Administration Guide

OpenText™ Brava!™ Enterprise

Install and configure the Brava! Enterprise components.

CLBRVW160612-ABE-EN-02

Administration Guide
OpenText™ Brava!™ Enterprise
CLBRVW160612-ABE-EN-02
Rev.: 2025-May-21

This documentation has been created for OpenText™ Brava!™ Enterprise 16.6.12.
It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product, on an OpenText website, or by any other means.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111
Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440
Fax: +1-519-888-0677
Support: https://support.opentext.com
For more information, visit https://www.opentext.com

Table of Contents

Chapter 1

Introduction

This guide provides an overview of the OpenText™ Brava!™ Enterprise product, information about installation and configuration, as well as troubleshooting tips to assist administrators and integrators with advanced customizable Brava! Enterprise features. This guide is designed for integrators and business process designers who are configuring their implementations to provide advanced capabilities on the server. A base knowledge of servlets, web servers, and web browsers technology is required.

For the latest documentation updates be sure to visit our product page on OpenText My Support (https://support.opentext.com/).

This chapter provides an overview of the Brava! Enterprise product to help you make informed choices when installing, configuring, customizing and troubleshooting the product. You will learn about the primary Brava! Enterprise components, including the Job Processor and Server, as well as security considerations and operating system requirements.

1.1 About Brava! Enterprise

Brava! Enterprise software is a web application for the sharing and collaboration of many file types, including raster, CAD, PDF, and Office documents. See “Supported file formats” on page 117 for a full list. The primary Brava! Enterprise components are the Brava! Server, Brava! Clients, Brava! License, and Job Processor. They can be installed on different machines with different operating systems in a production environment (see “Installation requirements” on page 13 for supported platforms). End users access the Brava! Clients through their browser.

Brava! Enterprise can be used alone as a client/server application for internet/intranet viewing and markup of files, or as a component of an enterprise document solution that can include any or all of the Brava! viewing and publishing products and third-party document management systems. Extensive customizations are available to allow Brava! Enterprise to work seamlessly within your organization.

! **Licensing**

The purchase of client license seats entitles you to a single Brava! Server, Job Processor, and License Server. Increasing the number of Brava! Servers, License Servers, or Job Processors in your environment involves an additional licensing fee that should be discussed with your Sales group Account Executive.

1.1.1 How Brava! Enterprise works

Brava! Enterprise is composed of these major components:

- Brava! Clients (HTML, 3D HTML, HTML tablet/phone, HTML Video, DocMerge, Preview, or ActiveX Client options)
- Brava! SDK
- Brava! Server
- Brava! Cache
- Brava! Job Processor
- Brava! License

The Brava! Clients are the user interface for viewing and marking up files. The Client communicates with the Brava! Server (using HTTP or HTTPS) to receive site configuration information and to access files. A custom markup service is set up by integrators to retrieve and save markups.

The Brava! HTML Client provides users the capability to load documents from the Brava! Server online using a web browser. Documents are processed with the Brava! Server as HTML output and are presented in this streamlined viewer for quick viewing, searching, publishing, redaction, and adding/reviewing a limited set of annotations, including Changemark discussion. The HTML Client also offers viewing options for compare mode, 3D files, video files, tablets, and mobile phones.

The Brava! ActiveX Client is an ActiveX control that is embedded in a container such as the Internet Explorer browser. Sample web pages are provided that load the Brava! ActiveX Client control into a browser. See "Sample web pages" on page 53. Properties such as the file to be loaded, the GUI components that are enabled, and other client attributes (such as the on-screen watermark) are established by setting key value pairs in the BravaXParams HTML parameters.

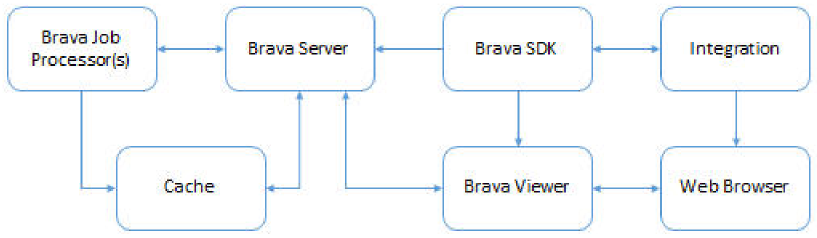
The Java and .NET SDKs are the primary integration points for connecting Brava! to other systems. They provide a programmatic interface for activities like uploading a file to a Brava! Server, telling the server how you want that file displayed, and then launching a viewer instance.

The Brava! Server is a servlet-based application that responds to Brava! Client requests to access files and publish documents. Brava! Server queues requests to publish native files for the Job Processor to retrieve. The Brava! Server maintains a cache of the published files to speed up retrieval for subsequent requests for the files.

The Brava! Cache stores the artifacts needed to view files in the clients. When a document is viewed more than once, having those artifacts in the cache means that the document does not need to be converted again. This makes it faster to display content in the clients on subsequent views of a document. The cache is not designed to hold the artifacts needed to render all documents in a repository. Instead, it is designed to be transitory in nature, and contains only the artifacts needed to render the most recently viewed documents. For more information about Brava! cache, see "Displaylist cache" on page 10 .

The Job Processor software is a scalable conversion solution that is used with Brava! Enterprise to publish drawings and documents to OpenText's common display list format. The Job Processor component is typically deployed to a dedicated server in a production environment. For high load deployments, multiple Job Processors on multiple servers can be used. The Job Processor publishes files in one of several ways. If the file being converted is one of the natively supported files (see "Supported file formats" on page 117), the Job Processor performs the conversion internally and requires no other application. If the file being converted is not one of the natively supported formats, an application must be registered on the Job Processor to perform a **PrintTo** action for that file type. For more information about how to set up applications to publish file types not natively supported by the Job Processor, see "Publishing using other applications" on page 91. Files can also be published using the Office Automation API.

The Brava! License is a servlet application that provides licensing capabilities to the Brava! Server for authorizing server and client licensing. The Brava! License only needs to be accessible to the Brava! Server and can be installed within the same servlet container.



The standard information flow is as follows:

1. An integration uses the SDK to notify Brava! that a file needs to be viewed. If the file is not already in the Brava! cache, Brava! can retrieve the file from a URI or the integration can send the file to Brava! using HTTP/HTTPS.
2. The integration tells Brava! how to render the file. A rendition can be as simple as converting a single file to the required format to display the file within the HTML Client, or it can be as complex as a compilation of one or more pages from multiple files into a single document. Markups, redactions, banners, and watermarks can also be burned into a rendition. A rendition can provide a custom view of one or more files for display in one of the clients while keeping the content of the original files unchanged and intact.
3. If a rendition matching the specified criteria does not already exist in the Brava! cache, Brava! places a rendition job into its job queue. Job Processors periodically poll the Brava! queue to see if jobs are available. Once a Job Processor completes a job; it immediately asks for a new job.

If no new job is available, Brava! Server will hold the request for several seconds waiting for a job to appear. After a timeout is reached with no new job appearing, a response is returned to Job Processor that no new job was found. The Job Processor then uses a back-off algorithm to wait a steadily increasing amount of time, up to the time specified in the `JobProcessor.config` file's *`job.sleep`* value, before polling for a new job. Since the Brava! Server can wait for new jobs on its own, the *`job.sleep`* parameter should always be set to 0.

4. After telling the Brava! Server how to render the file, the integration creates a client configuration that specifies which document to display and optional client behavior. Options include enabling or disabling different controls, specifying which markups to automatically load, search text to immediately highlight, and other options. This step can happen in parallel with steps 3 and 4. The integration does not need to wait for the Job Processor to complete its processing before continuing.

5. The integration displays the client in a web page, passing the configuration object to the viewer.

6. The client retrieves the necessary rendition artifacts from the Brava! Server and displays them for the user. If needed, Brava! Server can create single-page publish job to reduce viewing latency for artifacts requested by the client.

For a complete description of how an integration uses the SDK to communicate with the Brava! Server and what options can be set, see the Brava! SDK documentation available from OpenText support.

1.2 Displaylist cache

The displaylist cache allows documents published by Brava! Enterprise to be cached for faster future access.

The displaylist cache stores all files in the cache on the file system. This location can be a local disk drive, a SAN/NAS, or some other type of file store. (Microsoft DFS is not supported.) Each Brava! Server must have its own dedicated file system cache. They cannot be shared between servers, even if the cache is set to a shared network drive. The file system cache provides a range of options for managing its size and when documents are removed from the cache. For information about configuring the cache, see “Configuring Brava! Publication Cache” on page 102.

By default, the Windows installer shares the displaylist cache folder on the network with full read and write access for “Everyone”. This shared folder is installed, by default, to `%ProgramData%\OpenText\dlcache`. It is recommended that you restrict access to this network share. For example, you might want to use the same service account that is used for running the Brava! services. Additionally, on the local machine, you might want to further secure this folder. You can use NTFS permissions as long as the Job Processor is running under an account with those permissions.

Important
All Job Processors used must have full read and write access to this folder whether the service is running on the local machine or over the network on a separate machine.

1.3 Multiple Brava! Servers for improved availability and scalability

Multiple Brava! Server instances can be set up to improve the availability and scalability of Brava! Enterprise. Each Brava! Server has its own displaylist cache, so requests from the integration component and the viewer for a particular document view and subsequent publishing all need to be routed to the same Brava! Server.

The two main methods for accomplishing this routing is either through a proxy/load balancer, or by directly communicating from the integration component and viewers to the selected Brava! Server. For information about how to integrate to multiple Brava! Servers, refer to the **Load Balancing** page in the **Advanced Integrations** section of the BravaSDK documentation.

Note: Be aware that multiple Job Processors can be configured for each Brava! Server. For configuration information, see "Setting up multiple Job Processors for a single Brava! Server" on page 96 .

1.4 Customizing and integrating Brava! Enterprise

Several methods can be used to integrate with Brava! Enterprise:

- Various configuration options are available to customize the state of the Brava! Enterprise application. For instance, most of the toolbar buttons and sub-menu options, with their associated functionality, can be enabled or disabled. Date format, initial page to load, initial search text, and banners and watermark text are some examples of other states that can be pre-configured. It is possible to set configuration properties for ActiveX Clients for an entire site (see "Configuring the Brava! Server properties" on page 71) or each time either the ActiveX or HTML Client loads (see the Brava! SDK documentation).
- The file access and markup capabilities can be extended so that Brava! Enterprise can be integrated with Enterprise Content Management (ECM), Product Life Management (PLM) or project management solutions.
- Custom functionality can be invoked when particular events happen within Brava! Enterprise. See the `NotificationListener` HTML parameter in the *OpenText Brava! - Brava! Enterprise ActiveX Client Parameter Configuration Guide (CLBRVW-CAX)* for instructions on how to establish this customization.
- The Brava! Clients expose various methods that can be manipulated through HTML scripting language such as JavaScript.

See the Brava! Enterprise Programmer's Guide contained in the Brava! SDK documentation package for more information about customization.

Chapter 2

Installation

This chapter provides information about installing the Brava! Enterprise product. You will learn about the customization options regarding the installation, and how to install the Brava! Enterprise components. This chapter also covers how to test and verify that your installation is successful.

2.1 Installation requirements

Installing Brava! Enterprise components requires some knowledge of web server/servlet engine technology. Note that Brava! Enterprise maintenance support for execution on tested platforms and components is limited to the period that those platforms and components are supported by their vendors.

Refer to the product release notes document available on OpenText My Support (https://support.opentext.com/) (Brava! Enterprise Release Notes version 16.6.12.pdf) for detailed and the most up to date information about supported platforms, systems, and versions.

2.1.1 End User Machines

The Brava! HTML Client is downloaded when a user browses to a web page in which it is embedded.

When using the ActiveX Client, the end user machine downloads the Brava! Client ActiveX control (and dependent components) when it initially accesses a web page that references the Brava! ActiveX control. By default, the client components are installed to the user's HOMEPATH (for example: `C:\Users\<Username>\IGC\<version>\<GUID>`) It is possible to pre-install the ActiveX control. For more information, see "Permission to install the Brava! ActiveX Client control" on page 26 .

Note: An MSI package is available for deploying the Brava! ActiveX Client to end users. See notes in User Account permissions, and *OpenText Brava! - Brava! Enterprise MSI Public Properties Guide (CLBRVW-IBE)*.

The web package/MSI package must be run as the service account you intend to run the service.

ActiveX Client

Important
The ActiveX Client requires the machine to be updated with the latest Microsoft Universal C runtimes. These runtimes are a prerequisite for

installing the CAB, MSI, or Batch File for the client and can be downloaded from the following Microsoft support page:

https://support.microsoft.com/en-us/kb/2999226

- IE versions are supported on operating systems only where that version of IE is supported by Microsoft.
- For the ActiveX Client running on Windows, users must have the Internet Explorer browser set up with **Protected Mode** disabled. The safest way to do this is to add the viewer launch site to the **Trusted Site** list in IE.
- If UAC is enabled on the client system and IE is running in **Protected Mode**, the ActiveX control is unable to access the system's **PRINT SCREEN** key to monitor screen capture functionality. This means that disabling print permissions for documents disables the printing menus and full document printing functionality only. It does not obscure the document display if the **PRINT SCREEN** key is pressed.

2.1.2 Brava! Server

The (required) Brava! Server is a web application that runs within a servlet engine and is composed of various supporting configuration files.

- A **64–bit servlet container** (such as Tomcat) is required for Brava! Enterprise.

  Tomcat or another servlet engine must be installed prior to running the Brava! Enterprise installer.
- A **64–bit OpenJDK 17** installation is a requirement of Brava! Enterprise servlet container.

Note: Tomcat and Java are not bundled in the Brava! Enterprise installer. Users must install their desired web server and Java runtime before they install Brava! Enterprise.

2.1.3 Brava! License Server

The (required) Brava! License is a web application that runs within a servlet engine and is composed of various supporting configuration files.

- A **64-bit servlet container** (such as Tomcat) is required for Brava! Enterprise and should be installed prior to running the Brava! Enterprise installer.
- A **64–bit OpenJDK 17** installation is a requirement of Brava! Enterprise servlet container.

Note: Tomcat and Java are not bundled in the Brava! Enterprise installer. Users must install their desired web server and Java runtime as pre-requisites of a Brava! Enterprise installation.

2.1.4 Web Server

The Brava! Clients and dependent components are hosted on this Web server machine. A client end user machine downloads these components the first time it accesses a Brava! web page.

2.1.5 Job Processor

These required components consist primarily of a .NET application and various Windows EXEs and DLLs.

Microsoft .NET Framework 4.6.2 is a requirement of the Brava! Job Processor.

Note: .NET Framework is not bundled in the Brava! Enterprise installer. Users must install .NET Framework prior to installing the JobProcessor component.

The Job Processor installation includes installation of the **CSF Writer**, which is necessary for converting some file types.

Service account requirements

The service account used to run the Job Processor must have the following:

- **Folder/Network permissions:**

  Read and write access to the source and target destinations for jobs (this is the displaylistcache in Brava!), as well as permission to call the notification URL.

- **Privileged Executable Launching permissions:**

  The Job Processor service launches instances of separate executables to handle multiple threads of execution. These child processes must inherit access to the printing subsystem of Windows to successfully launch and interact.

- **Printer Configuration:**

  – CSF Writer must be installed and configured.

  – Administrative access to the CSF Writer printer driver is required.

  – The CSF Writer requires that Print Spooler services be set to **Automatic** or **Started**. The installation cannot start the Print Spooler service, required for CSF Writer, if the service is **Disabled**.

  – Permissions to manage the default printer.

- **Microsoft Office:**

  Access to the installed and activated version of Microsoft Office. To render Office formats (DOC, DOCX, XLS, XLSX, PPT, and PPTX), Office is required to be installed and licensed on the Job Processor machines, and applications used for publishing initialized - before running the Job Processor installation. We offer alternative technology that does not require Office as an additional option. Contact your Account Executive for additional information. See also “Installing Job Processor native applications” on page 30.

- **Performance Counter permissions:**

  The Job Processor accesses Windows performance counters to track job status. The service account might require being added as a member of the **Performance Monitor Users** group if the service account does not have full administrator privileges.

Notes

- The exact set of required permissions can vary, depending on which features are configured, which version of Windows is used, and which version of Office is used. Newer versions tend to be more restrictive and updates to Windows, Windows components, or Office can sometimes change the permissions required as well.
- We strongly recommend running the Job Processor with full administration privileges if possible, which is the scenario that is used for product testing.
- Microsoft Office requirements are important to avoid failure when publishing using either Office Automation or Print Publishing. These methods are defined as:

  Office Automation: processing Office documents by using the automation API provided as part of Office to either print to the CSF Writer printer driver or to save the Office files to XPS format, and then converting the result.

  Print Publishing: processing documents through an external application that prints them to the CSF Writer printer driver, and then converting the captured data from that print operation.
- The requirements listed for printer configuration are important to avoid problems when using Print Publishing.

2.1.6 Brava! Routing Service (optional)

Important

The Brava! Routing Service is a new component available in this release as an optional feature to allow customers to investigate and evaluate the benefits that it might provide to them. It will be refined in subsequent releases for broad adoption.

The Brava! Routing Service is a component that facilitates load balancing across multiple Brava! Servers without the need for a shared displaylistcache (shared database cache was deprecated in Brava! version 16.4).

It is recommended that more than one Brava! Routing Service is installed behind a load balancer along with more than one Brava! Server. This setup facilitates scalability and fault tolerance.

Note: For details about deployments requiring the Brava! Routing Service and how it functions, see “Routing Service Appendix” on page 119.

If you choose to include a Brava! Routing Service in your installation, see "Installing the Brava! Routing Service" on page 22 for details.

2.1.7 Hardware Requirements

Recommended hardware requirements on the Brava! Server and Job Processors include:

Brava! Server

- 15 - 60 GB of free hard drive space for displaylistcache. The amount required depends on how much caching of published files is required for your integration. (Note that DFS is not supported.)

Job Processor

- At least 4096 MB RAM (8192 MB or higher recommended).
- At least four CPU cores are recommended and more might be required in higher throughput environments.
- 700 MB available hard disk space is required for Job Processor and all installed files.
- Additional disk space is required for temporary files during publishing. Space requirement varies with number and complexity of documents served. We recommend at least 5 GB of temporary disk space on the C: drive for processing of files through the CSF Writer.
- At least 500 MB (depending on amount and size of original files) of hard disk space is required for temporary files created during the processing of requests.

  Note: The location of this temporary directory can be configured using the Windows Environment variables. To configure environment variables:

  1. In Control Panel, click **System and Security > System**.
  2. Click the **Advanced system settings** link, then click **Environment Variables**.
  3. In the **User variables ..** list, edit the value for `%TMP%` or `%TEMP%` to configure the temporary directory.

2.2 Linux installations

The Brava! Enterprise Linux installation script can install all or some of the following components:

- Brava! Server
- Brava! SDK
- Brava! License (must be run on a x86_64 CPU architecture)
- Linux Publisher

The publisher component's main function is to convert files from their native format to a format that can be viewed by the Brava! viewers.

To use the **Linux Publishing Agent**, refer to *OpenText Brava! Enterprise - Linux Publishing Agent Administration Guide (CLBRVW-SBE)*.

An alternate to using the publisher component for conversion is to use the Windows Job Processor. The Job Processor provides better fidelity for certain formats, such as Office, since it can take advantage of print publishing on Windows. If deciding to use the Windows Job Processor, see "Installing the Job Processors" on page 24 for instructions on installing this component. Note that the installation of the Job Processor on Windows prompts for the Brava! Server URL. If installing the Brava! Server component on Linux, you must configure a Samba mount between the Linux machine running the Brava! Server and the Windows Job Processor machine. Further instructions are provided in the README file of the Brava! Enterprise Linuxinstallation.

The Brava! Linux installation is primarily composed of a properties file and a bash script that updates tokens within files and copies them to their target locations. The Brava! Server, SDK, and License components all run as web applications (webapps) within a servlet container such as Apache Tomcat.

Note: The Brava! install script does not install a servlet container. That step is performed independently of these installation steps.

If deploying to a servlet container other than Tomcat and you need to create war files for the Brava! Web Apps, you'll need to have a jar executable installed on your installation machine. The jar is included as part of the OpenJDK-devel package.

To install the Brava! components on Linux:

1. Copy the file **Brava-16.6.12.0.{build}.tgz** to a temporary directory and extract it: **tar xvfz Brava-16.6.12.0.{build}.tgz**
2. After extraction, a README file is available that explains in detail how to proceed with the installation:

a. The first step involves editing the install properties in the `bravaInstall.properties` file. Within this file you can set properties to configure which of the Brava! components you want installed on this machine as well as other configurable values.

b. After updating the `bravaInstall.properties` file, run the `installBrava.sh` script as root, for example, `sudo ./installBrava.sh`.

c. Reply to the prompts to complete the installation.

Refer to the next sections "Installing Brava! Enterprise Components" on page 19 (for Windows installations) for instructions on installing the components that were not installed on Linux.

2.3 Installing Brava! Enterprise Components

The Brava! Server and Job Processor should be deployed on different servers. Multiple Brava! Servers and Job Processors can be used to provide scalability as needed.

Licensing note

The purchase of client license seats entitles you to a single Brava! Server, Job Processor, and License Server. Increasing the number of Brava! Servers, License Servers, or Job Processors in your environment involves an additional licensing fee that should be discussed with your Sales group Account Executive.

2.3.1 Distributed Install

The Distributed Install installs the Brava! Job Processor (multiple dedicated Job Processor servers are strongly recommended in a production environment), Brava! Server, Brava! SDK (which includes the Brava! Clients), and Brava! License.

The location of the components depends on the requirements for the Brava! Enterprise application. Any or all of the components can be installed individually. The installation should be completed on the physical machine where the application will run. The Job Processor and License Server can be installed on one or more machines. One License Server must be installed on at least one machine, but can be installed on more than one. To install on more than one machine, see "Configuring multiple Brava! License Servers" on page 99. For help in determining the best configuration for your environment, contact support.

Integration Considerations

If you have purchased more than one Brava! product or integration, installing these various products on the same server (or using one instance with multiple products or integrations) is not supported. Different Brava! integrations might rely on different versions of Brava! software, and those different versions must each run on their own server.

This restriction also applies to the Brava! ActiveX control that is used with various integrations. The integrations might include different versions of the Brava! ActiveX control and therefore both cannot be installed at the same time on a single client machine.

2.3.1.1 Select Installation Components

1. Double-click the installer file that you downloaded. The `Brava! Enterprise.msi/exe` file prepares the installation wizard.
2. On the **Welcome** page, click **Next**.
3. Read the OpenText End user License Agreement and select **I accept** if you have read, understand, and agree to the terms of the Licensing Agreement, and then click **Next**.
4. In the **Install Brava! Enterprise to:** page, use the **Change** button to select a custom location for the Brava! Enterprise files or click **Next** to accept the default location and continue.
5. In the **Custom** setup page, select which Brava! Enterprise components you would like to install on this machine. Install the components on different machines, depending on your scaling requirements. Multiple Job Processor machines increases scalability. The required Brava! License is typically installed on the Brava! Server machine.

   For details about installing the Brava! HTML Client files, see *OpenText Brava! - Brava! HTML Client Admin Guide (CLBRVW-AHD)*.
6. Brava! Server requires a servlet engine to be installed. If using Tomcat, it is configured to use the JVM you provide. To enable the default JVM, see “JMX” on page 33.

Licensing note

The purchase of client license seats entitles you to a single Brava! Server, Job Processor, and License Server. Increasing the number of Brava! Servers, License Servers, or Job Processors in your environment involves an additional licensing fee that should be discussed with your Sales group Account Executive.

2.3.2 Installing the Brava! Server components

1. Install the Brava! Server and optionally either the Brava! License or Brava! SDK on a separate machine than the Job Processors. To do this, from the **Brava! JobProcessor** list, select **This feature will not be available**. Components displaying an X are not installed.

   - At least one **License Server** must be installed, typically on the Brava! Server machine.
   - If you would like to install the Brava! SDK at this time, select **Brava! SDK** and select **This feature, and all subfeatures will be installed on local hard drive** from the list. The Brava! HTML Clients are installed as components to your Brava! Enterprise installation directory `...OpenText\Brava! Enterprise\BravaSDK\`
   - If you also want to install the Brava! ActiveX client, choose to install the **Include 2D ActiveX Client MSI package** option.
   - You can install the optional **Brava! Routing Service** on the Brava! Server or other machine. This service aids in routing requests to the same Brava! Server for a given session where the server has already cached the result. See "Installing the Brava! Routing Service" on page 22.

2. In the **Destination Folder** page, click **Change** to browse to the directory where you deploy Web Applications for your servlet engine. For example, if using Tomcat, this would typically be `C:\Program Files\Apache Software Foundation\Tomcat 10.1\webapps`. Click **Next**.

3. In the JAVAHOME page, click **Change** to browse to the directory of your Java installation. For example, if using OpenJDK 17, this would typically be `C:\Program Files\Java\jdk-17.<your version>\`. Click **Next>**.

4. In the **Brava! Server URL** page, the default servlet port 8080 is shown. This is the default port to use for Tomcat. If your application server is not Tomcat, or if you have configured Tomcat to use a non-default port, enter the application server's communication port number.

   Complete the three fields provided for the Brava! Server and click **Next**:

   URL Prefix: Enter either HTTP:// or HTTPS:// depending on whether the servers are configured to communicate using SSL.

   Hostname: Enter the fully qualified domain name of the server that is used for communicating with the Brava! Server.

   Port: Enter the port number that your application server is using (where Brava! Server is running/deployed to). See also "Job Processor Port configuration" on page 95.

   Example: Tomcat (Default): 8080

5. If you are installing the optional Routing Service (see "Installing the Brava! Routing Service" on page 22), in the **Brava! Routing Service URL** page, the default port 8081 is shown. If you have configured the Routing Service to use a

different port, enter that port number. Enter the fully qualified domain name for the server running the service. If installed on the Brava! Server machine, this value is the same as the **Hostname** used in the Brava! Server URL page.

