Teleprint

Java Application Reference Manual

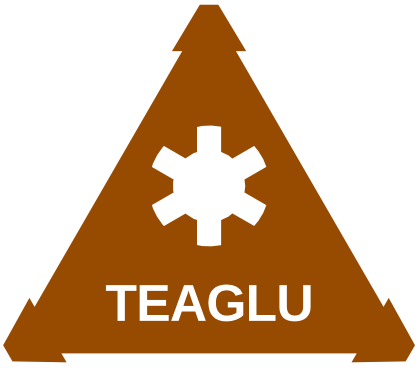

Table of Contents

Introduction....3
Scope....3
Target Audience....3
Installing the Application....4
Application Installation....4
Installation as a Container....4
Installation as a Web Archive....4
Application Configuration....5
Static Configuration....5
Dynamic Configuration....5
Reverse Proxy Configuration....5
Example HAProxy Configuration....6
Configuration Principles....7
Junctions....7
Listeners....7
Authenticators....7
JSON Web Token....7
API Key....8
Junction Key....8
RADIUS....8
Configuration Reference....9
Root Block....9
Authenticator Block....9
JSON Web Token Authenticator Block....10
API Key Authenticator Block....10
Junction Key Authenticator Block....10
Junction Key Junction Block....10
RADIUS Authenticator Block....11
RADIUS Server Block....11
Junctions Block....11
Listeners Block....11
Framed Block....12
Write Strategy Block....14
Quota Block....15
Quota Entry Block....15
Telemetry Block....16
Example Configuration....17
Integrating the Application....18
Credential Passing....18
Credential Passing Example....19

Introduction

Scope

This document describes the installation, configuration, and operation of the Device Web Connector Java Application.

Target Audience

The target audience of the installation and configuration sections of this document is anyone capable of installation, configuration, and secure operation of an appropriate public-facing server environment. This environment may be either containerized or a self-managed Java servlet container.

The target audience of the integration sections of this document is anyone capable of modification of an existing web application to implement handoff by means of a JSON Web Token.

Installing the Application

Application Installation

The dtserver java application can be installed as a container or as a Java web archive. Running as a container offers improved QA because you are using the same runtime and libraries that the application is tested with, but running as a Java web archive allows you more precise control of your operating environment, as well as the ability to operate on Windows.

Installation as a Container

The container image for the dtserver java web application can be pulled directly from registry.teaglu.com/apps/dtserver with the version number as a tag. Authentication to that registry may be done using the API password found in the Teaglu store under your profile.

The container will expose port 8080 as HTTP – it is expected that a load balancer or reverse proxy will sit in front of the container and handle TLS. Haproxy and AWS Application Load Balancer are commonly used in this role.

Installation as a Web Archive

The web archive (WAR) files can be downloaded from the Teaglu store. Installation as a web archive requires you to manage your own Java servlet container. Efficient and secure operation of a servlet container is beyond the scope of this guide.

If you are using this option to run on Windows,we recommend the Tomcat 9.0 Windows Service Installer from tomcat.apache.org. While this is not part of our release testing, this combination is known to work.

Application Configuration

Configuration can be done in either static or dynamic mode – the mode is determined by which environment variables have been set.

Static Configuration

In static configuration mode, the application uses an environment variable DTSERVER_CONFIG to determine where to read its configuration – this variable should be set to the full path of the JSON configuration file. In an Apache Tomcat installation this can be accomplished by creating a file in the bin directory of the installation named setenv.sh (for Linux/Unix) or setenv.bat (for Windows).

Dynamic Configuration

In dynamic configuration mode, the application uses the environment variables CONFIGURATION and SECRETS to point to a configuration source. The full options available are listed on the reference page of the configuration library at https://github.com/teaglu/configure.

In dymanic configuration mode, if the configuration is changed, listeners and junctions will be adjusted on the fly to match the configuration without restarting. Some options such as the servlet locations cannot be changed without a restart due to limitations in the servlet container model.

Reverse Proxy Configuration

In most production installations, a reverse proxy should be used to terminate the SSL connection and forward connections to the tomcat server. The reverse proxy should include websocket support.

Testing was done using the HAProxy open source software, but any reverse proxy should function as long as it is configured for proper forwarding of websockets and provides the required headers.

