

# WAR Deployment

Servoy Developer (6.1 and later) offers the option to export the Servoy Application Server to a so-called [WAR file](#). This WAR file can then be deployed on any [J2EE](#) standards compliant application server, like [Glassfish](#), [IBM WebSphere](#), [Jetty](#) or [Apache Tomcat](#) for example.

**NOTE:** If you use a Servlet spec 5.0 container, like Tomcat 10, you need to use a [Migration Tool](#) to convert the generated WAR to the new specification. Because by default the generated WAR will not run and you will encounter errors like: `java.lang.NoClassDefFoundError: javax/servlet/ServletContextListener`. Use the migration tool for this or keep using a Servlet spec 4.0 variant like Tomcat 9.

## Terminology

In this chapter the term "application server" is used to refer to two different things:

- **Servoy Application Server:** this refers to the specific libraries & software components created by Servoy that provide the Servoy specific functionality, like the Servoy Admin page, the hosting of Solutions that can be launched in the different types of Servoy Clients etc.
- (J2EE standards compliant) **application server:** this refers to any Java application server that follows the J2EE server specification and under which the Servoy Application Server can run

**NOTE:** You should not use the Servoy application server (the tomcat we ship) as your target for your WAR. This can result in errors and weird behavior because then 2 application servers are running at the same time in the same space that is not separated from each other. Two war deployments on the same server should be fine (they are separated by tomcat) you only have to make sure that the rmi ports are then different if you enabled rmi/smartclients.

**NOTE:** NGClient needs the support of websockets and then the java implementation that is based on: [JSR-356](#). For example Tomcat supports this from 7.0.43 and higher and Jetty from 9.1

**NOTE:** For memory management en tuning the server the same stuff applies as in [Memory Management](#) but the it can be that you are in a shared environment or don't have full control over the startup of your Server (like tomcat). But the deployed WAR is just a full blown servoy installation needing the same stuff as a servoy standalone application server.

See especially: [Big Server Configuration](#)

## Benefits

The benefits of the WAR export option are:

- Provides the ability to create a custom Servoy Application Server "package", containing specific plugins, beans, drivers and LAFs plus specific settings that can be deployed to an application server in one go
- Provides the ability to run a Servoy Application Server on any J2EE standards compliant application server.  
Contrary to the Servoy installation that results from running the All-In-One installer, the WAR export does NOT contain the Apache Tomcat application server. A WAR file is an archive file that follows containing an entire application
- Run multiple Servoy Application Servers on the same application server

## Differences with a standard Servoy Application Server installation through the All-In-One installer

- Doesn't contain Apache Tomcat as application server
- Doesn't contain the database engine or database files
- Doesn't support WEBDAV for Web Client template editing
- Doesn't support uploading (new versions) of plugins, beans, drivers of LAFs through the Servoy Admin page
- Doesn't support command-line upgrading

## Similarities with a standard Servoy Application Server installation through the All-In-One installer

- Still requires a servoy\_repository database connection to operate
- Can run all the different types of Servoy Client
- Has the full Servoy Admin page, except for the Library Upload functionality
- Solutions can be imported through the Servoy Admin page

## Database connections and .war deployment




The needed database connections (including the repository server DB connection) have to be operational (so already started) before deploying the .war file. This is not much different from a normal application server where you would also need the repository connection operational before starting the server.



If for some reason the .war deployment failed and you cannot access your solution/admin page as expected please check the J2EE server's log files as well as any relevant log files in `[user_home_dir]/servoy/server/[your_context_name]/...` For example a repository version in the repository DB that is incompatible with the version of Servoy the .war was built on could generate this problem.

## Creating a WAR export

- Do one of the following:
  - Go to **File > Export**. A window will appear. In the tree, select **Servoy > Export War File**. Click on the **Next** button.
  - Right click on the active solution in the Solution Explorer. Select **Export Solution > War Export**. A window will appear.
- Choose a name and a location for the export file. The name used for the export file automatically becomes the so-called "context" for the Servoy Application Server when installed on a application server, see [About "context"](#) for more info.

You can use the options described below, then click **Next**.