6. In the **Brava! License Server URL** page, enter the fully qualified domain name for the host running the Brava! License service. The **hostname** shown by default for the Brava! License Server is the same machine entered for the machine hosting the Brava! Server webapp.

7. In the **SMTP Server** page, enter the server SMTP information for sending emails from Brava!. Configuration of this page is optional and you can leave the field blank and click **Next** if you do not want to receive email notifications for the Brava! License.

8. Click **Install** to begin the chosen file installations.

9. Click **Finish**. The installation of the Brava! Server components is complete.

2.3.3 Installing the Brava! Routing Service

Important

The Brava! Routing Service is a new component available in this release as an optional feature to allow customers to investigate and evaluate the benefits that it might provide to them. It will be refined in subsequent releases for broad adoption.

The Brava! Routing Service is an optional component that assists in deployments that need two or more Brava! Servers for scalability and fault tolerance. This component does not need to be installed if only one Brava! Server is set up in your environment.

See “Routing Service Appendix” on page 119 for details about deployments requiring the Brava! Routing Service and how it functions.

The Brava! Routing Service is not installed by default. If you want to install it, you must select the **Install Brava! Routing Service** component as described in “Installing the Brava! Server components” on page 21.

Once the decision is made to install this component, there are some important configuration edits that must be done to specific configuration files across the Brava! systems. Certain choices can be made in the installer dialog boxes that will simplify this process, and the remainder of this section describes the best choices to make in the installer, and the required configuration file edits to make as a result of those choices.

Note: If you want to install the Brava! SDK to test against this load balanced configuration, install the SDK component on one host machine that is separate from the Brava! Routing Service and Brava! Servers. After setup, some edits must be made to the SDK pages for the configuration to work.

Installation tasks

1. Choose whether you are doing either HTTP or HTTPS. It **must be the same** for both the Brava! Routing Service and the Brava! Servers. Setting the correct URL format (URL prefix and relevant port number) in the installer page simplifies this.
2. Define your load balancer FQDNs for the virtual pool for both the Brava! Routing Services and Brava! Servers and save these for usage later.
3. Perform the installation on each machine using the specific hostnames/FDQNs that you are installing against (not the load balancing URLs). The configuration files will be edited later to set the proper load balancer URLs.
4. Complete a run of the installer on each machine that is to be used for a Brava! Routing Service and a Brava! Server.
5. On each Brava! Routing Service host, perform the following steps:

   a. Located in the Brava! Enterprise installation folder (`C:\Program Files\OpenText\Brava! Enterprise\` by default), click the `RoutingService` folder.

   b. In this folder, edit `cluster.exe` by doing the following:

      - Change `localhost` to the actual FQDN of this host machine.
      - Uncomment the commented out `<Member>` line and add the FQDN of each of any other Brava! Routing Services that you've installed (this tells them how to share data about which Brava! Servers have which publication in their cache).

   c. Also in this folder, click and run `RoutingServicew.exe` and do the following:

      - On the **Java** tab, under **Java Options**, scroll down and find `-Dbravaservice.pool.host=` and change the hostname after "=" to be the FQDN you chose earlier for the load balancer virtual pool for Brava! Servers. This tells the Brava! Routing Service where to send new (not known to be cached) requests. This should be the load balancer address for the Brava! Servers so the service can pick a server according to its balancing rules.
      - On the **Startup** tab, under **Arguments**, scroll down and find `-cluster-host localhost`, and change `localhost` to the actual FQDN of the host machine.
      - Click **OK**.

   d. Restart the Brava! Routing Service Windows OS service on this host.

   e. Move on to the next Brava! Routing Service host and repeat these steps #a-e.
6. On each Brava! Server host, perform the following steps:

   a. Located in the Brava! Enterprise installation folder (`C:\Program Files\OpenText\Brava! Enterprise\` by default), click the `Brava! Server` folder.

b. In this folder, edit `server.properties` by doing the following:

- Find the lines for `routing.service.callback.host` and `routing.service.callback.port`. Their values must match the hostname and port used for the BravaServer webapp. When the Brava! Server receives a publication request, it uses these two properties to tell the Brava! Routing Service how to route subsequent requests for specific publications or compositions.
- Save the file.

c. Restart the Tomcat Services for Brava! Server.

d. Move on to the next Brava! Server host and repeat these steps.

7. If you've installed the Brava! SDK for development or evaluation purposes, on the Brava! SDK host, perform the following steps:

a. In the deployed BravaSDK `webapp` folder selected during the BravaSDK installation (such as `C:\Program Files\Apache Software Foundation\Tomcat 10.1\webapps\BravaSDK`), click the `common` folder.

b. In this folder, edit the `common.*` files by doing the following:

- Find `bravaServerURL` and modify the value to be the full URL for the load balancer virtual pool for the Routing Service. This URL must have the protocol, host, and port.

Note: The formatting/syntax of each `common.*` file will vary. Using the incorrect formatting prevents the BravaSDK from functioning correctly.

Be sure to modify both the `common.js` file and either the `common.jsp` or `common.aspx` file depending on whether you will be using the Java API or the .NET API.

c. Restart the service used for the BravaSDK.

2.3.4 Installing the Job Processors

1. Run the Brava! Enterprise installer on a separate machine than the one where Brava! Server is installed and continue from the **Select Components** page. See "Select Installation Components" on page 20.
2. Select **This feature will not be available** from all component options except **Brava! JobProcessor**. Components displaying an X are not installed.
3. The Job Processor service components are installed by default. The CSF Writer is a required component for use with Job Processor print publishing. If you don't want the Job Processor to be installed as services, you can expand the **+** symbol and select your installation preference. If not installed, the service needs to be installed and started manually or as a service before Brava! can function.

CSF Writer notes

- Before you install the CSF Writer feature, uninstall any previous versions of IGC Writer and restart the machine..
- The print spooler service needs to be enabled and running in order to install the printer driver. If the print spooler service is disabled, the installer cannot start it.

4. In the **Brava! Server URL** page, enter the fully qualified domain name and port number of the machine where the Brava! Server is or will be installed.
5. If you have elected to install the Service, enter the account user name (as DOMAIN\Username) and password in the **Service Logon Information** page. Click **Next**.
6. Click **Install** to begin the installation of the Job Processor components and service.
7. If not detected (or incorrect version detected), the installer automatically installs the CSF Writer, which is a required component of the Job Processor.
8. At the end of the installation, the installer starts the Brava! services. Click **Finish**. The installation of the Job Processor and components and services is complete.
9. Test your installation. See "Testing and verifying your installation" on page 37 to start Tomcat* (or another servlet container).

Note: The Apache Tomcat service sign in must be run as an Administrator account rather than as a local account.

Licensing

By default, the Brava! Enterprise product installs a 5-user 30 day evaluation key. Once you obtain permanent keys from us, you must copy them to the installation directories to replace the 30 day evaluation keys. These three keys are included in the following two files, which must be updated after installation:

- The Job Processor `IGCKey.lic` file (located in your installed `C:\Program Files\OpenText\Brava! Enterprise\JobProcessor` directory.)
- The Brava! Server and Brava! Client keys contained in the `IGCKey.lic` file (located in your `C:\Program Files\OpenText\Brava! Enterprise\Brava! License` directory.)

Important
The purchase of client license seats entitles you to a single Brava! Server, Job Processor, and License Server. Increasing the number of Brava! Servers, License Servers, or Job Processors in your environment involves an additional licensing fee that should be discussed with your Sales group Account Executive.

Related Topics

- "Setting up multiple Job Processors for a single Brava! Server" on page 96

2.4 Installation additions

This section contains additional information for completing the Brava! Enterprise component installations.

2.4.1 Permission to install the Brava! ActiveX Client control

The following activities happen automatically on the end user's machine when they access a web page referencing the Brava! ActiveX Client control:

- If the machine does not have the specified version of the control installed, it downloads the `BravaClientXWrapper.cab` referenced in the *`codebase`* attribute of the page's <*object*> tag.
- The cab is downloaded to the Windows system **Downloaded Program Files** sub-directory. For example `C:\Windows\Downloaded Program Files`, and the cab components are installed in the user's HOMEPATH (`%USERPROFILE%`) by default. For example `C:\Users\<Username>\OpenText\<version>\<GUID>\` directory. As part of this installation, the `BravaClientX.dll` and `BravaClientXWrapper.dll` are registered, updating registry keys within the `HKEY_CLASSES_ROOT` key.
- Internet Explorer loads the Brava! Client control into the web page.
- The Brava! Client control checks to see that the proper version (based on the *`IntegrationVersion`* parameter) of the Brava! integration DLL is installed. This DLL handles client communication with the Brava! Server. It looks for this DLL in the `IGC` sub-directory of the user's HOMEPATH. If it is not found there, the client control downloads the DLL from the URL specified in the *`IntegrationURL`* parameter to the local directory.
- The client then proceeds to communicate with the Brava! Server to download the specified document (set using the *`DocSource`* or *`DocID`* parameter).

Note: You must completely remove the Client control before you can download a new control. To remove the control, you must manually delete and unregister the control by doing the following:

1. Through Windows Explorer, delete the Brava! Client control (`BravaClientXWrapper.inf`) from the `C:\WINDOWS\Downloaded Program Files` folder. Optionally, you can remove the control from IE **Manage Add-ons** and restart IE. Although not required, you can delete `%USERPROFILE%\OpenText\` to remove it from disk because it is no longer needed.
2. To unregister the `BravaClientX.dll`, from a Windows **Start > Run** command, type `cmd` and press **ENTER**.

3. At the command prompt, type `cd` to change directory to the homepath directory where the DLL is located (for example, `C:\Users\<Username>\OpenText\<version>\<GUID>`). Then type `regsvr32 /u BravaClientX.dll` and press **ENTER**. Repeat using `regsvr32 /u BravaClientXWrapper.dll` and press **ENTER**.

4. After removal, if you load a page in Brava! again, the latest client is downloaded and registered.

2.4.1.1 Required permissions

The Brava! ActiveX Client is a signed ActiveX control. For an end user to install the client control, their Internet Explorer browser needs to be configured to accept signed ActiveX controls, and must be enabled to run ActiveX controls. Select the **Custom Level** option from the Internet Explorer **Options > Security** tab to view the various IE security settings. The end user must also have the appropriate permissions to download and install the control. By default, most versions of Windows come with the **Standard User** and **Restricted User** roles. A user assigned to a group with the **Standard User** permission set will be able to install the Brava! Client control. However, an end user with only the **Restricted User** role will not.

To see what type of an account a user has:

1. In Control Panel, click **Users and Passwords**.
2. Select the user account and click the **Properties** button.
3. In the **Properties** dialog box, select the **Group Membership** tab for the user group.

User Account permissions

A standard user account can run Brava! Server if MODIFY and READ/WRITE permissions are granted to the following directories within the Brava! installation directory:

```
C:\ProgramData\OpenText\dlcache

C:\Program Files\OpenText\Brava! Enterprise\Brava! Server\markup\

C:\Program Files\OpenText\Brava! Enterprise\Brava! Server\stamptemplates\

C:\Program Files\OpenText\Brava! Enterprise\Brava! Server\
```

Apply one of the following options to allow end users to install the Brava! Client ActiveX control.

- Set the end user's **Group Membership** to one that has the appropriate rights, for example the Power User's Group. This is set from the **Group Membership** panel described in the preceding paragraph.

- A user with sufficient rights can install the control on the restricted user's machine. This can be accomplished by downloading `ClientInstall.zip` which is located in the Brava! Client Install directory. By default this is `..\webapps\BravaSDK\ActiveX\viewer\client`) to the end user's machine, unzipping it to an empty directory, and then running `BravaClientInstall.bat`. Once executed, an end user who only has **Restricted User** rights will be able to execute the Brava! Client control. See the `README.txt` file in the unzipped directory for more information.
- `Brava! Enterprise <version> Client.msi`: On Windows networks, another alternative is to deploy an MSI package, which provides a mechanism for the Brava! ActiveX Client control to be installed to multiple end users with no administrator privileges on their machines. During setup, select to install the **Include 2D ActiveX Client MSI package** option. See http://support.microsoft.com/kb/816102 (https://support.microsoft.com/en-us/help/816102/how-to-use-group-policy-to-remotely-install-software-in-windows-server) for further information about MSI package deployment.

The MSI package for the Brava! ActiveX Client can be optionally installed to the `C:\Program Files\OpenText\Brava! Enterprise\Client MSI Installer` directory through the Brava! installer. To deploy the `Brava! Enterprise Client.msi`, administrators must push down the MSI to the desired users profiles using Active Directory GPO or scripts, LANDesk, for example.

2.4.2 Installing Brava! components as a service

Brava! Server

- The Brava! Server runs as a web application within a host servlet engine.

  Note: The Brava! Server can be run as either a service or a console.

- To run the Brava! Server component as a service, the hosting servlet engine should be set up to run as a service. If using Apache Tomcat as your servlet engine, you can start Tomcat as a service or console.

Brava! Job Processor

- The Brava! Job Processor runs as its own application and can be run as either a service or a console.
- During the install for the Job Processor (**Custom Setup** page), you can choose to install the component as a service. If you decide to install the Job Processor as a service later, you must install it two more times. The first time to uninstall the component and the second time to reinstall the component as a service.

To manually start the Job Processor service:

1. In Control Panel, click **Services**.
2. From the **Service** list, select the Job Processor Service.

3. Right click **Job Processor Service** and click **Properties**. Click the **Log On** tab.
4. Click **This account** and enter a network user account with Administrator privileges that has write access to the Brava! Server displaylistcache share directory. Be sure to enter the user account domain. For example, `<domain>\<user>`. Enter the **Password** and **Confirm Password** fields and click **OK**.
5. The install sets the Job Processor Service to automatic and launches each time the machine boots up. If you switch the Job Processor Service to **manual** instead of **automatic**, you can start the service by highlighting the Job Processor Service in your **Services** list and clicking **Start**.

2.4.2.1 Job Processor security

The as-installed Job Processor service must be configured to run using an account that belongs to the local machine's Administrator user group. This is because the Job Processor service requires elevated privileges to provide print publishing features, which generally provide the best available fidelity to original source documents. If your organization prefers not to run services using accounts with elevated system privileges, and can accept a reduced level of fidelity in published documents, it is possible to configure the Job Processor to run using a service executable that does not require Administrator privileges.

To run the Job Processor service using an account without local system Administrator privileges:

1. Stop the Job Processor service.
2. Browse to the installed directory for the Job Processor, and locate the `jpservice.exe`, `jpservice-nonadmin.exe`, and `jpservice-admin.exe` files.
3. Delete `jpservice.exe`.
4. Copy and paste `jpservice-nonadmin.exe` into the same directory.
5. Rename the copied file from `jpservice-nonadmin - Copy.exe` to `jpservice.exe`.
6. Change the account used to run the Job Processor service to the desired non-Administrator account.
7. Ensure that all file shares accessed by the Job Processor have read and write access for the new non-Administrator account.
8. In a Command Prompt window (clicked to run as Administrator), run the following commands. Change the name to the actual Windows user account you are using. For example, `.\MyLocalNonAdmSvcAcct`:

   ```
   netsh http add urlacl url= http://*:7070/ user=<domain|local\user>

   netsh http add urlacl url= http://*:8080/ user=<domain|local\user>
   ```

   Note: If Brava! Server is installed on a separate machine from one or more Job Processors, each machine requires these commands to be executed for the user account specified.

9. Go to the `Job Processor\Igc.Loaders` folder and run the `loaders.configuration.exe` utility.
10. Expand the **OutsideIn2dl** loader and expand the **Associated Extensions**. Click each extension that is a Microsoft Office format extension and change the loader to **OTF2DL**.
11. When complete, click **Done**, then click **Yes** to close the dialog box and save all changes.
12. Start the Job Processor service.

2.4.3 Installing Job Processor native applications

The Job Processor publishes files in one of several ways. If the file being converted is one of the natively supported files, the Job Processor performs the conversion internally and requires no other application. If the file being converted is not one of the natively supported formats, an application must be registered on the Job Processor to perform a **PrintTo** action for that file type.

Brava! Enterprise does not require a separate (native) application to publish the file formats listed in the Supported Formats (http://www.opentext.com/file_source/OpenText/en_US/PDF/opentext-so-brava-enterprise-supported-formats-en.pdf) page, excluding those listed in the “Document/Additional Image Formats” section.

To publish other formats not on this list, or are listed in the “Document/Additional Image Formats” section and you are not licensed for the alternative technology option, the native applications are required to be installed on or accessible by the Job Processor machine. The Job Processor machine must have a **Print** command available in the native application for that extension. Install the native application (such as Word, Excel, or any other application that uses a **Print** command) on the Job Processor machine that will handle publishing jobs for that format.

It is necessary to install the applications directly on the Job Processor machine. Most applications register a **Print** (or **PrintTo**) command with the operating system for their file formats. The applications do not have to be running; the Job Processor automatically launches the application to perform the conversion and shuts it down when the job is complete. If needed, the application must have the initial setup done after installation so it opens the desired file type without error (Outlook MSG files, for example).

Microsoft Office requirement:

A licensed and activated version of Microsoft Office is required to be installed on the Job Processor machine prior to running the Job Processor installation to publish Office formats (see “Application Extension References” on page 31). Please refer to “Job Processor” on page 15 for additional information and to the product release notes for supported Office versions.

After Microsoft Office is installed, you must initially launch any of its applications that you intend to use for publishing, such as Microsoft Word. This is necessary to dismiss any first-launch dialog boxes or configuration pages and should be done using the sign in for the account configured to run the Job Processor service.

In addition, you should adjust any Trust Center settings (**File > Options > Trust Center > Trust Center Settings > Macro Settings**) to match your business rules, ensuring that you choose selections containing **Disable all macros without notification** as applicable. Without proper configuration, macro-based documents might not render.

If you plan to publish older Office formats, publishing might fail if the file block protection settings are enabled. This setting can be altered through **File > Options > Trust Center > Trust Center Settings > File Block Settings**. Any file checked as **Open** will fail.

Microsoft Outlook requirement:

To properly render MSG and EML files, a 64-bit version of Microsoft Outlook must be installed and initialized to the Job Processor prior to installing Brava! Enterprise.

Note: If no email account will be set up with your Outlook installation, Outlook must be configured to run without an email account to enable processing of MSG and EML files.

The following table provides a reference example for common file types and the native application they require to be installed on the Job Processor to use Print-publishing.

Table 2-1: Application Extension References

Application	File Extension
Microsoft Word	DOC, DOCX, DOCM, DOT, DOTX, DOCB, DOTM
Microsoft Excel	XLS, XLSX, XLT, XLSB, XLA, XLAM, XLL, XLW, CSV
Microsoft PowerPoint	PPT, POT, PPS, PPTX, POTM, POTX, PPAM, PPSX, PPSM, SLDX, SLDM, PP7
Microsoft Outlook	MSG, EML, OFT
Microsoft Access	ACCDB, ADB, ACCDE, ACCDA
Microsoft Publisher	PUB
Microsoft InfoPath	XSF
Microsoft Visio 2016	VSD, VDX, VST, VSX, VTX

2.4.4 Brava! Enterprise Monitoring

The following technologies are available for monitoring Brava! Enterprise.

2.4.4.1 Monitoring Tool

Note: This feature has not completed testing and is in an experimental phase.

The Brava! Enterprise Monitoring Tool is included as part of the BravaSDK web application, available from `<webapps>/BravaSDK/`. To access the tool, click the **Monitoring** button from the top of the Brava! Enterprise **Samples** page, enter the Brava URL to monitor, then press **ENTER**. The item is added to the list of services to monitor, available from the lower left panel of the tool.

To use this tool, the spring profile in `server.properties` must have the `admin` endpoints enabled (see *`spring.profiles.active`*), which should be enabled by default. You can use the **Token generator** button to generate an authentication token if needed. Click the **Instructions** button of the **Token Generator** page to view additional details. When a token key is generated, it can be copied and pasted into the **Authentication Token** field of the Monitoring tool.

The Monitoring Tool provides the current service information for the selected item. Use the various buttons in the left panel to view the current status of each subject.

Publishing
Displays the status of the publishing jobs that are currently in the various queues.

Display List Cache
Displays the status of the display list cache. This page provides options for you to search entries and clear the cache.

Licensing
Displays the current licensing status. This page provides details of the server and client licenses, including permissions enabled for each client license.

Errors
Displays a cached list of recent job error details. This page provides options for you to clear the error log and refresh the status.

Settings
Displays the current values that are set in `server.properties`.

2.4.4.2 JMX

The JVM within Brava! Enterprise servlet container can be enabled with Java Management Extensions (JMX). This lets you see the health of the JVM as well as a few BravaServer statistics.

To enable JMX in Tomcat:

1. Open the **Tomcat Service Configuration** application. On Windows, this application is located under Tomcat's `bin` directory.
2. Open the **Java** tab.
3. Add the following options, and then click **OK**:

   ```
   -Dcom.sun.management.jmxremote
   ```

   ```
   -Dcom.sun.management.jmxremote.authenticate=false
   ```

   ```
   -Dcom.sun.management.jmxremote.port=6666
   ```

   ```
   -Dcom.sun.management.jmxremote.ssl=false
   ```

4. Restart Tomcat.

If your JMX monitoring console is connecting from a remote machine, make sure that the port above (6666) is not blocked by a firewall.

Note: These options are global for the entire JVM. The reported data includes all web applications that are running within Tomcat that is running on the machine.

Viewing JMX Monitoring Data

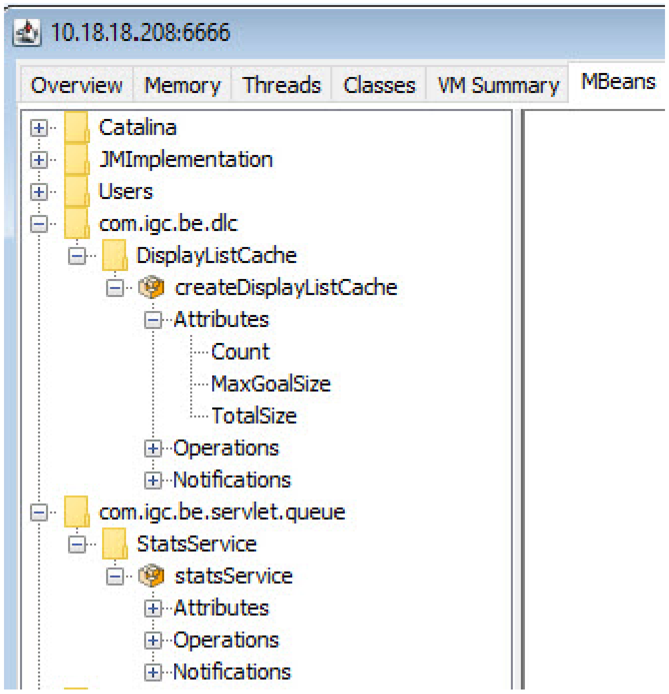
There are multiple JMX applications available for viewing JMX data. Applications that are included with the Java JDK include JConsole, VisualVM, and Java Mission Control. These can be found in the Java `/bin` directory.

JConsole

On Windows, `jconsole.exe` can be found within the JDK `/bin` folder. On Linux, the name of the exe is `jconsole`. When launching JConsole, you'll need to connect to the appropriate process. To do this, under the **Remote Process** section of the **Java Monitoring & Management Console > New Connection** dialog box, enter the server name and port specified in the Java options that were added to Tomcat in the JMX setup. For example, `localhost:6666`.

Once the connection is established, many JVM statistics become visible that can be useful when investigating application behavior. To see more specific statistics, connect to MBean server by clicking on the **MBeans** tab.

On the left panel, you can observe the MBeans that are currently registered. There are generic MBeans available on basically all JVMs, allowing you to perform tasks such as creating thread dumps and checking thread statistics. The **Catalina** section contains information about Tomcat. Brava! Enterprise specific statistics can be found under **com.igc.be**.



Currently, the only statistics provided by the Brava! Server are the number of entries in the dsplaylist cache and its size, as well as the number of entries for the various Job Processor job queues. JConsole only provides a static view of these statistics.

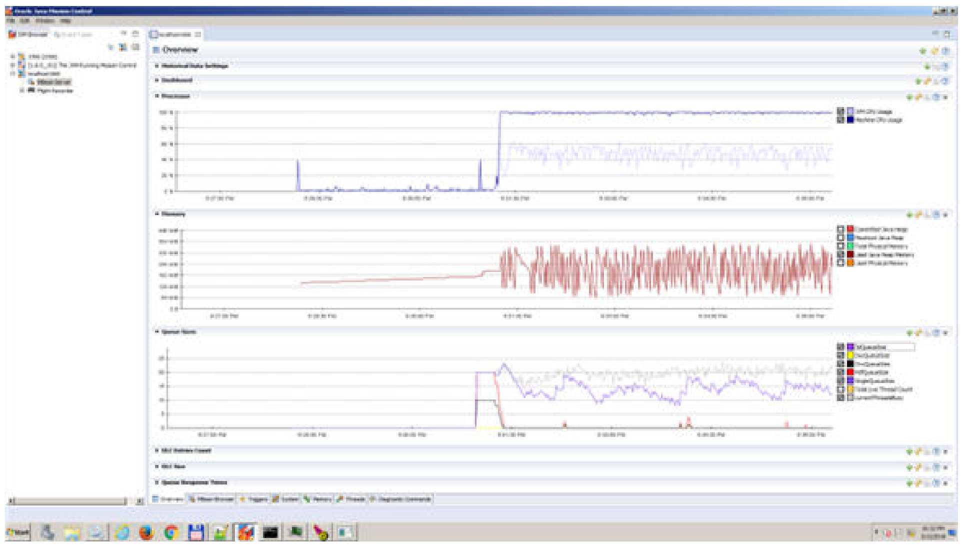
Java Mission Control

Java Mission Control provides a graphical view of JMX statistics over time. The name of this tool is `jmc` (or `jmc.exe` on Windows), and it is available in the JDK `/bin` folder.