The following headers are expected to be populated by the reverse proxy:

Header	**Value**
X-Forwarded-For	The source IP address of the client
X-Forwarded-Host	The hostname which was requested by the client
X-Forwarded-Port	The port number which was requested by the client

The following directives can be used in a frontend section of haproxy.cfg to accomplish this:

```
option forwardfor
http-request add-header X-Forwarded-Host %[req.hdr(Host)]
http-request add-header X-Forwarded-Proto https if { ssl_fc }
http-request add-header X-Forwarded-Port %[dst_port]
```

Example HAProxy Configuration

Below is an example haproxy.cfg file:

```
global
    maxconn 20000
    uid 2
    gid 2

    tune.ssl.default-dh-param 2048

defaults
    mode http
    option forwardfor
    option http-server-close
    timeout connect 5000
    timeout client 5000
    timeout server 5000
    timeout tunnel 2h
    timeout client-fin 5000

resolvers docker
    parse-resolv-conf

frontend https
    bind 0.0.0.0:80
    bind 0.0.0.0:443 ssl crt-list /usr/local/haproxy/etc/crt-list.txt

    acl secure dst_port eq 443

    http-request del-header ^X-Forwarded-For:.*

    http-request add-header X-Forwarded-Host %[req.hdr(Host)]
    http-request add-header X-Forwarded-Proto https if { ssl_fc }
    http-request add-header X-Forwarded-Port %[dst_port]

    redirect scheme https if !{ ssl_fc }

    use_backend dtserver if { ssl_fc_sni dtserver.yourcompany.com }
    default_backend dtserver

backend dtserver
    server all dtserver:8080 resolvers docker
```

Configuration Principles

Junctions

The goal is to connect two things: a printer or other device that might connect from anywhere in the world, and your application that wants to connect to it from somewhere on your network. These two things find each other by both referencing a meeting point named a junction.

In the case of printers, a junction has a one-to-one relationship with a print queue, but it does not provide job queueing. The queueing function is normally provided by a print server or is not relevant to the solution.

Listeners

A listener listens on a TCP port for incoming connections – for example, a physical print server or printer with a built-in network interface usually listens on port 9100.

For each listener you declare, the application will listen to a TCP port that you specify, and when it gets a connection on that port it will either connect it to a specific junction or accept the name of the junction on the connection.

Creating a listener bound to a junction creates a unique port that effectively behaves like a network print server. This allows you to use any existing solution that prints directly to a network printer.

Creating a listener not bound to a junction allows your application code to connect to a single port and specify the junction through the connection. This allows your application to create dynamic junctions and connect to them without individual connections being pre-configured.

The server running this application should be placed behind a firewall that only allows connections from inside your network, otherwise anyone on the internet can connect to the listening ports.

Authenticators

The authenticator module is responsible for making sure the web user is allowed to connect to a given junction. If the hand-off is integrated then the authentication information will be passed without user involvement, or the application can prompt for authentication.

There are several configurable authenticators that can be used. Typically the JSON Web Token authenticator is used for integrated hand-off, and all others are used for non-integrated handoff.

JSON Web Token

Using a JSON Web Token allows the calling application to hand off a key to our application that "vouches for" the user. The calling application builds a data structure saying what the user is allowed to do, and a timestamp when the ticket is valid, then signs it with a shared secret. Our application verifies the secret and timestamp are valid.

API Key

The API Key authenticator requires a single shared password to grant access to any junction. This method obviously leaks the authentication secret to the user, and is typically used in non-integrated hand-offs where a single password is desired.

Junction Key

The junction key authenticator is identical to the API key authenticator, except it requires a different password for each junction.

RADIUS

The RADIUS authenticator interprets the API string as a username:password pair, and authenticates the user using one or more RADIUS servers.

RADIUS integration can be used to integrate logins with existing infrastructure – in Windows AD environments this is usually done using Network Policy Server.

Configuration Reference

The configuration file is a JSON file containing a single JSON object, referred to as the root block. While the following sections contain all available options, it is usually easier to skip to the example section.

Root Block

Key	Type	Description
authenticator	Object	Configuration block which sets up authentication
junctions	Object	Configuration block listing each junction along with configuration parameters for each junction.
listeners	Object	Configuration block listing each listener along with configuration parameters for each listener.
framed	Object	Configuration block enabling the framed integration page
writeStrategy	Object	Configuration block for write-behind strategy
quotas	Object	Configuration block for quotas and limits
telemetry	Object	Configuration block for telemetry

Authenticator Block

The authenticator block is a single object under the main configuration block under the key "authenticator". Within the authenticator block the string key "type" determines the type of authenticator, and remaining keys are dependent on the type of authenticator.

The following are valid authenticator block types:

JSON Web Token Authenticator Block

The preferred production method of integration is using an RFC 7519 JSON Web Token because this method does not leave a re-usable token in posession of the end user.

Key	Type	Description
type	String	Must be the constant "jwt"
secret	String	The shared HMAC secret
requireJunction	Boolean	Whether a junction must be passed as part of the token
forwardSkew	Integer	The number of seconds the issued date of the token can be ahead of the time the token is evaluated. This defaults to 60 seconds to allow for some clock skew between servers.
backwardSkew	Integer	The number of seconds the issued date of the token can be behind the time the token is evaluated. This defaults to 1200 seconds to allow cases where the user does not immediately activate the connection.