Option	Description
<b>Include active solution and modules</b>	<p>Export the active solution and its modules. If unchecked, only the Servoy application server is exported.</p> <div>  If you check "Include active solution and modules", table changes and i18n, when selected, will be synced or imported into the application server when the WAR is first deployed.         </div> <p>If the active solution has pre/post import hooks, those will be called during the import process.</p> <p>This solution (and its modules) won't be imported into the repository, but are serialized as a binary file that is read in from the file system every time the application/war is started. If that solution is also in the repository then that one will be deleted and will not be used. The repository is still needed for user and sequence management.</p>
<b>Allow data model changes</b>	<p>If checked, it allows database structure in case the database structure on the server is different than the one in the export. e.g. table/columns changes or missing sequences</p> <p>It is also possible to specify the server names for which data model changes are allowed.</p>
<b>Skip database view updates</b>	If checked, it skips database view updates.
<b>Export based on DBI files only</b>	<p>Exports the database meta data information for the main solution and the modules (if needed) using only table definitions.</p> <div>  This method is used if at export the database is down (or has errors) so tables/servers are not accessible.         </div>
<b>Export all tables from referenced servers</b>	<p>If checked, it will export all the tables for any server used in the solution.</p> <p>If it is unchecked, only tables that are actually used in the solution will be exported.</p> <div>  if this option is unchecked, the tables which are not referenced by the solution will not be exported.         </div>
<b>Allow SQL keywords</b>	<p>It allows exporting tables which contain SQL keywords in table and column names. e.g. table having a column named "title"</p>
<b>Update sequences</b>	If checked the sequences will be updated for the servers involved in the export, otherwise they will not be touched.
<b>Override sequence types</b>	The sequence types will be overridden for the exported servers.
<b>Override default types</b>	If checked, the default values will be overridden for the exported servers.
<b>Export metadata tables</b>	Export all the metadata files in the workspace which are referenced by the solution.

<b>Check metadata tables</b>	<p>Check whether the workspace metadata files are in sync with the database tables. If it fails, then it throws an exception with the suggestion to update the meta data for the tables first, and the export does not continue.</p> <p>If it is left unchecked, the export will be done based on the existing workspace meta data files, without checking if they are in sync with the database tables or not.</p> <p>If checked and no metadata file is found in the workspace, the export will not continue, but throw an exception suggesting to update the meta data for the tables first.</p>
<b>Export solution sample data</b>	<p>If checked, data (from the current connection) for every table exported will be included. All rows or a limited number of rows can be chosen to be exported.</p> <div>  <b>Careful</b>        If the dataset is large, then exporting too many rows may fail.     </div> <div>  <b>Tip</b>        Even if sample data has been exported, if there is data already in a table when importing, it will not move sample data to the table on import.     </div>
<b>Export i18n data</b>	Exports the i18n keys used in the solution
<b>Overwrite repository group security settings with import version</b>	If checked, it overwrites the security settings for the groups which are exported and already present in the repository on the server.
<b>Export users</b>	Exports the Servoy users specified in user and group security
<b>Automatically upgrade repository if needed</b>	In case the repository version on the server is lower than the one of the developer used to export the solution, it automatically upgrades the repository.
<b>Create Tomcat META-INF /context.xml</b>	Tomcat specific option. Enable this option so that the options below can be enabled.
<b>Set antiResource Locking to true</b>	Set this option to true if running Tomcat on Windows and if having problems undeploying the war file. This option allows Tomcat to workaround the file locking feature of the OS and execute the undeploy. This option will impact the startup time of the war.
<b>Set clearReferencesStatic to true</b>	Set this option to true if a war undeploy or stop operation causes a memory leak. This option sets static fields to null, thus allowing them to be garbage collected.
<b>Set clearReferencesStopThreads to true</b>	Use with care. Set this option to true only as a last resort. Please read <a href="https://tomcat.apache.org/tomcat-8.0-doc/config/context.html">https://tomcat.apache.org/tomcat-8.0-doc/config/context.html</a> for more information.
<b>Set clearReferencesStopTimerThreads to true</b>	Set this option to true if after an undeploy or stop operation there are still Timer threads running.
<b>Minimize JS and CSS</b>	when set servoy will try to minimize the js of css files (when there are not already minimized so ending on .min.js) This is a much simpler minification then what a real minimizer does, so for component/service developers, if you have big support libs in your component or services packages use the minified version of that support lib. You can have reference the full version in the manifest and have the minified just besides it. Then in developer it will be easier to debug but at runtime the minified version will be taken.