To use the jmc tool:

1. Define a connection by selecting **File > Connect**, and then enter the host machine name and port configured for JMX.
2. Once the connection is established, available options for viewing statistics include the MBean Server and the Flight Recorder. Flight Recorder is a profiling/troubleshooting tool that is not covered here.
3. In the **JVM Browser** panel (left side), double click **MBean Server**. By default, JMC displays the JVM’s **Processor** and **Memory** graphs. Custom graphs can be added as described in the following example.

To produce a display such as shown in the example image below, perform the following steps:



1. Click the **Add** button on the top right corner of the window.
2. A new graph should appear on the bottom. Right click the graph and select **Edit titles > Graphic title** to change its name to “Queue Sizes”.
3. Click the **Add** button on the right side of the chart to add new MBeans that you want to visualize.
4. Filter by **StatsService** to retrieve your statistics.
5. Select one of the bean attributes (for example, `PdfQueueSize`) and define Type and Units if necessary.
6. By right clicking on the graph, you can set the time range to view, such as the last 10 minutes. It’s beneficial to set each of the graph time ranges to be the same for easy comparison.

2.4.5 Localizing Brava! Enterprise

Brava! Enterprise is installed with the following seven language translations. The Brava! ActiveX Client can be updated to use any of the following non-English languages:

Language	Three Letter Acronym	Country
English	ENU	USA
Simplified Chinese	CHS	Peoples Republic of China (PRC)
Spanish	ESN	Spain

Language	Three Letter Acronym	Country
German	DEU	Germany
French	FRA	France
Japanese	JPN	Japan
Korean	KOR	Korea

Two `bins` for each available language are installed at the location of the `BravaClientXWrapper.cab` on the web server (`\BravaSDK\ActiveX\viewer\client`). The bins for the English language are: `BravaClientXResource_ENU.bin` and `generic_ENU.bin`. One bin contains the Brava! Client resource DLL , while the other contains the Brava! Integration DLL.

When the Brava! Client is launched, the language that is currently set on the operating system is used to determine which set of language bins to download to the client machine. For example, if the operating system has Korean set, then the `BravaClientXResource_KOR.bin` and `generic_KOR.bin` are downloaded and used for running the Brava! Client.

Note: The required *`IntegrationUrl`* BravaXParam should not include the "Three Letter Acronym". This is formulated automatically by the Brava! Client. Please see "Brava! ActiveX required control parameters" on page 59 for details.

2.4.6 Uninstalling and repairing Brava! Enterprise components

At some point, you might want to uninstall or repair the Brava! Enterprise installation.

To uninstall Brava! Enterprise:

1. A reboot of the machine is required before uninstalling or upgrading your Brava! version. This is necessary to unlock the BlackIce printer driver that is used with the CSF Writer.

2. After the reboot completes, from Control Panel, click **Add or Remove Programs** to select and uninstall **Brava! Enterprise**.

To repair a Brava! Enterprise installation:

1. The command line examples below restore any files missing from your installation.

   Example 2-1: MSI repair example

   ```
   msiexec /fp "path\to\msi" /qb
   ```

Example 2-2: EXE repair example

```
path\to\exe /s /f /v"REINSTALL=ALL REINSTALLMODE=p /qb"
```

2. For more repair options, please see the following official Installshield and Windows Installer command-line documentation:

   - http://helpnet.flexerasoftware.com/isxhelp19/helplibrary/IHelpSetup_EXECmdLine.htm#Ref-Command-LineTools_2276000595_WP0586fParam
   - https://docs.microsoft.com/en-us/windows/desktop/msi/command-line-options

2.5 Testing and verifying your installation

After installing the Brava! components in your environment, refer to this section verify that your installation is set up and functioning correctly.

2.5.1 Brava! Server installation

There are two parts to the Brava! Server installation.

The first part is the Brava! Server web application that is deployed to the target servlet engine web application directory (for example `C:\Program Files\Apache Software Foundation\Tomcat 10.1\webapps`).

The second part is a directory structure that contains configuration files and directory holders for log files and published files that are cached. The default installation directory is `C:\Program Files\OpenText\Brava! Enterprise`. Make sure that the web application was deployed as specified during the installation process and that the Brava! Server Install directory was created in the location specified during installation.

Important

If you are updating your version of Brava!, you must delete your previously deployed folders (backing up whatever customizations were made) and then deploy the new web application files. You can then refer to your backed up files to restore any changes made. Use caution when updating with old customizations to ensure the new files have not changed. Please contact Support if you are unsure whether your backed up customizations can be ported forward.

2.5.2 Job Processor installation

You can have multiple Job Processors in use with Brava! Enterprise. Complete the following setup on the machines where you want the Job Processors to reside. These steps must be performed for each Job Processor machine.

- Install Job Processor
- Verify Launch

To install the Job Processor

1. Run the Job Processor installation on the machines where you want the Job Processors to reside.

   Important
   The Job Processor requires the user to be signed in as an Administrator.

2. From the component installation page of your Brava! Enterprise installation program, select the **Brava! Job Processor** component and follow the on-screen prompts. By default, the Job Processor installs to the following location, which is your Job Processor directory: `C:\Program Files\OpenText\Brava! Enterprise\JobProcessor`

The **OpenText** program group should now appear from **Programs > Start > All Programs > OpenText**. It contains shortcuts for launching the Job Processor, Sample pages, and the Brava! SDK.

Related Topics

- "Setting up multiple Job Processors for a single Brava! Server" on page 96

2.5.3 Job Processor verification

To verify the Job Processor installation

1. **Start your Servlet Container:** Make sure that your servlet engine (Tomcat for example) is up and running. Tomcat can be started up as either a service or a console.

   Note: This step is not needed if Tomcat is installed to run as a service. Instead, go to your Services Panel to verify that Tomcat has started.

   If Tomcat does not start, this is most often due to a port conflict. To resolve this issue, you can configure Tomcat to use a different port, or stop the process that is consuming the port and retry.

2. **Verify that the Job Processor is running:** The Job Processor should be installed as a service. On each Job Processor machine, go to your **Services** Panel to verify that the Job Processor has started.

3. **To manually start the Job Processors using console:**

a. `jpservice.exe` is for the Job Processor service and `jpconsole.exe` is for the Job Processor console.

Note: The service must be stopped before starting console.

b. To start using console, on each Job Processor machine, go to the Windows **Start** menu and select **All Programs > OpenText > Start Brava! Job Processor**.

Note: Not needed if the Job Processor is installed as a service. Instead, go to your **Services** Panel to verify that the Job Processor has started.

c. A Command Prompt window appears. If the launch is successful, you will see a script ending with "Controller Started".

4. **Verify web application deployment:** The Job Processor can display status information and can be used as a monitor tool. See "Monitoring the Job Processor" on page 98.

   On each Job Processor machine, open a web browser and go to `http://<hostname>:7070/config`, where <*hostname*> is replaced by the IP address or name of your servlet engine machine. You should see the **Job Processor Config** page in your browser window.

2.5.4 Brava! Client verification

1. When prompted for an install directory, the path to the document root directory of your web server should have been entered. The installation adds a BravaSDK web application which contains the different viewers, configuration files, and integration samples.

2. Set up your method of creating HTML pages for *each* file loaded. As an alternative to setting up HTML pages, a JSP or ASP page can be established. The JSP or ASP must be placed in a web server that serves those file types such as Apache Tomcat for a JSP page and IIS for an ASP page. Sample HTML, JSP, and ASP pages are installed with the Brava! Client SDK and contain some possible parameters. See "Sample web pages" on page 53 for more information.

3. After you have verified the loading of the server components, you can verify the client by opening one of the client sample web pages (`simple.html`, `overview.html`). If the control doesn't load, the Brava! Client control was not installed correctly (see "Permission to install the Brava! ActiveX Client control" on page 26 for more information). If the control does load, but the document does not load, examine the error message details. For further information to assist in troubleshooting, set the *`ClientLogging`* and *`ClientLogFile`* parameters. (See "Optional ActiveX Client parameters" on page 62 for more details).

4. Finally, verify that the outbound port that is configured for the Brava! Server URLs is not restricted by your end users network administrator. A good troubleshooting technique is to verify that all of the URLs listed in the HTML page, and in the Brava! Server `server.properties` and `client_precedence.properties` files are correct. (This can be verified by entering the URLs into the address field of your browser. You should receive either a prompt to download a file or a blank page, but not an error message).

2.5.5 Brava! Enterprise verification

1. Start the servlet engine where the Brava! Server and Brava! License web applications have been deployed.
2. Start the Job Processor application if not already started.
3. From within your browser, launch the `overview.html` page that was installed in to your client installation directory. For example, `http://localhost:8080/BravaSDK/ActiveX/samples/BasicFileLoad/overview.html`.
4. Click the link of a sample document to load and you should see that document successfully loaded into your browser.

If you experience any problems during your testing and verification, you can gather more information by doing the following:

- In the Job Processor admin web console accessed with the URL `http://localhost:7070/config`, update the *`log`* parameter to have a value of ON.
- The Brava! Server logs information. Please see “Brava! Server logging” on page 81 for details.
- The Brava! Client sets the parameters *`clientlogfile`* and *`ClientLogging`*. *`ClientLogging`* should be set to 0 or 1 to receive as much data as possible. See “Optional ActiveX Client parameters” on page 62 for additional details.

Chapter 3

Security considerations

In Brava! Enterprise, the Server is composed of a set of servlets that respond to HTTP URL requests with HTTP responses. This section discusses some of the ways to protect the information served from the Brava! Server. It also discusses protecting the machines running the Brava! components.

3.1 Protecting transmitted information

When opening a native file using the ActiveX Client, Brava! Enterprise does not send the original native file to the client unless *`ConvertOnClient`* is turned on for client-side publishing.

The Brava! Server sends our proprietary published format to the client, requiring the Brava! Client or another Brava! viewing product to view the image stream. If you require all the information transmitted between the client and the server to be protected, use a Web Server that supports SSL (secure socket layer) and HTTPS. To further protect content, native files can be pre-published to Content Sealed Format (CSF) using our client-side publishing products. CSF allows various rights to be established on the published file, such as the ability to print the file or copy text. It also allows you to set a file access password and an expiration date.

When using the Brava! HTML Client, document renditions are sent to the viewer as SVG or JPEG files. While SSL can be used to encrypt the files in transit, these files can stay in the browser’s cache in an unencrypted state. If this is a concern, using the ActiveX Client is recommended.

Alternatively, the *`browser.cache.expiration`* property, set in your `server.properties` configuration, can be set to 0 to prevent storage of downloaded artifacts in the browser’s cache.

3.2 Preventing unauthorized access to Brava! Services

This section discusses various methods for preventing unauthorized access to the Brava! Services.

The typical flow for viewing a document in Brava! Enterprise is that a user navigating a web-based content management system click a custom link that invokes an *integration* script or service. This integration service runs within the content management system's process space and is able to access the user’s credentials and determine the access control rights to the content management system's resources. The integration service determines what rights (view, annotate, publish) the current user has to the requested file. If no view rights exist, the script returns immediately with an error. Otherwise, the integration invokes various

Brava! Enterprise SDK API methods to initiate publication of the document by the Brava! Server. After the integration has initiated publication with the Brava! Server, it returns the configuration necessary for launching the viewer. After the viewer has initialized, it makes various calls to the Brava! Server to retrieve resources for displaying the requested file. The following sections describe ways that the Brava! Server end points are protected from inappropriate access.

3.2.1 Client roles

There are four general roles that can be distinguished for clients connecting to Brava! Server.

Client
Role for end-users, allowing the client to perform limited operations on the server as determined by the integration.

Integration
Role is responsible for providing content to Brava! Server, configuring viewers, and assigning rights.

Administrator
A privileged role capable of monitoring Brava! Server and performing administrative tasks such as deleting cache entries.

Job Processor
Role is responsible for retrieving jobs from the queue and returning job complete notifications back to the server. This is a privileged role and end-users should not be able to act as a Job Processor.

Historically, these roles were distinguished on the server side using IP address filtering. Using that approach might not be sufficient to correctly secure access to the resources stored on Brava! Server and, for example, blocking users from viewing content from other users was not easily achieved.

3.2.2 Token verification

If the Brava! Server is configured to do authorization validation using its `server.properties` file's *`auth.validation`* property, then it only processes integration service requests if the JWT is signed with the correct key (one that is known only to Brava! Server and administrators that are allowed to generate integration, job processor, and administrator tokens), hasn't expired, and has the appropriate rights. See “Establishing JSON Web Token (JWT) validation” on page 47 for more information about how this is set up.

Auth validation applies not only to integration and end-user endpoints but also to job processor and administration endpoints. Each endpoint might require a different type of token, which are always passed as BravaAuth HTTP headers.

Each token has expiration times that can be set explicitly. The expiration time for Client tokens are set by the server and are valid for 8 hours. Expiration times for Integration, Administrator, and Job Processor tokens should be set explicitly.

Note: Changing JWT key immediately invalidates all the previously generated tokens.

3.2.3 BravaSDK application

The BravaSDK application, which often is installed alongside BravaServer application on a servlet container, has several purposes. It contains artifacts of viewers and Brava SDK libraries, and it also contains multiple sample integrations with Brava! Server. It is very important to understand that the BravaSDK is not secure. The samples contain code that gives easy access to internal resources of the hosting server. It has been developed that way for simplicity and ease of use and modification.

Important
The BravaSDK application should never be hosted on a production environment and be accessible from untrusted addresses.

3.2.4 Services accessible to the integration service

The integration service communicates with the Brava! Server, using the Brava! SDK API, to upload files for conversion, define the characteristics of the publication (geometry data or annotations to burn in, for example), and sign the viewer configuration (in order to prevent it from being modified inappropriately outside of the integration service). There are two main ways these services are protected:

Token verification

Your integration should be configured with the proper integration token to be authenticated by Brava! Server as a valid integration. The token is passed to the BravaSDK using the *`auth.token`* property of the `BravaConnection` object.

Generating the integration token

1. From the directory where the BE API jar file and dependencies are located, in a CMD window run the following command: (Note that at least Java version 8 must be installed.)
   ```
   java -cp "./*" com.igc.be.auth.TokenGen --key <jwtkey> integration
   ```
2. Use the following command to obtain the full list of options:
   ```
   java -cp "./*" com.igc.be.auth.TokenGen
   ```
3. Alternatively, the token can be generated using the **tokengen** page from BravaSDK.
4. The type of token must be `integration`.
5. The generated token must then be passed to the *`BravaConnection > authToken`* property using the *`setAuthToken`* method.

IP Filter

For these services, the Brava! Server only responds to requests that come from machines whose IPs are defined in the `server.properties` file's *`legal.ip.to.get.sessionid`* property.

Note: The Brava! Server obtains hostname and IP information for carrying out some of its operations. It is possible for an attacker to spoof DNS entries. To reduce this risk, the DNS server should be trusted by or run within the firewalled environment that the Brava! Enterprise server components are running in.

3.2.5 Services accessible to the Job Processor

Token verification

When auth validation is enabled in Brava! Server, HTTP request made by the Job Processor must contain a verification token. As in all other cases, the token is provided as a BravaAuth HTTP header. The token first must be generated, which can be done by using be-common jar as an executable.

Generating the jobprocessor token

1. From the directory where the BE API jar file and dependencies are located, in a CMD window run the following command: (Note that at least Java version 8 must be installed.)
```
java -cp "./*" com.igc.be.auth.TokenGen --key <jwtkey> jobprocessor
```
2. Use the following command to obtain the full list of options:
```
java -cp "./*" com.igc.be.auth.TokenGen
```
3. Alternatively, the token can be generated using the **tokengen** page from BravaSDK.
4. The type of token must be `jobprocessor`.
5. The generated token must then be passed to the Job Processor configuration file:
   a. Windows:

      Add the following section to the `JobProcessor.config` file, replacing `index` and `token` with the proper values:
      ```
      queue.url.<index>.header.BravaAuth=<token>
      ```
   b. Linux:

      Add the following section to the `config/publisher.properties` file in the `Publisher Agent` directory:
      ```
      services.opentext-jobqueue-http.endpoints.<index>.headers.BravaAuth=<token>
      ```

IP Filter

Similar to the integration service, it is also possible to set up IP filtering for the endpoints called by the Job Processor. For these services, the Brava! Server only responds to requests that come from machines whose IPs are defined in the *`legal.ip.for.jp`* property of the `server.properties` file.

3.2.6 Services available to the Monitoring Tool

The Monitoring Tool requires access to administration endpoints on Brava! Server.

Token verification

Generating the administrator token

1. From the directory where the BE API jar file and dependencies are located, in a CMD window run the following command: (Note that at least Java version 8 must be installed.)
   ```
   java -cp "./*" com.igc.be.auth.TokenGen --key <jwtkey> administrator
   ```
2. Use the following command to obtain the full list of options:
   ```
   java -cp "./*" com.igc.be.auth.TokenGen
   ```
3. Alternatively, the token can be generated using the **tokengen** page from BravaSDK.
4. The type of token must be `administrator`.
5. The generated token must then be entered in the Monitoring Tool application after clicking the **Authenticate** button. The administrator token can also be passed directly to admin endpoints supported by Brava! Server as a BravaAuth HTTP token.

IP Filter

Admin endpoints use the same rules for IP filtering as the integration endpoints.

3.2.7 Services accessible to the Brava! clients

ActiveX Client

Brava! Server has a session manager which maintains a unique session for each document request.

Currently, the ActiveX Client is not using JSON web tokens for authentication. Session ID is a secret value. Access to session ID gives users access to all content associated with the document request and sharing the session ID means sharing access to that content. Each session has a limited expiration time.

The following steps describe how the Brava! Server and Client use session IDs to help prevent unauthorized access to server resources.

- The client makes a call to an URL, which generates a web page, such as JSP or ASPX. The client passes the *`DocID`* to be published as a parameter.
- The JSP/ASPX page makes a call to the Brava! Server to obtain a valid session ID for a specified length of time. The Brava! Server checks to make sure the request for the session ID comes from a machine with the IP set in the *`legal.ip.to.get.sessionid`* property in the Brava! Server `server.properties` file. The JSP/ASPX then returns an HTML page that loads the Brava! Client passing it to the *`sessionID`* as a parameter. See the `view.jsp` file in the `.../BravaSDK/ActiveX/Samples/BasicFileLoad` directory for an example of this.
- The Client browser loads the returned HTML page, which in turn loads the client viewer. The viewer communicates a request to the server to return the published document. It passes all parameters specified in the HTML page as part of the `ServerProperties` request and session ID and *`DocID`* parameters to the others.

Note: It is not possible to bind *`sessionid`* with a document when a `ServerProperties` request is made. Proper configuration of ActiveX requires that all of the parameters are provided to the server, including the *`docid`* information.

HTML based viewers

Token Verification

The HTML based viewers can be configured to pass a JWT to `BravaServer` services it calls. If the Brava! Server is configured to do authorization validation, then it only accepts viewer requests if the JWT is signed with the correct key (one that is shared only with the integration), hasn't expired, and has the appropriate authorization rights. The JWT is set as part of the viewer's initial configuration. See the following section, "Establishing JSON Web Token (JWT) validation" on page 47, for details on how this is initialized.

Restricting Cross-Origin Resource Sharing (CORS)

A common deployment architecture has the integration service running on one machine and the Brava! Server on another. Typically, the HTML page that launches the Brava! HTML Client and the call to retrieve the viewer's signed configuration are retrieved from the integration service; after viewer initialization, the Brava! HTML based viewers make calls to the Brava! Server. This results in a cross-origin resource sharing (CORS) situation, where the Brava! Server needs to accept requests from a client whose origin server is different than itself. By default, Brava! Server accepts requests from any other origin server. This access can be constrained by restricting the allowable origin servers within the Brava! Server `server.properties` file to the integration service's base URL.

The modification should be made to the *`cors.hosts`* property as follows:

```
cors.hosts=*
```

The *`cors.hosts`* property accepts a comma-separated list of the hosts that permit CORS requests.

Example: `cors.hosts=integrationhost1,integrationhost2`

Each host for the *`cors.hosts`* takes the form: `http://<IntegrationServer>:<IntegrationPort>`. This value is typically the base URL portion of the HTML viewer's initialization path property.

Note: It's possible to configure CORS handling at the servlet container level. For instance, a *`CorsFilter`* can be added as a filter in Tomcat's `conf/web.xml` file. If this is done, duplicate Access-Control headers are added to the responses returned from the BravaServer. Since browsers don't accept multiple Access-Control-Allow-Origin headers, it is necessary to remove the CORS configuration either at the servlet container level, or in the BravaServer webapp.

This configuration can be removed from the BravaServer webapp by uncommenting the `cors.disabled=true` property within BravaServer's `server.properties` file.

3.2.8 Establishing JSON Web Token (JWT) validation

For an integration to successfully have requests authorized, it needs to set the auth token property on the `BravaConnection` SDK object it instantiates. This auth token needs to be generated from the same JWT key that is set in the `server.properties` file. The type of the token should be "Integration".

Types of auth tokens:

client	Generated by the server during configuration signing and allows access to published artifacts and publishing of PDF/TIFF files from compositions.
integration	Allows uploading documents and signing configuration.
administrator	Allows access to admin endpoints and use of the Brava! "Monitoring Tool" on page 32.
jobprocessor	Allows popping jobs from the Brava! Server queue and sending "done" notifications from the Job Processor to Brava! Server.

3.2.9 Getting the SDK sample to work with authorization

1. Decide on the JWT key (64 character minimum).
2. Set the *`jwt.key`* property in the Brava! Server `server.properties` file to this value. If you're using an encrypted value, place it between the parenthesis in the `ENC()` string (a sample is provided in the `server.properties` file).
3. Generate an integration auth token using `http://<BravaServer>/BravaSDK/tokengen/` or from the command line using the Brava! Enterprise API jar.
4. In the Brava! Server `server.properties` file, set the value of the *`auth.validation`* property to `on` and restart the Brava! Server web application for the changes to take effect.

With these changes completed, the `/BravaSDK/HTML/samples/formats/formats.html` sample displays properly, though other samples fail until their corresponding integration script has their `BravaConnection` `setJwtKey` or `setEncryptedJwtKey` call uncommented.

3.2.10 Authorization rights set in JWT

The JWT sent to the BravaServer has one or more authorization rights set. For requests coming from the viewer, these rights default to VIEW and RASTER. To have alternative rights set in the JWT, the appropriate grant method needs to be invoked on the viewer's config SDK object. For instance, to allow a Brava! HTML Client viewer to have full access to Brava! Server services (rasters, publishing/printing), the integration should invoke `grantFullAuthorization()` on the instantiated `HtmlConfig` object. See the SDK HTML `GetFormatsConfig.jsp` script for an example, and refer to the Brava! SDK javadocs for other levels of grant authorization that can be established.

JWT authorization is used to protect endpoints that reside on the Brava! Server machine. Because Brava! Server does not provide endpoints for annotations and stamp templates (an integration's role) these features are not controlled using JWT in any way. Additionally, JWT authorization does not affect the viewer user interface. Any feature granted or revoked by the integration using different `grant*Authorization()` methods does not cause any UI buttons or menus to be shown or hidden.

3.3 Protecting the Brava! Server machine

It is important to restrict access to the Brava! Server since it contains published files, some of which might contain confidential information. Putting the Brava! Server machine behind a firewall reduces the chance of unauthorized access occurrences. If the Brava! Clients need access to the Brava! Server over the internet, one safe configuration option is to route all Brava! Server requests through an internet-accessible Web Server. The Brava! Server machine then sits behind a firewall and is configured to only receive HTTP requests from the Web Server and the Job Processor machines.   For further information about routing requests to the Tomcat servlet engine through a Web server, see Apache Tomcat 10 (10.1.10) - Connectors How To (https://tomcat.apache.org/tomcat-10.1-doc/connectors.html).

3.4 Protecting the Brava! License web application

The Brava! License web application (webapp) runs within a servlet container such as Tomcat. While the Brava! Server webapp needs to be accessible to all Brava! client machines, the Brava! License webapp only needs to be accessible to the Brava! Server machine, therefore, access should be restricted accordingly.

We suggest achieving this restricted access by configuring the `BravaLicense` servlet with a servlet filter. For instance, if running Apache Tomcat, the following filter, added to the `BravaLicense/WEB-INF/web.xml`, allows access only from localhost ip addresses:

```
<filter>
 <filter-name>Remote Address Filter</filter-name>
 <filter-class>org.apache.catalina.filters.RemoteAddrFilter</filter-class>
 <init-param>
 <param-name>allow</param-name>
 <param-value>127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1</param-value>
 </init-param>
 </filter>
 <filter-mapping>
 <filter-name>Remote Address Filter</filter-name>
 <url-pattern>/*</url-pattern>
 </filter-mapping>
```

With this configuration, if the `BravaServer` and `BravaLicense` servlets are installed on the same machine, and the *`lc.url`* property within the Brava! Server `server.properties` file has a value of `http://localhost:8080/BravaLicense`, then the Brava! Server webapp can communicate with the Brava! License webapp. However, Brava! License is inaccessible from other machines. For more information about the catalina `RemoteAddrFilter`, see:

Apache Tomcat 10 Configuration Reference (10.1.10) - Container Provided Filters (https://tomcat.apache.org/tomcat-10.1-doc/config/filter.html#Remote_Address_Filter)

3.5 Cr2dl Chrome security installation

Cr2dl (Chrome to DL) is the loader that is used for viewing local HTML files. It converts HTML files to PDF using the Chrome rendering engine, then loads the PDF using the Pdf2dl loader. Because arbitrary HTML loaded into Chrome has the potential to access data that it should not, Chrome must be secured prior to use by Cr2dl. The file restrictions required by Cr2dl are to apply a block list to Chrome that prevents loading any links from file, http, or https protocols. (See https://chromeenterprise.google/policies/#URLBlocklist.) As a result, the HTML files loaded by Cr2dl are passed directly into Chrome without loading from a link. This allows local HTML files to load, while preventing those files from accessing or referencing any other files. Note that many HTML files are designed to reference other files and can have an unexpected appearance under this restriction.

