# AmpJS Web Client

## Introduction

The AmpJS web client is a Coffeescript-based implementation of the Common Messaging Framework (CMF) protocol on top of the Streaming Text Oriented Messaging Protocol (STOMP). The client allows web browsers to connect directly to AMQP brokers without the need “proxy” connections through a Java or .NET bridge. The client is distributed as a collection of RequireJS modules and manages its internal dependencies through Twitter Bower. The readme file included with the project details how to install and integrate the client into a project.

## Architecture

The AmpJS client was written to closely resemble its Java counterpart. Consequently developers familiar with the class hierarchy of the Java client should recognize many of the same conventions and patterns used in the Javascript client. Detailed breakdowns of the client’s class hierarchy are included below in the Model Diagrams section. The client is built to be highly pluggable, allowing developers to swap out any of the built-in functionality with more customized implementations as necessary. A consequence of this highly configurable architecture is that a significant amount of “scaffolding” code is required even for common use cases. To simplify common usage of the AmpJS client a highly configurable factory called “ShortBus” has been provided. This factory allows developers to instantiate AmpJS with a single method call and change configuration properties as necessary.

## Messaging Overview

The AmpJS client is a single component with the AMP messaging architecture; therefore to fully understand the client it is necessary to understand how it interacts with other services within the system. Within AMP there are three major supporting services: the Global Topology Service (GTS), Anubis, and the AMQP Broker. As illustrated below in Figure 1 these services work together to both authenticate users and ensure that messages sent by the user are delivered to the correct recipient.

To send a message a user must first contact GTS and retrieve the routing information for any topics to which it will be publishing. GTS verifies the user’s identity transparently through SSL-based mutual authentication (X.509) and then takes care of any AMQP “plumbing” that may be necessary (creating queues and binding queues to the appropriate exchanges) and then returns the requested information to the client. Next the client contacts the Anubis service to request a temporary authentication token. Anubis independently verifies the user’s identity through X.509 certificates and then replies with a unique token that may be used by the client to authenticate itself with other services.

When the client initiates a connection with the AMQP broker it will pass the authentication token received from Anubis as part of its request. The broker will validate with Anubis that the username and token are valid and then permit the user access to the broker. Once the client has been granted access it is able to publish messages through the appropriate topics to back end services.

