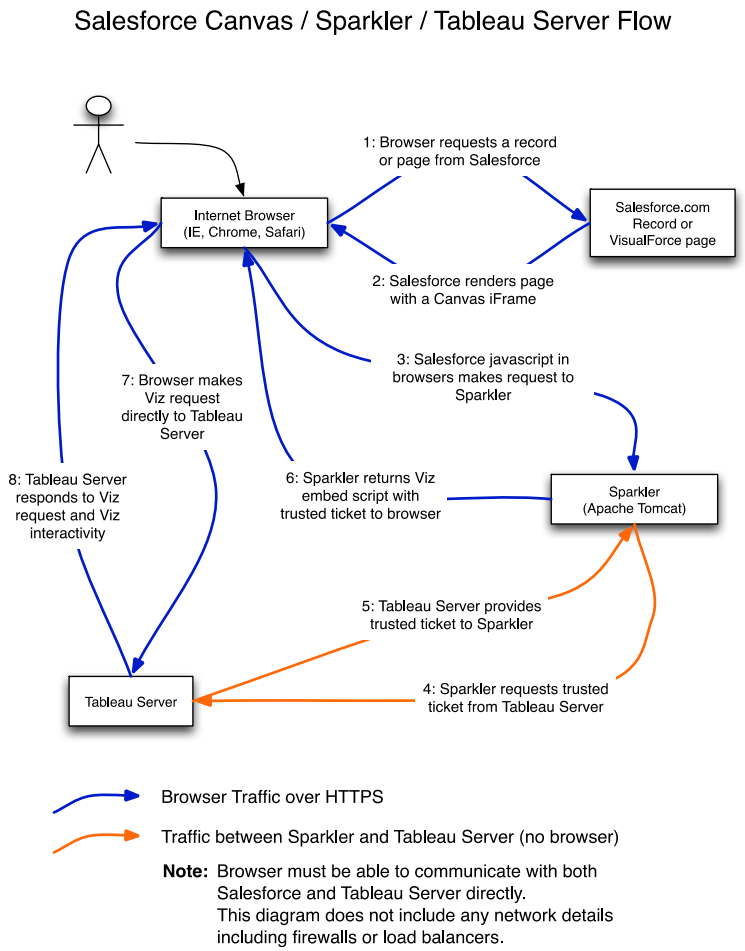
Contents

Introduction

This document describes how to embed Tableau dashboards in Salesforce.com using the Force.com Canvas framework. Canvas is Salesforce's framework for integrating third-party web applications into the Salesforce platform. For general information about Canvas, see Appendix N: Additional Information.

Tableau Server does not have built-in support for Canvas, but we have created a Java adapter named Sparkler to enable users to embed Tableau dashboards via Salesforce Canvas. This document describes how to deploy Sparkler on premises and how to integrate it with an on-premises Tableau Server using Trusted Authentication. If you are using Tableau Online, you can also use SAML for authentication.

The following diagram illustrates how Sparkler allows users to embed Tableau Server visualizations (views) in Salesforce.com. Note that there is also a flow from Salesforce.com to Sparkler in order to authenticate the application.



What's New

Version 1.04

The following changes have been made for v1.04 of Sparkler. For information about how to upgrade from an earlier version of Tableau Sparkler, see Appendix M: Upgrading to v1.04.

- Fixed a bug that prevented multi-site Tableau servers from working properly with Sparkler.

Version 1.03

The following changes have been made for v1.03 of Sparkler. For information about how to upgrade from an earlier version of Tableau Sparkler, see Appendix L: Upgrading to v1.03.

- Fixed a bug that prevented Mozilla from working properly with Sparkler.

Version 1.02

The following changes have been made for v1.02 of Sparkler. For information about how to upgrade from an earlier version of Tableau Sparkler, see Appendix K: Upgrading to v1.02.

Important: We recommend that all users update to v1.02 of Tableau Sparkler. Version 1.02 includes a security mitigation (see first point below).

- Security fix. This release includes a mitigation for a security vulnerability that potentially affected installations that used Tableau Server trusted tickets. In those installations, authenticated users could intercept the traffic to Sparkler and assert any identity. The fix for this vulnerability is that the Sparkler server process will not allow the client to assert their user identity. Instead, it maps a user identity from Salesforce to Tableau. For more information, see User Mapping and Security Considerations.
- Custom mapping. This release lets you create a custom mapping between a field in Salesforce and a Tableau user name. For more information, see Appendix J: Custom Mapping and the "signedIdentity" Parameter.
- Additional configuration values. The following new configuration values are defined in the `sparkler.xml` file. These configuration settings are used for user mappings.
    - `sparkler.sfdc.userIdentifierField`
    - `sparkler.sfdc.emailDomainsAllowed`
    - `sparkler.sfdc.signedIdentity`
    - `sparkler.tableau.signedIdentitySkewSeconds`

    For more information about these new configuration settings, see Configure Sparkler and Appendix C: Sparkler Configuration Parameters.

The following settings in the `sparkler.xml` file have been deprecated. If your configuration file contains these values, Sparkler will work, but the values are no longer used to establish a user identity.

- `ts.trustedticket.username`
- `ts.trustedsticket.siteid`

Technical Summary

Sparkler is a Java application that can be deployed on any Java application server that supports Java 8 or later. Though Sparkler can run on any Java App Server or Java servlet container, this document covers how to deploy Sparkler on Apache Tomcat 7 and provides configuration details for Windows and common Linux distributions.

Contents of the Package

The Sparkler distribution includes the Sparkler application as an Apache Tomcat WAR file, a Sparkler configuration XML file, this documentation, sample Tableau workbook files, and sample Salesforce VisualForce pages. The samples are designed to get you up and running quickly—see instructions in Appendix A: Using the Tableau Workbooks. However, these are just samples. For production deployments, you will need to provide your own Tableau workbooks and dashboards, as well as your own VisualForce pages.

Required Skills and Permissions

It is assumed that you have administrative access to both Salesforce.com and Tableau Server. Furthermore, you must be a system administrator on the system where the Sparkler application is being installed. We assume that are familiar with how to navigate a command line terminal, configure applications, set operating system environment variables, locate and read application logs, and install Tomcat as a service.

Pre-Install Checklist

To work with Sparkler and with the samples, you must have the following components, configured as listed.

Salesforce

- Salesforce users must be able to reach Tableau Server and Salesforce at the same time from the same browser. Both must be reachable via HTTPS.
- Salesforce (not the browser) must be able to communicate with the on-premises Sparkler adapter over HTTPS.

Tableau Server

- Automatic sign-in is not enabled.
- SSL is enabled. A commercial x.509 SSL certificate is highly recommended.
- Trusted authentication is configured on the server. (If you are working with Tableau Online, as an alternative you can use SAML, as described in Appendix I: Connecting to Tableau Online with SAML.)

On-premises Sparkler adapter

- Java 8 or later is installed.

- Tomcat 7 or later is installed.
- SSL is enabled. A commercial x.509 SSL certificate is highly recommended.
- The OpenSSL utility is installed. This is required on Windows in order to create RSA keys.
- A static IP is configured for the adapter. This is required for trusted authentication.
- The Sparkler adapter must be able to communicate with Tableau Server over HTTPS.

Supported Browsers

Tableau Server supports all major browsers, including Internet Explorer, Google Chrome, Mozilla Firefox, and Apple Safari. At the time of release, some issues were reported when users tried to use the embedded solution in Mozilla Firefox. We recommend testing your setup using Google Chrome. Verify that JavaScript and third-party cookies are enabled in the browser.

Accessing Sparkler from Public and Private Networks

Salesforce.com is a cloud solution that is accessible from the Internet. The on-premises Tableau Server/Sparkler solution runs on a corporate network. In order to ensure seamless integration, the end user (browser) must be able to access both Salesforce.com and Tableau Server in the same browser at the same time via HTTPS. If users are accessing embedded Tableau dashboards on site, generally no action is required. If users are accessing dashboards off-site (that is, from a client site), the user needs to be able to connect to the internal network using VPN, or Tableau Server needs to be publicly accessible from the Internet (see Proxy Servers in the Tableau Server documentation).

Support from Tableau Software

Sparkler is a supplementary application created and maintained by Tableau Software, Inc. Tableau will assist in basic configuration of the Sparkler app as it relates to communication with standard installations of Tableau Server. Tableau will support customer questions, but support is limited to the content in customer-facing documentation (including this document), in Tableau online help, and in Tableau blog posts.

Due to the complexity of customer implementations, such as network, firewall, and SSL certificate management configurations, customers are encouraged, but not required, to engage Tableau Professional Services.

The following items are explicitly **not supported**:

- Network connectivity, routing, firewalls, or load balancing between Salesforce.com and Sparkler.
- Network connectivity, routing, firewalls, or load balancing between Sparkler and Tableau Server.
- Issues related to SSL certificates that are issued by a certificate authority (CA) that is not trusted by default installations of Tableau Server, Oracle Java 8, or supported internet browsers.
- Installation of Java, Apache Tomcat, or other Java Servlet containers or operating system configuration.

- Salesforce.com configuration, beyond the documented Canvas App configuration, including custom Apex or VisualForce code.
- Browser-specific issues beyond standard support of Tableau Server.

SSL Certificates

Most of the instructions in this document assume that both Sparkler and Tableau Server are deployed using commercial x.509 SSL certificates purchased from a major CA such as Thawte or VeriSign. Although it is possible to use SSL certificates that are self-signed or signed by a corporate certificate authority for sandbox and proof-of-concept installations, you will need to perform additional configuration steps as described in this document.

Note: Self-signed SSL certificates *should not* be used in production environments.

An alternative for setting up SSL is to use a load balancer that supports SSL termination such as F5 Big-IP.

Single Sign-On Using SAML

If Salesforce.com and Tableau Server are both configured to support single sign-on using SAML, and if both use the same SAML identity provider, the user can access Tableau Server views directly in Salesforce.com without the use of Tableau Server trusted authentication. For more information about using SAML, see Appendix I: Connecting to Tableau Online with SAML.

Network Connectivity

Sparkler installation and operation *require* that:

- Salesforce.com can communicate directly with the Sparkler application over HTTPS. Salesforce will not allow communication with connected apps over HTTP. This document details how to enable SSL on Tomcat, but it is possible to run Tomcat over HTTP and have Salesforce communicate with Sparkler via an SSL proxy.
- The Sparkler application can communicate directly with Tableau Server over HTTP or HTTPS. Although the user's browser must communicate with Tableau Server over HTTPS, Sparkler can use HTTP.

The Salesforce.com Canvas Framework requires embedded applications to be secure and trusted. This requires a valid SSL configuration on both Sparkler and Tableau Server. If any browser warnings appear for either of the SSL configurations, embedded dashboards will not render in Salesforce. As noted previously, you can use self-signed certificates for testing, but users will need to accept security warnings. You should use X.509 SSL certificates signed by a commercial certificate authority for a production environment.

Each organization might have different operating system configurations, network configurations, security controls, proxies, load balancers, and other technology implementations that can affect the Sparkler integration. You should either be able to configure your environment or be able to consult with someone who can. This document cannot provide information for how to configure a wide variety of environments. However, the Sparkler application includes a number of end-to-end test

methods that are documented here that can help you evaluate the most common connectivity issues.