If using Cr2dl for processing HTML files, install the latest version of Chrome on your Job Processor(s) and follow the appropriate procedure below to apply the required block list.

To apply the block list on Windows – Method 1

1. Sign in with the account that the Job Processor will be running under.
2. Edit the Windows Registry:
   a. Navigate to the following key, creating any keys in the path that are missing: `HKEY_CURRENT_USER\SOFTWARE\Policies\Google\Chrome\URLBlocklist`
   b. Add a string value of "1" with the data of "`file://*`"
   c. Add a string value of "2" with the data of "`http://*`"
   d. Add a string value of "3" with the data of "`https://*`"

To apply the block list on Windows – Method 2

1. Sign in with the account that the Job Processor will be running under.
2. Using a text editor, put the following text into a new file. Save it as a registry file type with the name `cr2dl_install.reg` to a temporary directory.

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\SOFTWARE\Policies\Google\Chrome\URLBlocklist]
"1"="file://*"
"2"="http://*"
"3"="https://*"
```

3. Right-click the new `cr2dl_install.reg` file and click **Merge** to install its contents into your Windows Registry.

To apply the block list on Linux

1. Sign in with a root account.

2. Navigate to the following directory, creating any directories in the path that are missing: `/etc/opt/chrome/policies/managed`

3. Create a new text file named `URLBlocklist`.

4. Put the following text into the new file:

```
{
    "URLBlocklist": ["file://*",
     "http://*",
     "https://*"]
}
```

5. Save the file to the `managed` directory.

Configure Cr2dl:

1. In the Brava! Enterprise default installation, Cr2dl is not the default loader for publishing HTM and HTML files and you must use the "Loader Configuration Tool" on page 89 to configure this option. See *OpenText Brava! - Loader Configuration User Guide (CLBRVW-ULC)*.

2. Run the `loaders.configuration.exe` file from `<install dir>\Job Processor\Igc.Loaders\` to launch the Loader Configuration tool.

3. Expand the **Cr2dl > Parameters** section and set *`CromeLocation`* to the complete path to the local Chrome executable (`Chrome.exe` or `google-chrome`). Typically, this would be `C:\Program Files (x86)\Google\Chrome\Application` on Windows, or `/usr/bin` on Linux. Do not include the filename itself.

4. In the Cr2dl **Associated Extensions** section, add or change the htm and html extensions to use Cr2dl.

Note: If Cr2dl is configured for htm and html files and the Chrome security instructions are skipped, the default configuration of the loader will post a failure message as "Chrome not configured with secure file restrictions enabled. Please enable block-list security."

3.6 Other considerations

- **Visual Rights permissions**

  Brava! Enterprise provides the capability (enabled by default) for end users with the ActiveX Client to publish CSF files from native loaded files. All Visual Rights permissions retrieved from the server and applied to the output CSF file are enabled by default. If this presents a security issue for your setup, you can disable specific rights by returning a custom `security.xml` file. This customization is accomplished by configuring the *`getsecurityxml`* property. See the Brava! Programmer's Guide available in the BravaSDK for more information.

  Additionally, be aware that Brava! Enterprise provides the ability for the end user to interactively set the Visual Rights when they are publishing a CSF file. This capability is off by default. If this feature is enabled, there is no way to restrict which Visual Rights end users can apply to their published CSF files.

- **Job Processor permissions**

  If the Job Processor is running on a separate machine from the Brava! Server service, a network share to the Brava! Server displaylistcache folder needs to exist so that the Job Processor service can access it. See "Displaylist cache" on page 10. Note that Brava! Server installer automatically creates this network share. The service account running the Job Processor needs to have *read* and *write* access to this network share. For security, access to the network share should be restricted to the Job Processor service account.

- **Displaylist cache permissions**

  By default, the Windows installer shares the displaylist cache folder on the network with full read and write access for "Everyone". This shared folder is installed, by default, to `%ProgramData%\OpenText\dlcache`. It is recommended to restrict access to this network share. For example, you might want to use the same service account that is used for running the Brava! services. Additionally, on the local machine, you might want to further secure this folder (NTFS permissions can be used if the Job Processor is running under an account with those permissions).

Chapter 4

Brava! Configuration

This chapter covers how to administer the configuration of Brava! Enterprise. You will learn how to configure Brava! parameters to meet your custom specifications, set up the Server and the Job Processors, and monitor that process.

4.1 Configuring Brava! Clients

Brava! Enterprise can be configured to use both HTML clients and an ActiveX client. The ActiveX client is being phased out in favor of a fully featured selection of HTML clients. The end user client experience can be customized through the use of various sample web pages and optional parameters which can be configured to meet your needs.

4.1.1 Sample web pages

Brava! Enterprise displays documents or drawings by loading HTML pages within a web browser. The loaded HTML pages can be served from static HTML files on a web server, or can be dynamically generated HTML originating from a server-side script, a JSP page or an ASP page.

To use Brava! Enterprise, you must produce your own web pages (typically ASP or JSP pages) that contain links to the documents you want to make available for viewing.

Different sample pages can be found for each viewer within the BravaSDK application:

```
<webapps>/BravaSDK/<viewer>/samples
```

Important

For security reasons, the BravaSDK application should not be run on production environments.

A quick overview page that provides access to any of the available sample pages is available from `<webapps>/BravaSDK/`.

Refer to the online documentation within the BravaSDK for detailed descriptions of the code in these samples as well as how to construct custom pages for use within your application.

Following is a partial list of some of the available sample pages, along with a short description of each, for the ActiveX and HTML clients, as well as the Load Balancing set up samples. Sample pages are also available for the DocMerge, Preview, and Text Compare clients.

ActiveX samples:

`.../BravaSDK/ActiveX/samples`

`BasicFileLoad/overview.html`
: Demonstrates a simple scenario of loading a file statically from a predefined viewer configuration.

`Compare/<XXX>.jsp`
: Demonstrate different scenarios for comparing two different documents when one might be converted by the client and the other might be converted by Brava! Server.

`ConvertOnClient/<XXX>.jsp`
: Demonstrate different configurations for viewing different file types where the file is converted by the client instead of by Brava! Server.

> **Note:** To use the `DOC.jsp` and `PPT.jsp` samples, you must first contact your OpenText Account Executive to obtain a Brava! Client License key for Outside-in conversion and an `OfficeLoaders.bin` file that contains all of the support files for conversion.

`DynamicLoad/DynamicLoad.html`
: Demonstrates how to reuse a single viewer instance to consecutively load different files using an AJAX call to retrieve new viewer configurations. `GetActiveXConfig.jsp` is the custom service that is called to retrieve a new configuration.

`ESigDataItems/YYY/AdHocMode/overview.html`
: Demonstrate the AdHoc mode for creating an integration with ESignatures.

`ESigDataItems/YYY/TemplateMode/overview.html`
: Demonstrate a custom integration with ESignatures using predefined templates for signing.

`ExportFileSave/overview.html`
: Demonstrates configuring the viewer to use a custom publish export service for exporting published files to a custom location. `ExportXXX.jsp` are the custom services used for handling an export of the different available publish file types.

`FileWithXRefsUpload/filewithxrefs.jsp`
: Demonstrates uploading a file with xrefs to the Brava! Server such that the entire file, including xrefs can be properly displayed within the viewer.

`MarkupFileIO/overview.html`
: Demonstrates creating custom markup services for saving and retrieving markups to and from a custom file location or repository. `XXXMarkup.jsp` are the custom services called by the viewer to handle the different processes when handling markups.

`Measurement/<XXX>.jsp`
: Demonstrate different measurement configurations.

Multisource/<XXX>.jsp
: Demonstrate configurations for enabling on-demand page loading of documents comprised of multiple single page files. `GetSinglePageTif.jsp` is a custom service that handles serving the individual files as the viewer requests them.

PermissionsAndVisualRights/overview.html
: Demonstrates configuring the viewer to handle special permission and visual rights determinations using external services. `CsfPublishRights.jsp` and `perm.jsp` define the custom external services the viewer is configured to use for determining the permissions and rights.

PersistenceFilesLocation/<XXX>.jsp
: Demonstrate different scenarios for configuring the viewer to store user persistence setting files at customized locations.

PrePublished/YYY/<XXX>.jsp
: Demonstrate different scenarios for loading files that were already converted to our proprietary viewer formats, CSF and XDL.

Redaction/<XXX>.jsp
: Demonstrate different redaction script capabilities by loading different files optimized for certain pattern searches available in the default redaction scripts available.

RenditionRequest/<XXX>.jsp
: A document rendition is required to view a document within the viewers. Merging multiple files, burning in markups, applying banners, and publishing to different file formats are accomplished through a RenditionRequest.

Themes/<XXX>.jsp
: Demonstrate how to configure the viewer to use different default and custom theme skins.

ControlInitialized.jsp
: Demonstrates using the `ControlInitialized` event to trigger additional client-side processing.

CustomButtons.jsp
: Demonstrates creating custom UI elements with custom behavior.

DBUpdateString.jsp
: Demonstrates how to use the `ResolveStampEntityTokenSetJS` event for resolving `DBUpdateString` tokens that the viewer requests for resolution.

HTML Client samples:

`.../BravaSDK/HTML/samples`

3D
: Demonstrates launching the Brava! HTML Client 3D user interface.

`binder`
: Demonstrates launching the Brava! HTML Client viewing a binder of 3 different documents.

`client-api`
: Uploads a document, creates a composition rendition of it, and creates a simple default configuration to view it.

`compare-report`
: Uploads two source documents, has Brava! Server generate a compare report PDF, and creates a mostly defaulted signed configuration XML for the Brava! HTML Client to view that compare report document.

`content-suite-viewer`
: Demonstrates launching the OpenText bundled UI.

`custom-markup-storage`
: Demonstrate using custom markup storage scripts on the server; creating a custom `GetMarkup`, `MarkupList`, and `SaveMarkup` service for use by the HTML Client.

`esig/create-signing-template`
: Demonstrates launching the Brava! HTML Client with all of the markup tools hidden except the esig tool, allowing the user to create a new esignature markup template.

`esig/sign-document`
: Demonstrates launching the Brava! HTML Client with a preloaded markup file containing preset esignature elements, along with preset esigning rasters for the sample user. This shows how easy it is to present a simple user interface to the user allowing them to sign a document.

`formats`
: Uploads a source document, performs a simple publish of that source document, and creates a mostly defaulted signed configuration XML for the Brava! HTML Client.

`graphical-compare`
: Uploads two source documents, creates a special rendition of those documents enabling side-by-side and overlay graphical comparison in the Client.

`reload`
: Showcases the cad-compare feature, as opposed to the compare PDF report that Brava! Server can generate from two different documents.

`themes`
: Demonstrates launching the various Brava! HTML Client user interfaces (Smart View, Classic View, mobile platforms, color themes, and an accessibility view sample page). A measurement Takeoff sample page is also available in this folder.

`video`
Plays a video in the HTML Video Client allowing the user to load, create, and save markup files against the video file.

ActiveX and HTML Client load balancing samples:

- In the `Proxy` samples, the integration passes a docid hash in an HTTP header to a proxy. The proxy then takes the modules of the value based on the number of configured servers and passes the request on to one of the Brava! Servers.
- In the `Direct` samples, the integration computes which of the Brava! Servers to communicate with, and sets the `BravaConnection` appropriately (as well as method URLs for ActiveX).

If these pages are not placed on the Brava! Server machine, then the *`legal.ip.to.get.sessionid`* property in the `server.properties` file needs to be updated with the IP address of the machine the pages are placed on. Also, check to verify that the call from these pages to obtain a session ID from the Brava! Server is not routed through a proxy server (otherwise the IP of the proxy server needs to be added to the *`legal.ip.to.get.sessionid`* list). See "Configuring the Brava! Server properties" on page 71 for more information about setting the *`legal.ip.to.get.sessionid`* property.

4.1.2 Escape Codes

Brava! Enterprise uses URL encoding. Parameters added to the sample ASP or JSP pages must be URL encoded. For example, any spaces must be encoded as "%20".

Characters can be encoded with a % followed by its ASCII hexadecimal equivalent code.

Table 4-1: Escape Codes

Character	Escape code	Character	Escape code
SPACE	%20	.	%2E
!	%21	/	%2F
"	%22	:	%3A
#	%23	;	%3B
$	%24	<	%3C
%	%25	=	%3D
&	%26	>	%3E
'	%27	?	%3F
(	%28	[	%5B

Character	Escape code	Character	Escape code
)	%29	\	%5C
@	%40	]	%5D
`	%60	^	%5E
*	%2A	_	%5F
+	%2B	{	%7B
,	%2C	|	%7C
-	%2D	}	%7D
		~	%7E

4.1.3 Double byte character encoding

To process and display double byte characters properly, such as Asian, the HTML page contents should be UTF-8 encoded. Additionally, the character set encoding used by the IE browser to display the contents of the web page should be set to UTF-8. To find the encoding used for your currently loaded web page from Internet Explorer, select **View > Encoding** and check to see which encoding scheme is selected. The IE browser sets the encoding for a web page based on the following (listed in order of precedence):

1. If a Unicode byte order mark (a set of bytes at the beginning of a file that specifies the file's encoding) is set for the HTML page, Internet Explorer sets the encoding appropriately when it loads that page. The byte order mark for UTF-8 consists of the three bytes “EF BB BF”. This can be achieved by opening an HTML file with a text editor and saving it with UTF-8 encoding.
2. The charset specified in the content type of the HTTP header. Apache defaults to ISO-8859-1 for static pages it serves (set in `conf/httpd.conf`). For dynamic web page generation, this header can be dynamically set. For example, a JSP can set it with the following line `-> response.setContentType("text/html; charset= UTF-8").` See the `view.jsp` (located in the Brava! server web application) for an example.
3. If the byte order mark or the content type charset isn't set in the HTTP response header, then IE looks for the `<meta>` tag to see if it specifies a charset for the content type. An example line that sets the encoding to UTF-8 is: `<meta http-equiv="Content-Type" content="text/html; charset=utf-8">` This line must be the first line within the HTML `<head>` tag.

If using a form to submit information with double byte characters, then the submitted form elements use the encoding specified in the web form page `<meta>` tag if it exists (see point #3). The encoding set in the `<meta>` tag should match the encoding of the text in the form, and the web server that receives the form submission must be able to deal with that encoding scheme. The `choosefile.jsp`, in the Brava! Client Install directory, is an example form page that submits docID information to the `view.jsp` page (located in the Brava! Server web application) for processing. The character set used for this processing is UTF-8, which is the recommended encoding.

4.1.4 Brava! ActiveX required control parameters

Important

It is required that integrators use the composition *`setBravaXParams`* (server side configuration) for any bravax parameter. (See the Brava! SDK documentation.) The session ID must be defined in the client, and everything else should be set up through the integration *`setBravaXParams`* method or through the `server_precedence.properties` and `client_precidence.properties` files.

All additional bravaxparams which are passed in /Properties endpoints are ignored.

As previously mentioned, parameters are conveyed to the Brava! Client ActiveX control using the configuration file's BravaXParameters. Each parameter is located on its own line, with the parameter name located to the left of an equal (=) sign and the value to the right. If a parameter value is several lines long, new lines are specified with the character set "<\n>". The key specified on each line is referred to as a BravaX param. These parameters are either set in the Client HTML pages or in the `server_precedence.properties/client_precedence.properties` files (see "Optional ActiveX Client parameters" on page 62 for detailed information about these two files).

The following Brava! ActiveX parameters are required to use the Brava! ActiveX Client.

Note: These settings and the settings in the Optional ActiveX parameters sections have no effect on the HTML Client.

Client and Server control parameters

IntegrationURL	The URL to the `Integration.dll` which provides the Brava! Client ActiveX control with the functionality to connect to the server for file and markup access. Note that even though the `IntegrationURL` is specifying `generic.bin` or `generic.dll`, the actual file that is downloaded is specific to the user's language setting as `generic_<three letter acronym>.bin` or `.dll`. For example, If the current language is set to Japanese on the client machine, the file that is downloaded to the client is `generic_JPN.bin` or `generic_JPN.dll`.

IntegrationVersion	Specifies the version of the `Integration.dll`. This lets the Brava! ActiveX control know when a new `Integration.dll` file needs to be downloaded.
PublishDocument	The URL of the service that publishes a native file to our published format using the Job Processor. Example: `PublishDocument=methodurl,http://localhost:8080/BravaServer/Publish` “localhost:8080” should be the appropriate server name and port number that the webapp server is running on.
ServerProperties	The URL of the Brava! Server servlet that provides initialization information.
GetLastError	The URL of the Brava! Server servlet that provides error information about server failures.

Additional document control client parameters

Similar to the *`ServerProperties`* and *`GetLastError`* properties, there are several other properties that specify where clients make calls for various server functionality. These are specified in the `client_precedence.properties` file that is installed in the Brava! Server directory. These properties are described in more detail in the Brava! Enterprise Programmer's Guide included in the documentation folder within the BravaSDK.

The following parameters are used *only* in the client-side HTML BravaXParams and are not recognized in the precedence properties files.

Client only parameters

DocID	This required parameter uniquely identifies the document/drawing to be loaded into the Brava! Client viewer. Valid values are a unique ID, file system path, or URL. **Note:** URLs must be UTF-8 URL encoded. For example, `http://my/server/test%20file.jpg`.
CurrentURL	If the value of *`DocId`*/*`DocSource`* is a relative path, then the *`currentURL`* value is prepended to the relative *`DocId`*/*`DocSource`* path value.
DocSource	This optional value can be set to a file system or URL path. As with the *`DocID`* parameter, URLs must be UTF-8 URL encoded. If set, Brava! uses this path, as opposed to the value of the *`DocID`* parameter, to retrieve the file to be loaded. Typically this is only set if the *`DocID`* is set to a unique ID.

DocVersion	This is an optional parameter that indicates the version of the document being accessed. This parameter is typically used with custom integrations. If the *`DocVersion`* of a document has changed for example, the document is re-published. By default, to determine if a document was updated, the Brava! Server looks at the time stamp of the file (for *`DocID`* or *`DocSource`* at file locations) or at the document size (for *`DocID`* or *`DocSource`* at URL locations).
DocExtension	If the file is to be converted on the client, the *`DocID`* or *`DocSource`* value must end with an extension, for example .pdf or .tif, so the client can determine how to convert the file. If the *`DocID`*/*`DocSource`* does not have an extension, then the *`DocExtension`* parameter must be set to the extension corresponding to the file type. **Note:** This parameter only functions on files loaded using a URL path. Example:`DocExtension=pdf`

Download.<specified file>

The following parameters are used by the Brava! Client viewer to download specified files.

download.url.{name}

A URL pointing to a file to be downloaded to the user's "\IGC" directory. If the file is a zip file, it is unzipped. {name} can be any value.

download.version.{name}

The version of the file to be downloaded. This information is stored in the `version.txt` file located in the user's `<user profile>\IGC\<version>\` directory. The version number is used to determine whether an updated file needs to be downloaded. The value of {name} needs to match the value used for the corresponding *`download.url.{name}`* param.

Examples of using the download params are:

`download.url.Pdf2DL=http://<Server>/IGC/Pdf2DL.bin`

`download.version.Pdf2DL=2.5.7.2`

If the file specified by the *`download.url.Pdf2DL`* parameter is not yet downloaded, or if it has and its version is less than 2.5.7.2, then the Brava! Client downloads the file (`http://<Server>/IGC/Pdf2DL.bin`). Since this is a zip file, its contents are unzipped to the user's `\IGC\<version>` directory, and the `version.txt` file is updated to reflect version 2.5.7.2.

Example: `download.url.RedactionScripts=http://<server>/IGC/RedactionScripts.bin`

`download.version.RedactionScripts=7.2.0.3`

This example specifies location and version of Redaction Scripts bin file that can be downloaded from the web server and installed to the local client machines. Allows distribution of script files from server.

download.strictversionmatch.{name}

If set to "true" then if the value of the corresponding *`download.version.{name}`* parameter does not match the version listed in the local `version.txt` file, then the file specified by *`download.url.{name}`* is downloaded. If set to false (the default), the client viewer only downloads the file if the version number in the corresponding *`download.version.{name}`* parameter is greater than the value in the local `version.txt` file.

download.destinationname.{name}

If the file specified by the corresponding *`download.url.{name}`* parameter is not a zip file, then the file is renamed to the value of this parameter on the local machine when it is downloaded.

The following downloads the Changemark configuration file used for setting type and state options of the Changemark dialog box.

```
download.destinationname.ChangemarkConfig=ChangemarkConfig.xml
```

The following example downloads the standard theme file used for the Brava! ActiveX Client, setting "Blue-Theme" as the default theme:

```
download.url.Theme=http://<server>/IGC/Themes/Blue-Theme.xml
```

```
download.version.Theme=1.0.0.1
```

```
download.destinationname.Theme=BravaClientTheme.xml
```

4.1.5 Optional ActiveX Client parameters

Refer to the *OpenText Brava! - Brava! Enterprise ActiveX Client Parameter Configuration Guide (CLBRVW-CAX)* for a complete list of optional Brava! Client parameters (with their descriptions and possible values) that you can configure to meet your specific environment and custom requirements.

Important

It is required that integrators use the composition *`setBravaXParams`* (server side configuration) for any custom bravax parameter. (See the Brava! SDK documentation.) The session ID must be defined in the client, and everything else should be set up through the integration *`setBravaXParams`* method or through the `server_precedence.properties` and `client_precidence.properties` files.

All additional bravaxparams which are passed in /Properties endpoints are ignored.

The optional Brava! ActiveX parameters affect the state of the Brava! ActiveX Client. The parameter tables list the optional tags that can be used to customize your Enterprise configuration. The bold word in the left margin is the parameter name that precedes the "=" sign within the `BravaXParams` value in the `simple.html` example code. The descriptive text in the right column describes the parameter function, and the appropriate value for the parameter. Within an HTML page, the parameter value follows the "=" sign on each line within the `BravaXParams` value. See the `simple.html` as an example.

In addition to establishing these parameters on a per-file basis by adding them to the HTML that is rendered on the client, you can establish parameter values that affect all files on a site by setting properties in the `server_precedence.properties` or `client_precedence.properties` files located in the Brava! Server installation directory. A particular property can be set in one of three places – the client-side HTML, or in one of the two server-side precedence properties files, which contain lists of available parameters.

As described in the previous section, certain client control parameters can only be specified in the client-side HTML pages. If a property can be (and is) set in multiple places, the order of resolution is (from highest to lowest):

`server_precedence.properties`

HTML pages `BravaXParams`

`client_precedence.properties`

- `server_precedence.properties` : contains properties that set the client* state or behavior, for example, print banners or enable{button}. The properties defined here take precedence over those defined in the HTML web page or in `client_precedence.properties`.
- `client_precedence.properties:` contains properties that set the client* state or behavior; lower priority than those defined in the web pages or in `server_precendence.properties`.

Updates to the `server_` and `client_precedence.properties` files require a restart of the servlet engine.

! **Important**

If a property value in one of the server-side precedence files contains a file or directory path, then all back slashes must be escaped with another back slash. Also, to designate a quote in the value of a property, use the term `&quot;`.

4.1.6 Custom themes

Custom theme (or skin) files can be defined and applied to Brava! Enterprise, allowing you to specify the appearance (color) of the various elements that make up the user interface. A Theme is an XML-encoded, coordinated set of colors and images. The theme element has a name, a version number and locale.

You can apply a custom theme to the Brava! ActiveX Client by downloading the standard `BravaClientTheme.xml` file from the *`download.destinationname.Theme`* Brava! Client parameter (`download.destinationname.Theme=BravaClientTheme.xml`) and editing the XML to meet your custom requirements.

The theme XML file must reside on the local disk or be accessible on a shared network. Integrators can also set themes through the ActiveX API. For example, calling *`ApplyColorCustomization`* (“c:\themes\theme1.xml”, 0) through the API (see *OpenText Brava! - Brava! ActiveX Client API Interface Details Guide (CLBRVW-RAX)*).

A theme is defined in XML format (description follows). For additional details, refer to *OpenText Brava! - Brava! ActiveX Extensible Interface Guide (CLBRVW-SAX)*.

Regions and containers

Regions and Containers define the elements that can currently be skinned and include the following:

The region name identifies the area on screen to be “skinned”. Each region consists of a *`ForegroundColor`* and a *`Background`* element. The foreground color is applied to any text or static elements that do not have an explicitly set color. The background can be set to a single color or to an image file contained on disk. You can also specify whether the background is dark.

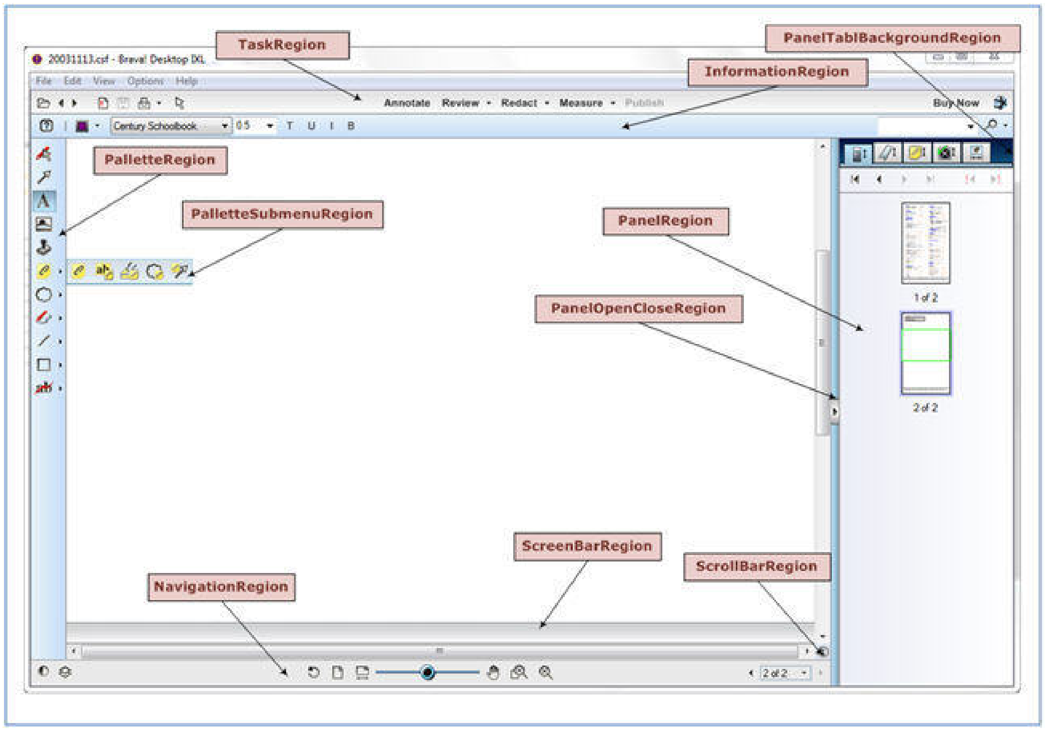
The ActiveX object contains two images for most buttons that appear on a skinnable control. Which image is used depends on whether the control background is marked as dark or not. When specifying an image to use, you can select an image in any of the following formats: EMP, GIF, JPEG, PNG, TIFF, Exif, WMF, and EMF.

