unblu 4.2

unblu 4.2 Collaboration Server Concise Setup Guide

Document Version 0.9

www.unblu.com

Phone: + +41 (41) 511 27 11

Email: info@unblu.com

Contents

1. Introduction

Note: This guide applies only to on-premise installations.

This guide will first introduce you to the various unblu features available then it will present the information you need to help you make the best decisions as to which features would add most value to your business model. Instructions on how to deploy those features within your own architecture are either provided here directly or in the form of links to the relevant information.

Note: Although this guide is ostensibly aimed at the technical experts who will deploy unblu we strongly urge your business process specialists to read the first nine sections of this guide. While the second part of this guide takes you through the actual installation, the first part outlines which technical features mean what, in terms of how your business will engage with visitors.

The first part of this guide attempts to put the installation into a context that may help you to envisage how the advantages of unblu's solutions might be maximized within your business model. The middle part of this guide deals with the actual deployment, offering instructions on how to set up each feature. In these sections you may be sent to different pages in our help for further instructions and explanations. The final part of this guide deals with minimal requirements, prerequisites and ports.

Before attempting to deploy anything you must ensure that your system is ready for unblu. This means all of the hardware requirements are met, your infrastructure is primed for deployment (for example, correct ports identified to be opened) and you have at least some preliminary ideas on how the system will work for you.

Each installation is slightly different and the decision as to which components should be installed is entirely at your discretion (bearing in mind that each decision will have an impact on the functionality available).

Some of these design decisions seem entirely technical, in the sense that your IT people would decide, for example, whether to deploy a reverse proxy and unblu Filter in light of their knowledge of your current infrastructure. The proxy and filter will, among other benefits, dynamically inject the unblu snippet only when required, thus reducing the load on the system. If you have session-specific resources or resources that are not accessible from within the agent's location the proxy and filter will also enable in-context browsing, which means your agents, even if they cannot normally access those resources, can join a visitor midway through their visit and have access to the complete context of the visit. For example, items in a basket will persist, as will login information. However, even such a seemingly technical decision has consequences on your business processes. If, for example, you have many proxies, the decision to employ the filter might be more problematic. Or, it may be that the resources required to set up the filter outweigh the assumed benefits of real-time code snippet injection.

Another example might be, for instance, whether to deploy the unblu User Management Synchronization Tool and, if so, how? The Synchronization Tool allows you to synchronize all user and team data so therefore, instead of setting up everything manually from within the Agent Desk, you can simply synchronize your already-existing user information from your own identity management system. However, the way you decide to define the synchronizations can have a serious effect on Agent Desk functionality. Synchronizing users and teams would mean that you can no longer properly manipulate team structures from within the Agent Desk. It is possible though to retain this core functionality by synchronizing 'users only' and leaving the team structures to be modified from within the Agent Desk. For more, see Set Up the User Management Synchronization Tool (Optional).

Most of the decisions that you will take when deploying unblu are pure business decisions. This is not just about load balancing or faster processing of data, it is about how you will shape the interactions between your venture and the visitors to your sites.

So, do you think your visitors might like the option of talking with your representatives face-to-face using our video solution? Might you be better able to steer them to a brighter financial future by perusing documents during sessions? Do you believe some visitors might be more liable to fill-in a loan application or purchase sovereign debt or write options on stock-index futures if one of your specialists is holding their hand every step of the way?

These are indeed loaded questions but they can only be answered by the people who design your business processes. That means that a deployment of unblu must involve technical people who can realize your vision but it is the *vision* that must be applied and technical considerations are merely the enabler of that vision.

Therefore, even before you consider whether our synchronization tool or the unblu Filter would benefit your processes you would want your business analysts and decision makers to work through all the possible consequences of each decision and how it will define how you engage with visitors and potential leads.

While many of the unblu features are optional, in truth they are optional only in the sense that you can still provide a useful visitor-engagement strategy without them. But in reality, with every feature deployed the visitor can be brought closer, the engagement sessions made deeper and more significant. Scaling features such as document sharing and video/audio across all visitor interaction will bring your expertise right into living rooms and train carriages and public parks on the PCs and phones and tablets of people hard pressed for time and perhaps more willing to engage on 'their own terms'.

In short, 'optional' means only that the software is highly-configurable. We would recommend you deploy all of the features that enhance the efficiency of your processes. If you find, after examining the evidence, that your visitors simply do not seem to want to see any of your documents, or that they eschew video and audio sessions then you might want to do without them. But until that time you can assume when we say 'optional' we mean the possibility exists to switch them off and on, not that they are simply an added extra.

You should also note that this concise server setup guide is only the first step to creating your optimal visitor-engagement strategy. After deploying the unblu Collaboration Server using this guide you will need to set up the database and the unblu agent desk. Finally, you will want to tweak the design using the vast configuration possibilities offered by unblu. Only then can you build the structures, teams, roles and session-forwarding strategies that are most suited to your goals and ambitions.

After setting up the server (that is, after completing the setup described herein) see https://www.unblu.com/en/doc/latest/database-setup when you want to set up the database.

When the database is in place see https://www.unblu.com/en/doc/latest/agent-desk-setup-and-user-guide to set up the agent desk and train your agents.

It may even make sense to look at the Agent Desk Setup Guide *before* you setup the collaboration server. In this way your ideas can 'propagate backwards' as you see what exactly it is that this setup is designed to enable and how it can work for you.

When you have a full default deployment working you can find more on configuration at https://www.unblu.com/en/doc/latest/unblu-server-configuration and

https://www.unblu.com/en/doc/latest/unblu-server-configuration-reference

Deploying on the unblu servlet container is a requirement for all on-premise installations. The other parts of the installation described herein are optional and depend on your current system setup and what unblu features you wish to deploy. For example, unblu can be deployed with a reverse proxy and the unblu filter, or with a proxy but without the unblu filter, or even without a proxy. These choices depend entirely on your needs. See The unblu Filter for a little more detail on the security and efficiency advantages of using a proxy and/or the unblu filter.

Another example might be whether to install the User Management Synchronization Tool or not. If, for example, you are a large organization and already have a centralized identity management system that manages staff information then it makes sense to employ the synchronization tool in order that your staff can transparently and securely login to the unblu system. If, however, you are a smaller organization with no in-place identity management solution you would not need the tool.

If you want to share documents with your visitors during co-browsing sessions or if you'd like to enable universal co-browsing you would need to employ the unblu Rendering Service. See Enable Universal and Document Co-browsing (Optional) for more.

If you want to enable video and audio chat as an option see Video / Audio Requirements

Note: See Defining the Appropriate Architecture for some pictorial examples of the architectural differences between typical deployments.

Note: The minimal hardware and software requirements are here: Minimal Requirements

With the advent of unblu version 4.2 our new Agent Desk offers a completely new way to engage with and manage your visitor co-browsing sessions. If you want to add user-level session statistics, canned responses and user/team management then you must install the unblu database. See Persistence Layer / Database for more information.

2. Persistence Layer / Database

Version 4.2. introduces new features such as user-level session statistics, canned responses and user/team management and session forwarding, which all require the presence of a persistence layer.

This is why unblu 4.2 now introduces (persistent) database support for the on-premise version of the product. While it is possible to continue to run unblu without a database (unblu can run with an in-memory, embedded database in cases where no persistence layer is deployed but this is not officially supported), the system will 'forget' everything if the unblu Collaboration Server is restarted.

The new persistence layer has increased the power of the unblu product by an order of magnitude. It is now possible to design your entire visitor-engagement strategy around the unblu Agent Desk.

Note: For a complete description of how to set up the database see:
https://www.unblu.com/en/doc/latest/database-setup

Note: For complete description of how to set up the agent desk see:
https://www.unblu.com/en/doc/latest/agent-desk-setup-and-user-guide

3. The unblu Filter

If your web site uses only static resources then you do not need the unblu Filter. Even if your site is login-protected, as long as there is no need (during a visitor-engagement session) to access either dynamic resources or protected resources then you do not need the unblu Filter.

If, however, you have any resources at all that are protected or dynamically generated then you will need the unblu Filter. There are associated advantages to using the unblu Filter but the technical requirement rests solely on the nature of the resources that will be accessed during visitor-engagement sessions.

Note: The filter is not required if you are sure you will never need to view session-specific data/images during a co-browsing session. This seems simple enough but it can be tricky to discern whether you will need the filter for session-specific resources. Make sure you are aware of what is 'session-specific' and what is not. In general, all personal or customer-specific data, especially in graphical form, is session-specific. See Session-Specific Content.

The nature of your use cases will determine whether you will need the unblu Filter or not. For example, if you have a public web site (with no login requirement) and all resources are open and available to all visitors then the unblu Filter is not required. This is the simplest case; with no protected resources nor any dynamically generated content to worry about the Agent can see what the visitor sees. However, it should be borne in mind that such a use case strongly implies a site with a lot of traffic. So, while the filter would not be required to perform the technical machinations that enable the Agent to see potentially sensitive information or access protected resources, the bandwidth efficiencies might make the unblu Filter an attractive option (as the unblu Filter would only inject the code snippet when a co-browsing session is started).

Note: Even if your site is login-protected, as long as you are sure that all of the resources across all visitor-engagement activity are static then you would not need the unblu Filter.

As use cases become more complex we must assume that there will be personal or sensitive data that will be shared during a session. For example, loans, mortgages, credit cards; everything that constitutes retail banking, can only be handled with static resources up to a point. That is; one might be able to engage with visitors, help them, direct them to offers or deals or other information that builds relationships, but when you need to actually do some business with visitors then you may need the unblu Filter.

For example, as we move up the 'engagement hierarchy' to investment banking or trade finance or mergers and acquisitions the complex engagement requirements themselves define your technical requirements. There are many complex activities that can still be performed without the unblu Filter. For example, your analysts can share charts or offer market insight to investors or your economists can outline their recommendations and strategies using static resources. Essentially, if you want to use unblu to do marketing and general relationship management, using static resources, you would not need the unblu Filter. You will need the unblu Filter if you intend to make transactions such as completing contracts or selling services; any engagement that requires your agents to share personal or sensitive information that must be protected will require the filter.

Using the unblu Filter can massively lower your web infrastructure requirements by dynamically injecting the unblu snippet only when you need it. Co-browsing will account for only a fraction of your website traffic so, if you have a lot of traffic, it makes sense if only the required unblu JavaScript files are loaded at the right time.

The decision on when to inject what is rule based. The rules are the main configuration element of the filter. The unblu Collaboration Server provides a JSON-based configuration, including rules and injection advice for the filter.

To deploy the filter see: Installing the unblu Filter (Optional)

The core purposes of the unblu Filter are:

1. Resource Histories

2. Dynamic snippet injection
3. Session-Specific Resources
4. Automatic uploads for document co-browsing

Resource Histories

The Resource History is a temporary cache of all data (DOM, CSS, images, etc) generated during an in-context session. The Resource History is the filter-related functionality that ensures that everything seen in the visitor browser can be seen in the agent browser. If you picture the Resource History as more than just a cache but as a replication machine, or resource replicator, that ensures everything (all resources) in the visitor browser can be replicated on the agent browser, securely, then you will better understand whether or not you require the unblu Filter's Resource History functionality (in order that your agents can see what the visitor sees, seamlessly and securely).

One of the main purposes of the filter is to capture the traffic coming from the organization's backend. The Document Object Model (DOM) is captured by the JavaScript in the visitor browser and the filter captures resources such as Cascading Style Sheets (CSS), images and PDFs, and uploads them.

Everything is passed through the unblu Filter to the unblu Collaboration Server then to the Agent Desk browser, thus ensuring security and enhancing compliance.

When a co-browsing session is started and the visitor browser requests images from your organization's backend, the filter is triggered as soon as the response arrives at the proxy. So, while the backend is supplying images to the visitor browser, the filter duplicates that content and uploads it to the unblu Collaboration Server (which then passes it to the agent browser).

Note: Without the presence of the unblu Filter within the filter chain it is impossible to view session-specific data. The absence of the filter will not break the session but any image content generated specifically for that session will simply not appear in the Agent's browser.

Note: The filter may be required if the agent browser is unable to access images or stylesheets from the agent's location.

Note: The default behavior is that the DOM and image data is stored in temporary memory for the duration of a session. It is not stored permanently.

- The Resource History is: A cache of all data generated during an in-context session.
- An in-context session uses a method we call 'Embedded Co-browsing'.
- Embedded co-browsing may require the unblu Filter.