Tested Configurations

We have tested Sparkler in a standard Sparkler configuration, which involves deploying Sparkler on a separate server from the server that is running Tableau Server. For details regarding installing Sparkler on the same server as Tableau Server, see Appendix B: Running Tomcat and Tableau Server on the Same Computer.

The examples described in this document have been validated against the following:

- The latest generally available release of Salesforce.
- The latest generally available release of Tableau Server.
- Sparkler deployments using Java 8 and Tomcat 7 (on both Windows and Linux systems).

User Mapping and Security Considerations

Tableau Sparkler requires that a Salesforce user is mapped to a user identity in Tableau Server. By default, you can map a Salesforce username, email name, or user ID to a Tableau Server user. You define the mapping by using settings in a configuration file that correspond to the type of mapping you want. (Details about using predefined mappings are provided later, during Task 3: Configure Sparkler.)

As of version 1.02 of Tableau Sparkler, as a security measure, Sparkler requires that the Tableau Server user is mapped from a signed Salesforce field. In addition, Salesforce signs the values that are used for these predefined user mappings. This helps prevent someone from tampering with the user identity during communications between Salesforce and Tableau Sparkler.

If the predefined set of user mappings doesn't work for your scenario, you can create a custom mapping. This lets you use code to map any value or values available in Salesforce to Tableau.

Note: If you use custom identity mapping instead of relying on the predefined mappings, it becomes your responsibility to make sure the user identity information is signed. We help you with this task by providing an Apex Class for Salesforce that includes code to sign the parameters.

More information about custom mapping is provided in see Appendix J: Custom Mapping and the "signedIdentity" Parameter.

Configure Sparkler

The following tasks walk through installation of Apache Tomcat 7 and the Sparkler App on Linux or Windows.

The configuration consists of these tasks:

1. Install Java using the standard JRE. Sparkler requires Java 8.
2. Install Tomcat. This is the Apache webserver for hosting Sparkler
3. Enable HTTPS on Tomcat.
4. Install Sparkler.
5. Configure Sparkler for HTTPS communication with Tableau Server.
6. Setup Tableau Server Trusted Authentication.

Note: As of version 1.02, this task includes steps for establishing user mapping.

Task 1: Install Java

To install on Windows, you use the standard installers for the latest Java JRE (available on the Oracle website) and Apache Tomcat 7 (available on the download page of the Apache Tomcat site). 64-bit is recommended. Make a note of the install location of both Java (indicated by the `JAVA_HOME` environment variable) and Tomcat.

Note: Sparkler requires Java 8 or later.

Check whether Java in installed

You can use command-line commands to check whether Java is installed and which version is installed.

Windows

```
C:>java -version
Java version "1.8.build"
Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot(TM) Client VM (build 25.31-b07, mixed mode, sharing)
```

Linux

```
[ec2-user ~]$ java -version
 java version "1.8.build"
 OpenJDK Runtime Environment (amzn-2.5.3.1.49.amzn1-x86_64 u71-b14)
 OpenJDK 64-Bit Server VM (build 24.65-b04, mixed mode)
```

If Java is not installed, you will get one of the following error messages:

Windows

```
C:>java -version
'java' is not recognized as an internal or external command, operable program or batch
file.
```

Linux

```
[ec2-user ~]$ java -version
 -bash: java: command not found
```

Install Java

If Java is not installed, or the version installed is lower than 1.8, download and install the latest version of Java from http://www.java.com. (At the time this document was written, the latest version was Version 8.)

Task 2: Install Tomcat

The next task is to install and configure Tomcat. The recommended configuration is to install Tomcat on a computer that is not running Tableau Server.

> **Note**: If you need to run Tomcat and Tableau Server on the same computer, you must configure Tomcat so that it will not conflict with Tableau Server. After you have followed the steps in the following procedure, see Appendix B: Running Tomcat and Tableau Server on the Same Computer for additional information.

Windows

1. Download Tomcat 7.0 Windows Service Installer from http://tomcat.apache.org.
2. Double-click the installer file and follow the on screen instructions, accepting all defaults. If you installed Java as described in the preceding task, you will see the Java installation directory selected as the default installation location.
3. From a command prompt, enter the following:

   ```
   services.msc
   ```

   This opens the **Services** window. At the top of the list you will see an entry for Apache Tomcat 7.0.

4. To start Tomcat, right-click this entry and then click **Start**.

   You will use the **Services** window to start and stop Tomcat later in the configuration process.

Linux

> **Note:** The example listings show Linux running on an instance of AWS EC2. However, EC2 is not required. You can run Tomcat on any Linux server or distribution.

1. Log in to the server.

2. Install Apache Tomcat7 by using the following commands:

```
[ec2-user ~]$ sudo yum install tomcat7

[root logs]# sudo yum install tomcat7-webapps
```

3. Verify that the Tomcat directories have been created by entering the following commands and checking that the directory listings match the example.

```
[ec2-user ~]$ ls /usr/share/tomcat7
bin  conf  lib  logs  temp  webapps  work

[ec2-user ~]$ ls /etc/tomcat7
Catalina              context.xml          tomcat7.conf
catalina.policy       logging.properties   tomcat-users.xml
catalina.properties   server.xml           web.xml
```

4. Start Tomcat7 by using the following command. Note that Tomcat can be stopped or restarted by ending the command with either `stop` or `restart`.

```
[ec2-user ~]$ sudo service tomcat7 start
[ec2-user ~]$                                [  OK  ]
```

TestTomcat

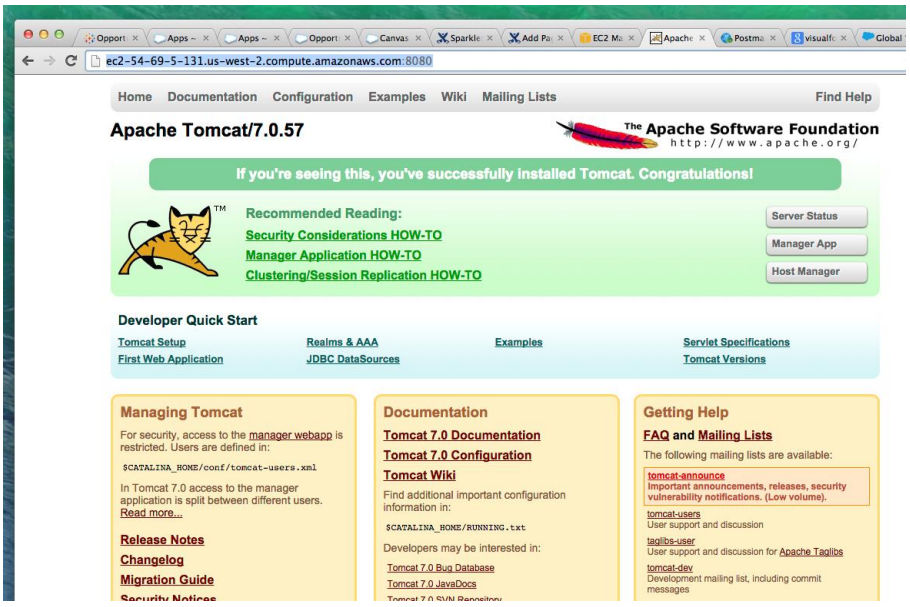
- Test that Tomcat is running by opening a browser and navigating to the following URL:

  `http://sparkler-hostname:8080/`

  For example, on an Amazon EC2 instance this would be:

  `http://ec2-hostname.us-west-2.compute.amazonaws.com:8080/`

  If Tomcat is running, you see a page like the following one:



Task 3: Enable HTTPS for Tomcat

You must set up Tomcat with a SSL certificate.

Note: Production installations of Tomcat should use a commercial SSL certificate. Otherwise, views will not render in Salesforce.com, because the browser will not trust the content.

1. If you have a certificate from a third-party CA, follow the instructions under "Installing a Certificate from a Certificate Authority" on the SSL/TLS Configuration HOW-TO page on the Apache Tomcat site. Then continue to step 2 of this procedure.

   If you do *not* have a CA certificate, use the Java `keytool` command to generate and install a self-signed certificate. The `keytool` command asks you a series of questions. Use these responses:

   - Use `changeit` for your password.
   - On the last step, press Enter to use the same keystore password.

   Windows

   ```
   c:\> c:\Program Files (x86)\Java\[javaversion]\bin\keytool.exe –genkey –alias
   tomcat –keyalg RSA –keystore "c:\Program Files (x86)\Apache Software
   Foundation\Tomcat 7.0\.keystore"
   ```

   Linux

   ```
   [ec2-user tomcat7]$ sudo su

   [root tomcat7]# cd /usr/share/tomcat7
   [root tomcat7]# $JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA -
   keystore /usr/share/tomcat7/.keystore
   ```

   The following listing shows the sequence of questions. This is the same on Linux and Windows.

```
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Sparkler
What is the name of your organizational unit?
  [Unknown]:  {Your Organization}
What is the name of your organization?
  [Unknown]:  {Your Company}
What is the name of your City or Locality?
  [Unknown]:  {Your City}
What is the name of your State or Province?
  [Unknown]:  {Your State}
What is the two-letter country code for this unit?
  [Unknown]:  {Your Country Code}
    Is CN=Sparkler, OU={Your Organization}, O={Your Company},
    L={Your City}, ST={Your State}, C={Your Country Code} correct?
   [no]:  y
Enter key password for <tomcat>
(RETURN if same as keystore password):
```

When the command finishes, the key is generated and installed in a specific directory.

Windows

```
c:\Program Files (x86)\Apache Software Foundation\Tomcat 7.0\.keystore
```

Linux

```
/usr/share/tomcat7/.keystore
```

2. Use a text editor to open the `server.xml` file.

Windows

Right click the following file and then choose **Edit**.

```
c:\Program Files (x86)\Apache Software Foundation\Tomcat 7.0\conf\server.xml
```

Linux

Use the following command:

```
[ec2-user ~]$ sudo nano /etc/tomcat7/server.xml
```

3. Uncomment the following section (remove `<!--` and `-->`)

```
<!--
<Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol"
maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS" />
-->
```

4. Edit the tag to include the location of the keystore file and the keystore password:

   Windows

   ```
   <Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol
   "maxThreads="150" SSLEnabled="true" scheme="https"
   secure="true" clientAuth="false" sslProtocol="TLS"
   keystoreFile="c:\Program Files (x86)\Apache Software Foundation\Tomcat
   7.0\.keystore" keystorePass="changeit"/>
   ```

   Linux

   ```
   <Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol"
   maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
   clientAuth="false" sslProtocol="TLS"
   keystoreFile="/usr/share/tomcat7/.keystore" keystorePass="changeit"/>
   ```

5. Save and close the file. (In Linux, use Ctrl+O, Ctrl+X.)
6. Restart Tomcat.

   Windows

   Open the **Services** window from the command line using the following command:

   ```
   c:\> services.msc
   ```

   Right-click **Apache Tomcat 7.0** and then click **Restart**.