```
<region name="TaskRegion">
<foregroundcolor value="#ffffffff"/>
<background type="image-id" dark="true" value="2907"/>
</region>
```

A layout consists of a set of Containers. At present, a maximum of twenty one containers can be defined. A Container specifies one of the pre-defined BravaX controls that can be used to host custom content.

```
<layout>
<containers>
<container name="CompareBar">
...
</container>
...
```

```
</containers>
</layout>
```



The following table lists the Regions and Containers that are recognized by the system and can be skinned in Brava! Enterprise ActiveX Client.

Table 4-2: Regions and Containers

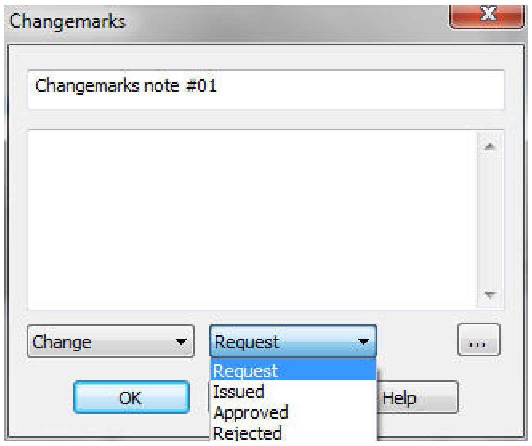
Region	Container	Description
TaskRegion	TaskBar	The toolbar that contains the **Annotate**, **Review**, **Redact**, **Measure**, and **Publish** buttons.
InformationRegion	InformationBar	The toolbar that contains the markup properties bar, measure results bar, and find bar.
	MarkupProperties	The toolbar containing the markup properties tools.
	TextSearch	The toolbar containing the text search controls.
	MeasurementResults	The toolbar containing the measurement results

Region	Container	Description
NavigationRegion	NavigationBar	The toolbar containing the page control, zoom slider, and navigation tool buttons.
ComparisonRegion	ComparisonBar	The toolbar containing the compare tools.
TextCompareResizeRegion	ComparisonBar	This region refers to the resize bar and mirrors how the panel resize bar works.
TextCompareHeaderRegion	ComparisonBar	This region refers to the header portion of the text compare panes (with the words Open File Compare File and two arrows).
ScreenBannerRegion	ScreenBanner	The bar that shows the screen banner.
ScrollBarRegion	HorizontalScrollBar	The bar containing the horizontal and vertical scroll controls.
PanelRegion	VerifyPanel	Panel containing the verify tools.
	ThumbnailPanel	Panel containing the document thumbnail pages.
	ChangemarkPanel	Panel containing the Changemark notes list and navigation tools.
	MeasurePanel	Panel containing the Measure and Takeoff results.
	BookmarkPanel	Panel containing the internal bookmark links.
PanelTabBackgroundRegion	PanelTabBackground	The container that holds all the panels.
PanelOpenCloseRegion	PanelOpenCloseBar	The vertical bar that toggles the visibility of the panels.
PalletteRegion	MarkupTools	The toolbar containing the markup tools.
	RedactionTools	The toolbar containing the redaction tools.
	MeasurementTools	The toolbar containing the measure tools.
	TakeoffTools	The toolbar containing the takeoff tools.
PalletteSubmenuRegion	PalletteSubmenu	The pop-out window that appears when someone activates a submenu on the markup toolbar.

4.1.7 Changemark config details

Used in the Changemark Discussion Panel, *`Type`* and *`State`* are attributes that can be assigned to the original Changemark content and to each reply that is added to a discussion. These *`Type`* and *`State`* are displayed for each Changemark note contained in the Changemark List, at the bottom of the panel.

When users create a Changemark note or reply to a Changemark discussion, option lists are available for selecting the *`Type`* and *`State`*. After the *`Type`* is selected, the *`State`* list shows only the *`State`* associated with the currently selected *`Type`*.



The Changemark dialog box initially selects the *`Type/State`* marked as the default type in `ChangemarkConfig.xml`.

The following *`Type`* and *`State`* are set in your default Brava! installation, but these values can be customized for your users by editing the `ChangemarkConfig.xml` file. In addition, you can set custom colors for each entry by specifying the RGB color values for each corresponding State listed in the file. The *`Type`*, *`State`*, and color information that is defined when users create the Changemark note (or reply to a Changemark note) is saved in the markup file and can be reviewed by other users.

The file is downloaded to the Brava! Server machine using the `ChangemarkConfig.xml` and is installed in the directory where the ActiveX cab file is downloaded (for example, `http://localhost:8080/BravaSDK/ActiveX/config/ChangemarkConfig.xml`) and the xml is directed to the user's HOMEPATH `C:\Users\<Username>\IGC\<version>\<GUID>\ChangemarkConfig.xml` directory specified by the client ActiveX page parameter *`download.destinationname.{name}`*. See "Optional ActiveX Client parameters" on page 62

Example: `download.destinationname.ChangemarkConfig=ChangemarkConfig.xml`

Table 4-3: Type and State values

Type	State	RGB value	Color displayed
Action	For Discussion (default)	193,235,255	Light blue
	An Idea	95,255,150	Light green
	Investigate	255,190,110	Orange
	Typo	255,205,160	Light orange
	Revision Error	255,250,180	Light yellow
	Confirm	180,140,200	Light purple
	Urgent	255,90,80	Red
Change	Request (default)	195,235,255	Light blue
	Issued	255,255,255	White
	Approved	95,255,150	Light Green
	Rejected	255,90,80	Red
	Superseded	235,230,130	Tan
	Released	0,0,0	Black
	Verified	180,140,200	Light purple
	Completed	0,0,0	Black
	Closed	100,200,255	Blue
	Notice	0,0,0	Black
Agreement	Acceptable (Default)	95,255,150	Light Green
	Can't do	0,0,0	Black
	Carve out	255,250,180	Light yellow
	Typo	255,90,80	Red
	Discuss Further	195,235,255	Light blue
	Make Reciprocal	195,235,255	Light blue
	Reword	255,90,80	Red
Issue	Undetermined (Default)	195,235,255	Light blue
	Critical	255,90,80	Red
	High Priority	255,230,95	Yellow
	Med Priority	255,240,215	Light peach
	Low Priority	195,235,255	Light blue
	Closed	0,0,0	Black

Type	State	RGB value	Color displayed
Status	Investigate (Default)	195,235,255	Light blue
	Pending	195,235,255	Light blue
	Working	195,235,255	Light blue
	Review	195,235,255	Light blue
	Completed	0,0,0	Black
	Closed	0,0,0	Black
Missing	Approval (Default)	255,90,80	Red
	Signature	255,90,80	Red
	Account No	255,90,80	Red
	Verification	255,90,80	Red
	Amount	255,90,80	Red
	Address	255,90,80	Red
	Other	255,90,80	Red
	Concern (Default)	195,235,255	Light blue
	Deliverable	195,235,255	Light blue
	Cost/Benefit	195,235,255	Light blue
	Feasibility	195,235,255	Light blue
	Resources	195,235,255	Light blue
	Estimate	195,235,255	Light blue
	Scheduling	195,235,255	Light blue

Notes

- If a user's configuration file defines state “X” to be colored green, but the markup entity they are reviewing has “X” defined as blue, the edit box appears blue. This occurs when different users have different types and states defined.
- The `State` font color is normally black. If the background color for the `State` has a luminance less than 80, the state displays using a white font.

4.1.8 Brava! HTML Client

The Brava! HTML Client provides users the capability to load documents from the Brava! Server online using a web browser. Documents are processed with the Brava! Server as HTML output and are presented in this streamlined viewer for quick viewing, searching, publishing, redacting, and adding/reviewing annotations, including Changemark discussion.

The BravaSDK application includes sample pages for launching a variety of user interface experiences, including 3D, Video, Graphical Compare, and Text Compare viewing applications. See "Sample web pages" on page 53. In addition, a mobile device version of the HTML Client is available that supports a limited feature set (view, search, annotation, and publishing to PDF). Refer to the Brava! Enterprise release notes document for mobile device support details.

Refer to the Brava! HTML Client integration documentation included in the SDK documentation system for details about each specific part of the XML configuration elements and information about integrating with the Brava! HTML Client.

HTML Client Print feature matrix

Important
Brava! HTML Viewer produces a PDF that is then sent to the user's internet browser for printing. The fidelity of these printed files is completely dependent on the browser's print capabilities.

To obtain the optimal PDF plug-in printing experience, update your client browsers to the most recent version available.

Depending on your browser configuration, the following **Print** features will be available in the HTML Client.

Table 4-4: HTML Client print feature matrix – with Acrobat Plug-in

Browser	Markups	Banners / Watermarks	Changemark Summary	Redaction Reasons	Monochrome /Grayscale
Chrome	X	X			X
Firefox	X	X			X
Safari	X	X			X (v.6+)

Table 4-5: HTML Client print feature matrix – without Acrobat Plug-in

Browser Without Acrobat Plug-In	Markups	Banners / Watermarks	Changemark Summary	Redaction Reasons	Monochrome /Grayscale
Chrome -With "Chrome PDF Viewer" Plug-in	X	X	X	X	X

Browser Without Acrobat Plug-In	Markups	Banners / Watermarks	Changemark Summary	Redaction Reasons	Monochrome /Grayscale
Chrome -No "Chrome PDF Viewer" Plug-in	X	X			X
Firefox	X	X			X
Microsoft Edge	X	X			X
Safari - With "WebKit built-in PDF" Plug-in	X	X	X	X	X (v.6+)
Safari - No "WebKit built-in PDF" Plug-in	X	X			X (v.6+)

4.2 Configuring Brava! Server

Once the Brava! Server and Client are installed, you need to configure the Brava! Server.

4.2.1 Configuring the Brava! Server properties

Configure the Server by adjusting settings in the `server.properties` file, located in `C:\Program Files\OpenText\Brava! Enterprise\Brava! Server` by default. The `server.properties` file contains configuration information for the server web application (the URL to communicate with the Job Processor, file size limits, time-outs, etc.). Read the `server.properties` settings in the following table and make any adjustments necessary.

! **Important**
The configuration values in *`displaylist.cache.root`* are used to communicate information between the Brava! Server and the Job Processor. Check this property value if you experience problems viewing files. Refer also to "Configuring the Job Processor for Brava! Server" on page 94.

The `server.properties` file resides in the installed `..\OpenText\Brava! Enterprise \Brava! Server` directory. The settings are described in the following table. Updates to the `server.properties` file require the servlet engine to be restarted.

Server Properties Settings

done.servlet.url	**Value:** <*URL*>: not set **Description:** The URL that Brava! Job Processor uses for job completion notification. **Example:** `done.servlet.url=http://MyMachine.mydomain.com:8080/BravaServer`

rest.docretriever.classname	**Value:** <*classname*>: not set **Description:** Classname for the Java class that implements the ISourceResolver interface. Specify this class to override how the server retrieves source documents. **Example:** `rest.docretriever.classname=com.mydomain.httpserver.GenericDocRetriever`
session.life	**Value:** <*minutes*> **Description:** Specifies the time, in minutes, that the session/composition lives from the last action. As long as the session/composition lives, users can access content stored on the server. **Example/default:** `session.life=480`
lc.url	**Value:** <*URL*>: not set **Description:** The address where the BravaLicense servlet (Brava! License Server) is running. Use the hostname of the license server machine. **Example:** `lc.url=http://bravalicenseserver:8080/BravaLicense`
routing.service.callback.host	**Value:** <*hostname or IP*> **Description:** The host or IP of the BravaServer machine for Routing Service to use for forwarding calls. See "Installing the Brava! Routing Service" on page 22. **Example:** `routing.service.callback.host=bravaserverhost.example.com`
routing.service.callback.port	**Value:** <*port*> **Description:** The port of the BravaServer machine for Routing Service to use for forwarding calls. **Example:** `routing.service.callback.port=8081`

legal.ip.to.get.sessionid	**Value:** <*ip address*>: not set **Description:** Indicates the exact IP addresses of the machines allowed to request a session ID. The default is set to the localhost IP. *`legal.ip.to.get.sessionid`* can have a single value, or multiple values separated by commas. *`legal.ip.to.get.sessionid`* supports the following five formats of addresses or address ranges for each value in the list: • Single IPv4 Address. For example, `192.168.1.79` • Wildcard IPv4 Address. For example, `192.168..` • Single IPv6 address, with elision. For example, `fe80::fc40:64dd:8a57:885` • Wildcard IPv6 address, no elision allowed. For example, `fe80:0000:0000:0000:fc40:::*` • An IPv6 subnet range in CIDR notation. For example, `2001:db8:1234::/48` **Note:** Elision refers to the fact that IP addresses can omit various entire hextets that are all zero, or leading zeros within certain hextets. For wild card representation, all 8 hextets must be present! **Example:** `legal.ip.to.get.sessionid=127.0.*.*`
auth.validation	**Value:** on/off **Description:** If set to on, various Brava! Server endpoints, including those called by the HTML viewer validate that it receives a valid JWT token in the BravaAuth http header, signed with the value in the *`jwt.key`* property. See the BravaSDK samples, and "Establishing JSON Web Token (JWT) validation" on page 47. **Example:** `auth.validation=off`
jwt.key	**Value:** <*secret key*> **Description:** A secret key that is used to encrypt and decrypt Json Web Tokens by the Brava! Server. This key will be used to generate client JWTs. The same key should be used to generate integration, jobprocessor, and administrator tokens. A change to the key will invalidate all previously created tokens. See "Getting the SDK sample to work with authorization" on page 48. **Notes** • This key string must be a minimum of 64 characters (a minimum of 512 bits). • If you are using non ASCII characters, they should be encoded using `\u####` formatting. **Example:** `jwt.key=ENC(k+0aoBh05B612NCaYEyviccRWTSGvpyYQUpCNaNfg3Eg94Y6ui3dFdng5WQnnRb2xC)`

cors.hosts	**Value:** *<hostnames>* **Description:** Comma separated list of hosts for which CORS is enabled. The default value"*" matches all host names, enabling all hosts. See HTML based viewers. **Example:** `cors.hosts=integrationhost1,integrationhost2`
spring.profiles.active	**Value:** FileSystem, FileSystem,async or FileSystem,async,admin **Description:** Determines the active profile option to use. `OnPremise.FileSystem` is the only type of cache that is used and is mandatory. When `OnPremise.FileSystem` is active, the server returns immediately if there are no jobs in the queue for the requested queue type. In this case, the JobGetter's *`job.sleep.n`* value should be higher than 0. When `OnPremise.FileSystem,async` is active, the server won't return immediately if there are no jobs in the queue for the requested queue type. Instead, it asynchronously waits for a task to appear. The wait timeout is specified by the *`queue.retrieval.timout`* setting. In this case, the JobGetter's *`job.sleep.n`* value should be 0 for the Job Processor to immediately request a new job. When `OnPremise.FileSystem,async,admin` is active, it is possible to connect to each Brava! Server instance individually using the Monitoring Tool. See "Monitoring Tool" on page 32. **Note:** The absence of an async profile does not change how the artifacts are provided to the clients. Artifacts are provided asynchronously whenever possible. **Example:** `spring.profiles.active=OnPremise.Filesystem` Brava! Server does not support storing markups or stamp templates. By default, the integration endpoints for markups and stamp templates are disabled.
prepublished.formats	**Value:** *<file extensions>* **Description:** The file formats that are already viewable and do not need to be published. **Example/default:** `prepublished.formats=csf,xdl`
publish.request.types	**Value:** *<file extensions>* **Description:** This setting contains a list of the types supported by Brava! Job Processor. The list should match the values for the *`jobprocessor.thread.manager.thread.types`* property in the `JobProcessor.config` file. **Example/default:** `publish.request.types=doc,drw,pdf,3d`

publish.request .extensions.<>	**Value:** *<file extensions>*
	
	Description: If a file has an extension listed in one of these properties, the type DOC, DRW, PDF, or 3D (specified in place of the <>) is used in sending publishing requests to Brava! Job Processor.
	
	Example/default: `publish.request.extensions.pdf=pdf,key, numbers,pages`
	
	`publish.request.extensions.doc=doc,dochtml,docx,htm,html, mhtml,mht,mpp,msg,pps,ppsx,ppt,pptx,rtf,vdx,vsd,vsdx,vsx, xls,xlsx,xlw`
	
	`publish.request.extensions.drw=000,3df,3ds,906,907,afp, arx,asm,bmp,cal,ccz, cg4,cgm,cgmct,cgmt,cit,cmi,csf,mi, dft,dc,dgn,dgn7,dls,dsf,dsn,dwf,dwfx,dwg,dxf,edc,emf,eps, fax,g3,gif,gp4,grp,hdp,hgl,hpgl,hsf,i3f,iam,ica,icd,icf, ics,iges,igs,ipt,iso,isf,jp2,pg,jpeg,jpm,m3r,mcs,mil,mrk, mvp,mvs,neu,neu.1,par,pcd,plt,png,prt,prt.1,ps,psd,psm, pub,ref,res,rle,rnl,rtl,sat,sid,sldasm,slddrw,sldprt,slp, stl,step,stp,tg4,tif,tiff,txt,vda,vrl,wdp,wmf,wrl,x_t, xdl,xgl,xml,xpr,xps,zgl`
	
	`publish.request.extensions.3d=par,psm,asm,sldprt,sldasm, iges,wrl,iam,ipt,stl,sat,acis,slp,xgl,zgl,hsf,3ds,ics, prt,xas,xpr,neu,x_t,vda,skp,step,stp`
	
	Note: SketchUp files (.skp) are supported with Job Processor deployments on Windows only.

extensions.wait .for.xdl.publish	**Value:** *<file extensions>* **Description:** This setting contains a list of file extensions where the Job Processor uses print publishing. If the extension is in the list, then the server waits for the XDL publish to complete before sending additional publish jobs. To define extensions exclusively for a loader profile, enter a list of extensions enclosed in square brackets. For example, `[profile:ext1, ext2]`. A joined setting might be formatted as: `ext1, ext2[profile1:ext1,ext3][profile2:]` **Example/default:** `extensions.wait.for.xdl.publish=doc, dochtml,docx,htm,html,mhtml,mht,mpp,msg,pps,ppsx,ppt, pptx,rtf,vdx,vsd,vsdx,vsx,xls,xlsx,xlw`
single.page.pub lishing	**Value:** on/off **Description:** Globally enables/disables single page publishing of XDL files. Enabled by default. Uncomment to disable this feature. **Default:** `#single.page.publishing=off`

single.page.publishing.thumbnails	**Value:** on/off **Description:** Enables/disables single page publishing for thumbnails. This setting is not affected by the *`single.page.publishing`* value. Enabled by default. Uncomment to disable this feature. **Default:** `#single.page.publishing.thumbnails=off`
single.page.publishing.evasion	**Value:** not set **Description:** If the evasion value is higher than the number of pages between the actively published page in the main svg job and the requested page, the system will not create a single page publish for that page. Setting the value to -1 disables the feature. Setting the value to 0 disables single page publish jobs for actively published pages. Actively published pages are tracked by target folder monitoring. If the main job hasn’t started yet, and single page publish is enabled, the single page publish is created even if the evasion value is relatively high. Uncomment to use this feature. **Example:** `single.page.publishing.evasion=1`
single.page.publishing.thumbnails.evasion	**Value:** not set **Description:** If the evasion value is higher than the number of pages between the actively published page in the main thumbnail job and the requested page, the system will not create a single page publish for that page. Setting the value to -1 disables the feature. Setting the value to 0 disables single page publish jobs for actively published pages. Actively published pages are tracked by target folder monitoring. If the main job hasn’t started yet, and single page publish is enabled, the single page publish is created even if the evasion value is relatively high. Uncomment to use this feature. **Example:** `single.page.publishing.thumbnails.evasion=20`
executor.threadpools.maxthreads	**Value:** <*number of threads*> **Description:** This setting creates a thread pool that reuses a fixed number of threads operating off of a shared unbounded queue, using the provided ThreadFactory to create new threads when needed. At any point, at most *`nThreads`* threads are active processing tasks. If additional tasks are submitted when all threads are active, they wait in the queue until a thread is available. If any thread terminates due to a failure during execution prior to shutdown, a new one takes its place if needed to execute subsequent tasks. The threads in the pool exist until it is explicitly shutdown. If this parameter is not present, BravaServer reverts to using a non fixed sized thread pool. **Example/default:** `executor.threadpools.maxthreads=300`

publish.request .timeout	**Value:** <*minutes*> **Description:** Specifies the number of minutes that the server waits for a publish request to complete. It is also possible to specify the time using the following pattern: Xh Xmin Xs Xms, where h=hours, min=minutes, s=seconds, and ms=milliseconds. **Example/default:** `publish.request.timeout=7`
publish.heartbe at	**Value:** on/off **Description:** Determines whether heartbeat files are created. This option is off (commented out) by default for optimal runtime performance. For advanced troubleshooting data collection, you can either turn on this parameter (remove hashtag #) to generate the `heartbeat.txt` files in the displaylistcache, or enable Job Processor logging (`log=on` in `JobProcessor.config`). **Example/default:** `#publish.heartbeat=on`
rasters.dir	**Value:** <*folder path*> **Description:** Indicates the full path to the raster image files. If not set, the default is used: `<server installation directory>/raster`. Uncomment to use this feature. **Example/default:** `raster.dir=<server installation directory>/ raster`
max.raster.dime nsion	**Value:** <*integer*> **Description:** Used as an override, this parameter is not included in the default `server.properties` file and can be added if needed. When a document is converted for viewing in the HTML Viewer, and that document contains a large raster image, such as a TIFF or PDF of a scanned image, image quality can be noticeably reduced when compared to the original document viewed in the source application. By default, Brava! uses a Max Raster Dimension size of 1440 dpi to prevent certain mobile device browsers from having memory load issues. A customer server deployment can override this default value globally on the server for ALL documents by setting this `server.properties` parameter. Integrations can override the default value on a per-document basis by using the *`MaximumRasterDimension`* property, of which details can be found in the SDK documentation. If *`MaximumRasterDimension`* is specified at runtime for a document composition, that runtime value overrides both the default (1440), and the value specified by *`max.raster.dimension`*. **Note:** Any document compositions generated before this value is changed are fetched from the displaylistcache. If you have fidelity issues with previously generated documents, stop the Tomcat service, clear the displaylistcache folder, and restart the Tomcat service. Documents re-generate with the new max raster value. **Example:** `max.raster.dimension=5000`

render.complexity.threshold	**Value:** *<integer>* **Description:** Used as an override, this parameter is not included in the default `server.properties` file and can be added if needed. Integrations can override the default value on a per-document basis by using the *`setRenderComplexityThreshold`* method in the document composition (see the Brava! SDK documentation). If *`RenderComplexityThreshold`* is specified at runtime for a document composition, that runtime value overrides both the default (300000), and the value specified by *`render.complexity.threshold`* in `server.properties`. **Note:** Cache must be cleared before the new value can take effect. **Example:** `render.complexity.threshold=300000`

Displaylist Cache Properties

displaylist.cache.root	**Value:** *<folder path>* **Description:** Indicates the location where the published (cached) files reside within the Brava! Server. The entire cache contents live in this directory. This setting must be absolute, including shared folder name if Brava! Job Processor is NOT on the same machine as the Server (only supports Brava! Job Processor running locally). Any directory path delimiters should be escaped backslashes (two consecutive backslashes “\\”) as opposed to a forward slash. If the Job Processor is not on the same machine as the Server, then this path should be a UNC share that is accessible to the Job Processor. Otherwise, it is best not to use UNC shared for performance and security reasons. If this is the case, both values should be the same. This share is created automatically by the installer. Restrict remote access to the share to only the service account for the Job Processor. **Example/default:** `displaylist.cache.root=\\<MachineName>\<ShareName>\<BravaServerInstallDirectory>\dlcache` or `displaylist.cache.root=\\\\$APP_SERVER_NAME$\\dlcache`
local.displaylist.cache.root	**Value:** *<folder path>* **Description:** This setting used the same as *`displaylist.cache.root`* but uses the local (Brava! Server File System) absolute path or environment variable instead. The Brava! Server uses this setting's value to read/write the local displaylistcache. If the displaylistcache is physically set up on a remote file share, then this entry must also be a UNC share. **Example/default:** `local.displaylist.cache.root=C:\ProgramData\OpenText\dlcache` or `local.displaylist.cache.root=$PROGRAM_DATA$\\dlcache`

displaylist.truncation.none	**Value:** true/false **Description:** When set to `false`, the Brava! Server truncates records as described in the previous entries. When set to `true`, the Brava! Server does NOT TRUNCATE cache entries at all. **Note:** When this value is set to `true`,, the cache grows without limit. While the only known limits on cache size are physical resources, we have not tested cache sizes above 300GB. Caches larger than 300GB might work, but we are not able to provide support for larger cache. **Example/default:** `displaylist.truncation.none=false`
displaylist.cache.maximum.size	**Value:** <*megabytes*> **Description:** The target maximum total size of the displaylistcache, in megabites. If the current size exceeds the threshold defined in combination with *`displaylist.cache.size.reduction.percent`*, the oldest published documents are removed from the cache until its size is less than the calculated goal. Default value is 10,000 MB (10 GB). **Example/default:** `displaylist.cache.maximum.size=10000`
displaylist.cache.size.reduction.percent	**Value:** <*percentage*> **Description:** The percent of *`displaylist.cache.maximum.size`* that defines the truncation goal. **Example/default:** `displaylist.cache.size.reduction.percent=90`
server.delete.cache.entries	**Value:** true/false **Description:** This integration specific setting determines whether the Brava! Server truncates files or if it only removes cached files from the cache tables so that they are no longer referenced. Once files are no longer referenced in the cache tables it is the responsibility of the integration to remove them from the filesystem. **Example/default:** `server.delete.cache.entries=true`
delete.files.root.dir	**Value:** <*folder path*> **Description:** This integration specific setting sets the directory where the server writes text files that contain lists of directories that are ready for deletion. **Example/default:** `delete.files.root.dir=%MACHINE%\\ deletedir`
displaylist.cache.save.interval	**Value:** <*seconds*> **Description:** The minimum number of seconds between the instances when the displaylistcache tables are saved to disk. **Example/default:** `displaylist.cache.save.interval=300`

displaylist.cache.backup.interval	**Value:** <*minutes*> **Description:** The minimum number of minutes between backups of the displaylistcache entries table. Backups are used in case the tables currently within the *`local.displaylist.cache.root`* become corrupt. **Example/default:** `displaylist.cache.backup.interval=30`
browser.cache.expiration	**Value:** <*seconds*> **Description:** Time, in seconds, that the browser should keep artifacts in cache. **Example/default:** `browser.cache.expiration=5000000`
displaylist.cache.max.backups	**Value:** <*minutes*> **Description:** This controls The number of backup copies of the displaylistcache tables to keep. **Example/default:** `displaylist.cache.max.backups=50`
displaylist.cache.max.files.to.delete	**Value:** <*whole number*> **Description:** This setting controls the maximum number of file system cache directory entries Brava! Server will queue for deletion during a single pass. This setting is independent of internal and external deletion of truncated files. **Example/default:** `displaylist.cache.max.files.to.delete=20`
number.cache.entries.queued.before.deletion	**Value:** <*whole number*> **Description:** When `server.delete.cache.entries=false`: This is the minimum number of cache directory entries that must be queued for deletion before any will be appended to output files as lists of directories ready for deletion. When `server.delete.cache.entries=true`: This is the maximum number of entries/folders that will be removed from the disk during a single truncation pass. **Example/default:** `number.cache.entries.queued.before.deletion=10`
single.page.publish.cleanup.delay	**Value:** <*minutes*> **Description:** Delay in minutes before cleaning up single page publishes from the `dlcache` directory. **Example/default:** `single.page.publish.cleanup.delay=2`

4.2.2 Editing the Precedence Properties file

By default, the `client_precedence.properties` file has a list of properties specifying (by value) which Brava! Server ASPX pages to hit to execute various functions. These values can be overwritten to invoke custom functionality (for more information, refer to the Brava! Programmer's Guide). The URLs specified here are invoked from the end user machines, so the server and port must be accessible from those machines (some network administrators restrict outbound access to a few very standard ports). Upon installation, these properties are put into the `client_precedence.properties` file, or alternatively, they can be set in the `server_precedence.properties` file.

As mentioned in “Optional ActiveX Client parameters” on page 62, BravaX parameters can be set in either the `server_precedence.properties` (higher precedence than HTML set property values) or the `client_precedence.properties` file (lower precedence than HTML set property values).

Note: Any directory paths entered as values should have the backslash directory delimiter escaped with an extra backslash.

4.2.3 Brava! Server logging

Brava! Server logging uses slf4j (standard logging facade for java) and logback. You can access the documentation for logback from http://logback.qos.ch/index.html.

Logback.xml details

Brava! Server logging is now controlled using an external configuration file, `logback.xml`, which is located in the webapps `\BravaServer\WEB-INF\classes` directory.

Line Details:

You can find the explanation of each section following the XML file sequence.

The first line in the default `logback.xml` automatically re-scans the logging configuration every 30 seconds, so changes can be picked up without restarting the server.