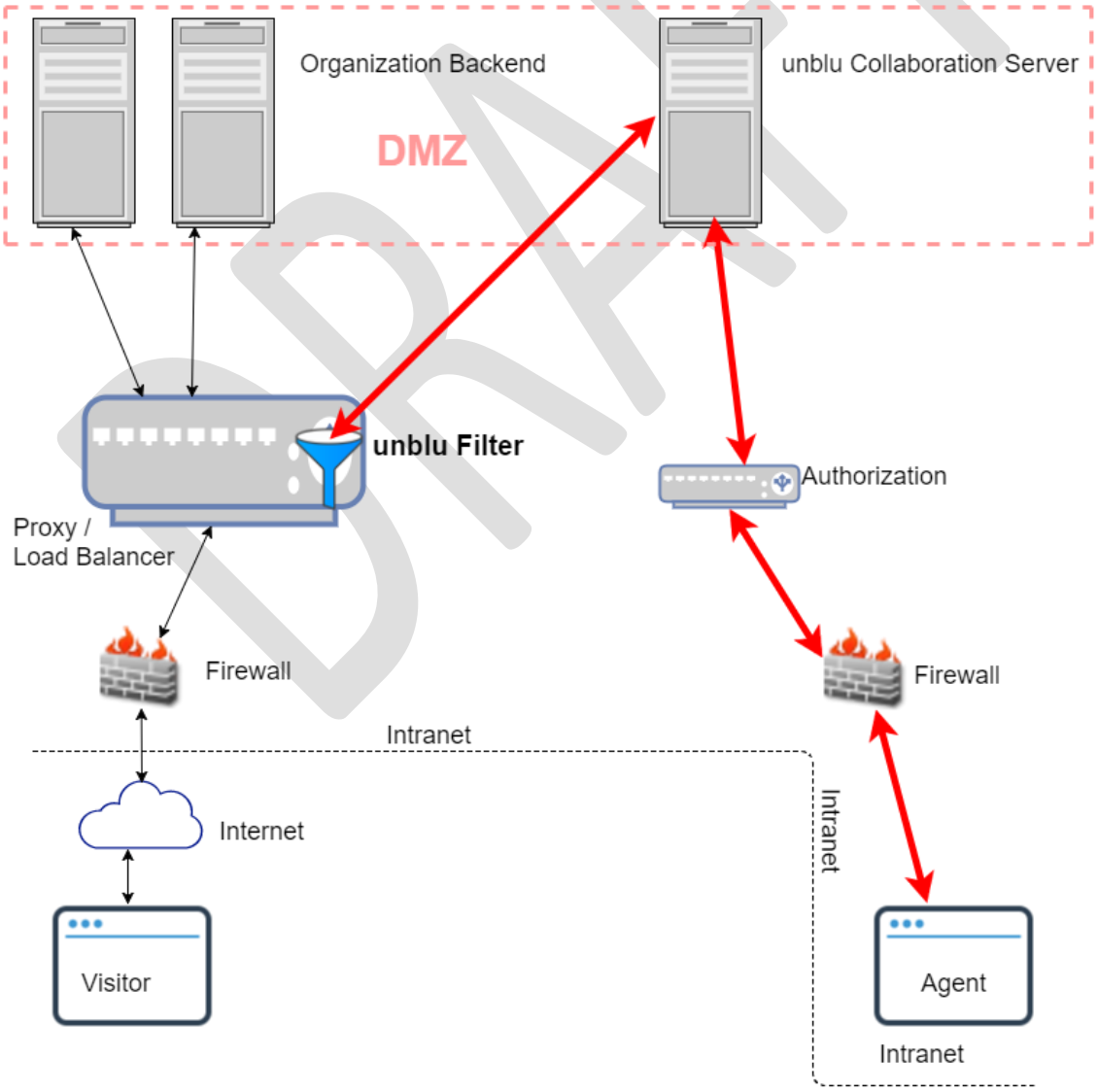
For more on Resource Histories see: Resource Histories Technical Detail

Embedded Co-browsing

Embedded co-browsing captures the visitor browser DOM within a defined domain scope.

Note: Universal Co-browsing uses a completely different method and allows co-browsing of public sites (where the domain does not have to be defined within the unblu system) but does not produce a Resource History. It is normal for organizations to require the capability for both Embedded and Universal Co-browsing. Universal co-browsing requires the addition of the unblu Rendering Service and adds document sharing as well as the ability to co-browse public sites. For more information on Universal Co-browsing see: https://www.unblu.com/en/doc/latest/enable-universal-and-document-co-browsing/

Embedded co-browsing, using the unblu Filter, ensures that the agent browser never has to access the organization backend. All communication with the agent browser is done through the unblu server and the unblu server only 'sees' the objects that have been previously uploaded by the filter. The diagram below provides an overview of the flow. (Red arrows represent unblu flows.)



Note: It is possible to implement the unblu Filter in the backend, without a proxy, but such special cases can be problematic so you should first talk to us if you feel you have good reason to attempt a 'backend implementation'.

When a visitor requests a web page they receive an HTML file. The browser parses the file then converts it into a Document Object Model (DOM). Once the browser has converted the HTML (plus any JavaScript modifications or enhancements) into a DOM, unblu captures that DOM and uploads it (regularly) to the unblu Collaboration Server. This only transfers the text and structure. In order to transfer the images and style sheets you may need the unblu filter. For each image described on the page, and for associated style sheets, it makes another request asking the unblu Collaboration Server for the content. The unblu Collaboration Server then sends the DOM, associated images and style sheets to the Agent browser.

Note: In order to minimize bandwidth and upload times, only changes are sent once a session has started.

The Agent browser **cannot** get the images from the organization backend server directly. Once a session is started, everything (the DOM, images, style sheet(s)) is downloaded from the unblu server.

Note: If you intend to use static resources almost exclusively but you have some resources that you want to be retrieved directly from your own backend server it is possible to configure 'exceptions', whereby normally inaccessible, session-specific resources can be retrieved from the backend server directly using patterns for resource URLs.

When the image element is transferred the unblu Collaboration Server creates an alternative URL for mapping purposes. The unblu server never communicates directly with the backend: all communication is done through the unblu Filter. For every filter-uploaded resource the unblu Collaboration Server generates an unblu-internal URL. The generated URL is the URL that the Agent browser sees. So, when the visitor makes a request the unblu server has a mapping to retrieve the actual image and send it to the Agent Desk browser.

Note: The URL (of the resource/image/CSS etc.) contained in the unblu Filter is sent to the unblu Collaboration server and it MUST match the URL (of the same resource) in the visitor browser. This URL can change according to the exact position of the unblu Filter within the filter chain. If the URLs do not match then the Resource History will not work. See Where to put the Filter for more.

As the actual images and style sheets are not present within the unblu server at the start of the session the filter must be present in the proxy where all the traffic between the visitor and the backend happens.

In-Context Sessions

Embedded co-browsing offers ‘in-context’ sessions. This means that ‘current’ browser information is kept intact. For example, a visitor has already logged into your web site and now wants a co-browsing session. All of the login information and browsing history persists into the session. This means your agents can ‘dive’ seamlessly into not just the web page but the whole context of what the user was doing. For example, shopping baskets will retain the items already there, or perhaps a visitor was struggling to fill-in a form and the session starts with the form half-filled. In this case the agent can see the half-filled form and help the visitor with the rest.

Session-Specific Content

Embedded co-browsing offers a solution to the possible problem of accessing sensitive content from within your intranet. Sometimes it is simply not possible to access content that may be secured or otherwise inaccessible from within a company’s intranet (which is where the Agent Desk resides).

On a technical level ‘session-specific’ can be defined as a protected web site with resources that are dynamically generated within the context of the session.

Unblu cannot identify precisely what is and what is not ‘session specific’ within your particular business domain. In order to ascertain whether or not something is session-specific you should think about the events you foresee taking place within engagement sessions. For example, anything that is unique or personal to a visitor is almost certainly session-specific: Contracts with names or other personal information, account details, charts or analyses prepared for specific customers, anything describing transactions. Ask yourself: Will there ever be a time when you want to conduct transactions using unblu? In truth, it is probably simpler to picture a system that does not require the unblu Filter and ask yourself if this is enough. Remember that unless you are absolutely certain that there will never be a single dynamic resource used during a session, you will need the unblu Filter.

Even if you believe that your resources are not ‘session specific’ it may still be that some content simply cannot be accessed by Agent Desks sitting behind your firewall. A simple test to ascertain whether the unblu Agent Desk can access such content directly from your web servers can be applied by attempting to load, for example, an icon from the page of your, for example, ebanking solution. If you can access the icon (i.e., it loads into the browser) from the internet but not from your intranet then you probably need to employ the unblu Filter in order to ‘bypass’ whatever policies you have in place to protect the server. (The unblu Filter ensures that the unblu Collaboration Server gets all of its content from the unblu Filter in the request queue and never attempts to communicate directly with your servers.)

Note: Even if you have identified that all of the resources that will be available during a session are ‘static resources’ there may be issues ‘hidden’ within deeper technical

processes. The people in your organization closest to those processes will be best placed to make informed judgements as to whether any given resource is static or dynamic.

Dynamic Snippet Injection

The unblu Filter can also be configured to 'decide' when to inject the unblu JavaScript snippet into your web pages. This is called 'Dynamic Injection' as the snippet is injected at runtime according to a rule set that you define. Dynamic Injection makes the process much more efficient as the unblu Filter decides when to inject which unblu JavaScript files and only does this when a co-browsing session has been started.

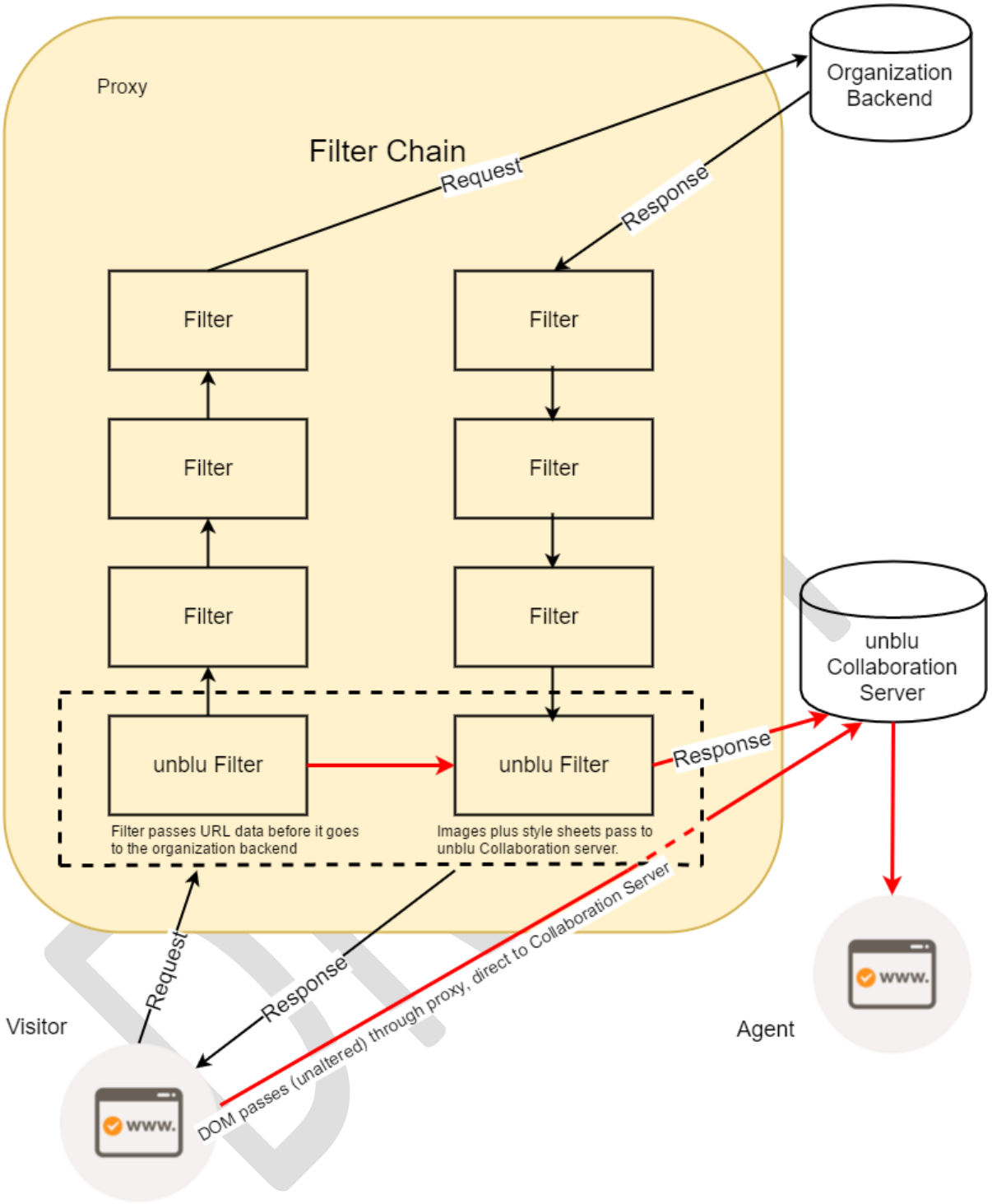
Note: The (computational/bandwidth) efficiency gained using dynamic snippet injection scales according to the scale of the traffic on your web site. If, for example, you have instrumented all of the unblu-relevant pages of your web site *without* the unblu Filter then each time a visitor browser makes a request the unblu snippet will load and execute and thus consume bandwidth and resources (bearing in mind that once the scripts are loaded they are cached and reused). Using dynamic snippet injection, *with* the unblu Filter in place, means communications between your web pages and the unblu server will only happen when 'unblu-specific' requests are made.

Where to put the Filter

The physical/logical placement of the unblu Filter within the proxy's filter chain is crucial. The reason for this is that the unblu Collaboration Server needs to process the exact URLs of the resources that are requested by the visitor browser. Often, within a filter chain, that URL, for whatever reason, can be altered on the way to the backend server.

Ideally, the unblu Filter should be as close to the front of the request queue as possible. Many potential difficulties can be easily resolved by placing the unblu Filter within the filter chain in such a way as it receives the data before any other filter has had a chance to make any changes to that data.

The following diagram shows the flows surrounding the unblu filter.



Note that placing the unblu Filter as close as possible to the visitor browser, within the proxy's filter chain, is a simple rule of thumb that you can use to offset possible problems. We cannot know what any organization has within its filter chain. And it may even be that, within a complex system, it is difficult to be sure of what is being changed where. Therefore, if you know, for sure, that the incoming requests will arrive at the unblu Filter without any modifications of the URL then that will work. But if you have any doubts at all concerning exactly what is happening within your filter chain, then simply placing the unblu Filter at the

appropriate position in the filter chain will offset possible problems arising from the integrity of the request not being perfectly intact.

When a co-browsing session has started, the unblu Filter will pick up the URLs from the request queue and the image and style sheet data from the response queue.

Note: The proxy must be configured such that all '/unblu' (slash unblu) requests go to the collaboration server.

The unblu Filter Flow

1. The visitor browser makes a request.

   In parallel with the unblu activity, the regular processes of the organization's web site take place: Requests are forwarded to the organization backend then responses delivered to the visitor's browser. This (regular) flow is represented in the chart(s) with the black arrows.

2. The request is intercepted by the unblu Filter and, if the rules apply (only images and style sheets), sent to the unblu Collaboration Server.

   The flow of unblu-specific requests is represented in the diagram by red arrows. The unblu Collaboration Server creates a URL 'map' which allows it to find the requested images and style sheets. As images and style sheets are not transferred with the Document Object Model (although the 'space' on the page where they should be placed is recorded) the unblu Collaboration Server uses this 'URL map' to find requested objects. As some objects may be held in the visitor browser cache the unblu server calculates what objects must be fetched to recreate the cached objects on the Agent browser.

3. The reconstructed page is sent to the Agent Desk browser.

Supplementary Information

Below are some links to the information you will need if you decide to deploy the unblu filter.

Filter Usage

For more on filter usage see: http://www.unblu.com/en/doc/latest/filter-usage

For the complete filter specification see: https://www.unblu.com/en/doc/latest/filter-specification

For (example) instructions on how to install the unblu Filter using Apache 2 see: https://www.unblu.com/en/doc/latest/native-apache2-unblu-module-and-unblu-core-filter-library/

Available Filters

Filter implementations for a number of well-known proxies are provided by and can be requested from unblu in binary or source code form (see below). This includes: native filter (C source code package); typically used for OEM integrations with proxy providers.