3. A list of all the plugins is shown. The plugins that will be used in the war can be left checked, the others can be omitted from the export.
4. A list of all the beans is shown. The beans that will be used in the war can be left checked, the others can be omitted from the export.
5. A list of all the laf files is shown. The lafs that will be used in the war can be left checked, the others can be omitted from the export.
6. A list of all the jdbc drivers is shown. The drivers that will be used in the war and are not provided by the application server in which the war will be deployed can be left checked, the others can be omitted from the export.

7. In the next dialog it is possible to choose which NG Components to export. The "exported components" list already contains the components required by the solution. However, in case the solution model is used to add components to forms/layouts, more components should be chosen from the "Available components" list.
8. Similarly to the components dialog, the exported services can be chosen.
9. Select a servoy properties file to be used for the export.
  - a. If no file is provided, a default file will be generated. An extra page will be spawned in the wizard, where a few settings need to be specified for the generated servoy properties file:
    - Allow running smart clients - if checked, both smart web clients will be allowed to run, and it must be noted that when restarting the application context in the web container, the RMI classes will not be garbage collected, and that may lead to out-of-memory errors. If it is left unchecked, only web clients will be allowed to run.
    - Port used by RMI Registry - default port is 1099.
  - b. If a servoy properties file is specified, make sure the `SocketFactory.tunnelConnectionMode` on 'Network Settings' page in Servoy Admin is set to 'http&socket'. See [WAR Deployment#About the servoy.properties file](#) for more info.
10. Click **Finish** (if no servoy properties file is specified, the needed settings for the default generated file have to be provided on the next page, then click **Finish**).
11. The `.war` file will appear in the specified location.

## Exporting multiply main solutions in one WAR.

If you use the option to export it with the active solution, only the active solution and its modules are exported as a runtime binary inside the WAR.

So other solutions that you have that are not in the module list are not exported. Currently there are 2 ways to also get those solutions in the WAR install:

1. Export the other solutions are normal .servoy exports and import them through the admin page, this can be done for all solutions so you create the WAR without any runtime solution, but be sure then that it WAR has everything the NGClient solution(s) need. So all the component and ngclient services that are used by all the solutions.
2. Create a wrapper solution around all the solutions and export that one as the active with its modules. (so the normal solution are the modules). Problem with this is that there can be conflicts between the solution because of the same form or scope names.

## About "context"

An application server can host many applications, where a Servoy Application Server is just one of the applications. In order to be able to access each individual application and prevent name clashes, each individual application runs in its own namespace or "context". If the application server runs on <http://localhost:8080> and the context of an application is "xyz", the assets of the application are exposed through <http://localhost:8080/xyz/.....>

In case of a Servoy Application Server that is deployed as a WAR file with the name "myCustomServoyServer" on an application server on <http://localhost:8080>, the typical url's of the assets a Servoy Application Server exposes are:

- <http://localhost:8080/myCustomServoyServer/servoy-webclient>
- <http://localhost:8080/myCustomServoyServer/servoy-admin>
- <http://localhost:8080/myCustomServoyServer/> - for SmartClient.

## About the servoy.properties file

The WAR export wizard in Servoy Developer allows the specification of a specific .properties file to be included in the WAR file. The settings in the .properties file that is specified will be included in the WAR export and when the WAR is deployed on an application server, the settings are applied. When the .properties file is not specified, the default settings for a Servoy Application Server will be included.

When updating an existing WAR deployment (see [WAR Deployment#Updating a WAR deployment](#) below for more info), only new default settings will be applied to the existing deployment, as to not override configuration changes made after the initial deployment.

On the application server, the settings are stored in the following location: `{user.home}/.servoy/server/{context}/servoy_server.properties`.