   Linux

   Run the following command:

   ```
   [ec2-user ~]$ sudo service tomcat7 restart
   [ec2-user ~]$                              [  OK  ]
   ```

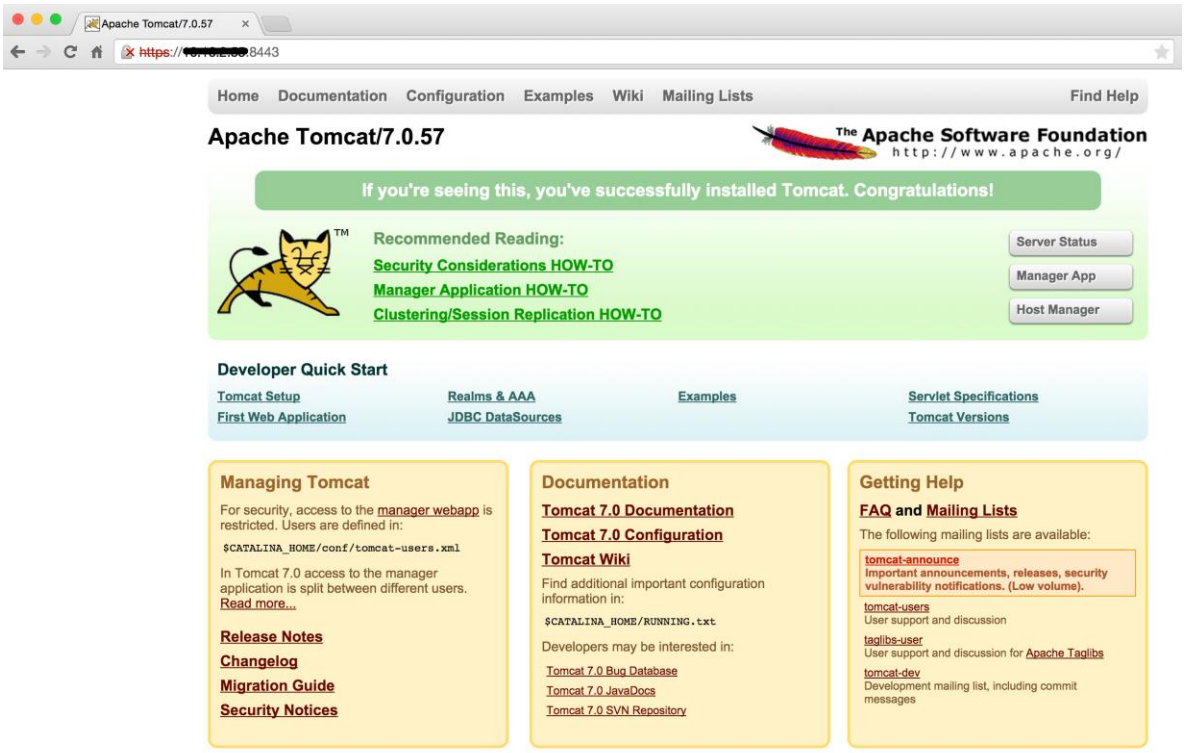
7. Test Tomcat over HTTPS by opening a browser and navigating to the following URL:

   ```
   https://sparkler-hostname:8443/
   ```

   Note that the URL uses HTTPS and port 8443. For example:

   ```
   https://ec2-hostname.region.compute.amazonaws.com:8443/
   ```

   If SSL is configured correctly for Tomcat, you see a page like the following one. If you are using self-signed certificate, you might see a browser warning. If so, accept it in order to proceed. (The red strike-through text for **https** in the URL in the following screenshot indicates this page is using a self-signed certificate.)



Task 4: Install Sparkler

The Sparkler package contains a Java WAR file with an accompanying `sparkler.xml` configuration file.

1. On your computer, navigate to the directory where you saved the Sparkler WAR file and XML files.
2. Copy Sparkler files to the Linux or Windows server where Sparkler will run. On Windows you may need to use a program such as WinSCP. On Linux, the command will look like this:

```
Myhost:~ username$ scp sparkler-1.0.2.war [USERNAME]@[YOURSERVER]:~/

Myhost:~ username$ scp sparkler.xml [USERNAME]@[YOURSERVER]:~/
```

3. Log in to the Linux or Windows server where you copied Sparkler.
4. Stop the Tomcat service if it is running

5. Change to the `webapps` directory.

   Windows

   Use Windows Explorer to go to the following path:

   ```
   c:\Program Files (x86)\Apache Software Foundation\Tomcat 7.0\webapps
   ```

   Linux

   Use the following command:

   ```
   [root home]# cd /var/lib/tomcat7/webapps
   ```

6. Copy the Sparkler WAR file to the `webapps` directory. (The XML file will be used later for configuration.)

   Windows

   Rename `sparkler-reference-1.0.0.war` in your home directory to `sparkler.war`. Then drag it to `c:\Program Files (x86)\Apache Software Foundation\Tomcat 7.0\webapps`.

   Linux

   Use the following command:

   ```
   [root webapps]# sudo cp /home/ user/sparkler-1.0.2.wa /sparkler.war
   ```

   You can deploy Sparkler to a different URL by changing the target location name of the file. The name of the file corresponds to the URL path for the application. For example, deploying the file as `sparkler-test.war` will result in the URL `https://sparkler-hostname:8443/sparkler-test`. In that case, since the target file name in the example is `sparkler.war`, the directory `/sparkler` will be created.

   Make a note of the exact file name for the `.war` file—for example, here, the name is `sparkler.war`. You'll need to use the same filename later when you work with the XML file (with a different filename extension).

7. Restart the Tomcat service.
8. Test Sparkler over HTTPS by opening a browser and navigating to the following URL:

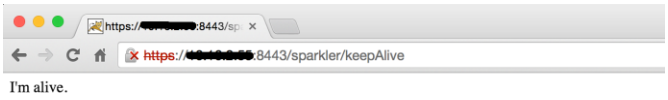
   ```
   https://sparkler-hostname:8443/
   ```

   Note: That the URL uses `https` and port 8443 and is case sensitive.

   For example:

   ```
   https://ec2-hostname.region.compute.amazonaws.com:8443/sparkler/keepAlive
   ```

   If Sparkler is configured correctly, you see a page like the following one:



Task 5: Configure Sparkler to Use a Tableau Server Self-Signed SSL Certificate (If Required)

By default, Apache Tomcat will not support SSL certificates that are not signed by a commercial certificate authority (CA). In order to support self-signed SSL certificates, or SSL certificates signed by an internal or corporate certificate authority, you must perform the following steps. If you have a certificate issued by a commercial CA, you can skip this task.

1. When you create or obtain your self-signed certificate, ensure that requests to Tableau Server will use the same hostname as the canonical name (CN) of the SSL certificate being used by Tableau Server:
    - Example request URL: `https://tableau.mycompany.com`
    - Example Certificate CN: `CN=tableau.mycompany.com`
2. On the Sparkler server, convert the self-signed SSL certificate to the DER format required by Java by running the following command.

    Note: The OpenSSL utility is preinstalled on Linux. If you're running Windows and don't already have OpenSSL installed, download and install it from the following site: https://www.openssl.org/related/binaries.html.

    ```
    openssl x509 -in <filename-of-SSL-cert-in-PEM-format> -inform PEM -out
    tableau.der -outform DER
    ```

3. Verify the DER-formatted file by running the following command.

    Windows

    ```
    %JAVA_HOME%/bin/keytool -printcert -v -file tableau.der
    ```

    Linux

    ```
    $JAVA_HOME/bin/keytool -printcert -v -file tableau.der
    ```

    If the file is correct, you see output similar to the following (`Owner` and `Issuer` will be unique to your key and organization):

```
Owner: CN=*.tableausoftware.com, OU=Netops, O=Tableau Software, L=Seattle,
ST=Washington, C=US
Issuer: CN=thawte SHA256 SSL CA, O="thawte, Inc.", C=US
Serial number: xxxxxxxxxxxxxxxxxxxxxxxxxxxxb5d4
Valid from: Thu Oct 23 17:00:00 PDT 2014 until: Sun Nov 29 15:59:59 PST 2015
```

4. Import the SSL certificate in DER format into the Java `cacerts` keystore by entering the following command.

   Windows

   ```
   %JAVA_HOME%/bin/keytool -import -trustcacerts -alias tableauserver -file full-path-to-DER-file/tableau.der -keystore %JAVA_HOME%/lib/security/cacerts
   ```

   Linux

   ```
   sudo $JAVA_HOME/bin/keytool -import -trustcacerts -alias tableauserver -file full-path-to-DER-file/tableau.der -keystore $JAVA_HOME/lib/security/cacerts
   ```

   If you are prompted for a keystore password, use the Java default password, which is `changeit`.

5. Verify that the SSL certificate was imported by running the following command.

   Windows

   ```
   %JAVA_HOME%/bin/keytool –list -v –keystore %JAVA_HOME%/lib/security/cacerts
   ```

   Linux

   ```
   $JAVA_HOME/bin/keytool –list -v –keystore $JAVA_HOME/lib/security/cacerts
   ```

   If the SSL certificate has been imported, you see the certificate listed as one of the certificates in the keystore. Search for the alias `tableauserver` that you used above in the output. (The certificate might be listed at the bottom.)

6. Restart Apache Tomcat.

Task 6: Set Up Tableau Server Trusted Authentication

To work with Sparkler, Tableau Server must be configured with trusted authentication enabled. (If you are using Tableau Online, you can use SAML for authentication. For more information, see Appendix I: Connecting to Tableau Online with SAML.)

Note: Automatic login must not be enabled on Tableau Server, because trusted authentication cannot be used with this feature enabled. If the server was installed with automatic login, you must re-run the installer and disable this feature after initialization.

---

1. Get the IP addresses of trusted clients that will be connecting to Tableau Server. The IP address of the server hosting Sparkler is the most important.
2. Get the IP addresses of any proxies or load balancers that are being used to route traffic to Tableau Server.

3. On the computer where Tableau Server is running, open a command window as an administrator.
4. Run the following command to change directories to the location where the `tabadmin` command is installed.

   ```
   cd c:\Program Files\Tableau\Tableau Server\version\bin
   ```

5. Set the list of trusted hosts by entering the following command:

   ```
   tabadmin set wgserver.trusted_hosts "10.88.36.216, 10.88.36.218, 10.88.36.220,
   10.88.36.221"
   ```

   Substitute your own list of trusted clients for `10.88.36.216` and the other placeholders.

6. Set the list of trusted proxies by entering the following command:

   ```
   tabadmin set gateway.trusted "10.1.1.69, 10.1.1.162, 10.1.1.157, 10.1.1.84"
   ```

   Substitute your own list of trusted proxies for `10.1.1.69` and the other placeholders.

7. Save the configuration settings and restart the server by entering the following commands:

   ```
   tabadmin configure
   tabadmin restart
   ```

8. When the server is restarted, verify that the trusted hosts have been added by examining the `tabsvc.yml` file in the following folder:

   ```
   c:\ProgramData\Tableau\Tableau Server\config\
   ```

   Verify that the following settings appear. (The values specific to your installation are in italics in this example.)