- apache reverse proxy filter (C source code package). Uses native filter under the hood.
- F5 BIG-IP (Note that integrating with F5 can be complex. If you want to use F5 then, in the first instance, please talk to us about it.)

In addition, the following proxies feature unblu Filter integrations provided by unblu and / or the proxy distributor:

- AdNovum Nevis Proxy
- Ergon Airlock Web Application Firewall

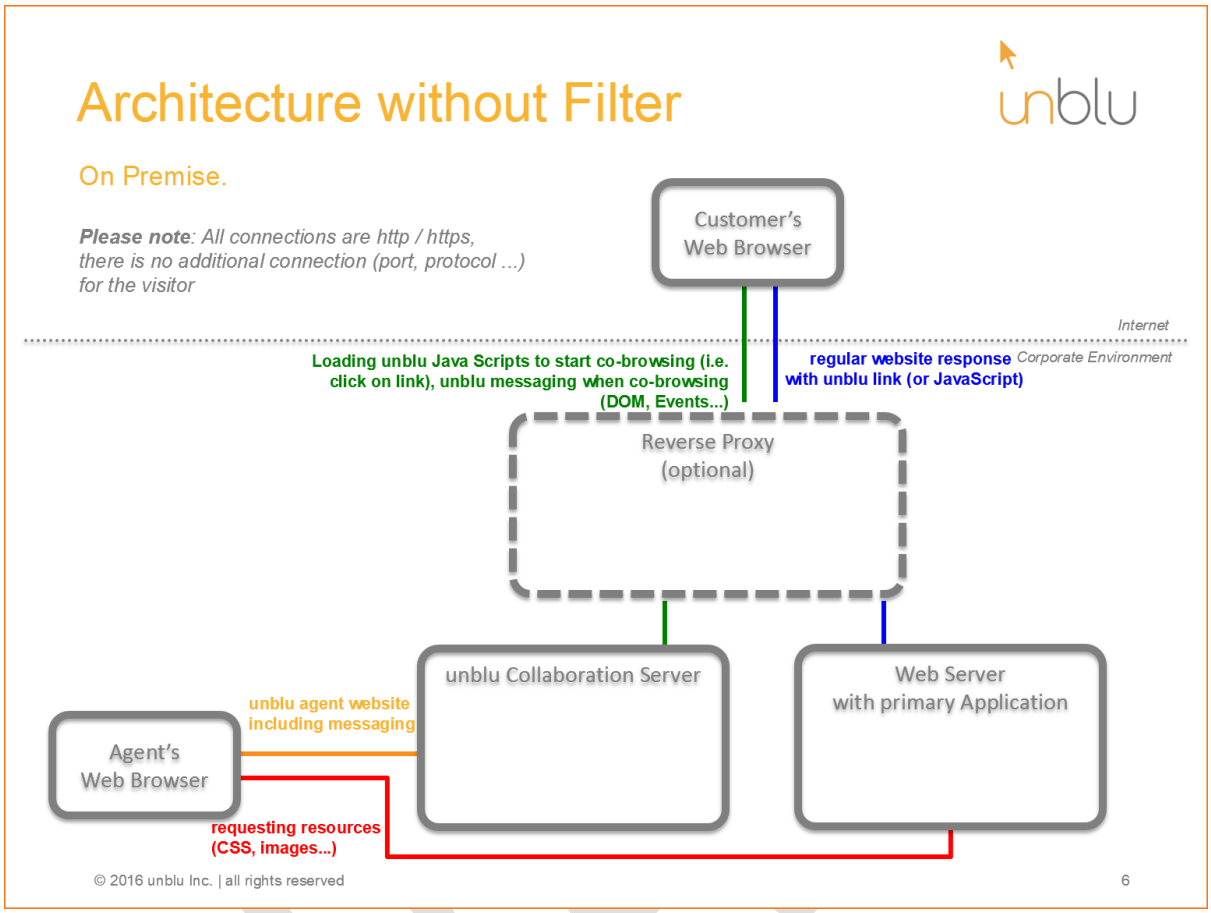
Reverse Proxy Integration

For more on integrating the proxy into your system see:
https://www.unblu.com/en/doc/latest/reverse-proxy-integration

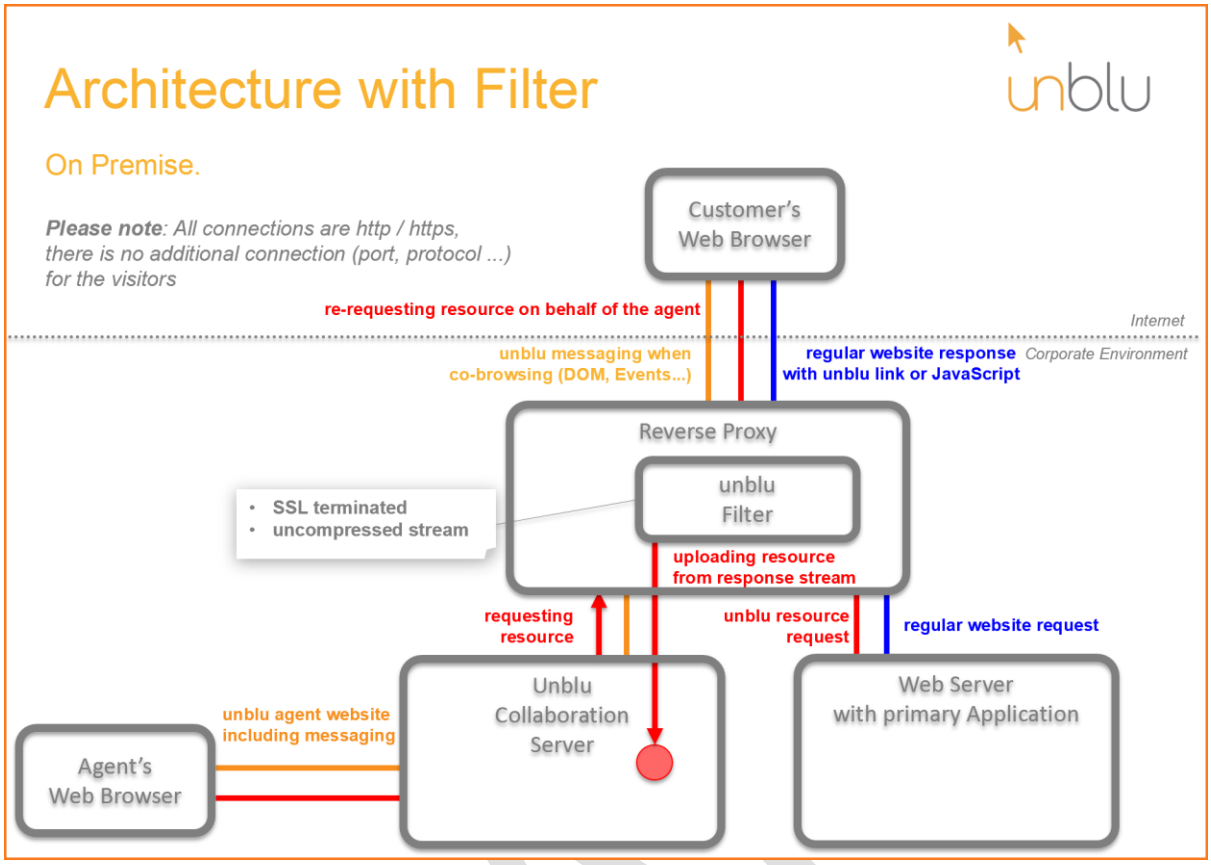
Limitations

For a complete description of all embedded co-browsing limitations see:
https://www.unblu.com/en/doc/latest/embedded-co-browsing-limitations/

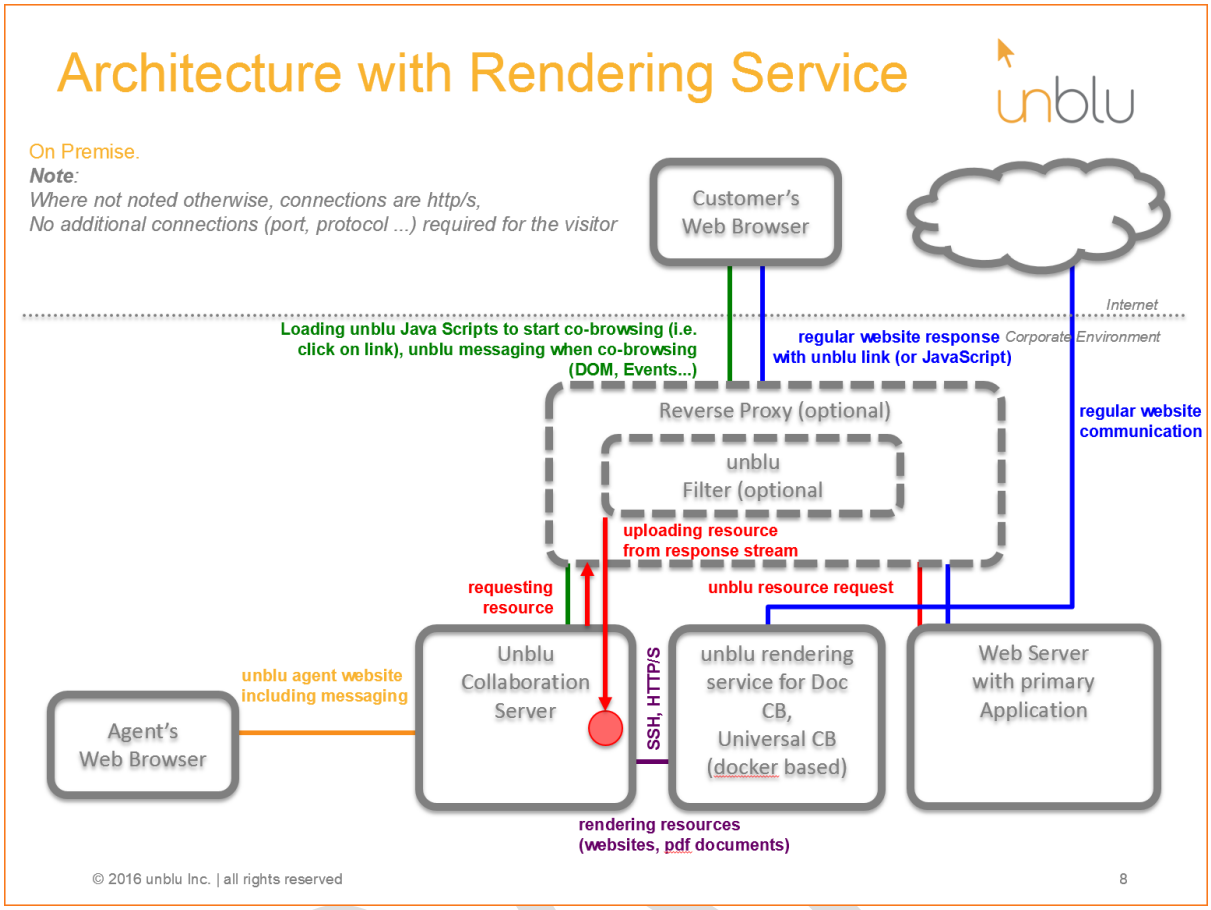
4. Defining the Appropriate Architecture



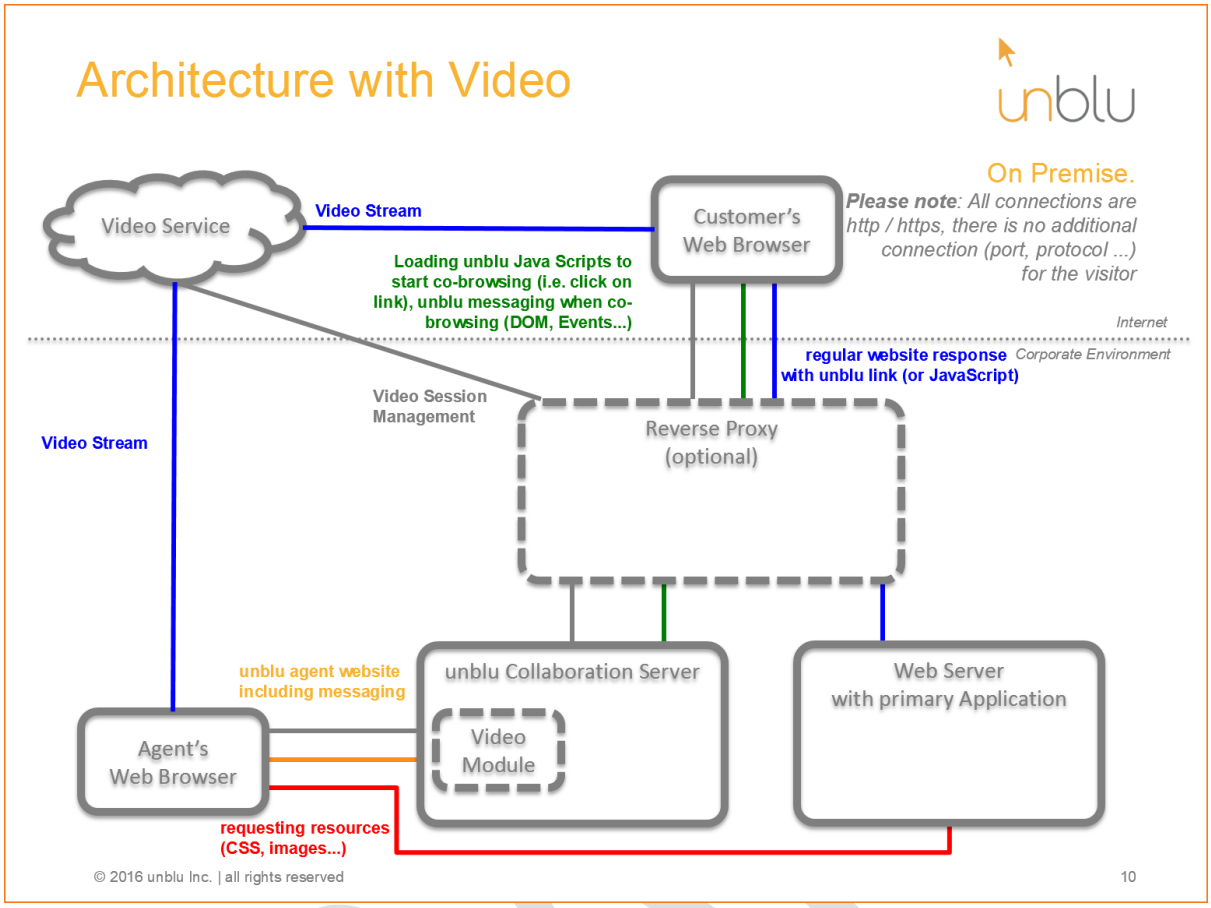
The diagram above shows a minimal installation of unblu in a corporate IT infrastructure without a reverse proxy (and no unblu filter). While this setup would give you secure co-browsing there would be no dynamic injection of the unblu snippet, making your system less efficient than it might otherwise be. Also, without the unblu rendering service you would not be able to define the scope of your co-browsing sessions as easily nor would you be able to share documents with visitors. With no video/audio service defined your agents would not be able to talk to visitors face-to-face.