API Key Authenticator Block

The API Key integration method uses a fixed API key to pass authentication from your application. This leaves the API key known to the end user, so should not be used in production if a more secure method is available.

Key	Type	Description
type	String	Must be the constant "apiKey"
apiKey	String	The shared API key

Junction Key Authenticator Block

The junction key integration method uses a per-junction fixed API key to pass authentication from your application. This leaves the API key known to the end user, so should not be used in production if a more secure method is available.

Key	Type	Description
type	String	Must be the constant "junctionKey"
junctions	Object	An object listing a junction key junction block for each junction

Junction Key Junction Block

The junction key junction block lists the keys which can be used for a specific junction. It has one key, which is an array of acceptable API keys.

Key	Type	Description

keys	Array	An array of strings listing valid keys

RADIUS Authenticator Block

The junction key junction block lists the keys which can be used for a specific junction. It has one key, which is an array of acceptable API keys.

Key	Type	Description
type	String	Must be the constant "radius"
servers	Array	An array of RADIUS Server Block objects

RADIUS Server Block

The junction key junction block lists the keys which can be used for a specific junction. It has one key, which is an array of acceptable API keys.

Key	Type	Description
host	String	The DNS name or IP address of a RADIUS server
secret	String	The RADIUS secret

Junctions Block

The junctions block is a required object block which appears under the key "junctions" in the root configuration block. Each key within the junction block declares a junction with the name of its key, and contains a block with configuration items for that junction.

The junction block is currently empty, so each junction entry should point to an empty block.

Listeners Block

The listeners block is a required object block which appears under the key "listeners" in the main configuration block. Each key within the listeners block declares a listener with the name of its key, and contains a block with configuration items for that listener.

If the "api" value is set to false, the listener will immediately connect to the listed junction and begin passing data. This will mimic the function of a network print server.

If the "api" value is set to true, the listener will listen for the junction name on the connection before connecting to the junction and passing data. Pass the junction name by writing it to the connection followed by a newline character. This allows the caller to connect to any junction without having to create a separate listener for each junction.

Listener Configuration Block Keys

Key	Type	Description
port	Integer	The TCP port number the listener will listen on
api	Boolean	Whether the port should present an API instead of raw data.
junction	String	The name of the junction that connections should be connected to

Framed Block

The framed block enables and configures the framed link page, which is meant to be embedded in an iframe of another application.

To pass authentication from another application, the iframe should be served from the original application, and a form should create a POST request to the framed endpoint which passes the junction and authentication.

To use the framed endpoint as an iframe without supplying authentication, the calling application should direct the iframe directly to the framed endpoint as a GET request. In this case the application will present a prompt for the junction and application.

Framed Configuration Block Keys

Key	Type	Description
root	Boolean	If this value is true, then the framed page will be presented at the root URL of the application instead of the status page.
title	String	Title to be used for the Bootstrap card header
junctionLabel	String	Label for the junction selector used when the endpoint is accessed via a GET request
authenticationLabel	String	Label for the authentication control used when the endpoint is accessed via a GET request
usernamePassword	Boolean	If TRUE then a separate username and password control will be shown, and their values will be combined with a colon (:) to be used as an authentication string. This matches the expected format of the RADIUS authenticator.
theme	String	An alternate Bootstrap theme to be used.
customCss	String	The full URL of a CSS file to be loaded into the page.
junctionList	Boolean	If TRUE then a list of known junctions will be presented instead of an input for the user to input the desired junction. This only has an effect when the endpoint is accessed via a GET request.
authenticationHidden	Boolean	If TRUE then the authentication field is created as a "password" field to enable masking. This only has an effect when the endpoint is accessed via a GET request.

Write Strategy Block

The write strategy configuration block configures the strategy for writing data received from one source to another. Within the write strategy configuration block the string key "type" determines the type of strategy, and remaining keys are dependent on the type of strategy.

Write Strategies

Strategy	Description
direct	Data is written to websockets by the thread the data was received on. This does not deal with the issue of back-pressure and should normally not be used in production.
dedicated	Each websocket has a dedicated write-behind thread. This guarantees no contention but may run out of threads in heavy use scenarious.
pool	A pool of write-behind threads is maintained, and data to be written is queued for the next available thread to write.

Only the "pool" strategy has additional keys beyond the "type" key.

Pool Write Strategy Configuration Block Keys

Key	Type	Description
poolType	String	The type of thread pool. The only current available value is "fixed".
threads	Integer	The number of threads to create for the thread pool.

For production operation, the recommended write strategy is a fixed thread pool, with the number of threads set to approximately 4 times the number of available CPU cores. For example, with a VM with 4 available cores, the number of threads should be set to 16.