   ```
   gateway.trusted_hosts: tableauserver.mycompany.com
   vizqlserver.trustedticket.log_level: ?debu
   ssl.key.file: C:\Program Files\Tableau\Tableau Server\SSL\sfdc-demo.key
   ssl.cert.file: C:\Program Files\Tableau\Tableau Server\SSL\sfdc-demo.cer
   gateway.public.host: tableauserver.mycompany.com
   vizqlserver.initialsql.disabled: false
   gateway.trusted: 10.1.1.69, 10.1.1.162, 10.1.1.157, 10.1.1.84
   wgserver.trusted_hosts: 10.88.36.216, 10.88.36.218, 10.88.36.220, 10.88.36.221
   config.version: 12
   service.init.state: start
   gateway.public.port: 443
   ```

   Note: You can view changes in this file, but you cannot change the values. To change values, you must use the `tabadmin set` command.

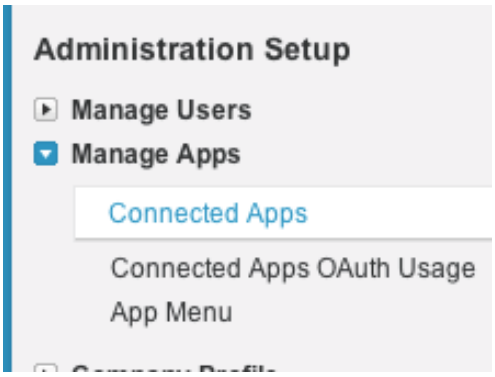
Configure Salesforce.com for the Sparkler Connected App

To configure the Sparkler Canvas Connected App in Salesforce, you need to be a Salesforce System Administrator. You also need to information about where Sparkler is deployed, such as the URL for the Sparkler host (and its port).

Task 1: Create the Connected App

1. Log in to Salesforce.
2. From the **<your name>** menu, select **Setup**.

3. Select **Create** > **Apps** > **Connected Apps** > **New**.



4. Make the following settings for the new app.

   Under **Basic Information**:

   - **Connected App Name**: The user-friendly name of your app.

   Note: The examples in this section use the name `Sparkler_Connector`.

   - **API Name**: The name of the app as it will be referenced in code.
   - **Contact email:** The contact email for this app.

Under **API (Enable OAuth Settings)**:

- **Enable oAuth Settings**: enabled (checked).
- **Callback Url**: `https://Sparkler-hostame:Sparkler-port/Sparkler-directory/keepAlive`

  Example: `https://sparkler.mycompany.com:8443/sparkler/keepAlive`

- **Selected oAuth Scopes**: The information that the app will request. The app will need access to your basic information (ID, profile, email, address, phone).

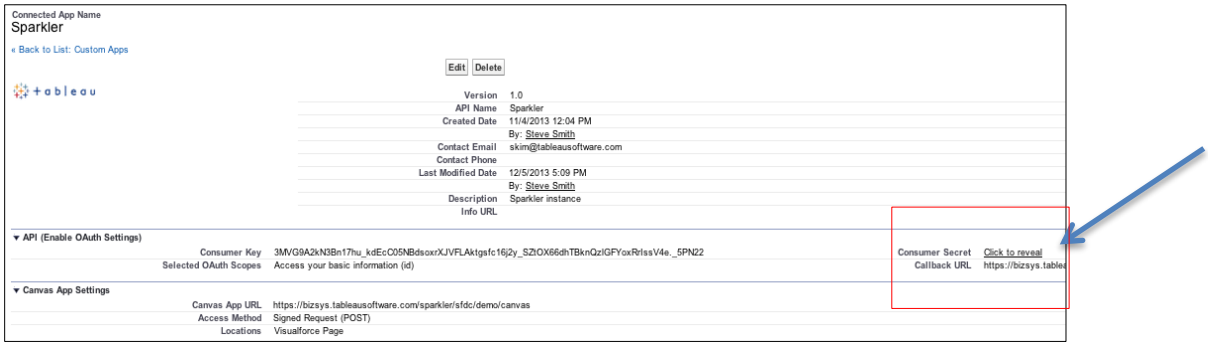
Under **Canvas App Settings**:

- **Force.com Canvas**: enabled (checked)
- **Canvas App Url**: `https://Sparkler-hostame:Sparkler-port/Sparkler-directory/sfdc/canvas`

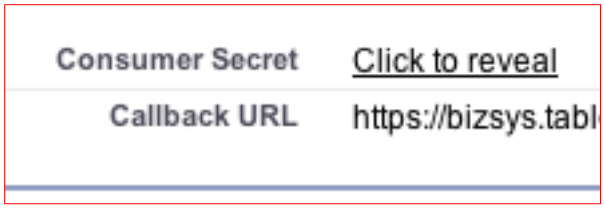
  Example: `https://sparkler.mycompany.com:8443/sparkler/sfdc/canvas`

- **Access Method**: Signed Request (POST)
- **Locations**: Chatter Feed, Publisher, VisualForce Page

5. Click **Save**.
6. Click **Setup** > **App Setup** > **Create** > **Apps** > **Connected Apps** > *App Name* to show the application page.



7. Click **Click to reveal** next to **Consumer Secret**. Salesforce displays the consumer secret for your app.



8. Copy this key and store it in a secure place. You will need it later when you set the value of `sparkler.sfdc.consumerSecret` in the `sparkler.xml` configuration file.

Task 2: Update the Connected App

1. In Salesforce, select **Setup** > **Create** > **Apps** > **Connected Apps** > **Manage** (to the left of the *Sparkler App Name*) > **Edit**.
2. Select **oAuth Policies** and then change **Permitted Users** to **Admin Approved Users are Pre-Authorized** and then click **Save**:
3. Select **Setup** > **Create** > **Apps** > **Connected Apps** > *Sparkler App Name* > **Manage** > **Profiles** > **Manage Profiles**.

   You won't see the **Manage Profiles** option until you've completed the preceding step.

4. Add one or more profiles, such as `System Administrator`.

Task 3: Configure Sparkler

Sparkler configuration is handled via the `sparkler.xml` file that comes with the Sparkler WAR file. For more details about configuration parameters, see Appendix C: Sparkler Configuration Parameters.

1. Copy the `sparkler.xml` file from the installation location to the configuration directory of your Tomcat installation on the Sparkler server.

   Windows

   Copy the file to the following location:

   ```
   C:\Program Files (x86)\Apache Software Foundation\Tomcat
   7.0\conf\Catalina\localhost
   ```

   Linux

   Use the following command:

   ```
   [root ~]# sudo cp ~/sparkler.xml /etc/tomcat7/conf/Catalina/localhost
   ```

2. Edit the `sparkler.xml` file.

   Windows

   Right-click `sparkler.xml` and then click **Edit**.

   Linux

   Use the following commands:

   ```
   [root ~]# cd /user/share/tomcat7/conf/Catalina/localhost
   [root localhost]# sudo nano sparkler.xml
   ```

The optional parameters are commented out. The `sparkler.xml` file contains descriptions of all optional parameters, which are also listed in Appendix C: Sparkler Configuration Parameters. You can enable these values by uncommenting the `<Environment>` tag.

3. Enter the value for the `sparkler.sfdc.consumerSecret` parameter. This was the value you stored earlier when you configured the Salesforce Connected App.
4. Enable the status page by uncommenting the `sparkler.app.statusPage` parameter and setting the parameter to `true`.
5. Configure the Salesforce field to map to the Tableau user. Set `sparkler.sfdc.userIdentifierField` to one of the following values:

   - `username`: This takes the Salesforce username and maps it directly to the Tableau username. For example, if the Salesforce username is `joe@example.com`, `joe@example.com` is used as the Tableau username as well.
   - `username.user`: This takes only the name but not the `@` character or domain name from Salesforce username as the Tableau username. For example, the Salesforce user name `joe@example.com` becomes `joe` as the Tableau username.
   - `email`: This takes Salesforce email address field for the user and maps it directly to the Tableau username. For example if `joe@example.com` is the Salesforce email name, `joe@example.com` is used as the Tableau username.
   - `email.user`: This takes only the characters to the left of the `@` from the Salesforce email address as the Tableau username. For example, if `joe@example.com` is the Salesforce email name, `joe` is used as the Tableau username.
   - `userId`: This maps the Salesforce user ID and directly to the Tableau username. For example, if `005x0000001SyyEAAS` is the `userId`, `005x0000001SyyEAAS` is used as the Tableau username.
   - `signedIdentity`: This takes the value of the `signedIdentity` setting and maps it directly to the Tableau username. You set this value if you're using custom mapping. For more information, see Appendix J: Custom Mapping and the "signedIdentity" Parameter.

   Important: If you use `signedIdentity` for user mapping, you must make sure that the user identity information is signed. For more information, see Appendix J.

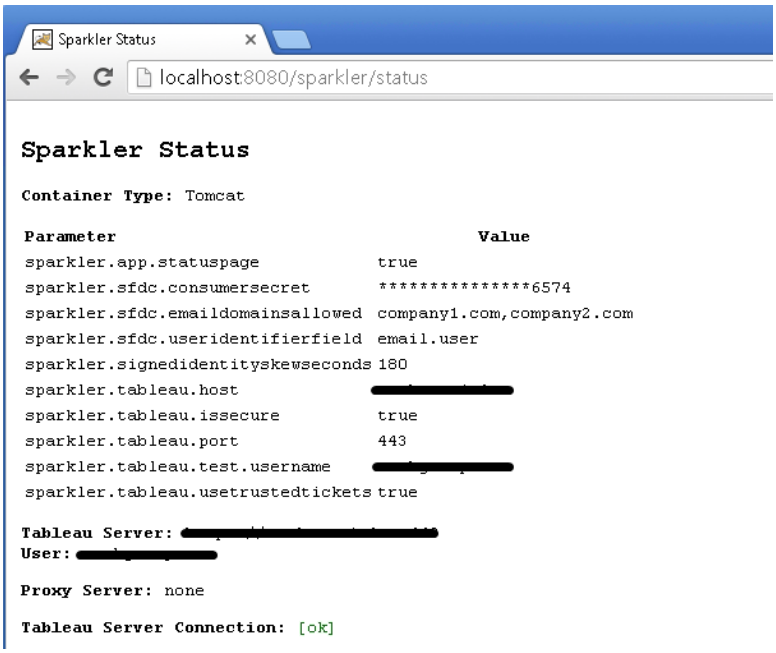
6. Configure the allowed email domains by setting `sparkler.sfdc.emailDomainsAllowed` to include the domains controlled by your company, such as `@example.com`.
7. Restart Tomcat.

Task 4: Validate Sparkler Configuration

1. In a browser, enter the following URL:

   `https://Sparkler-host:8443/sparkler/status`

   The page displays all of the parameters that were set, and indicates in red any required parameters that are missing or improperly formatted. In addition, the page lists the URL for Tableau Server trusted tickets and for the proxy server (if either or both has been set). If trusted tickets are enabled, a test request is made to Tableau Server and the results are displayed.



2. If any of the parameters are listed in red or if the Tableau Server Connection is not displayed with a status of **ok**, adjust the parameters as needed to resolve the issue. For information about debugging, see Appendix F: Debugging Sparkler Application Configuration.

Task 5: Test the Connected App

- In Salesforce, select **Setup** > **Canvas App Previewer** > *Sparkler App Name*.