The picture above shows the architecture with a reverse proxy and the unblu filter. In this case you have reduced network overhead as the snippet only runs when a session is started, thus saving precious resources. This design also provides you with Resource History data and in-context sessions, as well as general advantages regarding security.



The above picture outlines the architecture with the reverse proxy, the unblu filter, and the unblu rendering service all deployed. The Rendering Service enables document sharing and universal co-browsing.



Above is a pictorial representation of the architecture with the tokbox video and audio chat deployed. See Video and Audio Requirements for more on the video/audio chat feature.

5. Reverse Proxy Integration (Optional)

unblu can be used with or without a reverse proxy but there are a number of general benefits to using a reverse proxy such as load balancing and security, and it allows you to remove the need for direct access to your servers.

The main benefit of integrating a reverse proxy, in the case of the unblu product, is that it allows you to use the unblu Filter. The unblu Filter can be used inside the filter chain of a reverse proxy to inject the unblu snippet dynamically and accommodate co-browsing-specific communication automatically.

For maximum security we recommend filter integration due to its ability to capture and secure resources (such as CSS, images, session-specific pdf).

For more on the unblu Filter see: The unblu Filter and Installing the unblu Filter (Optional)

For more on configuring the reverse proxy see:

https://dev.unblu.com/confluence/display/UNBLUTGNEXTEN/Reverse+Proxy+Integration

and Configuring Apache 2 (Example)

6. User Management Synchronization Tool (Optional)

Given the presence of an LDAP (or similar) server, any enterprise that wants to use their current staff information, held in their identity management system, can map that information into the unblu Collaboration Server on a one-to-one basis. This means that, after the initial setup, existing organizational structures can be mapped to the unblu collaboration server. The tool can then be run according to a schedule in order to ensure that current staff data properly populates the unblu database.

Note: Employing the synchronization tool is not a purely 'technical' decision. Business decision makers and process design staff in your organization may want to read Set Up the User Management Synchronization Tool (Optional) in order to define the best strategy for your goals. For example, the way you configure the Synchronization Tool can have a significant impact on how you use the Agent Desk (as different syncing strategies directly affect how you interact with the Agent desk). In short: You can sync 'everything'; which will give your identity management system precedence, and thus the Agent Desk will reflect the team design already held within your system. Or, if you want to retain flexibility vis-à-vis the Agent Desk, then you can tell the sync tool not to synchronize teams. This will allow you to sync everything except teams, which you can then build and manipulate on-the-fly from within the Agent Desk interface.

You may also want to read about 'Single Sign-On' (SSO) in unblu:
https://www.unblu.com/en/doc/latest/unblu-sso-setup

7. Universal and Document Co-browsing (Optional)

Document co-browsing and universal co-browsing require the unblu Rendering Service appliance to be installed. The unblu Rendering Service is based on Docker (https://www.docker.com/). It requires a separate host to run Docker with the rendering service Docker image available at https://hub.docker.com/r/unblu/renderingservice/

Embedded Versus Universal Co-browsing

Embedded co-browsing is where we capture the Document Object Model (DOM), images and style sheets using the unblu Filter and the visitor stays inside their browser while we use JavaScript to replicate the browser on the agent side.

Universal co-browsing is where a headless browser runs on our server inside the rendering service and we render the graphical output to both the agent and visitor.

Note: Context sharing is not possible with universal co-browsing. Context sharing means that visitors do not lose their login or shopping cart information when they start co-browsing.

See Enable Universal and Document Co-browsing for more details surrounding the unblu rendering service installation and instructions on how to configure it.

8. Java Naming and Directory Interface (JNDI) (Optional)

From version 4.2 unblu offers the option to use the Java Naming and Directory Interface (JNDI) when setting up the unblu collaboration server. JNDI is an interface that allows you to bind names to objects (or references to objects) and to look up those objects by name. JNDI allows you to minimize the number of people who need to know the credentials for accessing a production database. Only the Java EE app server needs to know if you use JNDI. See Java Naming and Directory Interface Example for an example set up.

9. Order of Deployment

Before you start you should have Apache 2, apxs and your application server already deployed.

Note: For clarity's sake the examples herein assume an installation with Apache 2 and Tomcat but you may use any web and application server of choice.

To install unblu in one of the configurations outlined in this guide you need to deploy the following components:

- The unblu Collaboration Server which connects the visitor's computer and the agent's computer. See: WJAR Deployment
- An optional filter on the reverse proxy server which identifies co-browsing requests and forwards them to the unblu collaboration server. See: Installing the unblu Filter
- The database. See https://www.unblu.com/en/doc/latest/database-setup

Note: The three items above should be deployed first. The remaining features can be deployed according to the most convenient path for your particular system. Although it may make sense to deploy the unblu Agent Desk as soon as possible after the steps above in order that you can get unblu up and running.

- The optional User Management Synchronization Tool (identity management). See User Management Synchronization Tool (Optional)
- The optional unblu rendering service. See Configure and run the Rendering Service Appliance
- Optional video/audio chat. See Video and Audio Requirements
- Optional Java Naming and Directory Interface (JNDI). Deploy this if you want to more easily apply configuration values. See Java Naming and Directory Interface (JNDI) (Optional)

To complete the full installation in order that your agents can start engaging with visitors you must set up the agent desk: https://www.unblu.com/en/doc/latest/agent-desk-setup-and-user-guide

After these steps are completed you must configure the collaboration server. Note that 'Configuration' is outside of the scope of this concise setup guide. See https://www.unblu.com/en/doc/latest/unblu-server-configuration and https://www.unblu.com/en/doc/latest/unblu-server-configuration-reference

Note: After initial setup and configuration you can change the configuration at any time using configuration files or by setting values 'on-the-fly' through the unblu Agent Desk GUI.

10. WJAR Deployment

The installation is shipped within a single .WAR file. The shipped file is sometimes called a WJAR, because it is a valid .war file but can also be started directly as if it were a jar file (with something like `java -jar unblu.war`).

Note: Prior to the release of version 4.2 we shipped multiple types of products (SSO, staticuser, .war, .ear). Now, just one file is shipped which has all of the other types included.

Note: You should refer to your web server documentation for information on how to install the unblu web application on your web server and, by way of an example, you can find step-by-step (example) instructions on how to install unblu on a Tomcat server here: Installing unblu on a Tomcat Server (Example)

Note: If you intend to use an IBM WebSphere application server you must create an Enterprise Application Archive (EAR). See Running as an Enterprise Application Archive (EAR)

We recommend that you run unblu as the root application in your web server. If your web server allows you to specify virtual servers then you can use a virtual server to run unblu in a server root folder, even if it is not technically the root application.

There are several ways to run the wjar.

Running the WJAR on a Web Server

To start the wjar execute the following command:

```
java -jar product.com.unblu.war
```

This will automatically select

```
product.com.unblu.enterprise.universe.staticusers
```

and run it in a jetty webserver on an automatically selected TCP port in the range from 6060 to 7070. (To manually select the TCP port pass "`-Dport=<port>`" on the command line.)

To manually select a specific product execute the following:


```
java -jar product.com.unblu.war –product <productID>
```

Running as .WAR on a Java EE Application Server

As with the previous releases of unblu, copy the shipped file

```
product.com.unblu.war
```

into your Java Application Server.

If not configured via JNDI or system properties unblu will automatically select

```
product.com.unblu.enterprise.universe.staticusers
```

as the default to start in the Java Application Server.

Note: After you have installed the web application, you should be able to reach the unblu login page at `http:// localhost:<port number on which your server runs>`

Running as an Enterprise Application Archive (EAR)

Note: An EAR is a file with the .ear extension containing both .jar and .war files to be deployed on an application server. EAR files are used to deploy enterprise applications containing Enterprise Java Beans (EJBs), web applications, and 3rd party libraries. It is also a jar file. It has a special directory called APP-INF that contains the application.xml file. Note that APP-INF is WebLogic-specific and does not exist in the Java EE standard.

JNDI configuration is done as specified for your Java Application Server and will then apply to the running unblu product. See Java Naming and Directory Interface Example

Prior to running the desired unblu product as an EAR it is necessary to export either the whole or the selected product as EAR.

To export a specific product as separate wjar execute the following on the command line:

```
java -jar product.com.unblu.war --product <productID> --export wjar > target.war
```

When running the executable, the following command line options are available:

Options	Description
`-d,--diag`	Whether to include diag bundles into product execution/export.
`-e,--export <arg>`	Type of export. Possible options: 'ear' or 'wjar'
`-h,--help`	Display this help message
`-l,--list`	Format of output: <display name> (<product ID> version: <version> build: <build> branch: <branch>)

Options	Description
`-o,--output <arg>`	File to write the exported product to. If not given, output will default to STDOUT.
`-p,--product <arg>`	ID of the product to start. For example, product.com.unblu.platform
`-t,--test`	Whether to include test bundles into product execution/export.

Note: All of these options are also available for the JNDI configuration; without the dash(es). (Although not all of them are meaningful in the context of JNDI.)

11. Installing unblu on a Tomcat Server (Example)

This section describes how to install unblu on an existing Tomcat server. The installation described here aims to provide a straightforward way to ensure that unblu runs on the server. Depending on your server environment and your practices, you may choose to deviate from the steps suggested here.

Note: This example, using a Tomcat server, is designed as an illustrative guide only. unblu can be installed on virtually any web server. You should consult your web server's documentation to get the specific information you need to get up and running with unblu on your web server solution.

To install unblu manually on a Tomcat server:

1. Copy the `.war` file from the unblu distribution package into the folder `webapps` of the Tomcat folder.
2. Start Tomcat. After a few seconds, Tomcat unpacks the file automatically and extracts the files and folders. After Tomcat has finished unpacking, delete the `.war` file in the `webapps` folder.
3. Next, you need to make unblu the root Web application. Because Tomcat does not have configuration options for this, you need to do it manually by changing the folder name of the unblu web application. So, in the file system, go to the `webapps` folder, and rename the web apps as follows:

   Rename the folder `ROOT` to `ROOT-archive`

   Rename the `unblu` webapp folder to `ROOT`
4. Now, unblu runs as the root application in Tomcat. (You may have to restart Tomcat to make the change effective and to make sure that unblu runs properly.)

5. If you use the Tomcat web server for more than a quick test or demo, then remove all the other folders (except `ROOT`) from the `webapps` folder. This ensures that the Tomcat server runs only the unblu web application.

To test if unblu runs correctly:

1. With your browser, go to `http://localhost:8080` (or the address where the Tomcat server runs). If unblu runs correctly, you will see the login screen for the agent view.
2. Login with the username `unblu` and the password `secret`. unblu displays the code screen.

To enable a configuration file for unblu, proceed as follows:

- If you have received an `unblu.properties` file for your installation, put the file into the folder of the Tomcat server, for example into the `conf` folder. Make sure that the file can be read by the user running the Tomcat server.
- If you have not received a file create a new empty text file where you can store the configuration entries.
- To enable the properties file you need to define a Java system property named `com.unblu.propertyoverlay` that contains a link to the configuration file, in the form of `file:///full/path/to/unblu.properties`. In a default Tomcat installation you can do so with the following command:

  ```
  export JAVA_OPTS="$JAVA_OPTS -
  Dcom.unblu.propertyoverlay=file:///full/
  path/to/unblu.properties"
  ```

Note: In a default Tomcat installation you can put the command at the top of the catalina.sh file

```
<tomcatHome>/bin/catalina.sh
```

so that it is always executed when Tomcat starts.

12. Configuring Apache 2 (Example)

The `dist` package of `mod_unblufilter` provides a `config` directory containing a number of example configurations for various platforms.

Note: For a more complete description on how to build the Apache 2 module and unblu Filter see: https://www.unblu.com/en/doc/latest/native-apache2-unblu-module-and-unblu-core-filter-library/

The following example is ‘self-contained’. That is: With this configuration, all Apache 2 configurations required to run unblu via this apache server are included. In particular, this includes configuration of a reverse proxy path for `/unblu` requests, adding `mod_unblufilter` using the `mod_filter` Apache 2 module and the `mod_unblufilter` configuration itself.

Example Configuration

```
############################################################
# Configure reverse proxy to /unblu of unblu server

<IfModule !mod_proxy.c>
LoadModule proxy_module modules/mod_proxy.so
<IfModule !mod_proxy_http.c> LoadModule proxy_http_module
modules/mod_proxy_http.so
</IfModule>
</IfModule>

ProxyRequests Off

<Proxy *>
Order deny,allow
Allow from all
</Proxy>

ProxyPass /unblu/ http://unbluserver/unblu/
ProxyPassReverse /unblu/ http://unbluserver/unblu/
############################################################
# Configure unblu

LoadModule unblufilter_module modules/mod_unblufilter.so UnbluServerUrl
http://unbluserver/
UnbluConfigOrigin local
UnbluConfigDirectory /etc/httpd/unblu/unblu.conf
############################################################
# Advanced Unblu Configuration
# the following settings are defaults - uncomment and adapt them
#UnbluPublicPathPrefix /unblu
#UnbluSystemPathPrefix /sys-unblu
#UnbluServerAdminBackend rest/filterBackend
#UnbluDefaultCharset ISO-8859-1
```

```
#UnbluConfigOrigin local
#UnbluConfigDirectory /etc/httpd/unblu/unblu.conf
#UnbluConfigExpiration 3600
#UnbluMaxResponseSizeResource 100kb
#UnbluMaxResponseSizeInjection 10kb
###############################################################
# Add the filter module to the filter chain

<IfModule !mod_filter.c>
LoadModule filter_module modules/mod_filter.so
</IfModule>

# The following lines require filter_module to be active
# See http://httpd.apache.org/docs/2.1/mod/mod_filter.html
# for details about filters
# FilterProvider Syntax:
# FilterProvider filter-name provider-name  [req|resp|env]=dispatch match
# NOTE: do not change the provider-name (unblufilter) - it is hardcoded
# and connects this configuration to the filter module

FilterProvider unblufilter resp=Content-Type *

# FilterChain Syntax:
# FilterChain [+=-@!]filter-name ...
# NOTE: The @ before the filter-name ensures # that the filter is inserted at
# the first position

FilterChain @unblufilter
###############################################################
```

Configuration Parameter Description

Reverse proxy server

The following settings are generic and usually need no adaptation:

- `IfModule !mod_proxy.c`: loads `mod_proxy` if not loaded already
- `IfModule !mod_proxy_http.c`: loads `mod_proxy_http` if not loaded already
- `ProxyRequests`: Configure this as a reverse proxy (thus set to 'no' to disable forward proxy capabilities).