Quota Block

The quota block configures soft and hard limits for resources categories. Currently the only resource category with a quota is "control sessions".

Key	Type	Description
controlSession	Object	Configures the soft and hard limits on control sessions. Each control session corresponds to one end user connecting a resource.

Quota Entry Block

The quota entry block specifies the soft and hard limits for a resource.

Key	Type	Description
softLimit	Integer	Configures the soft limit. When the soft limit on the specified resource is reached, a warning is issued in the log file.
hardLimit	Integer	Configures the hard limit. When the hard limit on the specified resource is reached, no more resources of that type can be allocated.

Telemetry Block

Once per day the server calls back to our server and reports version information and runtime statistics. The telemetry block allows you to link the install to the email you use to purchase the product, or disable telemetry entirely.

Telemetry Block Keys

Key	Type	Description
enabled	Boolean	Enables or disables telemetry. Set this value to false to disable telemetry.
email	String	Specifies an email address to send with telemetry. This is used to associate the purchasing account.

The telemetry message does not contain any personal information – below is an example of a message that is sent by the telemetry system.

The telemetry call is sent at a randomized time between midnight and 5AM in Eastern Standard time.

Example Telemetry Message

```
{
	"application": {
		"code": "devtow-war",
		"version": "0.9.2",
		"build": "210708A"
	},
	"jvm": {
		"memory": 504,
		"osName": "Windows 10",
		"version": "16.0.1+9-24",
		"processors": 16
	},
	"controlSessions": {
		"maxAllocation": 7,
		"totalAllocation": 25
	}
}
```

Example Configuration

```
{
      "framed": {
            "root": true
      },
      "authenticator": {
            "type": "jwt",
            "requireJunction": true,
            "secret": "password1",
            "forwardSkew": 60,
            "backwardSkew": 1200
      },
      "junctions": {
            "label1": {}
      },
      "listeners": {
            "home": {
                  "port": 9102,
                  "junction": "label1"
            }
      },
      "writeStrategy": {
            "type": "pool",
            "poolType": "fixed",
            "threads": 16
      },
      "quotas": {
            "controlSession": {
                  "softLimit": 512,
                  "hardLimit": 1024
            }
      }
}
```

This establishes a single "junction" named "label1", a TCP listener on port 9102, and sends any print job received on port 9102 to client connected to the junction named "label1".

Integrating the Application

Credential Passing

The dtserver application is the application used for the public demonstration site, and can be used as a simple port redirector and integrated with your application in an iframe.

Your application or site should include an iframe and give the iframe contents which will use a POST call to submit the passed credentials to the java application. The following POST parameters are used:

Parameter	Function
junction	The name of the junction which should be connected to
authorization	The authorization string, which depends on the authenticator selected in the java application configuration.
dynamic	Set to true if the junction listed should be created dynamically, and destroyed when the control connection is complete.

The POST request should be sent to the URL "/framed" within the application context. If the application were installed on dtserver.yourcompany.com as ROOT.war to place it at the root path, the URL would be:

```
https://dtserver.yourcompany.com/framed
```

If the application were installed on dtserver.yourcompany.com as something.war, the URL would be:

```
https://dtserver.yourcompany.com/something/framed
```

The framed context can be placed at the application root by using the "root" key in the framed configuration block.

The following types of authenticators are available in the application:

Type	Authorization string contents
jwt	A JSON web token authorizing the user for a specific junction
junctionKey	A key which allows access to a specific junction. The secret junction key will be visible to the end user, so this does not provide secure authentication.
apiKey	A key which allows access to any junction. The secret key will be visible to the end user, so this does not provide secure authentication.

Credential Passing Example

For example, your application running at app.company.com might create the following JSON Web Token, which authorizes the holder to access the junction "bob_printer":

Header

```
{
      "alg": "HS256"
      "typ": "JWT"
}
```

Payload

```
{
      "iat": 1516239022,
      "junction": "bob_printer"
}
```

Using the shared secret of "password1" and the HS256 signature method, this creates the following RFC7519 web token:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE1MTYyMzkwMjIsImp1bmN
0aW9uIjoiYm9iX3ByaW50ZXIifQ.SASA8OzEK5k_s16XHLYkFd7I2IPeyJIcz76CRieG
LD4
```

In your application, create an iframe which immediately submits a form to the java application as shown in the following example:

```
<html>
      <body onload="window.forms[0].submit();">
            <form method="https://print.company.com/framed">
                  <input type="hidden" name="junction" value="bob_printer"/>
                  <input type="hidden" name="authentication"
value="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE1MTYyMzkwMjIsImp1bmN0aW9uIj
oiYm9iX3ByaW50ZXIifQ.SASA8OzEK5k_s16XHLYkFd7I2IPeyJIcz76CRieGLD4"/>
            </form>
      </body>
</html>
```