  This makes a request from Salesforce to Sparkler. If Sparkler is configured correctly, you see a page with information about the logged in Salesforce user, like the one shown in the preceding procedure.

  If you see a blank page, Salesforce is not communicating correctly with Sparkler. For help with debugging, see Appendix F: Debugging Sparkler Application Configuration, specifically the section Salesforce Canvas App Previewer.

If you see an error icon (in Chrome, this looks like a sad face), the issue might be that the browser does not trust the SSL certificate that Sparkler is using. For example, this can happen if you are using a self-signed SSL certificate.

Configure the VisualForce Page

This section describes how to embed a Sparkler Canvas App VisualForce Component in a Salesforce VisualForce page. The sample VisualForce pages can be found in the `examples/sfdc/vfp` directory of the distribution. For a list of parameters that you can set, see Appendix D: VisualForce View Embedding Parameters.

Types of VisualForce Pages

VisualForce pages allow users to view pages directly in Salesforce. There are two types:

- Extensions to Salesforce objects. These pages start with a single Salesforce record, and then add data and visual capabilities. The pages can generally access all of the fields in the object that the user has permissions for, in addition to any custom programmed fields. This is the most common type of VisualForce page.
- Custom pages. These are completely custom web views that are not associated with a specific record.

VisualForce pages can be accessed directly or embedded in a Salesforce record page layout.

Task 1: Create VisualForce Pages

1. Select **Setup** > **App Setup** > **Develop** > **Pages**.
2. Create the VisualForce pages for the Accounts Dashboard, Accounts, and Opportunities pages.
3. In the VisualForce markup section, copy and paste the VisualForce page from `examples/sfdc/vfp`.
4. After the pages have been created, click the **Security** link for each page and make sure the intended user has access to the page.

Task 2: Create a Tab Page

1. Select **Setup** > **App Setup** > **Create** > **Tabs**.
2. Create a new tab in VisualForce Tabs.
3. Verify that the tab you just created is shown on the toolbar. If it doesn't appear, click **+** on the toolbar and then click the **Customize my tabs** button.

Task 3: Embed the Dashboard in an Account/Opportunity

1. Enable the Accounts dashboard when a user selects an account by selecting **Setup** > **App Setup** > **Customize** > **Accounts** > **Page Layouts**.

   To use the Tableau-provided Sample workbooks, follow the instructions in Appendix A: Using the Sample Tableau Workbooks.

2. Edit the existing page layout or add new one, depending on your situation.
3. Under **Account Layout**, select the embedded Account VisualForce page and drag the page to where you want it.
4. Double-click the VisualForce page and if necessary, specify the height of the page and the width.
5. Enable the Opportunities dashboard when a user selects an opportunity by selecting **Setup** > **App Setup** > **Customize** > **Opportunities** > **Page Layouts**.
6. Repeat steps 2 through 4 to embed the dashboard for Opportunity.

Canvas App VisualForce Component

The information in this section is about the Canvas App VisualForce component, which is a way of embedding content easily in a VisualForce pages. The Canvas App VisualForce Component creates an `<iframe>` element into which the view is rendered.

The fields described below are part of the VisualForce component. The `parameters` field is the primary method of feeding data to Sparkler. In general, you pass only the following values:

- `applicationName` or `developerName`. This value is required.
- `ts.name`. This value is required.
- `height` and `width`.
- A subset of available parameters, typically to override defaults or values that are established using Sparkler configuration parameters.

The example that follows shows a complete example (all fields listed).

```
<apex:canvasApp
    applicationName="Sparkler_Connector"
    height="536px"
    width="654px"
    parameters="
    {
        'ts.name':'OpportunityDashboardDreamforce/OpportunityDashboard',
        'ts.width':'1000',
        'ts.height':'400',
        'ts.javascriptLib':
             'https://mytableauserverdomain/javascripts/api/viz_v1.js',
        'ts.hostUrl':'https://mytableauserverdomain/',
        'ts.siteRoot':'/t/myOrganizationSite',
        'ts.tabs':'no',
        'ts.toolbar':'yes',
        'ts.trustedTicket.host':'mytableauserverdomain',
        'ts.filter':'Opportunity_Parameter={!Opportunity.Id}'
    }" />
```

The following table describes the attributes in a `<apex:canvasApp>` element and the values that you specify for embedding a view in a Salesforce page. For more information about the `apex:canvasApp` component, see the following pages in the Salesforce developer documentation:

- apex:canvasApp Component
- Global Variables

Attribute	Description	More information
`apex:canvasApp`	Tells Salesforce to use the built-in Canvas App VisualForce component.	
`applicationName="Sparkler"`	Tells Salesforce to use the Connected App named `Sparkler`.	**Setup > Create > Apps > Connected Apps > Sparkler**
`height="536px"`	Tells Salesforce to render an HTML `<iframe>` element that is 536 pixels high.	
`width="654px"`	Tells Salesforce to render an HTML `<iframe>` element that is 654 pixels wide.	

The following table describes the attributes in the `<parameters>` element. The values in this element are passed to Sparkler and to the Tableau Server embed script.

Parameter	Description
`'ts.name':'OpportunityDashboardDreamforce/OpportunityDashboard'`	Specifies the name of the workbook and view to render.
`'ts.width':'1000'`	Tells the page to render the view at 1000 pixels wide. This value can be different from the value passed for the `<iframe>` element.
`'ts.height':'400'`	Tells the page to render the view at 400 pixels high. This value can be different from the value passed for the `<iframe>` element.
`'ts.javascriptLib':'https://mytableauserverdomain/javascripts/api/viz_v`	Tells the view embed code to use the specified

`1.js'`	JavaScript library. In most cases, you don't pass a value here; instead, you set the `sparkler.tableau.javascriptLib` configuration parameter for Sparkler.
`'ts.hostUrl':'https://mytableauserverdomain/'`	Specifies the URL of the Tableau Server where the view to be embedded is. In most cases, you don't pass a value here; instead, you set the `sparkler.tableau.externalURL` configuration parameter for Sparkler.
`'ts.siteRoot':'/t/myOrganizationSite'`	Specifies the site where the view is located. This value is included only if you are using a Tableau Server instance that has multiple sites.
`'ts.tabs':'no'`	Tells the view embed code to include tabs.
`'ts.toolbar':'yes'`	Tells the view embed code to include the toolbar.
`'ts.trustedTicket.host':'mytableauserverdomain'`	Specifies the name of the trusted ticket host. In most cases, you don't pass a value here; instead, you set the `sparkler.tableau.host` configuration parameter for Sparkler. If the value is included, the host name must match the server specified in the `ts.hostUrl` parameter. This setting allows Sparkler to communicate with multiple Tableau Servers, since the Sparkler default configuration allows only a single Tableau Server to be specified.
`'ts.filter':'Opportunity_Parameter={!Opportunity.Id}'`	Specifies a filter for the embedded view. In the example, the value is created by taking the ID of the Opportunity record from the VisualForce page on which the Canvas App component is embedded, and then putting it into a filter string. For example, if the `Opportunity.Id` value is `006i0000001vvYD`, the filter passed to the view will be `"Opportunity_Parameter=006i0000001vvYD"`

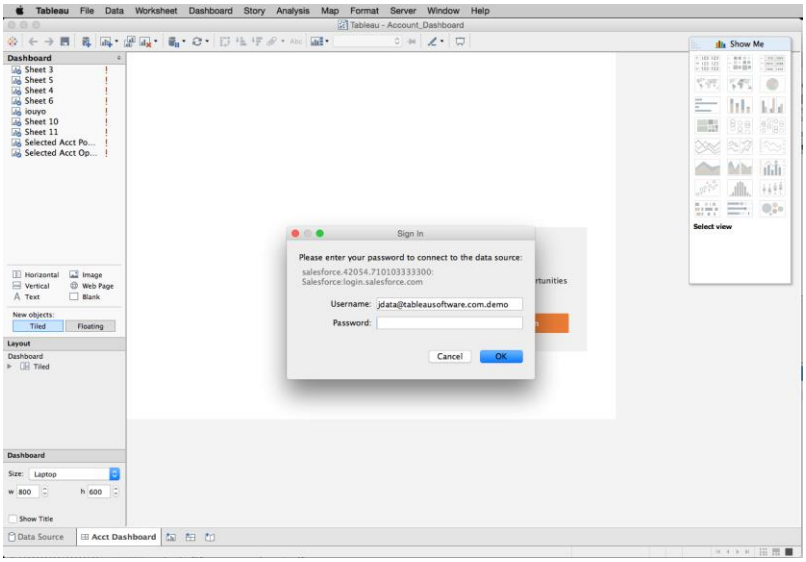
Appendix A: Using the Sample Tableau Workbooks

This appendix describes how to work with the sample workbooks that are included with the Sparkler installation. If you have workbooks you want to embed instead of the sample workbooks, you can ignore this section.

The following workbooks can be found bundled with Sparkler and can be published to your Tableau Server:

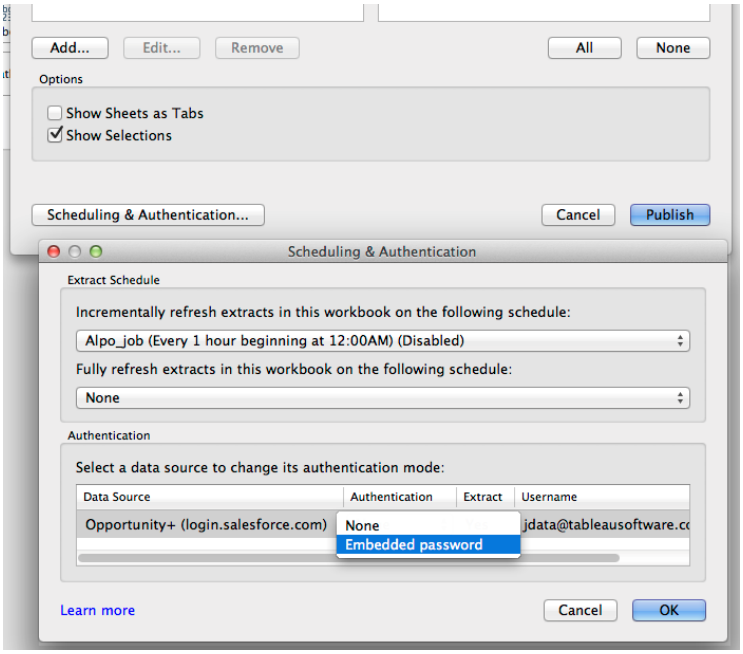
- `Accounts_Dashboard.twb` and `Quota for Accounts_Dashboard.xls` are referenced when a Salesforce user selects the **Accounts Dashboard** tab. Both of these files must be in the same directory.
- `Account_Dashboard.twb` is referenced when a Salesforce user selects an account.
- `Opportunity_Dashboard.twb` is referenced when a Salesforce user selects an opportunity.

The sample workbooks refer to data source paths, usernames, and passwords that will not be correct for your environment. Update the settings to reflect your Salesforce environment (see screenshot below). To do this, open each `.twb` file individually. You will be prompted for your Salesforce login. Change the default username from `jdata@tableausoftware.com.demo` to your username and enter your password. Validate that the resulting Dashboards populate with your Salesforce data.