- `Proxy section`: configure access control to proxy (allow all).

The following two lines require adaptation: `unbluserver` must be replaced with the hostname plus port (if non-standard) of the backend unblu collaboration server.

- `ProxyPass`: configure `/unblu` requests to be proxied.
- `ProxyPassReverse`: Make sure response headers are adapted to have a transparent reverse proxy unblu configuration.

The following settings are required and must be adapted accordingly:

- `LoadModule`: load `mod_unblufilter`
- `UnbluServerUrl`: hostname and path of the backend unblu Collaboration Server or a proxy, where the server can be reached.

The following settings are commented out (in the provided example configuration) because they represent default values:

- `UnbluPublicPathPrefix`: Must have a leading slash and otherwise match the setting of `com.unblu.identifier.publicPathPrefixPattern` in the unblu Collaboration Server (`unblu` by default).
- `UnbluSystemPathPrefix`: Must have a leading slash and otherwise match the setting of `com.unblu.identifier.systemPathPrefixPattern` in the unblu Collaboration Server (`sys-unblu` by default).
- `UnbluServerAdminBackend`: The URL of the admin backend to be used by the Apache 2 unblu filter. Typically fixed to the default `rest/filterBackend` (unlikely to change).
- `UnbluConfigOrigin`: Can be either local or remote (default).

  If local, the configuration file must be located at `UnbluConfigDirectory` (`UnbluConfigDirectory` is the path and filename in this case).

Note: `UnbluConfigExpiration` is ignored when local is selected.

  If remote, the configuration file will be downloaded from the remote Collaboration Server and stored in the directory given by `UnbluConfigDirectory` (which must be writable and readable in this case).

- `UnbluDefaultCharset`: Character set used if none is specified in an http (text) response.

- `UnbluConfigDirectory`: If `UnbluConfigOrigin` is set to remote, this is not required. If `UnbluConfigOrigin` is set to local, this must specify the path and filename of the configuration file.

- `UnbluConfigExpiration`: Time, in seconds, until the downloaded configuration of the unblu Collaboration Server expires and must be re-downloaded. Ignored if `UnbluConfigOrigin` is set to local.

`mod_filter` configuration:

- `FilterProvider: mod_filter` specific command to enable `mod_unblufilter` as a filter in Apache 2

- `FilterChain`: Specifies where in the filter chain unblu is called. unblu must be called before any URL transformations are performed (such as URL encryption or decryption).

13. Java Naming and Directory Interface Example

Up until release 4.2 it was necessary to use local system properties to configure the unblu collaboration server. We would strongly recommend that JNDI is used instead of local system properties.

You can set any allowed unblu property via JNDI. JNDI configuration should be performed as per your Java Application Server and it will then be applied to the running unblu product.

JNDI will work with any application server. When unblu starts it reads the JNDI properties; no matter how they are defined within the applications server. This is done using `com.unblu.propertyoverlay` which then references a configuration file.

You may still use configuration files when using JNDI but you must tell unblu where the configuration files are. This can be done using JNDI properties or system properties.

We cannot offer exact examples as the way JNDI is declared is non-standard but, purely for reference, here is a Tomcat example JNDI configuration for root context path installation:

Stored at:

```
$CATALINA_BASE_DIR/conf/Catalina/localhost/ROOT.xml
<Context>
      <Environment name="product"
value="product.com.unblu.enterprise.universe.sso" type="java.lang.String"
override="false"/>
      <Environment name="com.unblu.propertyoverlay" value"42x-sso.properties"
type="java.lang.String" override="false"/>
      <Environment name="com.unblu.textoverlay" value="42x-sso-
text.properties" type="java.lang.String" override="false"/>
</Context>
```


14. Installing the unblu Filter (Optional)

Now that unblu runs, you can add the unblu Filter to your reverse proxy server so that it forwards the co-browsing requests to the unblu collaboration server.

By way of an example, this section describes how to build the Apache 2 modules `mod_unblufilter` and `libunblufilter`.

Warning: When attempting to run Microsoft Internet Explorer with web servers there are some ('Keep-alive' timeout) issues to bear in mind. For more on this see: https://www.unblu.com/en/doc/latest/reverse-proxy-integration

Note: You can run the unblu Filter on a number of common proxies or web application firewalls, including AdNovum nevisProxy or Airlock or F5. It is even possible to run the filter in a custom installation. If you have any doubts as to whether the unblu Filter will run on your reverse proxy solution then contact us with your details.

Note: This document focuses on Unix/Linux OS (x86 and x86_64 platforms) but unblu supports a variety of other platforms.

For more on the unblu Filter see: https://www.unblu.com/en/doc/latest/filter-usage

Note: If you need to adapt the filter integration you can find a full specification here: https://www.unblu.com/en/doc/latest/filter-specification/

Source Packages

The distribution contains the following packages `(.tar.gz)` in source code form, which is based on `GNU Autobuild` for building and installation convenience under Linux and Unix:

`libunblufilter`

provides the core unblu functions, such as adding unblu's functions to the website.

`mod_unblufilter`

wraps the core functions and provides `libunblufilter` to Apache 2 as a filter.

General Package Dependencies

`mod_unblufilter` depends directly on `libunblufilter` (so you need to build `libunblufilter` first). In addition, it depends on the following:

`curl` (`curl` at runtime, `curl-dev` at build time)

`libexpat` (at runtime)

`uuid` (at build time)

`libidn` (`libidn` at runtime, `libidn-dev` at build time)

See the `README` file of the `mod_unblufilter` distribution package. It contains more detailed information about dependencies as well as package names for each distribution.

Apache Dependencies

Building `mod_unblufilter` requires Apache 2 as well as the Apache 2-specific development assistance tool `apxs`. (See http://httpd.apache.org/docs/2.2/programs/apxs.html for details). See the `README` file for details; especially concerning the required package name to include for your distribution.

At runtime `mod_unblufilter` will require the `mod_filter` module to be present (see http://httpd.apache.org/docs/2.2/mod/mod_filter.html). This is required for the sake of simplicity when it comes to configuring `mod_unblufilter` as a filter in Apache 2 in the required filter chain position.

Usually, the following are required to forward any requests to `/unblu/` (the unblu Collaboration Server - except if you use your own proxy setup):

`mod_proxy`

`mod_proxy_http`

libunblufilter Dependency

Building `mod_unblufilter` requires building `libunblufilter` first. In order to have a simple setup in the end, we recommend you include `libunblufilter` as a static library in `mod_unblufilter`. Thus, there is no runtime dependency to `libunblufilter.so`. Instead, `mod_unblufilter` is self-contained and only depends on typical system libraries (`libc` etc.).

Note: This is a recommendation, not a hard requirement. If you prefer to use shared objects, building and using them is possible with little extra effort. You can configure this during the build process.

Building

For both builds, `mod_unblufilter` as well as `libunblufilter`, it is possible to build `DEBUG` or `RELEASE` versions. They differ in that `DEBUG` is not optimized for speed, and stack traces / core dumps are more reliable. However, both build types will include symbols by default. In addition, `DEBUG` builds may have additional or slightly different diagnostic code enabled. We recommend you build both versions. In development and early test environments `DEBUG` builds are typically better. In late system integration or acceptance test environments, we recommend you use `RELEASE` versions.

Note: Production environments must use `RELEASE` versions only.

Building libunblufilter

In order to build `libunblufilter` unpack it with the following commands:

```
tar zxf libunblufilter.tar.gz

cd libunblufilter
```

If the tar command is not able to unpack the file directly, use gunzip first:

```
gunzip libunblufilter.tar.gz

tar xf libunblufilter.tar

cd libunblufilter
```

All build commands below will assume that you are in the extracted `libunblufilter` directory.

Release build

The following commands build a release version of `libunblufilter` and install it in the folders `/usr/local/include` (header file) and `/usr/local/lib` (or `.../lib64` if `x86_64` is the current architecture and build type and the directory exist). If you use another location, make sure that the build commands can access the file when you build `mod_unblufilter`.

```
mkdir -p build/RELEASE

cd build/RELEASE ../../configure --with-pic --disable-
shared

make

sudo make install
```

Options

- `--with-pic` is required to make sure that position-independent code is generated (this is required if a library is used in another library – as in the case here with the apache module). `Libtool` generally generates PIC code as well as non-PIC code. PIC code is used for the shared module, non-PIC for the static library. Here, both libraries need to be PIC, since both are intended for a shared library.

- `--disable-shared` will not build a shared object. This means that when you build a module later that depends on `libunblufilter`, the target will be the static library and not the shared module. This is the recommended build

configuration. Remove this option if you want to use `libunblufilter` as a shared module.

- `--libdir`: The library installation directory. If the build is `x86_64` and `/usr/local/lib64` exists, the default is `/usr/local/lib64`. You may override the default by specifying `libdir`.

Note: Specifying `libdir` will not affect the installation directory of the `unblu_filter_api.h` header file. This will still default to `/usr/local/include` Specify a prefix if you want to alter this, for example:

```
../../configure --with-pic --libdir=/usr/lib64
```

- `DESTDIR` (option): The prefix used when installing the library and the header file. Use this if the output should not be used directly but for a packaged example. Note that this will prefix the previous `–libdir` command as well.

The following example uses `DESTDIR`:

```
../../configure --with-pic --libdir=/usr/lib64 make
install DESTDIR=/tmp/libunblufilter
```

This produces a directory structure like `/tmp/libunblufilter/usr/lib64/libunblufilter.so` as well as `/tmp/libunblufilter/usr/local/include/unblu_filter_api.h`

Note: Installation into directories such as `/usr/lib` or `/usr/local/lib` may require root access. This is why in the above example the installation uses `sudo make install`.

Debug / test build

Building the debug version is generally the same as the release version, except that you add the option `--enable-debug` to the configure command. All other options are the same as for the release version.

```
mkdir -p build/DEBUG

cd build/DEBUG ../../configure --enable-debug --with-pic
--disable-shared

make

sudo make install
```

Building mod_unblufilter

mod_unblufilter is the Apache 2 (filter) module that is installed to Apache 2 to support unblu.

In order to build mod_unblufilter, unpack it as follows:

```
tar zxf mod_unblufilter-1.0.0.tar.gz

cd mod_unblufilter-1.0.0
```

If the tar command is not able to unpack the file directly, use gunzip first:

```
gunzip mod_unblufilter-1.0.0.tar.gz

tar xf mod_unblufilter-1.0.0.tar

cd mod_unblufilter-1.0.0
```

Once unpacked, build mod_unblufilter as follows:

```
mkdir -p build/RELEASE

cd build/RELEASE ../../configure --with-
apxs=/usr/sbin/apxs

make

make install DESTDIR=/tmp/mod_unblufilter
```

Options

- with-apxs: specify the path to the apxs tool. (If apxs is in standard directories or can be reached via $PATH, this is not necessary.)
- DESTDIR: Without specifying DESTDIR, the target of the installation is the Apache 2 modules directory as retrieved via apxs. For packaging builds this may not be intended. Instead, you may prefer to use a 'staged' installation in a different directory. In such situations, DESTDIR specifies a prefix to be used. In the above example if Apache 2 modules are located in /usr/lib64/httpd/modules, after installation, mod_unblufilter would be located under /tmp/mod_unblufilter/usr/lib64/httpd/modules/

Note: make install does not require root access rights if using DESTDIR with a target that is writable by the current user (such as /tmp). Such rights are required, though, if no DESTDIR is specified.

Building the debug version is generally the same as the release version, except that you add the option --enable-debug to the configure command. All other options are the same as for the release version.


```
mkdir -p build/DEBUG

cd build/DEBUG ../../configure --enable-debug --with-
apxs=/usr/sbin/apxs make

make install DESTDIR=/tmp/mod_unblufilter
```

15. Set Up the User Management Synchronization Tool (Optional)

In on-premise installations, enterprise customers often have an existing central user identity management system where all modifications are applied using the enterprise's own processes.

Note: We have used 'LDAP' here in order to highlight the concepts of the User Management Synchronization Tool, but it is possible to use other identity management access technologies. If you prefer to use something else (for example, a REST-based identity management system) then talk to us about it.

The tool syncs from your own identity management system to the unblu database. **This is a one-way synchronization.** That means that if you decide to use the Synchronization Tool you must choose which attributes/entities should be driven by the Synchronization Tool accessing your identity management system and which attributes/entities are to be handled from within the unblu Agent Desk, or from within the `unblu.properties` file.

It is important to put some serious thought into choosing the right synchronization strategy as there are consequences. For example, it makes sense, in a large complex organization, not to have many systems carrying duplicate user data. You already have the data ordered within your own identity management system so syncing should be a perfect solution. However, if you decide, for example, to 'sync all' (teams and users) then this would require you to have confidence that current team structures are entirely appropriate for how you want to use the unblu product(s). This is because when you ascribe precedence to your own identity management system you will not then be able to 'manipulate' teams from within the unblu Agent Desk.