If `{user.home}` is a read-only folder, then a `SERVOY_USER_HOME` system variable (or in 2019.03 this can also be an Environment variable) must be defined to point to a writable folder. You can give this directly to the JVM as well using `-DSERVOY_USER_HOME=...` (in case of Tomcat you can put this in `JAVA_OPTS`; see Tomcat documentation on how that can be changed). Starting from Servoy version 8.2, it is also possible to set this in the servoy.properties file used for the WAR export, by setting "servoy.user.home". If this 'servoy.user.home' is used in the properties file then property substitution (system properties or environment variables value replacement) is not supported. This value must be a valid folder by itself, use the `SERVOY_USER_HOME` system property or since 2019.0 that same property can also be set as an environment variable.

NOTE: when using an existing servoy.properties file, check if the property "SocketFactory.tunnelConnectionMode" is set to one of the following values: "Http&Socket", "Http" or "Socket". The "2waysocket" option is not fully supported when deploying using a WAR: deploying multiple Servoy WAR's on the same server or a hot redeploy does not work when using "2waysocket". When not using an existing servoy.properties file the "SocketFactory.tunnelConnectionMode" is set to "Http&Socket".

**note:** When you want the location of the leading properties file of a war/server deployment: The default admin homepage tells you the full path under the Server Information section: Settings file for this server: xxxxxx

## About Web Client templates

A Servoy Application Server deployed as a WAR file does not support WEBDAV, thus it is not possible to edit Web Client Templates. Any templates that require editing need to be edited in Servoy Developer. The WAR Export functionality in Servoy Developer will automatically export all modified templates.

## NGClient grouping/minizing

With Servoy 8.1 if you export a war the exporter will group all css and js files through a few urls, these urls that are created are unique for this WAR instance. Servoy will send caching headers on those urls that will let the browser and proxies know that they can be cached "forever". This way the browser doesn't have to redownload those resources when hit again.

The grouping tries to use the minimized version of they are there on the file system. So a component or service can have libs that are the normal js files for easier debugging in the developer, but besides it it can have the same js name but ending on \*.min.js. Those will be taken by the grouper, this way the download is even way less smaller. All other files can be minimized by Servoy on export, but this is a very simple minimizer.

All the templates of the components that are inside the war are also grouped in 1 big js file, that is again grouped. So all the template files are also immediately available in the browser without the need to download them one by one.

If for some reason a component doesn't work in grouping mode then you can turn this use of the generated grouping files off at the admin page: servoy.ngclient.enableWebResourceOptimizer set that to false. Better is to improve the component itself to let the Servoy grouper know that it should exclude this component or 1 of its support libs. For this the spec has a few options, the libraries section can have a "group":false besides name/url/version and the spec itself can have also that same property. See [Specification \(.spec file\)](#)

## Deploying a WAR file

Deploying a WAR file depends on the application server used. Some application servers host a admin webpage to manage WAR deployments. On other application servers the deployment can be as simple as placing the WAR file in a special 'webapps' folder of the web server, after which the WAR automatically gets deployed by the application server. For details see the documentation of the application server used.

Some application servers automatically expand the WAR file, which means that the content of the WAR file is automatically unzipped into separate files/directories in the appropriate directory of the application server.

When running multiple Servoy Application Servers through WAR deployment on the same application server it is vital to make sure each Servoy Application Server is configured to use a different RMI port and use a different set of databases



### Port conflicts

If you keep your developer open while testing war deployment on the same machine, please take care of possible port conflicts. For example if your J2EE server is Tomcat it will probably try to use 8080 which is already being used by Servoy Developer. So you will not be able to access it from the browser on that port. Just close developer before deploying or [change the ports](#) (see your J2EE server's documentation or this topic if it is not Tomcat) on one of the servers.

## Updating an existing WAR deployment

A deployed WAR file can't be modified. Therefore the ability of a Servoy Application Server to upload new (versions of) plugins, beans, drivers and LAF's is disabled.

If updates are required to plugins, beans, drivers, LAFs, Web Client templates or Servoy Application Server version updates, a new WAR export needs to be made from Servoy Developer. The generated WAR file can then be deployed using the WAR deployment mechanism that most application server have to upload WAR files and (re)deploy them.

When upgrading an existing WAR deployment, make sure to use the exact same context, otherwise when the WAR file gets deployed, it doesn't update the existing deployment, but instead deploys the WAR as a new application under the new context.

## Trouble Shooting

Admin page not accessible

- Is database running?
- Using 2waysocket?