In addition, you can modify the `Quota for Accounts_Dashboard.xls` spreadsheet with your own data, and see it reflected in the `Accounts_Dashboard.twb` workbook when you refresh.

For each `.twb` file, publish the workbook to your Tableau Server. In the **Publish** dialog box, click **Scheduling and Authentication**, enter the refresh frequency, and specify that you want to embed credentials. Tableau Server will not be able to refresh the extract if you do not embed your credentials. Click **OK** and then click **Publish**.



For more information about publishing workbooks, see How to Publish Workbooks to a Tableau Server in the Tableau documentation.

You can also create your own workbooks and publish them to Tableau Server. For information about how to connect to Salesforce, see New in Tableau 8: Salesforce Direct Connector on the Tableau blog.

The workbooks have URL Actions that allow the user to navigate to an opportunity or an account. Update the actions to reflect the appropriate Salesforce instance for your company (for example, `na15.salesforce.com`).

The accounts dashboard (`Accounts_Dashboard.twb`) filters based on the user's full name. If you want it to filter using the user's ID, email, or username instead, remove the name filter and create a different filter. You may also need to modify the drop down showing the salesperson's name to fit your situation. For more details about filters, see Appendix N: Additional Information.

Troubleshooting the sample workbooks

- If you see the error `There was a problem connecting to …`, log in to Salesforce and reset your security token through **Setup** > **My Personal Information** > **Reset security token**. The new token will be sent to you by email.
- If you see the error `Tableau Data Engine Error 4`, close Tableau, re-open the `.twb` file, and ensure that you are entering your Salesforce username and password into the first dialog box that appears.
- After you connect to Salesforce, if the dashboard appears blank, hold the mouse pointer over the top right corner of any worksheet in the dashboard and click **Go to worksheet**. From the

worksheet, find the fields on the left with red exclamation points. One by one, right-click these and choose **Replace references**.

Appendix B: Running Tomcat and Tableau Server on the Same Computer

If Tomcat and Tableau Server are running on the same computer, and Tomcat is running without SSL enabled, Tomcat must be modified to run on a different port than the default of 8080. Running Tomcat without SSL is a non-standard configuration and would only occur if Salesforce is communicating with Sparkler via an SSL proxy.

1. Edit the `server.xml` file.

   Windows

   Use a text editor to open this file:

   ```
   c:\Program Files (x86)\Apache Software Foundation\Tomcat 7.0\conf\server.xml
   ```

   Linux

   Use the following command:

   ```
   [ec2-user ~]$ sudo nano /etc/tomcat7/server.xml
   ```

2. Change the `port` value from 8080 to 9090.

   ```
   <Connector port="9090" protocol="HTTP/1.1"
                  connectionTimeout="20000"
                  redirectPort="8443" />
   ```

3. Restart Tomcat.

Appendix C: Sparkler Configuration Parameters

The following table lists Sparkler configuration parameters. These are set a in the `sparkler.xml` file.

Parameter	Description	Default
`sparkler.sfdc.consumerSecret`	The consumer secret from the Salesforce Connected App configuration.	
`sparkler.tableau.useTrustedTickets`	A value that instructs Sparkler to perform trusted ticket requests against Tableau Server. If this value is true, the trusted ticket username must be passed in every embedded Salesforce Canvas App component.	false
`sparkler.tableau.host`	The hostname of the Tableau Server instance. Does not include `https://` or `http://`.	null
`sparkler.tableau.port`	The port on which Tableau Server is listening.	80
`sparkler.tableau.isSecure`	A value that indicates whether SSL is enabled on Tableau Server.	false
`sparkler.tableau.test.userName`	The Tableau Server username to use for testing the trusted ticket request on the Sparkler status page.	null
`sparkler.tableau.externalURL`	The external URL for Tableau Server to use in the embedded view HTML that's sent to the user's browser. This is not the URL used by Sparkler to communicate with Tableau Server. Only set this parameter if the external URL for Tableau Server is different than that used for trusted tickets. This must be a complete URL and begin with either `http://` or `https://`.	null
`sparkler.tableau.trustedTicketSiteId`	The URL of the Tableau Server site that uses Sparkler, if the server has multiple sites. (Do not set this parameter if the server does not have multiple sites.) For example, if the value of the `sparkler.tableau.siteRoot` configuration parameter is `/t/mysite`, this value is `mysite`.	null
`sparkler.tableau.siteRoot`	For Tableau Server instances that contain multiple sites, the site that the user and the view belong to.	null
`sparkler.tableau.showTabs`	The value used as the `showTabs` value for embedded views.	false
`sparkler.tableau.showToolbar`	The value used as the `showToolbar` value for embedded views.	false
`sparkler.tableau.javascriptLib`	An alternative location for JavaScript library that is used to embed views.	`/javascripts/api/viz_v1.js`

`sparkler.proxy.host`	The host name or IP address for the proxy server. If this value is specified, the `sparkler.proxy.serName` and `sparkler.proxy.password` parameters are required.	null
`sparkler.proxy.port`	The port on which the proxy server is listening.	80
`sparkler.proxy.isSecure`	A value that indicates whether the proxy is running with SSL.	false
`sparkler.proxy.userName`	The proxy server username.	null
`sparkler.proxy.password`	The proxy server password.	null
`sparkler.log.level`	The log level for Sparkler logging. This setting is used for development purposes, and can be ignored for production installations.	
`sparkler.log.rootLevel`	The root level logging for Sparkler. This setting is used for development purposes, and can be ignored for production installations.	
`sparkler.app.statusPage`	A value that enables the Sparkler status page. When this value is true, a warning message appears on all pages that contain embedded views. This feature should be disabled in production environments.	false
`sparkler.sfdc.org.`***`{org}`***	A value equivalent to `sparkler.sfdc.consumerSecret` for the URL, where ***`{org}`*** is replaced with a value specific to your installation. For more information, see the listing for the `/canvas/sfdc/{org}` endpoint in Appendix D: Sparkler Endpoints.	
`sparkler.sfdc.userIdentifierField`	One of the following values: • `username` • `username.user` • `email` • `email.user` • `userId` • `signedIdentity` Signed identity is used when you map custom fields, use a single Salesforce account for all users who access Tableau, or access multiple Tableau sites.	`email.user`
`sparkler.sfdc.emailDomainsAllowed`	A comma-separated list of domains that are allowed as part of the `email` or `email.user` values. This value is used only if `sparkler.sfdc.userIdentifierField` is set to `email.user`. If an email domain isn't in this list, a request	

	that includes that email domain will be rejected. The domain names must include the leading `@` character. Example: `@tableau.com, @tableau.test.com`	
`sparkler.tableau.signedIdentitySkewSeconds`	The clock skew that's allowed between Salesforce and Sparkler—that is, the number of seconds' difference between the servers. Setting this value accounts for small differences between the clocks in the different processes. This value is used only if `sparkler.sfdc.userIdentifierField` is set to `signedIdentity`. You do not need to set this value unless the default of 300 seconds is inadequate for your environment. **Note**: Do not set this value to a high number. The default (300 seconds, or 5 minutes) is usually more than enough to account for small difference in the clocks. Setting the clock skew to a high value can potentially allow a replay attack, in which a user attempts to send the same request at a later time.	`300`

Appendix D: VisualForce View Embed Parameters

Embedding views with Sparkler is based on the native Salesforce functionality for this task. Sparkler parameters allow data to be passed to Sparkler, which in turn allows them to be embedded in a view-embed script. For more information about embed parameters, see Tableau Server Embed Parameters in the Tableau Server documentation.

The following table lists the parameters you can set.

Parameter	Required?	Description
`ts.name`	Yes	The name of the workbook or view.
`ts.width`	No	The width of the embedded dashboard itself. This can be different from the width of the `<iframe>` element created by the Canvas App component. The default is the `width` parameter provided to the Canvas App VisualForce component.
`ts.height`	No	The height of the embedded dashboard itself. This can be different from the height of the `<iframe>` element created by the Canvas App component. The default is the `height` parameter provide to the Canvas App VisualForce component
`ts.javascriptLib`	No	The JavaScript library used for embedding. The default is the value in the `sparkler.tableau.javascriptLib` configuration parameter.
`ts.hostUrl`	No	The URL of the Tableau Server instance. The URL must include a trailing slash. The default is a URL created by Sparkler using the `sparkler.tableau.host`, `sparkler.tableau.port`, and `sparkler.tableau.isSecure` configuration parameters. This default URL will be formatted like this: `http://`***`sparkler.tableau.host`***`:`***`sparkler.tableau.port/`*** or `https://`***`sparkler.tableau.host`***`:`***`sparkler.tableau.port/`***
`ts.siteRoot`	No	For Tableau Server instances that use sites, the site that the user and view belong to. The default is the value of the `sparkler.tableau.siteRoot` configuration parameter.
`ts.tabs`	No	Whether to show tabs. The default is the value of the `sparkler.tableau.showTabs` configuration parameter if that variable is set; if that parameter has no value, the default is `no`.
`ts.toolbar`	No	Whether to show the dashboard toolbar. The default

		is the value of the `sparkler.tableau.showTabs` configuration parameter if that variable is set; if that parameter has no value, the default is `no`.
`ts.trustedTicket.host`	No	The hostname for Tableau Server using trusted authentication. The default is the value of the `sparkler.tableau.host` configuration parameter if that variable is set; if that parameter has no value, the default is null.
`ts.trustedTicket.signedIdentity`	No	A value that is used to pass the signed identity for custom mapping. For details, see Appendix J: Custom Mapping and the "signedIdentity" Parameter.
`ts.filter`	No	The filter to pass to the embedded view. For more information about filters, see _Appendix L: Additional.

Appendix E: Sparkler Endpoints

The following table lists all Sparkler URLs that can be accessed via a web browser for embedding dashboards, testing, and debugging Sparkler.

Endpoint	Description
`/keepAlive`	Tests that the application is running. Returns a static message.
`/sfdc/canvas`	Uses the consumer secret set by the `sparkler.sfdc.consumerSecret` environment variable. This is the default path tested by the Salesforce Canvas Tester.
`/sfdc/canvas/{org}`	Tests the end-to-end functionality of a signed request originating from Salesforce. A request made using this endpoint uses the consumer secret set by the `sparkler.sfdc.org.`***`{org}`*** environment variable. The value of ***`{org}`*** in the environment variable is converted to lower case by Sparkler, and that is how it appears on the status page. For example, the endpoint `/sfcd/canvas/myorg` uses the value set by `sparkler.sfdc.org.MYORG` or `sparkler.sfdc.org.myorg` environment variable. This endpoint is rarely used by most Sparkler implementers.
`/status`	Displays all configured Sparkler parameters, as well as any required parameters that are missing, and parameters that have invalid formatting. If trusted authentication is enabled in Sparkler, Sparkler displays the address of the Tableau Server configured for trusted authentication. If the proxy server address and credentials are set, Sparkler displays these values as well. This endpoint also displays the status of a test connection made to Tableau Server for trusted tickets. If this test connection failed, the failure message is displayed.

Appendix F: Debugging the Sparkler Application Configuration

This appendix contains helps you debug configuration issues in Sparkler. For additional information about Sparkler configuration, see the section Task 3: Configure Sparkler earlier in this document. For reference information, see Appendix C: Sparkler Configuration Parameters and Appendix E: Sparkler Endpoints.