## Macintosh HD:Users:drew:Desktop:ampjs.png

Figure 1 Messaging Architecture

## Model Diagrams

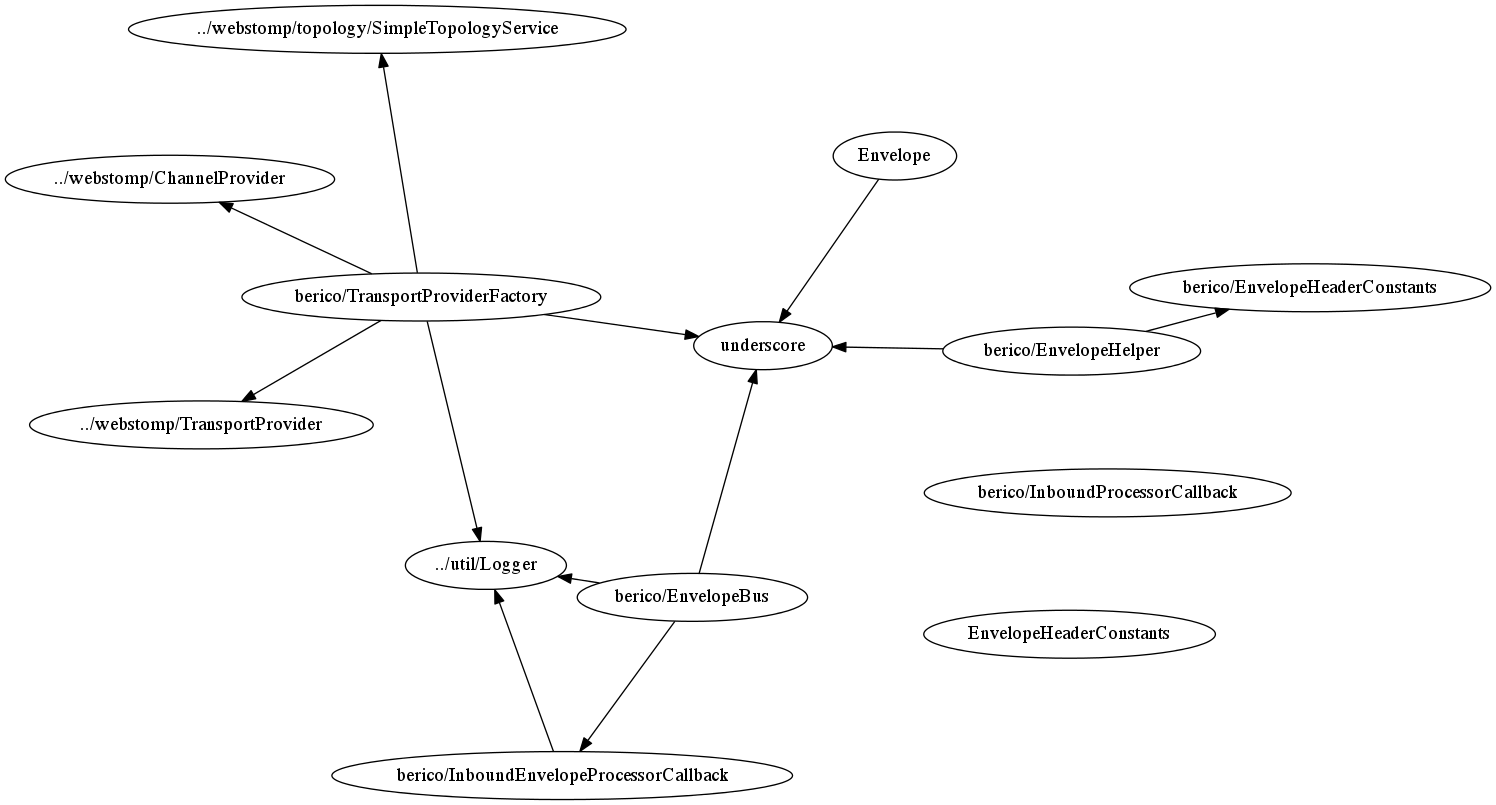