```
<configuration scan="true" scanPeriod="30 seconds">
```

These properties set the filenames and folder paths for both logs.

```
<property name="SERVICE_FILES_PATH" value="C:\\Program Files\\OpenText\\Brava!
Enterprise\\Brava! Server"/>
<property name="LOG_BASE_NAME" value="brava-enterprise-service"/>
```

The following configures a rolling file appender for the Brava! Server main log and allows the log files to be sifted by day and split into multiple files for a day when the log size exceeds 2 MB. You might notice the patterns between all the loggers are the same. An appender specifies how you want to output log entries in terms of format, files (or console) and other similar rules.

```
<appender name="rolling" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${SERVICE_FILES_PATH}/log/${LOG_BASE_NAME}.txt</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${SERVICE_FILES_PATH}/log/${LOG_BASE_NAME}-%d{yyyy-MM-dd}.
%i.txt</fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy 
class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>2MB</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
        </rollingPolicy>
        <encoder>
            <pattern>%d{"yyyy-MM-dd'T'HH:mm:ss,SSSZ"} [%thread] %-5level %logger{36} - 
%msg%n</pattern>
        </encoder>
    </appender>
```

The “truncating file appender” creates a new log (and empties the old one) each time the Brava! Enterprise Service is restarted. The configuration is basically the same as the previous appender minus the daily log and file size factors, plus an <append> element. This appender is commented out by default.

```
<!-- truncating file appender - creates a new log file each time the service is 
started -->
    <appender name="file" class="ch.qos.logback.core.FileAppender">
        <file>${SERVICE_FILES_PATH}/log/${LOG_BASE_NAME}.txt</file>
        <append>false</append>
        <encoder>
            <pattern>%d{"yyyy-MM-dd'T'HH:mm:ss,SSSZ"} [%thread] %-5level %logger{36} - 
%msg%n</pattern>
        </encoder>
    </appender>
```

This is the console appender. It allows messages to be written to the console. It uses the same log formatting pattern as the previous two appenders. The pattern in this example is for JRE7.

```
<appender class="ch.qos.logback.core.ConsoleAppender" name="stdout">
        <encoder>
            <pattern>%d{"yyyy-MM-dd'T'HH:mm:ss,SSSXXX"} [%thread] %-5level %logger{36} 
- %msg%n</pattern>
        </encoder>
    </appender>
```

The default logging level is set in the “root” element of the `logback.xml` file, but the root can be overridden with specific settings. Some examples exist, but are commented out in `logback.xml`, to get the user started.

```
<!-- type-specific settings override log level specified in root element -->
    <logger name="com.igc.be" level="error" />
    <logger name="org.springframework.beans" level="error" />
    <logger name="org.springframework.jdbc" level="error" />
    <logger name="org.springframework" level="trace" />
```

This section sets the default logging level and is equivalent to *`log.level`* in `server.properties` of previous releases, however the values are new. (See following table.) The log levels are sent to the `logback.xml` appenders.

The equivalent logging levels are mapped as follows:

Previous java.util	New slf4j
ALL	ERROR
FINEST	TRACE
FINER	TRACE
FINE	DEBUG
CONFIG	INFO
INFO	INFO
WARNING	WARN
SEVERE	ERROR

To change the level, go to the root element within the `logback.xml` file and change the level property using one of the these slf4j values. In the following example, the default of `error` was changed to `warn`.

Example:

```
<root level="warn">
<appender-ref ref="rolling" />
<appender-ref ref="stdout" />
 </root>
```

4.2.4 Advanced logging

4.2.4.1 Properties

The following context-based properties are available that can greatly simplify troubleshooting client-server interactions.

HOSTNAME
Logback's built in HOSTNAME property can be used to encode the NETBIOS name of the service host in each log entry. At the expense of bloating local log files, this can simplify the amount of processing required by management agents. This property is always available. This property must be formatted as `${HOSTNAME}`.

Request Properties: With a properly configured **ContextLoggingFilter** (see "Configuring the ContextLoggerFilter" on page 84 for instructions), the following properties are available for each inbound request.

These request property names must be formatted as `%X{req.remoteHost}`.

req.remoteHost
The IP address of the computer that originated the request.

req.user.Agent
The user agent that originated the request.

To write these properties to the log, they must appear in a pattern element in your `logback.xml` file.

Example 4-1:

```
<pattern>%d{"yyyy-MM-dd'T'HH:mm:ss,SSSZ"} [${HOSTNAME}] [%thread] [%X{req.remoteHost}]
[%.-15X{req.userAgent}] %-5level %logger{36} - %msg%n</pattern>
```

When no requests are pending, log entries appear as (note empty braces for host and user-agent):

```
2014-09-19T12:17:09,214-0700 [zardoz] [Timer-4] [] [] DEBUG
c.i.httpservice.RetractableArm - FSM reports idle state.
```

When processing a request, log entries take the following form:

```
2014-09-19T12:17:21,346-0700 [zardoz] [Timer-4] [172.12.0.109] [Java/1.6.0_4] DEBUG
c.i.httpservice.MotorEmulator - FSM reports ignitions engaged.
```

4.2.4.2 Configuring the ContextLoggerFilter

The **ContentLoggingFilter** is disabled by default. To enable *`req.remoteHost`* and *`req.userAgent`* in your environment, follow these steps:

1. Stop the Brava! Enterprise Service.
2. Copy the pattern element from the "Usage" example in "Properties" on page 83 into `./BravaServer/WEB-INF/classes/logback.xml`. Place it within the appenders that you want to capture these properties for. Modify the pattern to suite your needs. This console appender uses the pattern we defined earlier (see Logback.xml details):

   `logback.xml`

```
<appender class="ch.qos.logback.core.ConsoleAppender" name="stdout">
    <encoder>
        <pattern>%d{"yyyy-MM-dd'T'HH:mm:ss,SSSZ"} [${HOSTNAME}] [%thread]
[%X{req.remoteHost}] [%.-15X{req.userAgent}] %-5level %logger{36} - %msg%n</
pattern>
    </encoder>
</appender>
```

3. Install the **ContextLoggingFilter** by removing the comments ("<!--" and "–>") that surround it in `./BravaServer/WEB-INF/web.xml`.

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>com.igc.be.appconfig.spring.BravaEnterprise</param-value>
  </context-param>
```

```
<!--
  <filter>
    <filter-name>contextLoggingFilter</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
  </filter>
  <filter-mapping>
    <filter-name>contextLoggingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
 -->

  <filter>
    <filter-name>iPMonitorFilter</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
```

4. Start the Brava! Enterprise Service.

4.2.4.3 Filtering irrelevant stack trace lines in logs

When an exception error is logged, a huge number of stack trace entry lines are printed. The logs can be adjusted to filter out various stack trace entries through configuration of the `pattern` element in your `logback.xml` file.

Example 4-2: Filtering of package or class

```
<pattern>%d{"yyyy-MM-dd'T'HH:mm:ss,SSSZ"} [%thread] %-5level
%logger{36} - %msg%n %m%n%rEx

{full,org.springframework,org.apache.catalina,java.lang.reflect.
Method,jdk.internal.reflect}

</pattern>
```

Example 4-3: Full stack trace

```
<pattern>%d{"yyyy-MM-dd'T'HH:mm:ss,SSSZ"} [%thread] %-5level
%logger{36} - %msg%n</pattern>
```

4.2.4.4 Execution timing services

An Execution Timing logger is available in the Brava! Server which logs Execution Times of important endpoints (services). This logger is important for understanding trends in performance under different load conditions, over time. If you are using the third party Splunk Enterprise application with Brava! Enterprise, it is the driving force to many of the Dashboards in the Splunk app.

By default, this service is turned off.

When activated, timing entries for many important services are logged to `Brava! Server/log/brava-enterprise-executiontime-monitor.txt`. The Execution Timing Service messages of interest are queued and written on a dedicated thread at a later time, allowing execution to continue unencumbered by IO.

To turn on execution timing for Splunk:

1. In the `server.properties` file, the `Splunk` profile must be added to the *`spring.profiles.activate`* property.

   For example: `spring.profiles.activate=OnPremise.FileSystem,async, Splunk`

2. Restart the servlet container that the BravaServer webapp is running in.

3. In the `/BravaServer/WEB-INF/classes/logback.xml` file, set the *`level`* attribute of the `executiontime-monitor` element to `TRACE` (which is its preset value). If the level is set to any value other than `TRACE`, then no entries are logged to the `/ brava-enterprise-executiontime-monitor.txt` file.

   The server does not require a restart after changing this value, providing a convenient method for toggling execution timing logging on and off.

The format of entries logged to the `brava-enterprise-executiontime-monitor.txt` file is:

<timeStamp> <logLevel> <clientIP> <timestampBegin> <timestampEnd> <executionTime> <timingobjectID>

Example:

```
2016-04-22T09:51:24,599-0700 TRACE 192.168.1.12 1461343884250 1461343884598
SetActiveXViewerParams 0.348
bd943d59aa30f7ede40da0fbbdbfb390062c4ed2609ec38aa2f18812961c83f3
```

Where the fields are interpreted as:

`Timestamp` of actual writing of the statement

`Log Level` - always TRACE

`Client ip address` - sometimes it is “Unknown” - when it is a timing statement of an internal service

`Timestamp Function Begin` - *`java System.currentTimeMillis()`* value at beginning of service

`Timestamp Function End` - *`java System.currentTimeMillis()`* value at end of service

`ExecutionTime` - in seconds of the service

`Timing Object Id` - can be *`requestid`*, *`sessionid`*, *`compositionid`*, *`docid`*, etc. - these are internal ids used for correlating timing events in the Splunk app

4.2.5 Configuring Brava! Server for HTTPS

To configure your server for SSL using HTTPS, it is best to specify the correct protocol (HTTPS) and port in the appropriate fields at the time of installation.

If the installation was initially run specifying to use HTTP, numerous paths must be updated in the properties files location in the installation folder (`C:\Program Files\OpenText\Brava! Enterprise\Brava! Server`). In addition, the viewer sample files location within the BravaSDK must be updated to work with HTTPS, and numerous paths must be updated in the various sample JSP files.

4.3 Configuring the JobProcessor

4.3.1 Editing the Job Processor configuration file

This section describes the various parameters in the `JobProcessor.config` file, located in your installed Job Processor directory `C:\Program Files\OpenText\Brava! Enterprise\JobProcessor` by default). This file controls different variables in the Job Processor software product. The default values are shown in the given examples. Changing any of these values requires a restart of the Job Processor.

The parameters at the end of this section (starting with *`queue.url`*) are defined by queue. The properties for a particular queue have a queue number appended to them, for example: *`queue.url.2`*, *`jobgetter.classname.2`*, *`job.sleep.2`*, *`job.error.sleep.2`*.

Table 4-6: Optional Job Processor Parameters

Parameter	Description
`log`	Enables or disables job logging by the Job Processor. The log file location is displayed in the Job Processor Console window and is typically `C:\Documents and Settings\<JP USER ID>\local settings\application data\Informative Graphics Corp\` or `C:\Users\ <JP USER ID>\AppData\Local\Informative Graphics Corp\rie_nie_logfile.xml` Logs can be viewed by visiting `http://JPMachine:7070/log`. See "Monitoring the Job Processor" on page 98. **Default:** `log=off`

Parameter	Description
`log.size`	The maximum number of jobs that are written to the log file at any time. Older jobs are removed from the log as newer jobs are processed. If performance is an issue, logging can be turned off by setting the *`log=off`* parameter. **Default:** `log.size=200`
`dataport`	Used for monitoring purposes, this is the port that the Job Processor uses to listen for browser connections. Point your browser at `http://<JPMachine>:7070/config` to view configuration information. **Default:** `dataport=7070`
`thread.drw`	The (unlimited) number of jobs from the DRW queue to concurrently publish. **Default:** `thread.drw=3`
`thread.pdf`	The (unlimited) number of jobs from the PDF queue to concurrently publish. **Default:** `thread.pdf=3`
`thread.single`	The number of jobs that simultaneously publish single page requests. Brava! sends a single page publishing request (as opposed to the standard full document publishing request) for any file type (that it has a native driver for) that can be more than one page in length. Single page requests are sent in order to expedite a response to the client for a requested page. **Default:** `thread.single=3`
`thread.doc`	The number of jobs from the DOC queue to publish. This value should not be modified unless you are licensed for, and using, Outside-in of OTF technology. **Default:** `thread.doc=2`
`thread.3d`	The number of jobs from the 3D queue to publish. 3D is assigned to any publishing request for file types listed in the server's *`publish.request.extensions.3d`* properties. **Default:** `thread.3d=3`

Parameter	Description
`queue.url.#`	The hostname of the Brava! Server which has an internal publish request queue. Used by the *`IGC.JobProcessor.PollingHttpJobGetter`*. **Example:** `queue.url.0=http://MyMachine.MyDomain.com:8080/BravaServer/Pop`
`jobgetter.classname.#`	The name of the Job Getter used to retrieve jobs for the Job Processor to process. There can be more than one job getter for each Job Processor. Example for one Job Processor to work with Brava! Server: `jobgetter.classname.0=IGC.JobProcessor.PollingHttpJobGetter` `queue.url.0=http://BUILDBE70:8080/BravaServer/Pop` `job.sleep.0=1500`
`job.sleep.#`	The time (in milliseconds) between querying the Job Processor queue for jobs awaiting publishing. This property has relevance when the query has successfully retrieved a job, or has timed out. **Default:** `job.sleep.#=0`
`job.error.sleep.#`	The time (in milliseconds) between querying the Job Processor queue for jobs awaiting publishing after an error is received from the previous query. A delay here reduces the amount of job requests from the Job Processor when the Brava! Server is in a bad state. **Default:** `job.error.sleep.#=1500`

4.3.2 Loader Configuration Tool

Brava! recognizes supported file formats, their file extensions, and any related parameters, some of which can be configured through the Loader Configuration Tool.

Prior to the 7.3 product version release, loader configuration was done through the `myrdrv.ini` INI file. This single file contained all of the different parameters for all of the different loaders, as well as the association information for which loaders should be used for which extensions. This file required manual editing for any changes needed.

The current Loader Configuration Tool (`loaders.configuration.exe`) is a standalone executable that provides a simple user-friendly interface to the multiple XML configuration files that control loader behavior. As the Tool starts up, it compares the latest installed “source” loader configuration files with the current user's “target” loader files. If it finds a difference in versions, it updates the user's files with the installed source files and reports any changes found.

For usage details, loader updating, and more information, refer to the tool's documentation: *OpenText Brava! - Loader Configuration User Guide (CLBRVW-ULC)*. The executable and its properties file are located off the product's install directory in the `..\JobProcessor\Igc.Loaders\` folder.

4.3.3 Configuring the Job Processor to process particular request types

The Brava! Server publishing requests that are sent to the Job Processor are divided into various types. These types are primarily defined by the *`publish.request.types`* property in the server's `server.properties` file (by default DRW, DOC, and PDF). The other type of publishing request the Brava! Server sends to the Job Processor beyond those listed in this property is *single*. With single, Brava! sends a single page publishing request (as opposed to the standard full document publishing request) for any file type that has a native driver and that might be more than one page in length. Single page requests are sent in order to expedite a response to the client for a requested page.

It is possible to configure a Job Processor to listen only for particular publishing request types. For instance, to improve client response time, a separate Job Processor machine can be configured to listen only for publishing requests of type single. This configuration is accomplished by removing the *`thread.*`* values for all types except *`thread.single`*. Therefore, your `JobProcessor.config` file will contain only *`thread.single=1`*, and the lines *`thread.drw`*, *`thread.doc`*, and *`thread.pdf`* will be removed. Similarly, if only one machine is licensed with a particular set of applications used for print-publishing those application formats, that machine would be configured to listen for request types configured for that application’s extensions, while a second Job Processor machine would exclude that request type.

Related Topics

- “Setting up multiple Job Processors for a single Brava! Server” on page 96

4.3.4 Automatic File Recognition

When Brava! Enterprise tries to load a file with no extension or an undocumented extension (an extension which is not listed in the Loader XML configuration files), Brava! Enterprise tries to determine the file format. The `autorecognize tests.txt` file needs to be edited to add a test for the file format. The text file includes detailed instructions on how autorecognize works and how to edit the file to add new tests. The file is located in the Job Processor install directory. For example, `C:\Program Files\OpenText\Brava! Enterprise\JobProcessor)`

4.3.5 Using Xrefs with the Job Processor

You can use Xref files in conjunction with the Job Processor. Xrefs are files that are referenced by any number of DWG files and allow you to add common information to only one file. This eliminates the repetitive task of adding identical information to multiple files. To use Xrefs with the Job Processor, you must modify the Xref path setting in the Job Processor installation.

To set Xref paths:

1. Browse to the installed Job Processor directory (`C:\Program Files\OpenText\Brava! Enterprise\JobProcessor` by default).
2. Run the Loader Configuration Tool (see *OpenText Brava! - Loader Configuration User Guide (CLBRVW-ULC)*).
3. In the [DWG2DL] section, update the following parameter to point to your Xref file directory:

   ```
   XrefPath=c:\ACAD\xref
   ```

4.3.6 Publishing using other applications

To publish file formats that are not on the Supported File Format list (see "Supported file formats" on page 117), an application that can print the format you want to publish must be available to the Job Processor. Generally, it is best to install the applications directly on the Job Processor machine since most applications register a **Print** or **PrintTo** command with the operating system for their file formats. If the application is shared on another machine, you must register the **PrintTo** action on the Job Processor machine to point to that application. The application does not have to be running; the Job Processor automatically launches the application to perform the conversion and shuts it down when the job is complete. If needed, the application must have the initial setup done after installation so it opens the desired file type without error (Outlook MSG files, for example).

Note: If the native application is not installed, or if the installed native application does not have a **Print** or **PrintTo** action, then you must create a jdocprint action.

Perform the following steps to register a **PrintTo** action in Windows. This example sets up the OpenText Brava! Desktop client application to publish Microsoft Word documents.

1. Open Windows Explorer.
2. From the **Tools** menu, select **Folder Options**.
3. In the **Options** dialog box, click the **File Types** tab.
4. In the **Registered File Types** list box, find and highlight the format you want to create a **PrintTo** action for. (If it is not there, continue following these steps to create a new type.)
5. Click **New**.
6. In the **File Extension** text box, type the extension for the format and select **OK**.
7. Choose **Advanced**.
8. Choose **New** to add an action and type `jDocPrint` in the **Action Name** field.
9. In the **application used to perform action** box, type the full path to the application's executable file. Some applications require additional command line switches or commands in this line. See the documentation for the application you are using for information about setting up this command. For example, `"C:\Program Files\OpenText\Brava! Desktop\BravaDesktop.exe" -dde ff_load("%1") -dde print(EXECUTE_PRINT_QUICK) -dde delete_ item("%1")`.
10. If necessary, select **Use DDE** and fill in the necessary boxes according to the application's instructions.
11. Choose **OK**, and then **Close**.

Publishing Microsoft Outlook MSG files

To publish MSG files, Microsoft Outlook needs to be installed and initially set up on the Job Processor machine with *no* personal mail account information set and *no* Exchange server information set. After this initial setup, close Outlook. Processing MSG files should now launch in Outlook and successfully print-publish.

You must perform this initial setup to publish MSG files, otherwise, they do not publish and a critical error message is logged.

Since the CSF Writer might forcefully close Outlook during the print publishing process, there should be no email accounts configured in the Outlook application on the Job Processor machine because those accounts can become corrupted. If the Job Processor is having problems converting MSG files, do the following:

1. Shut down the Job Processor.
2. Make sure CSF Writer and Outlook processes are not running (go to the Windows **Task Manager** and end the processes if they show up there).

3. Manually attempt to open Outlook and make sure it can launch without reporting errors.
4. Close Outlook.
5. Restart the Job Processor.

Outlook attachments

By default, when viewing an Outlook message file that contains attachments, if any of the attachments is of an unsupported file format, a warning displays and that file attachment is ignored. Any supported attachments for that message are displayed and the Outlook message is processed. If you prefer to disallow the loading of any Outlook message files that contain one or more unsupported attachments, a setting is available which can be modified to prevent the message file from loading if unsupported attachments are detected.

To do this, run the loader configuration tool and edit the *[Eml2dl]* parameter *`IgnoreErrorsOnAttachments`*. See "Loader Configuration Tool" on page 89. When set to FALSE, the message file is not loaded if an unsupported format is detected.

Publishing multiple Excel worksheets

In order for the Job Processor to publish all worksheets in an Excel document (instead of only the first sheet), the following two registry settings must be modified:

```
HKEY_CLASSES_ROOT\Excel.Sheet.8\shell\printto\ddeexec
```

Modify:

```
[open("%1")][print(1,,,,,,,,,,,2,"%2")][close()]
```

To

```
[open("%1")][print(1,,,,,,,,,,,3,"%2")][close()]
```

and

```
HKEY_CLASSES_ROOT\Excel.Sheet.8\shell\printto\ddeexec\ifexec
```

Modify:

```
[open("%1")][print(1,,,,,,,,,,,2,"%2")][quit()]
```

To

```
[open("%1")][print(1,,,,,,,,,,,3,"%2")][quit()]
```

Caution

Extreme care should be taken when modifying the Registry as incorrect changes can cause serious problems that might require reinstallation of Windows. Your Registry settings can be backed up by clicking on **My**

Computer in Regedit and choosing **Export Registry File** from the **File** menu.