The general method for debugging is to enable and use the Sparkler status page. If the status page has not been enabled in the `sparkler.xml` configuration file, the status page displays the following message:

```
Status page disabled. Set configuration parameter 'sparkler.app.statuspage' to 'true'
to enable.
```

To get to the Sparkler status page, in your browser, enter the following URL:

```
https://SparklerHostName:8443/SparklerPath/status
```

where:

- `SparklerHostName` is the hostname where Sparkler is deployed.
- `SparklerPath` is where the Tomcat web App is deployed. Usually this is `sparkler`.

Sparkler errors and debug messages are logged in Tomcat's `catalina.out` file. In Windows, this file is located in the following folder:

```
Tomcat-install/logs/catalina.outtall/logs/catalina.out
```

On Linux, this file is usually in the following location:

```
/usr/share/tomcat7/logs/catalina.out
```

You might need to be an administrator (Windows) or root (Linux) to view this file.

In most instances, the status page should be sufficient for debugging and there is no need to view the log files directly.

Testing the Sparkler Consumer Secret

In your browser, enter the following URL:

```
https://SparklerHostName:8443/SparklerPath/status
```

On success, you see the consumer secret displayed in the list of parameters. This value appears in black and has all but the last four digits masked:

```
sparkler.sfdc.consumersecret***************5639
```

If `sparkler.sfdc.consumersecret` is not set, you see the parameter highlighted in red:

```
sparkler.sfdc.consumersecret
```

Set the `sparkler.sfdc.consumersecret` configuration parameter in the `sparkler.xml` file. Tomcat detects that the file was modified and restarts the application automatically. If Tomcat does not restart automatically, restart it manually.

Testing the Trusted Ticket for Sparkler

In your web browser, enter the following URL:

```
https://SparklerHostName:8443/SparklerPath/status
```

On success, you see the following at the end of the status page:

```
Tableau Server Connection: [ok]
```

The sections that follow describe various error conditions that might arise when you perform these tests.

Issue: Using Self-Signed SSL Certificate that has not been added to Java

If Sparkler has been configured with SSL using a self-signed certificate, Sparkler cannot communicate with Tableau Server without additional configuration. In that case, this is the case, you see an exception like the following at the bottom of the status page:

```
Tableau Server Connection: [fail]
Reason: unable to find valid certification path to request targets
```

Follow the steps under Configure Sparkler to Use Tableau Server Self-Signed SSL Certificates.

Issue: SSL Certificate CN does not match Tableau Server Host Name

If your SSL certificate does not match the hostname Sparkler is using to communicate with Tableau Server, you see a message like the following at the bottom of the status page:

```
Tableau Server Connection: [fail]
Reason: hostname in certificate didn't didn't match
```

Check the `sparkler.tableau.host` environment variable and update it as necessary to match the SSL certificate.

Issue: Incorrect Trusted Ticket Site ID

If you have an incorrect trusted ticket site name, you see an HTTP 302 redirect message at the bottom of the status page:

```
Tableau Server Connection: [fail]
Reason: Invalid trusted ticket response. HTTP response: 302 Found
```

Verify that the `sparkler.tableau.trustedTicketSiteId` environment variable has been set to a valid site ID.

Tip: View the site in Tableau Server and look at the URL.

Issue: Incorrect Trusted Ticket Username

If you have an incorrect trusted ticket username, you see an error that indicates `-1` as the value of the ticket:

```
Tableau Server Connection: [fail]
Reason: Invalid trusted ticket. ticket=-1
```

Verify that the `sparkler.tableau.test.userName` environment variable is set to the name of a valid Tableau Server user.

Issue: Incorrect Trusted Ticket Configuration in Tableau Server

If the host where the Sparkler app is running not correctly listed as a trusted ticket host or gateway in the Tableau Server configuration, you see `-1` as the value of the trusted ticket:

```
Tableau Server Connection: [fail]
Reason: Invalid trusted ticket. ticket=-1
```

Review the recommendations in Appendix G: Debugging Tableau Server Trusted Tickets.

Issue: Incorrectly Configure Trusted Ticket Host

If the trusted ticket host is incorrectly configured, you see an exception like the following:

```
Name or service not known

Tableau Server Connection: [fail]
Reason: tableauserverx.mycomany.com: nodename nor servname provided, or not
known
```

Verify the value of the `sparkler.tableau.host` environment variable to ensure that it is correct.

Salesforce Canvas App Previewer

When the Salesforce page is loaded, the Salesforce Canvas App Previewer sends a request to Sparkler using the consumer secret and the shared encoding key. Sparkler decodes this request and logs the results.

If Sparkler and Salesforce are correctly configured, you see log entries like the following. (In some cases, entries have been truncated to save space.)

```
2015-02-17 11:47:03,086 DEBUG (http-bio-8443-exec-32)
[org.apache.http.client.protocol.RequestAddCookies.process:122] - CookieSpec
selected: best-match
2015-02-17 11:47:03,087 DEBUG (http-bio-8443-exec-32)
[org.apache.http.client.protocol.RequestAuthCache.process:75] - Auth cache not
set in the context
2015-02-17 11:47:03,087 DEBUG (http-bio-8443-exec-32)
[org.apache.http.impl.conn.PoolingHttpClientConnectionManager.requestConnection
:215] - Connection request: [route: {s}-
>https://sbseabizx1001.tableausandbox.com:443][total kept alive: 1; route
allocated: 1 of 2; total allocated: 1 of 20]
...
2015-02-17 11:47:03,162 DEBUG (http-bio-8443-exec-32)
[org.apache.http.wire.wire:86] - http-outgoing-1 << "r0bi4Wbyp1M66CN7k4tUXFYN"
2015-02-17 11:47:03,162 DEBUG (http-bio-8443-exec-32)
[org.apache.http.impl.conn.PoolingHttpClientConnectionManager.releaseConnection
:276] - Connection [id: 1][route: {s}-
>https://sbseabizx1001.tableausandbox.com:443] can be kept alive for 5.0
seconds
2015-02-17 11:47:03,162 DEBUG (http-bio-8443-exec-32)
[org.apache.http.impl.conn.PoolingHttpClientConnectionManager.releaseConnection
:282] - Connection released: [id: 1][route: {s}-
>https://sbseabizx1001.tableausandbox.com:443][total kept alive: 1; route
allocated: 1 of 2; total allocated: 1 of 20]
```

Issue: Sparkler Consumer Secret Set Incorrectly

If `sparkler.sfdc.consumerSecret` is incorrectly set, you see the following on the Salesforce page:

```
SPARKLER ERROR: Salesforce.com consumer secrets did not match.
```

Issue: Sparkler Consumer Secret Not Set

If `sparkler.sfdc.consumerSecret` is not set, you see the following on your Salesforce page:

```
SPARKLER ERROR: Salesforce.com consumer secret was not set in Sparkler.
```

Appendix G: Debugging Tableau Server Trusted Tickets

In order for Sparkler to acquire trusted tickets from Tableau Server, Tableau Server must trust the IP addresses for any gateways and for the host used by Sparkler. You configure these as trusted IP addresses as described under Task 5: Set Up Tableau Server Trusted Tickets.

Debugging Trusted Tickets

If Sparkler can communicate with Tableau Server but cannot acquire a trusted ticket, it receives a value of `-1`. This may be because not all IP addresses are trusted.

To see which IP addresses are being seen by Tableau Server during trusted ticket requests, you must read the Apache Access Logs in your Tableau Server install. Apache Access logs are generally located in the following folder:

```
c:\ProgramData\Tableau\Tableau Server\data\tabsvc\logs\httpd\
```

Entries in the log look like the following example:

```
10.1.5.213 - - [06/Jan/2015:19:29:52 -0800] 80 "POST /trusted HTTP/1.1"
"154.99.5.133, 10.1.5.213" 200 24 "58" 46800 VKyoMArlFc0AAAvAeSUAAAGh
```

In the example, the first IP address (`154.99.5.133`) is the address of the host. This value must be listed the `wgserver.trusted_hosts` Tableau Server configuration variable.

The second IP address (`10.1.5.213`) is the address of the gateway. This value must be listed in the `gateway.trusted` Tableau Server configuration variable.

To make these settings, use the `tabadmin set` command, as in the following example:

```
cd c:\Program Files\Tableau\Tableau Server\8.2\bin
tabadmin set gateway.trusted "10.1.5.210, 10.1.5.211, 10.1.5.212, 10.1.5.213"
tabadmin configure
tabadmin restart
```

In this example, the first three addresses in `gateway.trusted` were already set, and the fourth is the new one. (All of the addresses are placeholders; substitute values from your own network.)

Appendix H: Connecting to Tableau Server via SSL Proxy

If Tableau Server does not have SSL enabled, but communication to the server is handled via an SSL proxy, the proper request headers must be set by the proxy to ensure that all Tableau Server links and redirects are generated properly. For information about how to do this, see Cannot Connect to Tableau Server through SSL Proxy in the Tableau Knowledge Base.

Appendix I: Connecting to Tableau Online with SAML

Sparkler can be used to embed Tableau Online views in VisualForce pages without using trusted tickets, provided both Salesforce and Tableau Online are configured to use SAML with the same Identify Provider (IdP). The exact details of this configuration vary depending on your IdP and fall outside the scope of this document. In summary, this is what you need:

1. Tableau Online must be configured for SAML. You can access SAML configuration settings by logging in to your Tableau Online site with an administrator account and navigating to **Settings** > **Authentication**. Follow the instructions on the page and your IdP's documentation to configure Tableau Online with your IdP and to enable users for single sign-on.
2. Salesforce must also be configured to provide single sign-on with SAML, using the same IdP. For instructions, see Configuring SAML Settings for Single Sign-On on the Salesforce site.

For more information, see Site Authentication in the Tableau Online documentation.

Once users can authenticate with both Tableau Online and Salesforce using SAML, Sparkler can be used to embed views without using trusted tickets. To use Sparkler, open the `sparkler.xml` configuration file and check the following settings:

- `sparkler.tableau.useTrustedTickets`. Set to `false`. (This is the default.)
- `sparkler.tableau.host`. Set to the host name for your Tableau Online site, such as `online.tableau.com`. Do not preface the host name with `https://`.
- `sparkler.tableau.port`. Set to 443. (Tableau Online runs SSL on port 443.)
- `sparkler.tableau.isSecure`. Set to `true`. (Tableau Online runs SSL.)
- `sparkler.tableau.siteRoot`. Set to the path to your site, such as `/#/site/mysitename`. You can see this path in the URL when you are signed in to Tableau Online.

You can now embed views in VisualForce pages without specifying any server or login parameters in the page itself. When users load a view for the first time in Salesforce, they might be prompted by Tableau Online for their email address. However, they are not prompted for additional login information aside from their initial Salesforce credentials. Subsequent views will not make additional prompts.

Appendix J: Custom Mapping and the "signedIdentity" Parameter

Sparkler requires that a Salesforce identity is mapped to a Tableau Server identity. In the sparkler.xml file, you can set the `sparkler.sfdc.userIdentifierField` parameter to map a Salesforce username, email, or user ID to a Tableau Server user. (For details, see Task 3: Configure Sparkler.)