Note: You may be able to enjoy the 'best of both worlds' by syncing users but not teams. In this way you could still use the unblu Agent Desk to create (team) structures on-the-fly that will persist through synchronizations. Syncing users but not teams can be an ideal strategy for a large organization who may be unsure as to exactly how they want to structure their visitor-engagement teams. When you are sure you have defined exactly the right structure (by 'experimenting' inside the unblu Agent Desk), you can then define the teams' structure on your identity management system. In this way you can use the flexibility of the unblu Agent Desk in combination with the efficiency of drawing data and data structures from your own identity management system. A caveat here is that handling teams from within the unblu Agent Desk means that you would have to assign users manually, which can be time consuming and prone to error. (Also note that the Synchronization Tool cannot

reproduce nested team structures; nesting must be defined from within the unblu Agent Desk.)

The User Management Synchronization Tool can insert or delete teams but it cannot update them. 'Team' is a standalone entity; if there are no users assigned to a team then that team, as far as the unblu Collaboration Server is concerned, does not exist. If you try to rename a team from within the LDAP directory unblu will drop/delete that team and create a new team, so your previous team settings would be lost.

Note: The User Management Synchronization Tool is integrated into the unblu Collaboration Server but it will not start by default. In order to use the tool you must first configure it in the `unblu.properties` file. See Synchronization Tool Configuration for more on how to get up and running with the User Management Synchronization Tool.

Note: You can also configure the Synchronization Tool from directly within the unblu Agent Desk at the Account level. (Note that while configuring the Synchronization Tool via the Agent Desk is possible it is probably only useful for multi-tenancy installations; as it requires 'Superadmin' credentials.) See Per Account Configuration.

When the User Management Synchronization Tool starts it queries the identity management system using LDAP for users. Users are (uniquely) identified by an ID (sourceID) supplied by the identity management system. This sourceID is used to keep the LDAP users and unblu users in sync.

If you decide to synchronize from your identity management system using LDAP, Synchronization Tool precedence means that all attributes defined for synchronization must be set on the LDAP server in order to populate the unblu Collaboration Server database. If you make changes from within the unblu Agent Desk those changes will be lost the next time the synchronization tool runs (if you have defined that attribute to be synced).

The first time the tool runs it will sync all users and associated attributes. After this initial run the database will remain relatively unchanged with only the differences being synchronized.

As an example, if you currently have a team of 100 generalists dealing with incoming requests and a second team populated by specialists, this same structure will exist within the unblu Collaboration Server after the synchronization tool is run. (In this example the first team would deal with the initial visitor engagement then forward those visitors accordingly to the second team, or members within the second team.)

The identity management system using LDAP should also contain the 'Authorization Role' (Admin, Supervisor, Registered User, etc.) that the unblu Collaboration Server requires to grant users the appropriate permissions.

Other information, such as languages or canned responses, can be set from within the unblu Agent Desk and will persist if not bound to a user or team. (If they are bound to a user or team and the Synchronization Tool deletes that user or team then that information will be

lost). You can, for example, define these settings at the Account or Global level. But if you, for example, add a new user or team from within the unblu Agent Desk, the next time the synchronization tool runs those users or teams will be lost.

Note: The default behavior of the tool is to sync 'everything' but user and team parameters can be synced separately. This means you can either sync users or, if the team structure has changed but not user information, you can sync just 'teams'.

Note: The 'Authorization Role' identifier is an attribute of the user, not an attribute of the team. Technically, one could create teams populated with many Supervisors, if one wished. This means that the rule of 'one supervisor per team', for example, must be enforced by you, and 'written' from your LDAP system to the unblu Collaboration Server. However, illogical as it may seem on the face of it, these design options allow you to, for example, create a whole team of supervisors in order that they can 'see' everything 'downstream'. This would mean a team of supervisors at the top level that can see all users and child teams below them.

Note: Other information, such as First Name/Last Name, email address, etc., are handled as 'soft identifiers'. This means that the information is not a technical requirement and that the synchronization tool will accept any information with the correct data type (and the correct data type = 'string').

General Functionality

The tool has the following base functionality:

- Is integrated into the unblu collaboration server.
- Provides a scheduler mode that accepts a cron-like action scheduling.
- Loads entities (users, teams) from a central User Management System.
- Stores new retrieved entities (users, teams) in unblu.
- Deletes entities (users, teams) from unblu that no longer exist in the central User Management System.
- Synchronizes changes from users in the central User Management System to unblu.
- Only separate syncing of users or teams is allowed. (When syncing 'all': teams are synced first then users.)
- Is configurable (mostly for the entity reader/writer parts, which can require a larger number of options that do not make sense as command-line arguments).
- Logs to the standard unblu server log.

User Model

The User Model describes how to create the implementation (internally to your organization) in order that data can be transferred from the source (your identity management system using LDAP) to the target (the unblu Collaboration Server).

A user entity model consists of the following parts:

- **SourceId:** A unique identifier which allows you to reliably sync between the central User Management System and unblu (such as employee #)
- **UserName:**
  UserName is the identifier within the unblu system. (It must be identical to the JEE Principal or Trusted Header UserId and must also be specified in the User Management System.) Note that UserName has an unknown structure and format. The only thing that can be assumed is that it is a string (which *does* include it being a number). In SSO implementations, the UserName is what the corresponding system delivers as output from the authentication process. In reality, that can be technical IDs like i12345, names like 'Kay Huber' or anything else that can be represented by a string.
- **Authorization role** (See Authorization Roles)
- **TeamName**
- **Email** (optional)
- **FirstName** (optional)
- **LastName** (optional)

Authorization Roles

Note: Authorization Roles are purely technical 'roles'; in that the unblu Role hierarchy bears no relationship to any internal roles in your organization. The roles should be granted according to how you intend to use the Agent Desk.

Current Authorization Roles are: `SUPER_ADMIN, ADMIN, SUPERVISOR, REGISTERED_USER`.

Authorization Roles are hierarchical (`SUPER_ADMIN` is the highest / root node, `REGISTERED_USER` the lowest role).

In the context of the synchronization tool, only `ADMIN, SUPERVISOR` and `REGISTERED_USER` are relevant. `ADMIN` has rights related to the account, all teams and all users. `SUPERVISOR` has rights related to his team and contained users and `REGISTERED_USER` only has rights regarding himself.

For more on Authorization Roles see: https://www.unblu.com/en/doc/latest/agent-desk-setup-and-user-guide

Team

A team is a grouping entity for users:

- Any user can only belong to exactly one team.
- A newly-created user that has no particular team applied belongs to a 'default team'.
- Teams can be nested. (But the Synchronization Tool cannot reproduce nested teams.)
- Teams are relevant for applying unblu configurations (it is a configuration scope).
- A configuration applied for team A automatically also applies to nested child teams A1 and A2 and all users belonging to A, A1 or A2.

Tool Actions

The Synchronization Tool supports the following actions:

1. Synchronization of 'team' information.
2. Synchronization of 'user' information.
3. Synchronization of 'everything' (that is: includes 1. and 2. in that order)
4. Scheduled mode.

The tool allows you to specify a list of synchronization schedules. Each synchronization schedule consists of a cron-like expression describing when to execute an action (`syncteam`, `syncuser`, `syncall`).

Entity Source (Internal Identity Management System)

The tool has the functionality to 'load' entities from an entity source. Entities to load are `team` or `user` entities. The design of the tool is done in such a way that loading the entities can be implemented using various technologies (i.e., there is an interface for an 'EntitySource' part with an initial implementation using LDAP to load entities).

LDAP Access

LDAP is the initial protocol to use to retrieve users from a central user management system.

https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol

The tool supports access to the identity management system using normal LDAP or with startTLS.

The tool first `binds` and then executes searches with the given configuration (see LDAP-Based Entity Reader Configuration) in order to load team or user information.

With the loaded information it creates internal user or team representations and maps the retrieved attributes to the model's attributes.

Entity Target (unblu Database)

The tool has the ability to 'store' entities in an entity target (in our case; the unblu Database). Entities to store are `team` or `user` entities.

The writing process consists of the following actions per entity type (team or user):

1. Retrieve all entity IDs that currently exist in the entity writing target (i.e.,, the database) -> `entityIDsBeforeWrite`
2. For each entity read from the source (e.g. LDAP):
    - Check if an entity for the given entity ID already exists
    - If it does exist, update it
    - If it does not exist, create it
    - From `entityIDsBeforeWrite` remove the given entity ID
3. Delete all entities that are still in `entityIDsBeforeWrite` (they no longer appear to exist in the entity source / LDAP).

Synchronization Tool Configuration

All configuration is based on unblu configuration properties.

Scheduling Configuration

If the tool is running in scheduling mode it must be configured using the following configuration keys:

`com.unblu.addons.synctool.runAtCronExpressions`: A list of cron expressions strings. Each string must contain a valid cron (time) expression and an action. The action is one of `syncteam, syncuser, syncall`.

For quartz cron expressions see the quartz documentation. Additionally to the quartz cron expressions the expression `@reboot` may be used to indicate that the action should be executed once when the unblu Collaboration Server starts.

Example:

To enable the sync tool add (and adapt) the code below to your `unblu.properties` file:

```
com.unblu.addons.synctool.runAtCronExpressions=["@reboot
syncall","* */15 * * * ? syncall"]
```

Note: The code above will synchronize everything (syncall=teams and users) every 15 minutes. All other configuration options (server, teams, roles) are listed below.

Entity Source Configuration

LDAP-Based Entity Reader Configuration

The following parts of the LDAP reader are configurable:

`com.unblu.addons.synctool.ldap.serverHostname`

`com.unblu.addons.synctool.ldap.serverPort`

`com.unblu.addons.synctool.ldap.connectionSecurity`: Indicate whether the connection to the identity management system using LDAP should be done using `LDAPS, STARTTLS, NONE`.

`com.unblu.addons.synctool.ldap.serverUsername`: User to use to bind; may need the `"cn="`-Prefix.

`com.unblu.addons.synctool.ldap.serverPassword`: Password to use to bind.

`com.unblu.addons.synctool.ldap.baseDN`: The root DN for all records.

`com.unblu.addons.synctool.ldap.userFilter`: The filter to use when retrieving user entities.

`com.unblu.addons.synctool.ldap.userSearchScope`: LDAP Search Scope: `OBJECT, ONELEVEL, SUBTREE`.

`com.unblu.addons.synctool.ldap.roleFilter`: LDAP filter to retrieve all unblu roles (if they are a groupOfNames); must contain %role% as a wildcard for all roleIdentifiers.

`com.unblu.addons.synctool.ldap.roleSearchScope`: LDAP Search Scope: `OBJECT, ONELEVEL, SUBTREE`.

`com.unblu.addons.synctool.ldap.teamFilter`: The filter to use when retrieving team entities.

`com.unblu.addons.synctool.ldap.teamSearchScope`: LDAP Search Scope: `OBJECT, ONELEVEL, SUBTREE.`

LDAP Attribute Mapping

The following configuration is required to map LDAP attributes to entity model attributes.

User

`com.unblu.addons.synctool.ldap.userIdAttributeName`: The attribute name for a field/string that uniquely identifies a user (for instance; Employee Number).

`com.unblu.addons.synctool.ldap.userNameAttributeName`: The name of the attribute whose value is the user's username.

Note: This is the identifier that needs to be passed to unblu for authentication (JEE Principal/ Trusted Headers UserID).

`com.unblu.addons.synctool.ldap.userEmailAttributeName` (Optional) property which contains user's email.

`com.unblu.addons.synctool.ldap.userLastNameAttributeName` (Optional) property which contains user's last name.

`com.unblu.addons.synctool.ldap.userFirstNameAttributeName` (Optional) property which contains user's first name.

Team

`com.unblu.addons.synctool.ldap.teamIdAttributeName`: The name of the attribute which contains the team name.

`com.unblu.addons.synctool.ldap.teamMemberAttributeName`: (default 'member') the name of the attribute(s) which contains users within that team.

Authorization Role

`com.unblu.addons.synctool.ldap.roleMemberAttributeName`: (Default 'member') the name of the attribute(s) which contains users that have that role.

`com.unblu.addons.synctool.ldap.superadminRoleIdentifier`: The name of the group that contains super admins. Will be substituted for %role% in the roleFilter.

`com.unblu.addons.synctool.ldap.adminRoleIdentifier`: The name of the group that contains admins. Will be substituted for %role% in the roleFilter.

`com.unblu.addons.synctool.ldap.supervisorRoleIdentifier`: The name of the group that contains supervisors. Will be substituted for %role% in the roleFilter.

`com.unblu.addons.synctool.ldap.registeredUserRoleIdentifier`: The name of the group that contains agents. Will be substituted for %role% in the roleFilter.

`com.unblu.addons.synctool.ldap.defaultRole (optional)`: The default role for users that otherwise are not assigned a role (One of: `SUPER_ADMIN`, `ADMIN`, `SUPERVISOR`, `REGISTERED_USER`).

Multitenancy Support

Note: Normally, all unblu Collaboration Server users are associated with a single account but multitenancy allows users to be assigned to multiple accounts.

The synchronization tool supports multitenancy usage. That is, syncing different unblu accounts, each with (potentially) different configurations. For instance, each account could connect to a different LDAP system.

There are two minor limitations:

1. The sync schedule can only be set globally.
2. Usernames must be globally unique (no two customers may use the username 'bob', for instance).

Configuration

Global Configuration

Note: 'Configuring at the account level' means configuring from within the unblu Agent Desk using Superadmin credentials.

To enable multitenancy support, most of the configurations from above should likely be made at the account level (and not globally in the server configuration file), with the following exceptions:

`com.unblu.addons.server.synctool.SyncTool.multitenancySyncMode` should be set to true to enable the multitenant sync.

`(com.unblu.addons.server.synctool.SyncTool.multitenancySyncMode=true)`

`com.unblu.addons.synctool.runAtCronExpressions` must be set globally as it cannot be set on a per account level.

Any configurations from the 'Entity Source Configuration' section can be set globally, **with the exception of** `com.unblu.addons.synctool.ldap.baseDN`. This configuration must be set at the account level and it indicates that this account indeed should be synced by LDAP. If an account does not have this property set, it is assumed that the account should not sync from any LDAP system but is configured entirely within the unblu server.

Per Account Configuration

To enable the LDAP sync at the account level the property `com.unblu.addons.synctool.ldap.baseDN` must be configured at the account level.

All the other settings from the 'Entity Source Configuration' section may be configured globally or individually on a per account basis. For instance, if only one LDAP server is used to sync all accounts, it might make sense to configure the identity management system using LDAP hostname, port and other server-specific settings globally and to set the LDAP structure individually, per account.