4.3.7 Configuring CSF Writer for the Job Processor

CSF Writer contains the API through which the Job Processor accesses and publishes documents. There are INI settings that enhance the performance of publishing various file formats. The registry keys are listed in the following table, along with the default settings. The time values represent seconds.

In a server configuration, the relevant `BIPrint.ini` file is the one located within the CSF Writer install path (typically at `C:\Program Files\OpenText\Brava! Enterprise\CSFWriter\BIPrint.ini`).

Refer to *OpenText Brava! - CSF Writer Publishing Guide (CLBRVW-UCW)* for detailed information about these settings.

4.3.8 Configuring the Job Processor for Brava! Server

The Job Processor is a scalable conversion solution used with Brava! Enterprise to publish documents and drawings to the proprietary common display list format. End users select a file to view through their browser which is ultimately sent to the Job Processor for publishing. The published file then displays in the end user's browser using the Brava! Enterprise clients. The Job Processor can be used to publish any file type. To convert files, it either uses drivers, which directly read the binary files, or uses print drivers to print the file from its native application.

The Brava! Server caches files published by Job Processor, allowing subsequent loads to be faster.

Note: When the cache directory exceeds its allotted size (which is configurable), older files are deleted.

Editing the Brava! Server `server.properties` file

For the Brava! Enterprise cache to work with the Job Processor, the following required parameters must be set. For details about other optional custom parameters you can set, refer to "Configuring the Brava! Server properties" on page 71.

In a text editor such as Notepad, open the file `server.properties` which resides in the installed Brava! Server directory (`<Windows Drive>:\Program Files\OpenText\Brava! Enterprise\Brava! Server`) by default. The installation sets the values of the various server parameters based on information you provide during the installation. If incorrect values are entered, or if you need to change your configuration, edit the following parameters to your specific environment:

```
# Where the cached files reside within the Brava! Server
#NOTE: This setting must be absolute, including shared folder name if the Brava!
Enterprise publisher (Job Processor) is NOT on the same machine as the Server
```

```
displaylist.cache.root=\\\\$APP_SERVER_NAME$\\dlcache
# URL of the servlet of the Job Processor to hit for file publish completion
done.servlet.url=http://BravaServer:8080/BravaServer/Publish
```

Important

The port number entered for the URL properties should match the port number used by your servlet container.

Related Topics

- "Job Processor Port configuration" on page 95

4.3.9 Job Processor throughput

In the process of converting files, the Job Processor writes out many temporary files to the Windows temp folder, which by default is on the c: drive. For example, `c:\windows\temp` . The throughput of the Job Processor is limited by the write speed of the temp folder. Significant improvements in performance can be made by setting the `%TMP%` and `%TEMP%` environment variables to a folder on a driver with high IO, such as a disk based on 6 drives configured for RAID 0.

4.3.10 Job Processor Port configuration

The Job Processor receives jobs from the Brava! Server using the server's HTTP interface. By default, this is port 8080 on the Tomcat server, so port 8080 must be open on the Brava! Server and available to the Job Processor (configured through the Job Processor parameter `queue.url.0=http://localhost:8080/BravaServer/Pop)`.

To change the port Tomcat uses:

1. Open the folder <*Tomcat install*>`\conf` on your computer.
2. Using a text editor, open `server.xml` and edit the following line for the port that you want to use:

   To use port 8888 instead of 8080, change:

   ```
   <Connector port="8080" protocol="HTTP/1.1"
   ```

   to:

   ```
   <Connector port="8888" protocol="HTTP/1.1"
   ```

Related Topics

- "Connections" on page 104

4.3.11 Setting up multiple Job Processors for a single Brava! Server

It's possible (and often preferred) to have multiple Job Processors set up to provide publishing capabilities to a single installation of Brava! Server. This section will help you to successfully distribute publishing responsibility.

CSF Writer note:

Publishing with CSF Writer is designed to process one document at a time, for each machine, even when multiple publishing threads are configured.

To improve throughput with CSF Writer publishing methods, such as using the Bi2dl loader, additional Job Processor machines can be added. Alternatively, some formats can be configured with other loaders, `Otf2dl` for example, that can be run in parallel on a single Job Processor machine. See *OpenText Brava! - Loader Configuration User Guide (CLBRVW-ULC)*.

Background

Although often referred to as the "Job Processor", this implies an installation of the Job Processor and its associated Brava! Server. Each Brava! Server manages publish requests, and any number of Job Processors query the Brava! Server for available publishing "jobs" they can handle. This architecture allows publishing power to be increased to accommodate very small to very large systems with very little administrative responsibility.

An added feature of the Job Processor component is that it can be configured to handle only specific types of jobs. This can allow more complicated arrangements where, for example, a few very fast computers with very fast network connections are set up to handle PDF publishing, while a few slower computers can handle the relaxed requirements of other file types. Or, it can be used to direct types that require Microsoft Office to be installed to only a small subset of computers for which that product has been purchased, while using other machines to handle other types.

One unique ability of Brava! is that often the first thing a user sees after requesting a document is the result of a single-page publish. When Brava! Server receives a request to display a document, it sends two publish requests: one single-page job and one complete-document job. In some cases (with HTML Client publish requests for example), a thumbnail request is also sent. The single-page job normally completes in a very short amount of time and is delivered to the client immediately upon completion. For as long as it takes to publish the complete document, any time another page is requested by the client, another single-page job is sent for processing. After the complete document job is completed, the result is cached and the Brava! Server then uses those results directly to deliver the pages to the client, no longer requiring it to submit single-page job requests.

Prerequisites

Before creating multiple Job Processors to work with one Brava! Server, the Brava! Server must be set up. These can normally be installed at the same time, using the same installation program, and most often on the same computer. Once a Brava! Server is installed and running, a Job Processor can be installed elsewhere (or on the same machine) and directed to connect to the Brava! Server.

! **Important**
All Job Processors must be UNC-accessible to and from the Brava! Server to allow the configuration to work properly. Although requests are handled using HTTP, actual file transfers occur using the UNC mechanism.

Setup

Perform the following steps for each Job Processor that will communicate with a single Brava! Server.

1. Using the installation program, install the Job Processor as described in "Installing the Job Processors" on page 24. This can be on any computer system that has both HTTP and UNC access to the machine on which the Brava! Server is installed.

2. Verify the installation by checking to see that the program now resides on the machine (typically at `C:\Program Files\OpenText\Brava! Enterprise\JobProcessor`).

3. In the `JobProcessor` directory, locate the `JobProcessor.config` file. Open it and acquaint yourself with the following sections:

```
thread.single,thread.drw,thread.pdf,thread.doc,queue.url.0
```

4. To direct the new Job Processor to connect to the existing Brava! Server, change the *`queue.url.0`* property. The server name and perhaps the port number will change, but the ending portion `/BravaServer/Pop` will remain the same. For example, if you installed Brava! Server on a computer called "phoenix", with the servlet engine (normally Apache Tomcat) using port 8080, then this property will have the value `http://phoenix.domain.com:8080/BravaServer/Pop`.

5. To define which types of files this Job Processor can handle, change the single, DRW, PDF, and DOC thread properties:

   a. The Job Processor threads define which groups of files this Job Processor can handle. These groups are declared in the *`publish.request.types`* property and defined in the *`publish.request.extensions.*`* properties found in the `server.properties` file in the Brava! Server installation directory. For example, `C:\Program Files\OpenText\Brava! Enterprise\Brava! Server`. Each group is defined as a comma-separated list of file extensions.

   b. To allow only a certain group of file types, list only that group in the `jobprocessor.config` file, and comment out all the other thread properties (start the line with a "`#`"), leaving the ones that match the thread properties you want to keep.

For example, suppose you want this Job Processor to handle only PDF and DRW file groups. The thread properties would look like this:

```
#thread.single=5
thread.drw=5
thread.pdf=1
#thread.doc=1
```

Modifying the thread counts, depending on the count of cores/virtual cores on the server, can make the Job Processor more efficient.

6. In some cases, it is also desirable to handle single-page jobs separately from complete-document jobs. This is often the case when large PDF files are the most common document type. Since these take a considerable amount of time to process, it becomes important that single-page publishing jobs occur on a different machine than the complete-document job, so that it executes more quickly.

   a. To set up single-page publishing, you use the *`single`* thread group. It is treated the same as the other thread groups, except that it encompasses all file types and limits to single-page requests.

   b. This set of properties would enable only single-page requests to be handled by the Job Processor:

   ```
   thread.single=5
   #thread.drw=5
   #thread.pdf=1
   #thread.doc=1
   ```

   c. This set of properties would instruct the Job Processor to handle all types EXCEPT whole-document PDF jobs (but would still handle single-page PDF jobs):

   ```
   thread.single=5
   thread.drw=5
   #thread.pdf=1
   thread.doc=1
   ```

4.3.12 Monitoring the Job Processor

Each Job Processor provides status monitoring, accessible by pointing a browser at the Job Processor machine. By default, the Job Processor listens to port 7070 for browser connections. You can change this port in the `JobProcessor.config` file using the *`dataport`* parameter. The `JobProcessor.config` file can be updated directly to modify the Job Processor configuration. To make changes, stop the Job Processor, edit the file, and restart the service in order for the changes to take effect.

The following pages are available to monitor the Job Processor:

`http://JPMachineName:7070/stats` - This page shows general statistics for the jobs processed by the Job Processor.

`http://JPMachineName:7070/log` - A log of the publish results for each job the Job Processor has processed. This is empty unless the value for *`Log`* is set to `ON` in the `JobProcessor.config` file. This page is available with information only after logging is enabled in the `JobProcessor.config` file, the service is restarted, and the file is processed by the Job Processor.

This page shows the Job Processor Configuration.

`http://JPMachineName:7070/config`

This page shows the Job Processor Service Log.

`http://JPMachineName:7070/servicelog`

4.4 Configuring the Brava! License

The Brava! License is installed as a Servlet running inside a servlet container, such as Tomcat. It needs to be started simultaneously or prior to the Brava! Server in the same manner as the Brava! Server. It needs to be accessible to the Brava! Server and can be installed within the same Servlet container.

Failover to the other listed Brava! License Servers in `server.properties` functions if any particular Brava! License Server goes down (see "Configuring multiple Brava! License Servers" on page 99).

4.4.1 Configuring multiple Brava! License Servers

It is possible to configure Brava! Server to connect to multiple Brava! License Servers to provide redundant failover capability in the case of accessibility loss to any one Brava! License Server. To configure Brava! Server to communicate with more than one Brava! License, add the URL of each additional Brava! License Server (comma delimited) to the *`lc.url`* property located in the `server.properties` file.

Example 4-4: lc.url

```
lc.url=http://machine1:8080/BravaLicense,http://machine2:8080/
BravaLicense,http://machine3:8080/BravaLicense
```

Ensure that each URL is specified such that the Brava! Server can correctly identify and communicate with the Brava! License. This can require using a fully-qualified domain name.

When a client license is required due to a user request to view a document using Brava!, the Brava! Server attempts to obtain a license from one of the listed Brava! License Servers. If communication with the Brava! License Server fails, another is tried until either a license is obtained or the list of Brava! License Servers is exhausted. If communication with a Brava! License Server is lost, the Brava! Server should regain access shortly after communication is restored without requiring a restart of the Brava! Server.

While not required, it is recommended to initialize the Brava! Server after a restart. This can be accomplished by making the following call to the Brava! Server from any web browser located on a machine with an IP address in the allowable range for obtaining a session ID (see the *legal.ip.to.get.sessionid* property in the `server.properties` file).

```
 http://<bravaserver>:<port>/BravaServer/Secure?igcmethod=session&
sessionlife=28800000
```

If initialization is successful, the response returned is a 32-bit integer. In a load balanced environment with multiple Brava! Servers, this call should be made to each individual Brava! Server, bypassing the load balancer. If no Brava! License Servers are available at the time of initialization, the Brava! Server initializes incorrectly due to the lack of a valid server license and requires a restart, even though Brava! License Servers can be subsequently made available.

4.4.2 Configuring Brava! License monitoring preferences

Defining preferences for receiving license-related notifications according to your desired criteria allows you to properly plan for changes (licenses, deployments, or upgrades) without incurring usage disruptions.

You can monitor license utilization and notify administrators in the event that configured thresholds of license usage are exceeded. Checks of license utilization are initiated by the notification service based off the configured *notification.frequency* and if the *notification.thresholdPercentage* is exceeded for any client license, an email is sent to each email address from within the *notification.adminEmails* property (comma separated, no spaces). Emails are sent using the *notification.smtpServer* utilizing the *notification.fromEmail* property.

The following configurable properties can be specified in the `Brava! License/notification.properties` file:

```
notification.adminEmails=$COMMASEPARATEDNOTIFICATIONEMAILLIST
notification.smtpServer=$NOTIFICATIONSMTPSERVER
notification.thresholdPercentage=$NOTIFICATIONTHRESHOLD
notification.frequency=$NOTIFICATIONFREQUENCY
notification.fromEmail=$NOTIFICATIONFROM
notification.licenseServletAddress=$LICENSESERVLETADDRESS
```

SMTP Server Address: (*notification.smtpServer*) The host address of the smtp server (`smtp.host.com`).

License Servlet Address: (*notification.licenseServletAddress*) All license utilization information is retrieved from the valid /License servlet address defined in this property value.

Example 4-5: License servlet address

```
notification.licenseServletAddress=http://bravaserverurl.com:8080/
BravaLicense
```

Email Addresses: (*enotification.adminEmails*) A comma separated (no spaces) list of email addresses to which notifications are sent.

Example 4-6: Email address

```
notification.adminEmails=jdoe@mydomain.com,bjohnson@mydomain.com,
csmith@mydomain.com
```

From Email: (*notification.fromEmail*) Specifies the email address from which notifications are sent using the notification SMTP server.

Example 4-7: From email

```
notification.fromEmail=noreply@mydomain.com
```

Threshold percentage: (*notification.thresholdPercentage*) A threshold expressed as a percentage in the range [0-100]. When license usage for any given license capability set exceeds that threshold, an email is sent, at the notification interval, to the specified email address list. If set to 100, a notification is only sent when license requests are denied because there are no available licenses.

Example 4-8: Threshold percentage

```
notification.thresholdPercentage=80
```

Frequency of Notification: (`notification.frequency)` A minimum frequency, specified in hours, with which an email is sent. For example, if the value is set to 12, notification emails are sent every 12 hours at most. Email is not sent each time the threshold is exceeded within that twelve hour period. Only positive integers can be specified.

Example 4-9: Notification frequency

```
notification.frequency=2
```

Notes

- Administrators receive a notification for each client license capability set where the threshold is exceeded. The notifications are not for the cumulative set of client licenses in the system.
- The email notification sent is formatted as follows:

  Subject: Brava! license threshold exceeded

Body: One or more of your Brava! Enterprise client license has reached the maximum threshold of <threshold> utilization on the license server `http://<servername>:<serverport>/BravaLicense/License`.
If this happens regularly, you might be in need of additional licenses. Please contact your account manager.

4.5 Configuring Brava! Publication Cache

The Brava! Publication Cache feature allows documents published by Brava! Enterprise to be cached for faster future access.

4.5.1 File system cache

The cache, sometimes referred to as the dlcache or displaylistcache, is a set of folders under a single parent folder. Brava! maintains a set of indices of all content it knows about in the cache. Brava! maintains these indices in memory when it is running and periodically copies them to disk. The copies on disk are read into memory when Brava! starts. Brava! does not rebuild the indices by reading the contents of the cache directory. Unless otherwise specified, all settings are in the `server.properties` file in the Brava! Server install directory.

4.5.1.1 Cache truncation

The Brava! file system cache is designed to be transitory in nature, and contains only the artifacts needed to render the most recently viewed documents. As such, it can be set to have a maximum size. When the size grows above that point, older content is deleted from the cache to shrink its size below the threshold.

If that content needs to be viewed at a later date, the original source document needs to be uploaded to the Brava! Server and converted again. For usage details of the properties used with file system cache truncation, see truncation server properties in "Configuring the Brava! Server properties" on page 71.

4.5.1.2 Single page publish truncation

If a Client attempts to view a document before it has been completely converted by the Job Processor, Brava! issues a command to the Job Processor to publish a single page of the document that can be immediately viewed.

The results of those single page publish commands are stored separately from the results of publishing the complete document. Cleaning up these cache entries is controlled by the parameter *`single.page.publish.cleanup.delay`*.

4.5.1.3 Brava! deletion of truncated files

Brava! deletes truncated files when the setting *`server.delete.cache.entries`* is set to `true`. In this case, whenever the Brava! Server is notified by the Job Processor that a new publish action has completed, Brava! checks its in-memory cache records to determine if the new content will push the size of the cache over the value specified in *`displaylist.cache.maximum.size`*. If so, Brava! truncates and queues up its oldest cache entry for deletion. After *`number.cache.entries.queued.before.deletion`* entries are queued, Brava! deletes that number of cache entries all at once.

This can result in the size of the Brava! cache temporarily exceeding *`displaylist.cache.maximum.size`* as entries are queued for deletion. No more than *`displaylist.cache.max.files.to.delete`* are deleted during a single Job Processor notification event. If there are fewer files queued for deletion than the value specified in *`displaylist.cache.max.files.to.delete`*, Brava! Server does nothing and waits for another publish completion action. This caching scenario is suitable for most deployments.

4.5.1.4 External deletion of truncated files

Brava! can be configured to allow an external process to delete truncated files when *`server.delete.cache.entries`* is set to false. In this case, when Brava! detects that the cache size exceeds its upper limit, it begins building an internal list of cache entries to delete. When the number of entries in this list exceeds *`cache.entries.to.delete.per.file`*, it writes the folder path of each entry to a file in *`delete.files.root.dir`*, with one entry for each line. Each file is given a TMP extension as it is written and renamed to have a TXT extension when Brava! has finished writing to it.

An external process can monitor this directory for TXT files and delete any cache entries specified within them. That external process is then responsible for deleting the TXT file. This cache deletion scenario should be used in high throughput environments where Brava! might be I/O bound. You can move the deletion of files to a separate process or to a separate server. This can reduce wait time while files are being deleted and will ensure that the Brava! Server remains responsive.

4.6 Configuring Tomcat for high load

When the Brava! Server is running on Tomcat under high load, it might be necessary to update Tomcat's configuration. You can change some key parameters to tune the number of connections and the amount of maximum heap space.

4.6.1 Connections

Historically, request for artifacts that are yet to be published were blocking Tomcat (servlet) threads. Currently, most of these blocking requests are processed asynchronously and the Brava! Server application can work effectively with much smaller maximum amount of threads in the thread pool. Under very high load, clients might experience connection failures. If needed, you can use one of the following options to increase the number of available connections for Tomcat:

Option 1: Increase number of connections in the default connector

1. In Tomcat’s `conf\server.xml` file, locate the `<Connector` element for the port that Brava! clients are connecting to. By default, this port is 8080.
2. Update or create the `maxThreads` attribute with a sufficient value. For example, `maxThreads=1000`.

Option 2: Add a new connector for Job Processor and Brava License communications

1. A new Tomcat connector can be configured for Job Processor and Brava License communications. The main advantage being that it reduces the likelihood for timing out of a Job Processor job or Brava License call. In Tomcat’s `conf\server.xml` file, add a new connector element with a different port than the default one that is configured for the Brava! viewers. For example:

   ```
   <Connector port="8081" protocol="HTTP/1.1" connectionTimeout="20000"
   redirectPort="8443" maxThreads="200" />
   ```

2. Update the Brava! Server port for the properties related to connecting from the Job Processors in the Brava! Enterprise configuration files. For example:

   `server.properties` in the Brava! Server install directory:

   ```
   done.servlet.url=http://<Brava! Server>:8081/BravaServer
   ```

   ```
   lc.url=http://<Brava! Server>:8081/BravaLicense
   ```

   `JobProcessor.config` in the Job Processor install directory:

   ```
   queue.url.2=http://<Brava! Server>:8081/BravaServer/Pop
   ```

   Note: The adjustment to the `lc.url` for Brava License calls is only necessary if the BravaLicense webapp is running in the same servlet container as the BravaServer webapp.

Important

When using Tomcat, you should specify a higher `asyncTimeout` value in the `server.xml` connectors. The default value of 30000 (30 seconds) is the maximum amount of time that the server will wait for artifacts to be published before it will time out the asynchronous HTTP request. Therefore, the `asyncTimeout` value must be at least the value of the publishing timeout value defined in `server.properties` to allow the Job Processor enough time to provide artifacts.

This configuration requires that you manually modify the Tomcat `<TomcatDirectory>/conf/server.xml` file such that the connector includes a line specifying the `asyncTimeout` value desired when using your alternate Tomcat instance.

Example:

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443"
       asyncTimeout="600000" />
```

4.6.2 Memory

For production systems, it is recommended that this be increased to 2048 MB. If a `java.lang.OutOfMemoryError` is reported in the Brava! Server or Tomcat logs, this value needs to be increased.

To configure the maximum heap size for Tomcat on Windows

1. Open the Tomcat configuration GUI application which is located within the Tomcat `bin` directory.
2. From the application, select the **Java** tab.
3. Enter the desired value in the **Maximum Memory Pool** field, then click **OK**.

Chapter 5

Tips and troubleshooting

This chapter contains troubleshooting tips, temporary solutions, and known limitations for using the Brava! Enterprise components.

5.1 Brava! Server tips

- When files publish to XDL and launch through Brava!, you will see a "DisplayList Server" Command Prompt window launch for the first 4 or 5 files that are published (through `BDLGenServer.exe`). This is normal Brava! behavior and the windows can be ignored, or you can choose to view the displayed publishing information for any file generated.
- Ensure that the HTML page to launch the Brava! Control is configured properly (see "Configuring Brava! Clients" on page 53).
- Ensure that the Brava! Server is configured properly (see "Configuring Brava! Server" on page 71).
- Ensure that the servlet engine that hosts the Brava! Server web application is restarted when changes are made to the `server.properties` file. Changes to the `server_precedence.properties` or `client_precedence.properties` also require a restart.
- When Brava! is used in a production environment with Apache Tomcat as the servlet engine, it is recommended that the Apache Tomcat JVM Heap Size setting be increased from the default (128 MB min - 512 MB max) to at least 128 MB min - **1GB** max to prevent the Brava! Server from becoming unresponsive.
- When viewing CAMCAD files converted with the Brava! Client, strings of inverted text in the original file remain inverted.
- Conversion of password protected PDF files is not supported on the server. You can, however, set the parameter *`ConvertOnClient=true`* to convert password protected PDF files on the Client.
- If you would like to customize the search macro file (`search_macros.xml`) by adding to or changing the existing search criteria, the file can be edited in Notepad. The XML file is downloaded with the client CAB file and can be found in your HOMEPATH directory. Be sure to make a backup copy of the XML file before performing any edits.
- The following hotkeys are available for navigating CAMCAD (*.cc) files with the Client (when Select tool is active):

  [mouse click] = select entire geometry assembly and display its

  attributes

  [mouse click] +**[CTRL]** = select specific geometry and display its attributes

[mouse click] + **[SHIFT]** = select a specific piece of a geometry

and display its attributes

5.1.1 Updated DllPath functionality

The *`DllPath`* BravaXparam no longer needs to be set manually as a parameter because the value of this setting is now automatically detected and populated.

The client cab file is downloaded to the Windows system `Downloaded Program Files` sub-directory (for example, `C:\Windows\Downloaded Program Files`), and when a user requests a file for viewing in Brava!, the cab components are installed in the user's HOMEPATH `<user profile>\IGC\<version>` directory.

On a Windows 10 client machine it would look something like:

`C:\Users\<user>\IGC\<version>` and `C:\Users\<user>\AppData\Roaming\IGC\Brava! Client`.

If needed for reference, the following information provides use-case details involving the current placement of the viewer files (previous dllpath use-case) and the persistence settings (previous `dl.xml` settings to the new *`usersettingspath`* `viewconfig.xml` settings).

Use Cases

Example 5-1: Brava! Enterprise / Brava! Client use cases

Single user running Brava! Client for each machine.

With persistence

- Default Install/Download Location and Default Persistence Location. (MOST COMMON)

  `DllPath` (implicitly set to) => `%USERPROFILE%\IGC\<version>\`

  `UserSettingsPath` Not Set => `%USERPROFIILE%\Application Data\IGC\Brava! Client\`

  `PersistUserSettings` Not Set => `true`

- Custom Install/Download Location and Default Persistence Location

  `DllPath` (implicitly set to) => `C:\Program Files\IGC\Brava! Client\<version>\`

  `UserSettingsPath` Not Set => `%USERPROFIILE%\Application Data\IGC\Brava! Client\`

  `PersistUserSettings` Not Set => `true`

- Custom Install/Download Location and Custom Persistence Location

`DllPath` (implicitly set to) => `C:\Program Files\IGC\Brava! Client\<version>\`

`UserSettingsPath` => `C:\Program Files\IGC\Brava! Client\<version>\Brava! Client\`

`PersistUserSettings` Not Set => true

- Default Install/Download Location and Custom Persistence Location

  `DllPath` (implicitly set to) => `%USERPROFILE%\IGC\<version>\`

  `UserSettingsPath` => `C:\Program Files\IGC\Brava! Client\<version>\Brava! Client\`

  `PersistUserSettings` Not Set => `true`

Without persistence

- Default Install/Download Location

  `DllPath` (implicitly set to) => `%USERPROFILE%\IGC\<version>\`

  `PersistUserSettings` => `false`

- Custom Install/Download Location

  `DllPath` (implicitly set to) => `C:\Program Files\IGC\Brava! Client\<version>\`

  `PersistUserSettings` => `false`

Example 5-2: Multiple users running Brava! Client for each machine

With persistence

- Each user gets a unique directory of persistence files and settings.

  Custom Install/Download Location and Default Persistence Location.

  `DllPath` (implicitly set to) => `C:\Program Files\IGC\Brava! Client\<version>\`

  `UserSettingsPath` Not Set => `%USERPROFIILE%\Application Data\IGC\Brava! Client\`

  `PersistUserSettings` Not Set => true

- All users share the same directory of persistence files and settings.

  Custom Install/Download Location and Custom Persistence Location.

  `DllPath` (implicitly set to) => `C:\Program Files\IGC\Brava! Client\<version>\`

  `UserSettingsPath` => `C:\Program Files\IGC\Brava! Client\<version>\Brava! Client\`

  `PersistUserSettings` Not Set => `true`

Without persistence

- Custom Install/Download Location

  `DllPath` (implicitly set to) => `C:\Program Files\IGC\Brava! Client \<version>\`

  `PersistUserSettings` = `false`

Notes

- Custom Install/Download Location requires msi script or custom `bravaclientinstall.bat` to install the Brava! Client pieces. The default `BravaClientXWrapper.cab` only installs into the `%USERPROFILE%\IGC \<version>\` directory.
- Settings files include: `Measure.ini`, `Reasons.ini`, and `ViewerConfig.xml`
- Install/Download files include: `BravaClientX.dll`, `BravaClientXWrapper.dll`, `BravaClientXWrapper_ENU.dll`, `BravaClientXWrapper_ENU.chm`, `Integration_ENU_7.<ver>.dll`, `search_macros.xml`, `ChangemarkConfig.xml`, `BravaClientSkin.xml`, `RedactionScripts/*.xml`, `StampImages/*.png`, etc.
- The directory specified in the *`UserSettingsPath`* must physically exist, for the parameter to work properly. The running Brava! Client does not create new directories for this parameter.
- See the sample html files in the `PersistenceFilesLocation` directory for more specific examples.

5.2 Job Processor publishing tips

- You can increase publishing performance and fidelity in some cases by changing the value of *`forcetojpg`* to TRUE. This setting is contained in the **System** section of the "Loader Configuration Tool" on page 89.
- For advanced troubleshooting data collection, you can either turn on the *`publish.heartbeat`* parameter in the `server.properties` file to generate the `heartbeat.txt` files in the displaylistcache, or enable the Job Processor logging by turning on the *`log`* parameter in the `JobProcessor.config` file.
- **Antivirus exclusions:**

  We highly recommend that you add any server directories that are actively managed by Brava! Enterprise and the Job Processor to the whitelist of your antivirus software. Excluding the directories that hold the display list cache and other Brava! Enterprise components from your antivirus scans can significantly improve performance.

Folders to exclude from antivirus scans:

– The Job Processor temporary directory. By default, this is the system temporary folder.

– The Job Processor data folder. By default, this is: `C:\ProgramData\OpenText\JobProcessor\`

Specifically, the `.request` folder.

- **Updating Loaders:** You should check for availability of the most recent loaders from time to time, and download them to production as a best practice recommendation. Issues are often resolved in loaders, which are released separately from the main product release. You can download the latest from My Support (https://support.opentext.com/):

  Deployment instructions for your setup are provided in the Release Notes document distributed in each loader package.

- **Office file formats:** The Job Processor offers several loaders to process common Office documents which provide a varying range of benefits. The default BI2DL loader, which requires and uses installed and licensed native applications to provide the required content, offers the highest fidelity. Several other loaders are available with supporting libraries; some internal to OpenText and others from third-parties. These include LOTODL, OTF2DL, and the license fee based OutsideIn2DL loader.

  For more information and to review their benefits, see the latest loader and Knowledge Base article for Office based formats on My Support: https://support.opentext.com/csm?id=kb_article_view&sysparm_article=KB0720088

- Add ActiveX URL to the IE trusted sites list or the published file will not appear in the location in which it was saved. When it is added to the trusted sites, the published file is then saved.

  To add the URL to the trusted sites list:

  1. From the Internet Explorer **Options** dialog box, select the **Security** tab.
  2. Click **Trusted Sites**, then click **Sites**.
  3. On the **Trusted Sites** dialog box, enter the URL of the Brava! ActiveX Client and click **Add**.
  4. Click **Close**, then **OK**.

- For the most trouble-free operation of the ActiveX Client with default security settings, the web server launching the client must be a **Trusted Site** (see previous bullet). If security settings aren't the defaults, then the security level can't be the highest allowable (High) and **Protected Mode** must be disabled. The symptom prompting this change is that files associated with the ActiveX Client placed in a Virtualized Profile directory inside `Temporary Internet Files` causes problems accessing components of the client application.

- The default DPI value of embedded raster images is 600 DPI when published to CDL. If you require greater resolution, you must edit the value of *`MaxRasterDPI`* in the **System** section of the Loader Configuration Tool. See also *`max.raster.dimention`* parameter in "Configuring the Brava! Server properties" on page 71.

Ensure all values are correctly entered into the `JobProcessor.config` file:

- Always use UNC paths for the source and target locations on requests.
- Use correct capitalization and spelling on requests. This value must match the corresponding task/variable in the XML file.
- The push URL path used for pre-publishing must be UTF8 URL encoded. Therefore, a push URL that contains Korean or other double-byte characters in the path must first be UTF8 URL encoded before it can work.
- The encoding standards for input files that we support with the TXT2DL loader are ANSI, UTF8, UCS16LE, and Shift_JIS.
- Ensure that the Job Processor is restarted when changes are made to the config file.

5.2.1 Troubleshooting

`Memory errors`

If you experience memory usage problems, one preventative measure is to turn off hyperlink extraction on Word documents. To do this, set the *`NoWordLinks`* registry entry to 1.

`Excessive temp files`

Microsoft Word leaves behind temporary working files. If the number of temporary files becomes too large, Microsoft Word does not launch and the Brava! Job Processor is not able to convert Word documents using the CSF Writer. You can run the `DirCleanup.exe` to clean up temporary Word files. This utility is located in the Job Processor directory. It can take one or more paths as arguments, and cleans up the temporary Word files in those directories. A log file is written to `C:\NetItDirCleanup.log`.

Example 5-3: Usage example

```
DirCleanup.exe  "C:\WINDOWS\Temp" "%userprofile%\Local Settings\
Temp"
```

`Unable to view Office documents when running the Job Processor as a Service on Windows Server machines.`

You can modify the following registry value to run the Job Processor as a service on Windows Server:

1. Run **regedit** and locate the following registry key;

   ```
   HKEY_CURRENT_USER \Software\Microsoft\Windows\CurrentVersion\
   Explorer\User Shell Folders
   ```

2. Double-click the **Cache** registry entry and type the following in the **Value** data box:

   `%USERPROFILE%\AppData\Local\Microsoft\Windows\Temporary Internet Files`

3. Click **OK** and then close the **Registry Editor**.

4. If the `Temporary Internet Files` folder is not visible, it can't be accessed from services and publishing Office documents will fail. If this is the case, change the **Cache** value from:

   `“%USERPROFILE%\AppData\Local\Microsoft\Windows\Temporary Internet Files”`

   to

   `“%USERPROFILE%\AppData\Local\Microsoft\Windows”`

-21 error

If the Job Processor is set up to run as a service, the Job Processor service must have an administrative account specified in the **Properties** dialog box of the **Services** panel. If not set, you might receive a -21 error.

You can check to see if a -21 error is a publishing error or services error by shutting off the Job Processor service in the **Service** panel and then starting Job Processor manually. If publishing is successful, the problem is the service configuration.

Tomcat errors

If the Tomcat installation is set to use the Local Account, the servlets running in the service cannot access the UNC shares. The Tomcat service must be configured to use a domain account with admin privileges.

If Tomcat fails to start and you are receiving a “SocketExceptions” or “binding” error, you can use **Netstat** (command line activity: `netstat -oab`) to determine where that specific port is already being used. Alternatively, the port can be configured in the `server.xml` file to use a different port.

Macro-based documents are failing

You should adjust any application Trust Center settings (**File > Options > Trust Center > Trust Center Settings > Macro Settings**) to match your business rules, ensuring that you choose selections containing **Disable all macros without notification** as applicable. Without proper configuration, macro-based documents might not render.

Old Office formats are failing

If attempting to publish older Office formats, print/XPS publishing can fail if the file block protection settings are enabled. This setting can be altered through your Office application file options. Note that you should be logged in as the JobProcesser service account user when making this update:

1. From your Office application, go to:

File > Options > Trust Center > Trust Center Settings > File Block Settings

2. Any file checked as **open** will fail.

```
Office documents do not render in the viewer
```

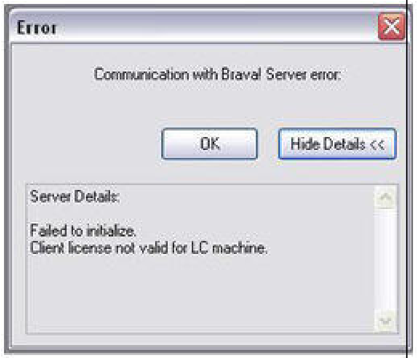
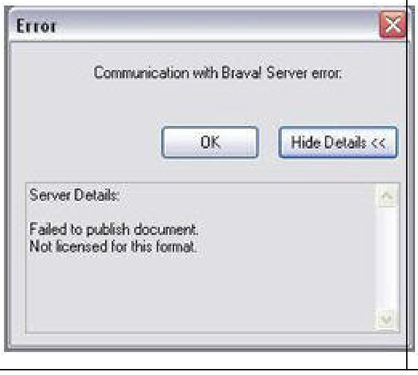
If you are configuring the application to run as a different user than the one which you are currently logged in as (user account for Windows services logon), you might be required to perform a reboot after installation to reload the user-specific changes to the registry.

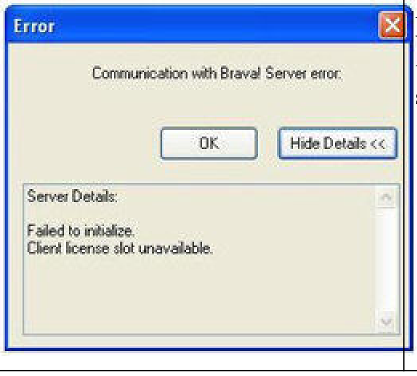
5.3 Licensing tips

This section lists error messages that end users would encounter with a licensing issue in Brava! Enterprise.

Expired license

This error message appears when the Brava! Server License is not valid or has expired. It is worth noting that a large number of issues, including firewall issues, can also cause this message to appear.

Type	Dialog	Cause
Incorrect Server	 Error Communication with Brava! Server error: OK Hide Details << Server Details: Failed to initialize. Client license not valid for LC machine.	This error message can appear if the Client license the user has is "tethered" (licensed by IP or HOSTNAME) to another Brava! Enterprise Licensing Server.
Unsupported File Format	 Error Communication with Brava! Server error: OK Hide Details << Server Details: Failed to publish document. Not licensed for this format.	This error message can appear if the user is trying to load a file that is supported by Brava! but not licensed to the current Client license.

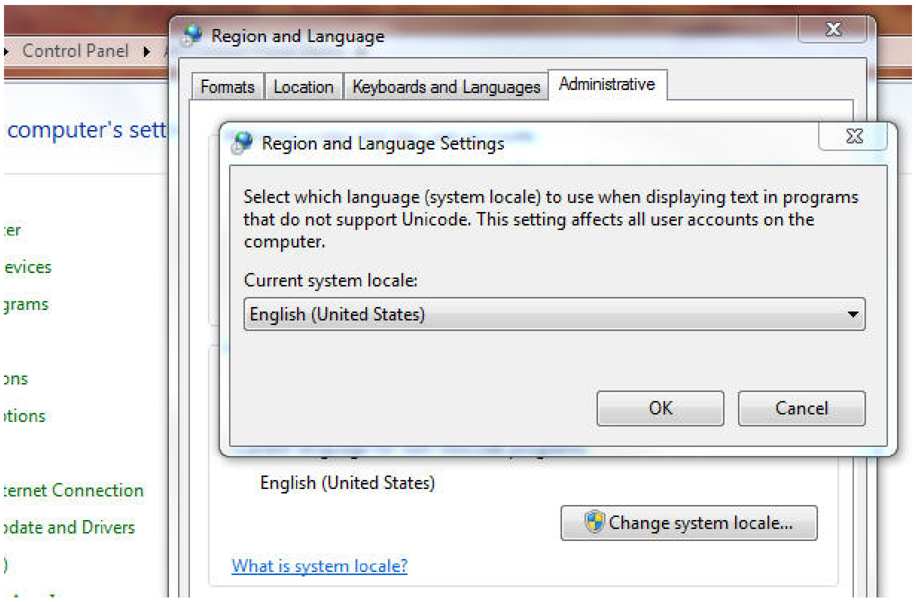
Type	Dialog	Cause
Maximum Users Exceeded	 Error Communication with Brava! Server error. Server Details: Failed to initialize. Client license slot unavailable.	A user might see this message when the maximum user cap for their particular Client license slot is reached.

5.4 HTML Client tips

- **Problem**: Some characters in my localized version of Brava! HTML Client do not display correctly.

  Solution: When running Brava! HTML Client on a non-English system, the server locale setting needs to be set to American English since some European languages use a comma as the decimal separator which is not supported in SVG. To do this:

  1. On the Server serving up the HTML pages for the Brava! HTML Client , open Control Panel.
  2. Select **Region and Language**, then click the **Administrative** tab.
  3. Click **Change system locale**, and on the **Region and Language settings** dialog box, select **English (United States)**.
  4. Click **OK** to close through the dialog boxes and save your settings.



Select which language (system locale) to use when displaying text in programs that do not support Unicode. This setting affects all user accounts on the computer.

Current system locale: English (United States)

English (United States)

- **Problem**: I can't select text in SVG files in Firefox.

  Solution: Upgrade to the latest Firefox version for support of native text selection of SVG files.

- **Problem**: The HTML Demo page, or other pages, do not launch on a localized installation.

  Solution: Filenames of pages cannot contain unicode characters unless your server can handle these characters on requests. For example, Tomcat cannot handle these requests by default and `URIEncoding="UTF-8"` needs to be added to the Connector element in the `server.xml` to enable Unicode filenames.

- **Problem**: Some text is not highlighted when using the Export to Word feature for Brava! Changemarks.

  Solution: Brava! supports highlighting some types of text that cannot be highlighted using Word. Examples are bullets, formatting characters, and text boxes. If these text types are used with a Brava! Changemark, they might not be highlighted correctly when exported to Word.

- **Problem**: Text markup entities do not render correctly when using the Edge browser.

  Solution: This issue is acknowledged by Microsoft. See issue 3723601 (https://developer.microsoft.com/en-us/microsoft-edge/platform/issues/3723601/) for details.

Chapter 6

Appendix

6.1 Supported file formats

Brava! Enterprise does not require a separate (native) application to publish the file formats listed in the Supported Formats table. To publish other formats not on this list, the Job Processor and the native applications are required. The Job Processor machine must have a Print-to command available in the native application for that extension. You can install the native application (such as Word, Excel, or any other application that uses a Print command) on the Job Processor machine. For information about how to set up applications to publish file types that do not have a **Print** or **PrintTo** action and are not on the following supported file types list, see "Publishing using other applications" on page 91.

Brava! Enterprise Supported Formats

Brava! Enterprise does not require a separate (native) application to publish the file formats listed in the supported formats page, excluding those listed in the "Document/Additional Image Formats" section. See Supported Formats (http://www.opentext.com/file_source/OpenText/en_US/PDF/opentext-so-brava-enterprise-supported-formats-en.pdf).

6.2 Provide the online help on a local help server (Private Help Server)

The online help for this module is delivered using the OpenText Global Help Server (GHS) system, which provides your users with live access to the latest version of the help. If you cannot use the GHS system, for example, if your site does not have internet access, you can install the OpenText Private Help Server (PHS), a local version of the help system that can host your OpenText online help on your organization's network. After the PHS is installed, you can then configure your OpenText module(s) to forward all online help requests to your PHS. For detailed information about installing the PHS, see *OpenText Help System - Private Help Server Administration Guide (OTHS-AGD)*.

Notes

- The Private Help Server can support multiple OpenText modules. If the Private Help Server has already been installed within your organization to support another OpenText module, you can add additional OpenText module online helps to that installation.
- If you are replacing a previous PHS installation, see Section 2.5 "Updating a Private Help Server installation" in *OpenText Help System - Private Help Server Administration Guide (OTHS-AGD)*.

- If the server you want to use for the PHS installation cannot connect to the internet, see Section 1.1 “Deploying online help files in an environment without Internet access” in *OpenText Help System - Private Help Server Administration Guide (OTHS-AGD)*.

Once the PHS is installed or upgraded, you can use its Online Help Deployer to download online helps from the GHS system by entering the help deployment codes listed in “Help deployment codes” on page 118. For information about using the codes, see Section 3 “Adding product online help to the Private Help Server” in *OpenText Help System - Private Help Server Administration Guide (OTHS-AGD)*.

Table 6-1: Help deployment codes

Code	Product
CLBRVW160612-ABE	OpenText Brava! Enterprise 16.6.12

To configure the Brava! HTML Client to use an alternate help URL:

1. Create a client configuration to generate a signed config, for example: `.../webapps/BravaSDK/HTML/samples/formats/GetFormatsConfig.jsp`
2. Add the local network help URL to the Brava! HTML Viewer signed configuration to launch an alternate help file.

   ```
   htmlConfig.setGHS(Providers.ClientConfig().createGHS("<URL>", "",
   ""));
   ```

   For example:

   ```
   htmlConfig.setGHS(Providers.ClientConfig().createGHS("http://docsapi.
   staging.opentext.com/mapperpi", "", ""));
   ```

To configure the Brava! ActiveX Client to use an alternate help URL:

1. You can configure the alternate help URL through the Brava! ActiveX client configuration parameters. See *OpenText Brava! - Brava! Enterprise ActiveX Client Parameter Configuration Guide (CLBRVW-CAX)* for usage details.
2. In the *`AltHelpURL`* setting, enter the alternate help server URL that you prefer to use. For example: `althelpurl=http://docsapi.staging.opentext.com/mapperpi`
3. You can also use *`AltHelpLang`* and *`AltHelpTenant`* to specify a language or tenant for the help. For example, `althelplang=de`

   The Brava! Client help files are localized in the following languages:

   - DE (German)
   - EN (English)
   - ES (Spanish)
   - FR (French)
   - JA (Japanese)
   - KO (Korean)

- NL (Dutch)
- PT (Portuguese)
- ZH (Chinese)

If an alternate URL or language is not specified, Brava! clients launch the default online Production GHS English help from the OpenText servers: `http://docsapi.opentext.com/mapperpi`.

6.3 Routing Service Appendix

! **Important**
The Brava! Routing Service component as an optional feature that allows customers to investigate and evaluate the benefits that it might provide to them. It will be refined in subsequent releases for broad adoption.

For a Brava! Enterprise deployment to be able to support multiple Brava! Servers for scalability and fault tolerance, all operations related to a given desired operation (either viewing a document, or publishing to a new format, etc.) require all HTTP/HTTPS requests to go to the same Brava! Server. Previously (version 16.6.0 and earlier), Brava! used a (now deprecated) shared cache feature that used a database to handle this functionality.

To handle proper routing of these requests in a deployment where only local filesystem displaylistcache is used on each of multiple Brava! Servers, an optional Brava! Routing Service component is available. See "Installing the Brava! Routing Service" on page 22.

6.3.1 Deployment

Figure 6–1 depicts a typical deployment of multiple load balanced integration servers working with multiple load balanced Brava! Servers, plus the addition of multiple load balanced Brava! Routing Services.

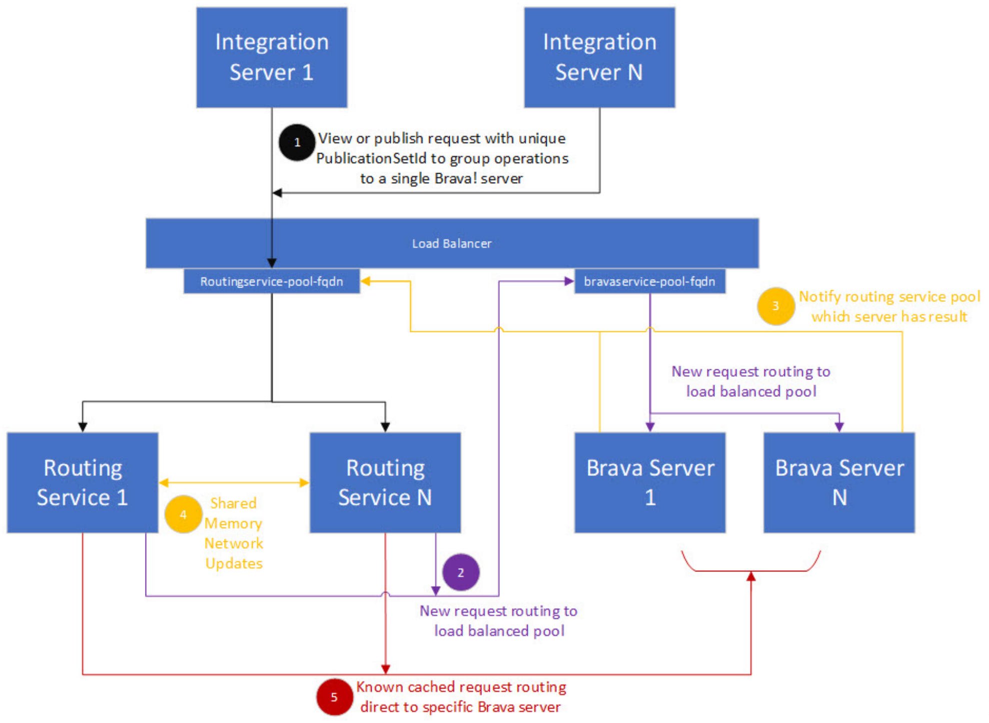

Integration Server 1
Integration Server N
1
View or publish request with unique PublicationSetId to group operations to a single Brava! server
Load Balancer
Routingservice-pool-fqdn
bravaservice-pool-fqdn
3
Notify routing service pool which server has result
New request routing to load balanced pool
Routing Service 1
Routing Service N
Brava Server 1
Brava Server N
4
Shared Memory Network Updates
2
New request routing to load balanced pool
5
Known cached request routing direct to specific Brava server

Figure 6-1: Routing Service Deployment Diagram

1. In the above diagram, when the Brava! Routing Services are installed, they are added to the load balancer as a virtual server. This single virtual server URL should be the URL configured in the integration server or servers as the sole known path to Brava!.

2. Each Brava! Routing Service is then configured to know the virtual server URL of the Brava! Servers at the load balancer. All new document viewing / publishing requests are sent by the specific Brava! Routing Service instance that picks up the request, to the Brava! Service load balancing pool, and a load balancer selected Brava! Server picks up the request.

   In addition, each Brava! Server has configured the URL to the load balancer Brava! Routing Service pool and reports back with which specific (unique FQDN) Brava! Server handled the request. Once one of the Brava! Routing Services picks up this notification, it “broadcasts” this information to the other Brava! Routing Services using Hazelcast IP discovery/communication. The list of Brava! Routing Service instances is therefore a configuration item as well (for each Brava! Routing Service that’s installed), so they know all the other members of their pool to share information with.

3. Once the Brava! Routing Service Pool has this shared list of which specific Brava! Server has the converted artifacts in it’s local filesystem displaylistcache, all subsequent requests for the same specific publication (denoted by an integration choosing a unique `PublicationSetId` for each document or collection of unique

documents being viewed/published) can be routed directly to the specific Brava! Server.

Failover notes

Due to the shared data mechanism between Brava! Routing Services, one or more of these can fail and the pool continues to function.

If a given Brava! Server instance fails, and the load balancer removes it from the pool, the overall deployment is fault tolerant in that the next request to publish the same unique set (based on `PublicationSetId`) will detect the missing Brava! Server instance and try to send to another in the pool at the load balancer.

6.3.2 Known limitations

Known limitations of the current version of the Brava! Routing Service include:

- **Integration support:** Any given product integration of Brava! Enterprise must utilize the `BravaConnection` API and specify a unique `PublicationSetId` that identifies a given type of operation (such as view or publish to new format) for a given set of one or more specific documents. If this does not happen, the existing API used generates a random `PublicationSetId` value and, while this ensures operations all go to a given server to complete without failure, it does not guarantee any future cache hit reuse against servers with the same converted artifacts.
- **Routing tables persistence:** The current implementation of the Routing Service does not have any persistence, and does not have any "resync" capability when a single member is restarted. This means that if all Routing Services are restarted around the same time, the entire state of the routing tables are lost. For example, if using a cluster with two Routing Service members running for a period and one is restarted, and then a day later the other is restarted, all documents viewed before the first member restart (that have not been viewed since) will have their mappings lost. What this means from a functional standpoint is that it's possible that those documents will be republished.
- **Memory usage:** The Routing Service retains its tables within memory with no mechanism to clean up entries (other than providing a means for integrators to remove mappings from the routing service tables using the `DELETE /route/:id` endpoint). This means that within environments that process a large volume of unique documents, the memory usage for the Routing Services can become an issue.
- **ActiveX not supported:** At this time, the Brava! ActiveX viewer is not supported in environments configured to use the Routing Service. Any integration deployed in front of Brava! Routing Services must only use the HTML viewer or make separate provisions for launching the ActiveX viewer to be pointed to a separate standalone Brava! Server.

Chapter 7

Copyright Notices and Acknowledgements

This software includes third-party component software distributed by OpenText to you pursuant to specific third-party license agreements, whose terms and conditions are as set forth in your license agreement with OpenText and/or the Terms and Conditions of Embedded Products. Copies of such Embedded Software Licenses relating to the use and distribution of such Embedded Products can be found in the `\OpenText\Brava! <product>\Licenses` directory located in the product install directory. You agree to comply with all such Embedded Software Licenses which apply to the Software licensed to you by OpenText.

Please refer to the **LegalNotices.txt** document included in your product installation to view all third-party software copyright notices and acknowledgements that are associated with this product and its components.