If these options are not suitable for your scenario, you can specify a custom mapping, which allows you to control which field is used to map to the Tableau user. In addition, it allows you to define which Tableau site to use to gather the visualization from. However, you also take on responsibility for making sure the identity information is signed in order to help prevent identity tampering.

This appendix describes how to configure custom mapping and make sure the value is signed.

There are a few cases where custom mapping *must* be used:

- If you are using a custom field to map to a Tableau user that is not the Salesforce username, email, or user ID.
- A single Tableau user is used regardless of which Salesforce user initiates the request (federation).
- You have multiple sites on your Tableau server that use Sparkler. If you are using multiple Tableau sites, but only one of them is using Sparkler, you set the `sparkler.tableau.trustedTicketSiteId` value to the site that is using sparkler and you do not need to use Custom Mapping.

Overview of the mapping process

The basic approach to creating a custom mapping is that in your Salesforce application, you craft the custom field to map to the Tableau user, sign that parameter with a call to `TableauUtilities::signedIdentity`, and then send the resulting value to Sparkler. Sparkler checks the signature of the custom field before passing it on the Tableau server.

To implement custom mapping you must do the following:

1. Create a Visual Force page to embed the Tableau visualization into. You can extend an existing Salesforce VFP or component, you can create your own page that is a VFP controller extension, or you can create a VFP custom page.
2. Create the mapping of Salesforce identity to the Tableau user. This is done by setting `ts.trustedTicket.signedIdentity` to the value you want to map.
3. Create a Visual Force Apex class as the controller.
4. Create a new Apex class for the Tableau Utilities class. This is used for signing your custom field.

About Signing

It's important to understand that signing protects only certain parameters that are passed between Salesforce and Sparkler. The predefined values for `sparkler.tableau.trustedTicketSiteId` are

automatically signed. Other configuration parameters (for example, `ts.filter`) are not signed, and you should not pass any information using those parameters that could be tampered with. Never use any other parameters to pass information that's used for authentication or authorization.

Task 1: Configure Sparkler to use custom mapping

1. Edit the `sparkler.xml` file.
2. Set `sparkler.sfdc.userIdentifierField` to `signedIdentity`.

Task 2: Create the custom setting for the secret

1. In Quick find, enter type in “custom settings”.
2. Select **New**.
3. Create a custom setting with the following values:
   - **label**: `TableauSparklerConsumerSecret`
   - **visibility**: `protected`
   - **type**: `list`
4. From the `TableauSparklerConsumerSecret` custom setting, add a custom field with the following values:
   - **Field Label**: `SecretValue` (this is the literal value that you enter)
   - **Type**: `Text 20`
5. Add an element into `TableauSparklerConsumerSecret` custom setting with the following values:
   - **Name**: `defaultSecret` (literal value)
   - **SecretValue**: *(Your Canvas App consumer secret)*

Task 3: Create the Apex class that Salesforce uses to sign parameters

1. Create a new Apex class by choosing **App Setup** > **Develop** > **Apex Classes** > **New**.
2. Name the new class `TableauSparklerUtilities`.
3. Using a text editor, open the following file in the Sparkler installation package:

   ```
   Examples\Salesforce\Apex\ TableauSparklerUtilities.apxc
   ```

4. Copy the contents of this file into the newly created `TableauSparklerUtilities` class, completely overwriting the existing text in the file.
5. Save and close the file.

Task 4: Create a test class for "TableauSparklerUtilities"

1. Create another Apex class (**App Setup** > **Develop** > **Apex Classes** > **New**).
2. Name the class `TableauSparklerUtilitiesTest`.
3. Using a text editor, open the following file in the Sparkler installation package:

   ```
   Examples\Salesforce\Apex\ TableauSparklerUtilitiesTest.apex
   ```

4. Copy the contents of this file into the newly created `TableauSparklerUtilitiesTest` class, , completely overwriting the existing text in the file.
5. Save and close the file.

Task 5: Create an Apex controller class

1. In Salesforce, create a new controller class for your Controller Extension. Follow the instructions that start at What are Custom Controllers and Controller Extensions? in the Salesforce documentation.
2. Copy the contents from the following file:

   `\Examples\Salesforce\apex\MyAccountController.apxc`

   The example code shows how to extend the `Account` object. It sets the username to the literal string `standard_user1`.

3. Save and close the file.

Task 6: Update your VisualForce page to use the signed identity

- Update your VisualForce page that has the Tableau embedded view to use a signed identity.

  You can use code from the following example:

  `Examples\Salesforce\Visualforce\Accounts verbose-use-signed-identity.vfp`

Task 7: Verify

- Open your Salesforce VisualForce Page and verify that the Tableau visualization has been loaded correctly.

The "TableauSparklerUtilities::generateSignedIdentity" method

The `generateSignedIdentity` method in the `TableauSparklerUtilities` class is used to generate signed parameters, which are required for custom mapping to work. The method signature is:

```
String generateSignedIdentity(String username, [String Siteid])
```

This method returns a signed identity. If you passed a site ID to the method, that ID is returned as well.

Parameters

Parameter	Type	Description
`username`	String	The username that you want to map to a Tableau user.
`siteId`	String	Specifies the ID of the site that contains the view to embed in Salesforce. This parameter is used only if the view is in a site different from the default site on Tableau Server

Appendix K: Upgrading to v1.02

This section describes what you need to do in order to upgrade your Sparkler installation to v1.02.

Task 1: Update the Sparkler binaries

1. On your computer, navigate to the directory where you saved the Sparkler WAR file and XML files from the update package.
2. Copy the Sparkler files to the Linux or Windows host where Sparkler will be installed. On Windows you may need to use a program such as WinSCP. On Linux, the command will look like this:

   ```
   Myhost:~ username$ scp sparkler-1.0.2.war [USERNAME]@[YOURSERVER]:~/

   Myhost:~ username$ scp sparkler.xml [USERNAME]@[YOURSERVER]:~/
   ```

3. Log in to the Linux or Windows server where you copied the Sparkler files.
4. If Tomcat is running, stop it.
5. Change to the `webapps` directory.

   Windows

   Use the following command:

   ```
   cd c:\Program Files (x86)\Apache Software Foundation\Tomcat 7.0\webapps
   ```

   Linux

   Use the following command:

   ```
   [root home]# cd /usr/share/tomcat7/webapps
   ```

6. Copy the Sparkler WAR file to the `webapps` directory. (The XML file will be used later for configuration.)

   Windows

   Rename `sparkler-reference-1.0.2.war` in your home directory to `sparkler.war`, drag it to `c:\Program Files (x86)\Apache Software Foundation\Tomcat 7.0\webapps`.

   Linux

   Run the following command:

   ```
   [root webapps]# sudo cp /home/ user/sparkler-reference-1.0.0-SNAPSHOT.war
   /sparkler.war
   ```

   You can deploy Sparkler to a different URL by changing the target location name of the file. The name of the file corresponds to the URL path for the application. For example, deploying the file as `sparkler-test.war` will result in URL `https://sparkler-`

`hostname:8443/sparkler-test`. In that case, since the target file name in the example is `sparkler.war`, the directory `/sparkler` will be created.

Take note of the exact file name for the .war file—for example, here, the name is `sparkler.war`. You'll need to use the same filename later when you work with the XML file (with a different filename extension).

7. Restart the Tomcat service.
8. Test Sparkler over HTTPS by opening a browser and navigating to the following URL:

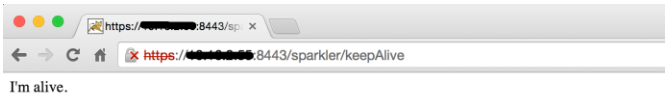
   `https://sparkler-hostname:8443/`

   Note: That the URL uses `https` and port 8443 and is case sensitive.

   For example:

   `https://ec2-hostname.region.compute.amazonaws.com:8443/sparkler/keepAlive`

   If Sparkler is configured correctly, you see a page like the following one:



Task 2: Configure Sparkler

1. Open the `sparkler.xml` file from the configuration directory of your Tomcat installation on the Sparkler server.
2. Add the following settings to the `sparkler.xml` file. For details, see Task 3: Configure Sparkler.
   - `sparkler.sfdc.userIdentifierField`
   - `sparkler.sfdc.emailDomainsAllowed`
   - `sparkler.sfdc.signedIdentity`

Appendix L: Upgrading to v1.03

If you are upgrading from Sparkler v1.0 to v1.03, follow the directions in Appendix K: Upgrading to v1.02.

If you already have v1.02 installed, follow the directions for Task 1 ("Update the Sparkler binaries") in Appendix K: Upgrading to v1.02. You do not need to perform Task 2 ("Configure Sparkler").

Appendix M: Upgrading to v1.04

If you are upgrading from Sparkler v1.00 to v1.04, follow the directions in Appendix K: Upgrading to v1.02.

If you already have v1.02 or v1.03 installed, follow the directions for Task 1 ("Update the Sparkler binaries") in Appendix K: Upgrading to v1.02. You do not need to perform Task 2 ("Configure Sparkler").

Appendix N: Additional Information

Tableau Software Online Help

- Tableau Desktop documentation
  http://onlinehelp.tableausoftware.com/current/pro/online/en-us/help.htm
- Tableau Server documentation:
  http://onlinehelp.tableausoftware.com/current/server/en-us/help.htm
- Publishing Workbooks to Tableau Server:
  http://onlinehelp.tableau.com/current/pro/online/en-us/publish_workbooks_howto.html
- Obtaining an SSL Certificate for Tableau Server
  http://kb.tableau.com/articles/knowledgebase/creating-ssl-certificate-and-key-tableau-server
- Tableau URL Actions
  http://onlinehelp.tableau.com/current/pro/online/mac/en-us/actions_url.html
- Tableau Server Trusted Authentication
  http://onlinehelp.tableau.com/current/server/en-us/trusted_auth.htm
- Tableau Server with Proxy Servers
  http://onlinehelp.tableau.com/current/server/en-us/proxy.htm
- Building Parameterized Filters
  http://onlinehelp.tableau.com/current/pro/online/mac/en-us/parameters_filters.html

Other Tableau Resources

- Tableau Salesforce Connector
  http://www.tableau.com/about/blog/2013/3/new-tableau-8-good-selling-salesforce-and-tableau-21507
- Sparkler/Tableau Server/Salesforce Canvas Solution
  http://community.tableau.com/thread/141178
- Using Parameterized Filters in Embedded Vizzes
  http://kb.tableau.com/articles/knowledgebase/view-filters-url

Salesforce.com Documentation

- Canvas framework document:
  http://www.salesforce.com/us/developer/docs/platform_connectpre/Canvas_framework.

pdf

- Salesforce Canvas App Visual Force Reference
  http://www.salesforce.com/us/developer/docs/platform_connect/Content/canvas_app_vf_component_ref.htm

- Salesforce VisualForce Global Variables
  http://www.salesforce.com/docs/developer/pages/Content/pages_variables_global.htm

OpenSSL Commands

- The Most Common OpenSSL Commands
  https://www.sslshopper.com/article-most-common-openssl-commands.html