Edit settings at the account level

1. Log into the unblu Agent Desk as Superadmin.
2. Click on **Manage Accounts** below your account name.
3. In the Account list, click the **impersonate** button for the account you want to configure. The website will reload and you should now be logged in as a Superadmin of that account.
4. Select **Settings**, then **Account**. The **Account** window opens.
5. From the **Account** window menu select **Settings** then, in the filter search box, enter **ldap** and click the **Advanced** button. You now have all of the LDAP settings available for configuration.
6. When you are done with entering configuration settings, click **Save**.

The account should now be configured and will initiate an ldap sync the next time the sync process runs, according to the globally-configured schedule.

Note: You should monitor the server log for potential sync errors due to misconfiguration.

Limitations

Multitenancy Syncing has two limitations compared to syncing a single account:

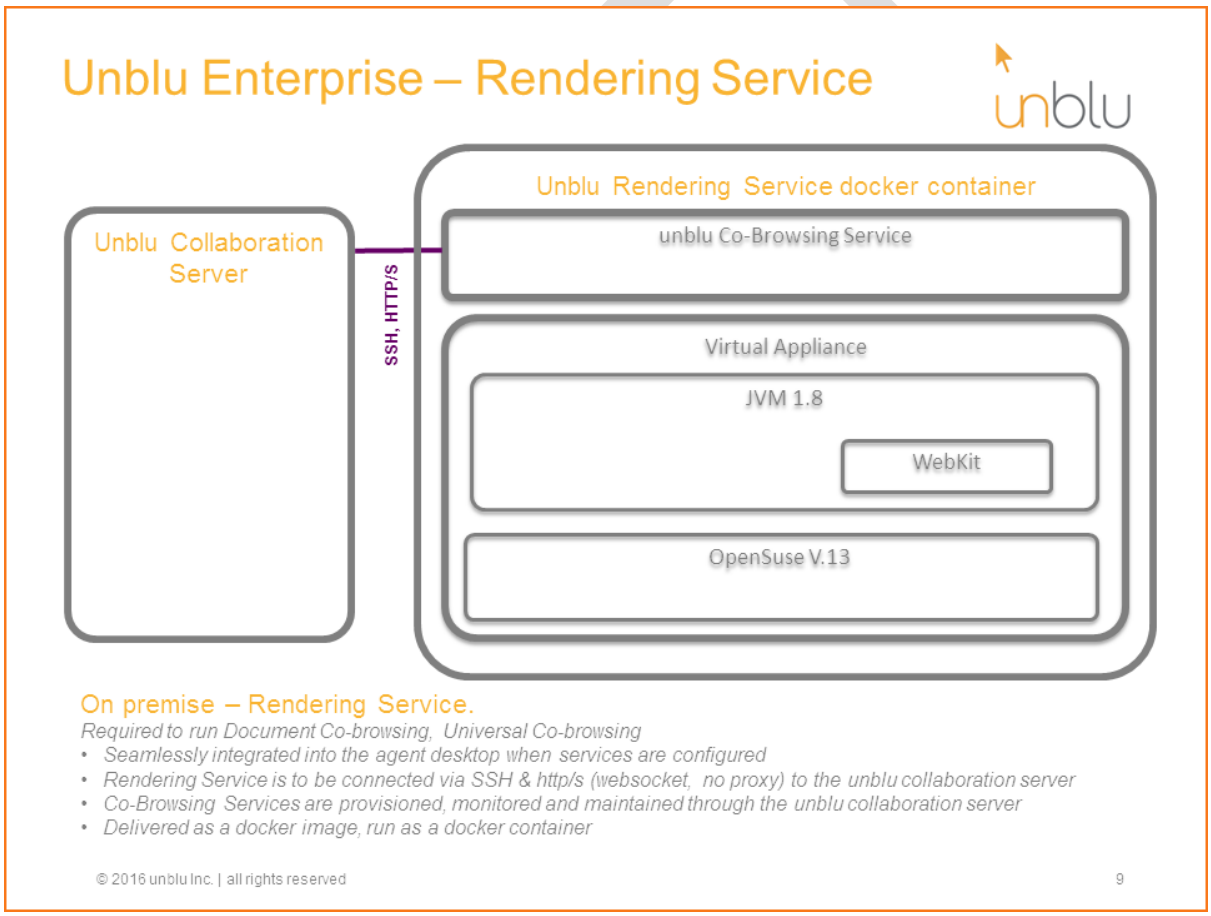
1. The sync schedule (defined in `com.unblu.addons.synctool.runAtCronExpressions`) must be

configured globally and cannot be set on an individual account basis. All accounts are synced according to this schedule. Syncing is performed sequentially, i.e., one account is synced after another.

2. Usernames must be globally unique, i.e., no two accounts may use the same username. If the name is not unique the existing user will be preserved and syncing of the account containing the new user will fail (due to the username already existing).

16. Enable Universal and Document Co-browsing (Optional)

The unblu rendering service appliance is a helper appliance that augments the unblu Collaboration Server with the capability to perform universal co-browsing and document sharing.



Network Connectivity Requirements to run the Rendering Service

SSH unblu Collaboration Server to the unblu Rendering Service

The unblu Collaboration Server must be able to connect to the unblu rendering service using the SSH protocol on a tcp port on the Docker host that is mapped to the **tcp port 22** in the unblu Rendering Service container.

HTTP(s) from the unblu rendering service to the unblu Collaboration Server

The unblu Rendering Service container must be able to connect to the unblu Collaboration Server using http(s) with the URL provided in the `com.unblu.hbworker.headlessBrowserReverseBaseUrl` configuration property.

HTTP(S) unblu Rendering Service to Internet / Intranet

If the unblu Rendering Service is used for universal co-browsing then the unblu rendering service container needs to be able to access all web servers (using HTTP(S)) that need to be available within universal co-browsing. Note that this requirement does not apply when the rendering service is only used for document sharing.

Configure and Run the Rendering Service Appliance

The unblu rendering service appliance is installed / started using the Docker command line tool with the following command:

```
# <name>: name of the docker container that is created on the docker
# host
# <port>: tcp port mapping for the unblu/renderingservice on the docker
# host
#         (the unblu collaboration server will connect to the
#          unblu/renderingservice through this tcp port)
# <password>: password for the unblu collaboration server to connect to
#             the unblu/renderingservice
docker run -d --name <name> -p <port>:22 -e PASSWORD=<password>
unblu/renderingservice
```

This command creates and starts a container from the unblu rendering service image on the Docker host.

To stop the renderingservice container:

```
# <name>: name of the docker container as given in the "docker run"
# command above
docker stop <name>
```

and to start the container again:

```
# <name>: name of the docker container as given in the "docker run"
# command above
docker start <name>
```

The unblu rendering service Docker image is stateless and does not store any persistent data, thus it is also possible to use a transient container (create a new container on every start of the system) instead of using a persistent container.

If you want to remove the rendering service use the following command:

```
# <name>: name of the docker container as given in the "docker run"
# command above
docker -t 10 stop <name>; docker rm -f <name>
```

Configure the unblu Collaboration Server to use the unblu Rendering Service

With the rendering service started you can configure the unblu Collaboration Server to use it with the following configuration properties:

```
# <collaborationServerBaseUrl>: fully qualified base url of the unblu
#          collaboration server (reachable from the renderingservice),
#          i.e. https://unbluserver.mycompany.com
#          (format: <protocol>://<host>[:<optionalPort>]
# <dockerHost>: host name or IP of the docker host where the
# unblu/renderingservice
#              container is running
# <renderingServicePort>: tcp port of the unblu/renderingservice
# container
#                        as mapped using the -p
# <renderingServicePort>:22
#                        argument in the "docker run" command
# <password>: the password of the unblu/renderingservice
com.unblu.hbrunner.runnerStrategy=HB_RUNNER_POOL_SINGLE_VA
com.unblu.hbworker.headlessBrowserReverseBaseUrl=<collaborationServerBa
seUrl>/sys-unblu
com.unblu.hbrunner.vaHostname=<dockerHost>
com.unblu.hbrunner.vaSSHPort=<renderingServicePort>
com.unblu.hbrunner.vaPassword=<password>
```

While the above block deals with the technical requirements to establish connections between the unblu Collaboration Server and the rendering service, the following property is required to enable universal and document co-browsing in unblu as a whole:

```
# enable universal co-browsing options in agent desk
com.unblu.collaborationsession.headlessBrowserEnabled=true
```

Test Connectivity of Docker to Collaboration Server

Once Docker has been started with `docker run` (see above), try to connect from the unblu Collaboration Server into Docker as follows:

```
# <port>: port on which docker host is providing access to the image
#         (corresponds to <port> from docker run command)
# <dockerHost>: host name or IP of the docker host where the
#               unblu/renderingservice container is running
# <password>: the password specified when starting the unblu rendering service
#             with docker run
ssh -p <port> unblu@<dockerHost>
Password: <password>
```

Within the Docker image you should also check whether the connection back to the unblu Collaboration Server works. For that purpose make sure the unblu Collaboration Server has been successfully started. Once it is started, login to the Docker image using ssh (see above) and execute the following:

```
# <collaborationServerBaseUrl>: fully qualified base url of the unblu
# collaboration server.
#  Must eventually match with what is specified in the unblu.properties file
#  (see com.unblu.hbworker.headlessBrowserReverseBaseUrl above)

curl <collaborationServerBaseUrl>/sys-unblu/rest/product/all

# example output from the above command:
product.com.unblu.core_4.1.7.RELEASE
product.com.unblu.headlessbrowser.model_0.0.1
product.com.unblu.thirdparty_1.0.0
product.com.unblu.enterprise.universe.base_4.1.7.RELEASE
product.com.unblu.domcap_0.0.1
product.com.unblu.dispatcher_3.0.0
product.com.unblu.filemanager_0.0.1
product.com.unblu.platform_2.0.2
product.com.unblu.proxy_1.0.3
product.com.unblu.runtime.jetty_1.0.3
product.com.unblu.node_4.1.7.RELEASE
product.com.unblu.cloud.storage.immutable_3.0.0
product.com.unblu.zookeeper_3.3.3
product.com.unblu.enterprise.universe.staticusers_4.1.7.RELEASE MAIN
product.com.unblu.chat_0.0.1
product.com.unblu.authenticator_3.0.0
```

```
product.com.unblu.cassandra_1.2.18
product.com.unblu.headlessbrowser_0.0.1
```

Diagnostic Tools

Some other network diagnostic tools that may help to analyze problems when you are connected with SSH into the unblu Docker image (example):

```
# check if dns lookups works
# <collaborationServerHostname>: Hostname or IP of the unblu collaboration
# server
Nslookup <collaborationServerHostname>


# check if ping works. Note: must be called with sudo. Operation is not
# permitted otherwise.
# <collaborationServerHostname>: Hostname or IP of the unblu collaboration
# server
Sudo ping <collaborationServerHostname>
```

17. Resource Histories Technical Detail

In unblu, resources represent pieces of data which are (usually) not directly contained in a visual but referenced from it (using a URI). Typical resource types are binary resources like images, documents (pdf, doc, etc.) or multimedia content (videos, audio, etc.). Besides those, also textual resources - e.g., style sheet (CSS) files - and theoretically even HTML itself can be classed as resources.

There are some exceptions to this rule:

The following styles are typically directly contained in a visual and then transformed into resources on the unblu server side:

- Styles in HTML attributes
- Styles in `<style>` tags

Resource Storage and Access

An unblu resource consists of two parts:

1. The **resource** itself.
2. The data contained - called a **blob** (**b**inary **l**arge **ob**ject).

Note: The word ‘blob’ is a ‘backronym’. The data type ‘blob’ described data too large and diverse to be managed on older systems. The name was originally taken from the 1958 film of the same name. Current computer systems now can handle such large files easily, thus

the blob was designated the **B**inary **L**arge **Ob**ject retrospectively, when it became a viable technology.

The Resource Object

The resource object provides the following information:

- **uuid** representing the resource
- **uri** of the original resource
- **mime type**
- **charset** (if textual resource)
- **state** (can be `PENDING`, `REQUESTING`, `REQUESTED`, `INVALID`, `MATERIALIZED`, `DELIVERABLE`)
- **origin** (can be `CSS_PROPERTY`, `CSS_IMPORT`, `STYLE_ATTRIBUTE`, `TAG`, `OTHER`)
- reference to a **blob** containing the actual data of the resource

The Blob Object

A 'blob' contains the actual data of the resource. There are two kinds of blobs:

1. **Basic blob** (or just blob).
2. **Typed blob** (based on a basic blob).

In addition there exists a "dummy" blob, which is marked with a specific id:

```
d2874f28-96e3-416b-bfed-ecb932b064fa
```

The purpose of this (dummy) blob is to indicate that there is 'no data available'. The dummy blob is used in resources, where the blob is not available (yet).

Basic Blob

Information provided by basic blobs:

- **Checksum** (currently a CRC32 checksum)
- **Length** (in bytes)
- **creation date**
- **binary data**

Typed blob

Information provided by typed blobs:

same as **blob** plus:

- **id**
- **mime type**

Storage

'Resource storage' is split into three areas:

- Resource store
- Blob store
- Resource table

The first two refer to their respective objects (see above). The resource table is the resolution table when looking for resources with a known backend URI (but not a UUID). By default, all three stores are located in memory. The resource and resource table stores have session scope - that is, their content is 'dropped' when the unblu session ends. The blob store on the other hand is global - thus the same resources in multiple sessions are stored only once in memory.

Resource Request URI

When the Resource History is turned off, URIs arriving at the agent browser directly point to the original image / element of the original backend webserver. Such URIs are directly requested by the agent browser.

On the other hand, when the Resource History is turned on, visuals arriving at the agent browser do not contain URIs pointing to the original backend webserver. Instead, the URIs are converted to a specific format pointing to the unblu server.

Resource URI with the Resource History turned on:

```
http://unbluserver/<restricted-path-prefix>/player/resource/res/<blob-
uuid>#<resource-uuid>

Example:
http://localhost:7777/unblu/player/resource/res/d38fb660-2b62-4364-8a5b-
2fe76de5e78f#8a17c8b5-7b6e-4dc1-a141-5664e6eec307
```

Note that the resource-UUID is never sent to the unblu server when the resource is requested from an agent browser, since the so called 'fragment' is only used in the browser. The reason to have it in our URIs is that it is appended, e.g., in a URI contained in a CSS. Thus, if the CSS is parsed on the server, the resource UUID can be extracted and the relevant information (especially inbound / outbound references) retrieved and processed. As soon as the CSS arrives at the agent browser, the browser will request the CSS but without the resource UUID – (in fact, the resource UUID is not required).