Figure Bus Package

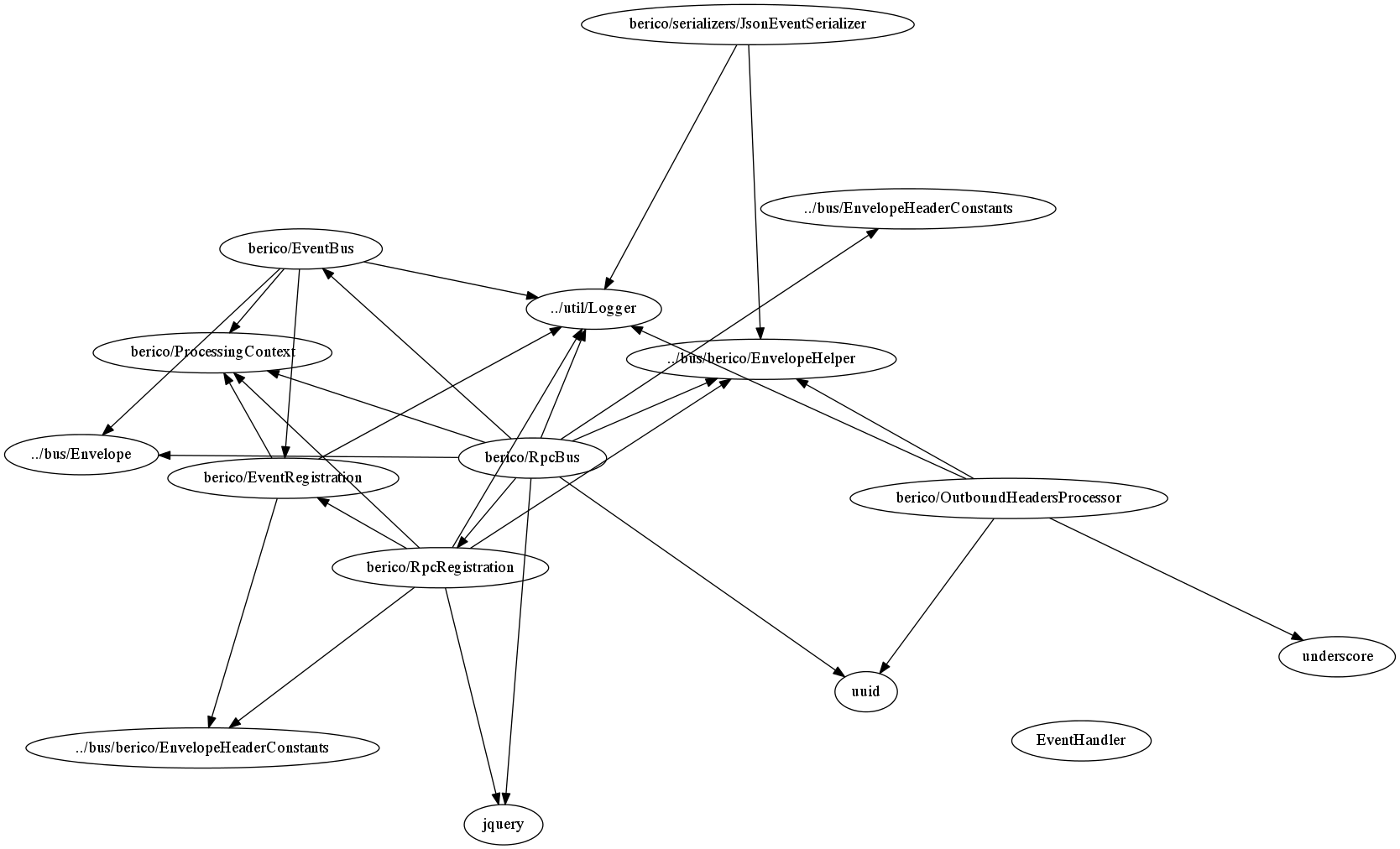


Figure Eventing Package