The fact that it is actually the blob being retrieved and not the surrounding resource is important. Since blobs are stored only once, the agent browser also only has to retrieve them once (and then have them locally cached). If there are many resources (e.g., many URIs) with the same content (blob) then caching is extremely effective. In extreme cases it is possible that the agent browser can load a web page faster than the visitor browser. This can happen if the original webpage contains hundreds of images with differing URIs but always the same data. In the agent browser, all of those resources would have the same blob, resulting in the same URI and thus would be retrieved only once.

Activating Resource Histories

In unblu, resources are handled fundamentally differently depending on whether the Resource History is turned on or off. The purpose of the Resource History (when turned on) is to store all resources that are visible on the visitor browser.

The main reason for doing this is to check, validate, and filter resources that will be displayed on the agent browser.

How to Activate Resource History

In your unblu properties file, add the following lines:

```
com.unblu.visual.resourcehistory.enabled=true
```

The following configuration is optional: If set to true, missing resources will be loaded by the unblu server and thus require the unblu server to be able to access the backend web server (which is not always desired or possible).

```
#com.unblu.visual.resourcehistory.loadMissingFromBackend=false
```

Behaviour with the Resource History switched off

Without the Resource History enabled, resource URIs in visuals are left as-is and transferred unchanged from the visitor browser to the agent browser. Thus, the resources themselves are not transferred to the unblu server nor are they stored somewhere. Instead, the agent browser will request the resource directly from the original backend web server on demand.

Note: Even though the URIs remain as-is in the visitor browser, they are checked on the unblu server and only let through if they correctly point to a resource on the backend webserver. Thus, it is not possible to 'inject' a link to some obscure resource on some unknown web server, or at least, this resource will not be requested by the agent's browser.

Note: This is the fastest way to access resources.

Behaviour with the Resource History switched on

With the Resource History turned on, the agent browser only interacts with the unblu server. There is nothing requested from anywhere else. All resources thus must be uploaded to the unblu server once they are discovered in a visual (e.g., as an image or stylesheet link tag).

This configuration provides the maximum level of security. If the visitor's browser sends a link to some other file, the agent's browser will not be able to resolve that resource - that is, it will request the resource from the unblu server which has no such resource and returns a '404 not found' message.

Note that browser caching is a problem for this approach. Typically, resources are transferred to unblu, e.g., from a reverse proxy where all data to the end user is flying by. Usually, traffic is only monitored by unblu once an unblu session has been started. That means that it may well happen that an end user surfs on the web site, retrieves images and

stylesheets and has them stored in his browser cache from that moment on. Once the unblu session starts, the resources no longer fly by the reverse proxy and thus do not get transferred to unblu. Unblu will detect this and send the visitor browser commands to re-request such files. However, this behaviour may take some time and thus, the performance is typically slower with the Resource History enabled than without.

Resource Processing

Depending on the resource type, a resource will be 'processed' prior to being used, or it is used in its raw / original form. A typical (and currently the only) processor is the CSS processor.

CSS Processor

The main purpose of the CSS processor is to identify URI reference locations and convert them to unblu resource URIs. The unblu server features two kinds of CSS processors:

1. Full CSS parser-based
2. 'Simple' regex-based

The 'Full CSS parser-based' processor is used when the Resource History is turned on. Basically, it simulates a CSS parser as present in the browser and filters the CSS. The full CSS parser thus not only scans for URIs, in addition it also drops unknown / unsafe / problematic CSS rules.

The 'Simple regex-based' CSS processor only scans for URIs and verifies / replaces them.

Dependency Processing

Resources can have incoming (inbound) or outgoing (outbound) references. A CSS resource, for example, can have outbound references to background images and inbound references to HTML files or other CSS resources.

These references have an impact once a resource changes. Resources can have the following states:

- `PENDING`: A resource with a certain URI has been seen, e.g., in a visual, but is not present in the resource store. It is expected to arrive on the unblu server later on.
- `REQUESTING`: A resource is supposed to have arrived on the server but did not arrive (so far). It is requested from the unblu client again.
- `REQUESTED`: A missing resource has been requested.
- `INVALID`: A missing resource is invalid - e.g., not available, 404 or similar.
- `MATERIALIZED`: A resource has arrived but may need processing.
- `DELIVERABLE`: A resource has arrived and has been processed. It is ready for delivery.

If a resource changes its state, e.g., from PENDING to DELIVERABLE, it typically includes changes in its data (that is, the contained blob has changed). If the blob changes, the URIs in

visuals (and possible dependent resources, like CSS files) need to be updated. Thus, a status update always leads to a cascade of resource updates which, in the simplest case, means there is nothing to do (no dependencies) or, for example, if a picture changes this can lead to reprocessing of the imported CSS, the default CSS, as well as the part of the visual where the default CSS was included in the HTML.

18. Minimal Requirements

Hardware Recommendations for unblu Collaboration Server

A single server setup (optionally with a hot standby) is the most robust, lowest cost to maintain, solution required to run a co-browsing infrastructure. To accommodate 10-20 concurrent co-browsing sessions (this means active usage of co-browsing in parallel), the following minimum requirements apply:

- 10 GB hard drive
- 1 Recent CPU core running at 2.4GHz or higher
- 4 GB RAM
- 4 Mbit bandwidth

Any additional 10-20 concurrent sessions require 1 additional core, 4 additional GB of RAM, and 4 additional Mbit of bandwidth.

With the outlined specs below, it is possible to scale up to 250 concurrent sessions. In our experience with existing large-scale clients in the financial services space, 250 concurrent sessions support up to 2500 agent seats.

We recommend you run a fully-productive server with a hot standby configuration to cover redundancy.

- Recent 16 core CPU with 2.4GHz or better
- 64 GB RAM
- Gigabit Ethernet or fiber channel network
- SSD (min 100 GB available space on top of OS)

Software Server Requirements

The unblu Collaboration Server is built to be deployed to any Servlet Container that supports Servlet API Version 2.5 or higher running on Java 8 or higher.

Recommended Containers

- Tomcat Version 7 or higher
- WebSphere 6.1 or higher
- WebLogic 10.0 or higher

Web Server

If you are using the Apache 2 web server, in order to build the `mod_unblu` filter the Apache 2-specific development assistance tool `apxs` is a requirement.

See: http://httpd.apache.org/docs/2.2/programs/apxs.html

Network Requirements

The unblu Collaboration Server needs to be accessible to both visitors and agents. In minimal configuration the server does not communicate with other servers and thus does not require Internet access (although Internet access may be required for certain setup procedures).

If additional modules are deployed see the following sections to find out which network connectivity requirements apply.

Universal and Document Co-browsing Requirements

The optional unblu document sharing component requires the unblu Collaboration Server to be installed with the unblu rendering service.

The unblu Collaboration Server must be able to communicate with the unblu rendering service via SSH protocol on **port 22**.

The unblu rendering service must be able to communicate with the unblu Collaboration Server on **ports 80** and **443** in both directions (inbound and outbound).

Note: Either no proxy must be between these two servers, or the proxy must support the WebSocket protocol.

Hardware for the unblu Rendering Service based on Docker image

The Docker image is available on the Docker marketplace. The unblu Collaboration Server needs to have access to the Internet for installation.

Resource consumption is entirely dependent on the size of the documents loaded, dynamic elements of websites visited and the amount of scrolling performed on the target.

Recommended starting configuration:

- Recent 16 core CPU with 2.4GHz or better
- 32 GB RAM
- Gigabit Ethernet or fiber channel Network
- SSD with 100 GB

Supports:

- 32 parallel rendering sessions with constant throughput

Note: Loaded documents which are not being scrolled do not generate load.

19. Video and Audio Requirements

The unblu Collaboration Server communicates with the tokbox cloud service to offer video and audio facilities. Currently, video is not provided as an embedded on-premise service. Tokbox provides an API that we use to allocate video sessions.

This may seem like a problem for financial institutions who prefer everything to sit behind their own firewalls. **However, using the tokbox cloud service is no different, in principle, from making a telephone call through a provider.**

Running Unblu's audio / video service requires the Browser to access the audio / video chat services over specific ports in order to work. A network administrator in your organization can configure these firewall settings:

Minimal Requirement:

The minimal Requirement is that TCP **port 443** is open. Some firewall/proxy rules only allow for SSL traffic over port 443. You will need to make sure that non-web traffic can also pass over this port.

Better Experience:

In addition to the minimal requirements being met, we also recommend that UDP **port 3478** is open. The User Datagram Protocol (UDP) favours low latency over error-correction and is well-suited to audio and video communications.

Port 3478 only accepts inbound traffic after an outbound request is sent. The connection is bidirectional but is always initiated from the corporate network/client so it is not possible for an external entity to send malicious traffic in the opposite direction.

Best Experience:

For the best possible experience, we recommend that UDP **ports 1025 - 65535** be open.

You can test if a network meets the connectivity requirements by using the OpenTok Connectivity Doctor: http://www.tokbox.com/tools/connectivity/ (you must use Chrome). There is also a Connectivity Doctor app in the Apple App Store as well as the Google Play Store.

Whitelist the following domains:

*.tokbox.com

*.opentok.com

Whitelist the following HTTPS verification servers for the unblu HTTPS certificate:

oscp.godaddy.com

crl.godaddy.com

Bandwidth Requirements

- Video: 300 kbps per stream (recommended lowest level)
- Audio: 50 kbps per stream (recommended lowest level)

Browser Requirements

Unblu is optimized to run on modern browsers. We do not support browsers older than the following:

- Internet Explorer 9 (or newer)
- Google Chrome 40 (or newer)
- Apple Safari 7 (or newer)
- Mozilla Firefox 30 (or newer)

Note: The browsers listed here will give you optimal performance:

Running Unblu's audio / video service requires WebRTC support which is provided by the following Browsers:

- Google Chrome (latest release version)
- Firefox (latest release version)
- Internet Explorer 10 and 11 (with the Opentok Plugin for Internet Explorer provided at the start of an audio / video chat)
- Microsoft Edge

- Google Chrome for Android (latest release version)
- Firefox for Android (latest release version)

Server Communication with Opentok

The unblu Collaboration Server must be able to communicate with https://api.opentok.com which is a service located on the Internet. Thus, **port 443** must be open for outbound traffic to `api.opentok.com`.

20. Mobile Requirements

iOS Tablets: Safari on iOS 7 (or newer). In-page chat UI renders top right of viewport.

Android Tablets: Chrome on Android 4.2+. In-page chat UI renders top right of viewport.

Smartphones: Android (Chrome) and iOS supported in principal. Usability restrictions apply for small screens.

21. unblu Collaboration Server Network Ports

In addition to the ports mentioned throughout this document it may be necessary to configure the various ports used by unblu itself or its components (Cassandra, Zookeeper). The following sections list the addresses and ports used by unblu. The proper configuration of server and client parts is explained.

Zookeeper Configuration

Zookeeper Server

```
com.unblu.zookeeper.server.hostname=AAAA (default: localhost)
com.unblu.zookeeper.server.clientPort=xxxx (default: 2181)
```

Zookeeper Client

```
com.unblu.zookeeper.client.hosts=AAAA:xxxx (default: 2181)
```

Note: If you change `AAAA` or `xxxx` for either the server or client part, you **must** configure the other part correspondingly! Changing `AAAA` in the zookeeper server part to `unbluserver.intra` thus **must** include `unbluserver.intra` in the `client.hosts` configuration!

Cassandra Configuration

Cassandra Server

```
com.unblu.cassandra.server.rpcListenAddress=BBBB (default: localhost)
com.unblu.cassandra.server.rpcPort=yyyy (default: 9160)
```

Other Cassandra server ports must be different if multiple unblu instances are started on the same physical machine but they do not have to be considered for the client settings below:

```
com.unblu.cassandra.server.nodeListenAddress=CCCC (default: localhost)
com.unblu.cassandra.server.storagePort=uuuu (default: 7000)
com.unblu.cassandra.server.sslStoragePort=vvvv (default: 7001)
com.unblu.cassandra.server.nativeTransportPort=wwww (default: 9042) ->
available as of 3.4.2 (as of 3.3 included in the product but not configurable)
```

Cassandra Client

```
com.unblu.cassandra.connection.host=BBBB
com.unblu.cassandra.connection.port=yyyy
```

Note: If you change BBBB, or yyyy for either the server or client part, you **must** configure the other part correspondingly!

22. Further Reading

This section contains links to content that may not be fully covered in this concise guide.

Unblu Server Configuration

For more on configuring the unblu Server see:
https://dev.unblu.com/confluence/display/UNBLUTGNEXTEN/Unblu+Server+Configuration

For a configuration reference guide see:
https://dev.unblu.com/confluence/display/UNBLUTGNEXTEN/Unblu+Server+Configuration+Reference

For information on how to validate your unblu Server configuration see:

https://dev.unblu.com/confluence/display/UNBLUTGNEXTEN/Unblu+Server+Configuration+Validation

Advanced unblu Server Configuration

For a guide to more complex and dynamic setups see:

https://dev.unblu.com/confluence/display/UNBLUTGNEXTEN/Advanced+Unblu+Server+Configuration

unblu Server Security

For more on unblu Server security see:

https://dev.unblu.com/confluence/display/UNBLUTGNEXTEN/Unblu+Server+Security

Single Sign-on Setup

To integrate unblu into your current authentication system see:

https://dev.unblu.com/confluence/display/UNBLUTGNEXTEN/Unblu+SSO+Setup

Logging

For more on unblu Server logging see:
https://dev.unblu.com/confluence/display/UNBLUTGNEXTEN/Unblu+Server+Logging for

For more on the session log output format(s) see:
https://dev.unblu.com/confluence/display/UNBLUTGNEXTEN/Session+Log+Output+Format

Text Localization

To use localization in your unblu Server installation see:

https://dev.unblu.com/confluence/display/UNBLUTGNEXTEN/Unblu+Server+i18n+Configuration

For a reference guide to localizing text see:

https://dev.unblu.com/confluence/display/UNBLUTGNEXTEN/Unblu+Server+Localized+Text+Reference

For advanced localization configuration see:

https://dev.unblu.com/confluence/display/UNBLUTGNEXTEN/Advanced+unblu+Server+i18n+Configuration