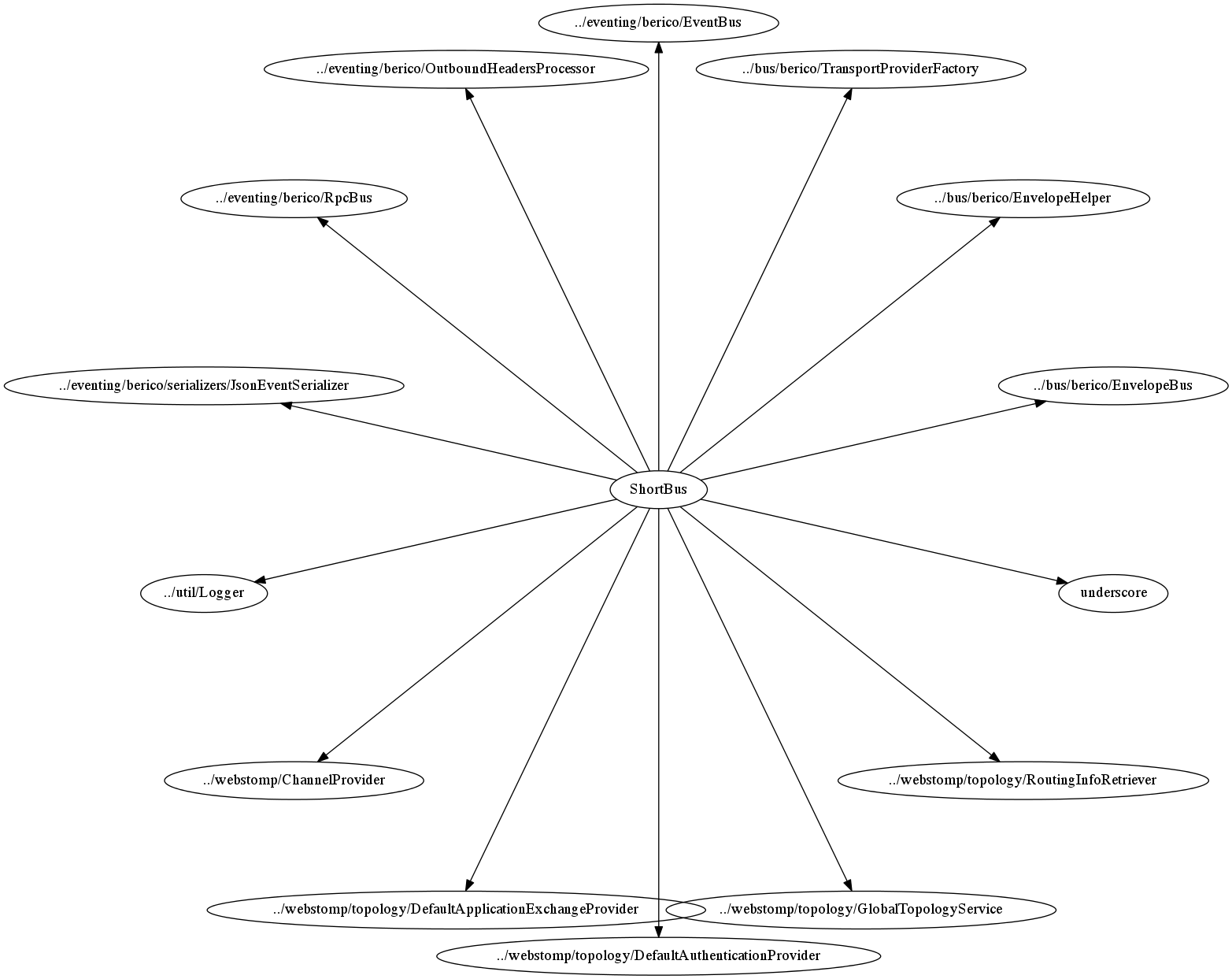


Figure Factory Package

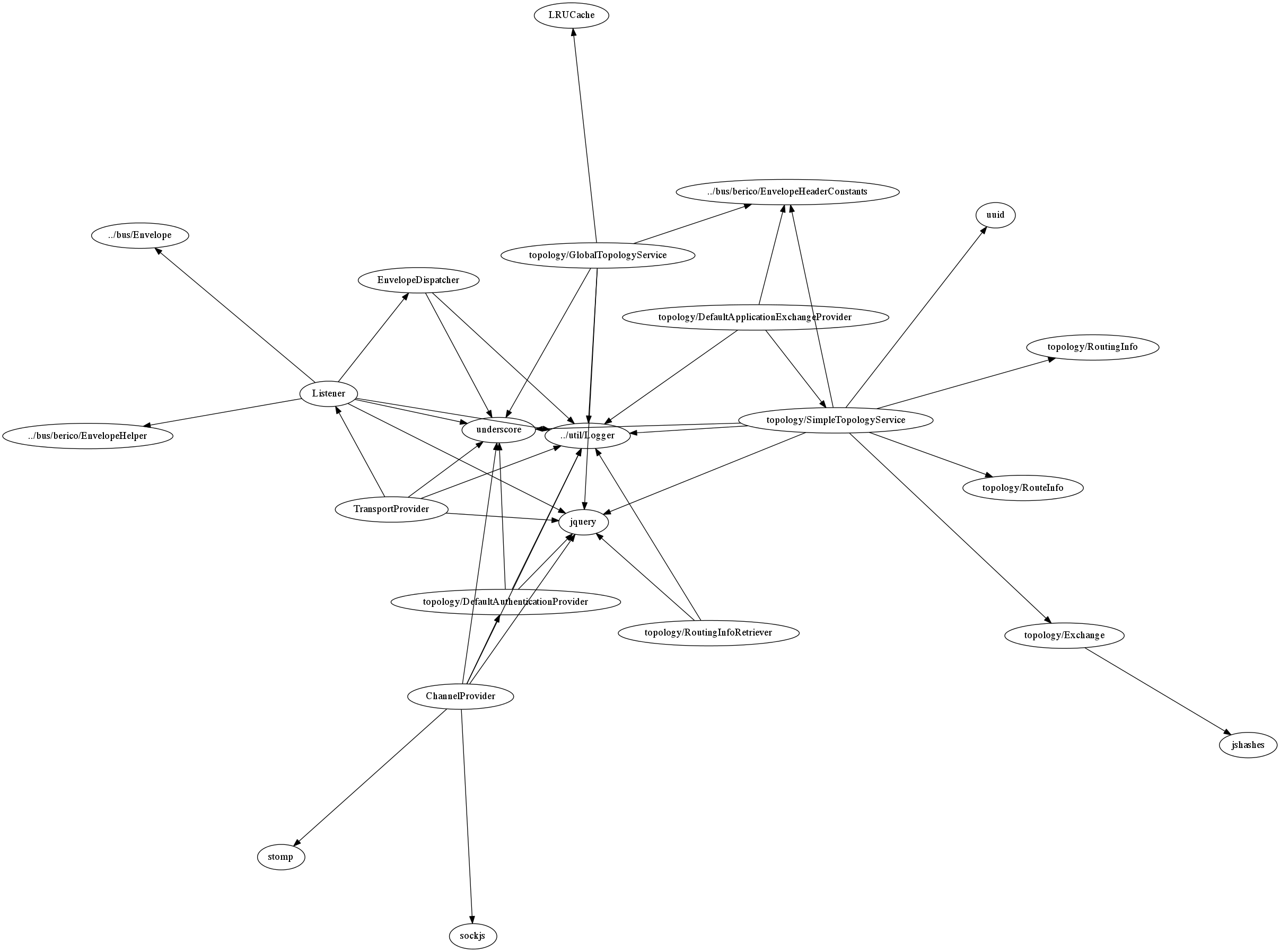


Figure Webstomp Package