Command line utilities for Exporting of Solutions and Unit Testing

Servoy allows you to export workspace solutions into .servoy files or .war files as well as to run jsunit tests on these using command line.

NGClient, WebClient and Smart Client ".war" export

This exporter will export NGClient, WebClient and SmartClient solutions as .war files. There is also an exporter specific for mobile typed solutions that also generates a .war file but that one is separate is is detailed later.

To see the help message do the following:

1. Change the current directory to <servoy_install>\developer\exporter
2. execute (.bat/.sh depends on your OS): war_export.bat -help

This will output the help for the Command Line War Export Application. To do a war export a command similar to the following should be executed:

```
war_export.bat -s <solution_name> -o <out_dir> -data <workspace_location> -defaultAdminUser <admin username> -
defaultAdminPassword <admin password> [optional_args]
```

Output of "war_export.bat -help" is:

```
WAR exporter. Exports workspace solutions into .war files.
USAGE:

   -help or -? or /? or no arguments ... shows current help message.

                OR

   -s <solution_name> -o <out_dir> -data <workspace_location>
   Note: <solution_name> can  also  contain multiple solution names  separated  by comma. In
         that case, one .war file will be generated for each solution in that  list. Some of
         the options below  (for example -warFileName)  do not make sense when multiple .war
         files are to be created. There is a note on each such option.

   -defaultAdminUser <user name for admin page when no admin user exists>
   -defaultAdminPassword <password for defaultAdminUser> [optional_args]

        Optional arguments:

        -verbose ... prints more info to console
        -p <properties_file> ... path and name of properties file used to start exporter.
             Default: the 'servoy.properties' file  from 'application_server'  will be used.
        -as <app_server_dir> ... specifies where to find the 'application_server' directory.
             Default: '../../application_server'.
        -pl alternate project locations; solution, resources and  other needed projects will
             be searched for in subfolders (deep) of the given 'workspace_location' as well.
                                                                    For example: if the
             workspace needs to contain projects from different git repositories,  those can
             be checked out in '<workspace_loc>', '<workspace_loc>/a', '<workspace_loc>/b/c'
             and so on.
        -ie ignore build errors.  CAUTION! the use of this flag is discouraged; it can cause
             invalid solutions to be exported.
        -sb skip build. No build markers will be generated. This can greatly decrease
             export time. CAUTION! the use of this flag is discouraged;
```

```
     it can cause invalid solutions to be exported.
-dbi ... export based on dbi files (even if database servers are available)
-active <true/false> ... export  given solution  (and its modules) as part  of .war.
     That means the (current if more) solution  from -s  parameter and  its modules.
     If false,  it will create a war with no solution  included. If false,  you will
     have to import  solutions using the admin page. It does  not make sense  to use
     false if multiple solutions were given at -s.
     Default: true
-pfw <properties_file> ... path & name of properties file to be included in the war.
     Default: the 'servoy.properties' file  from 'application_server'  will be used.
-b <bean_names> ... the space separated list of (smart / web client) beans to export
     Default: all beans from application_server/beans are exported.
     You can use '-b <none>' to avoid exporting beans.
-excludeBeans <bean_names> ... the  list of beans to excluded from  the export e.g.:
     -excludeBeans bean1.jar bean2.zip
     Default: none is excluded.
-l <lafs_names> ... the space separated list of look-and-feels (smart cl.) to export
     Default: all lafs from application_server/lafs are exported.
     You can use '-l <none>' to avoid exporting lafs.
-excludeLafs <lafs_names> ... the list of lafs to be excluded  from  the export e.g:
     -excludeLafs laf1.jar laf2.zip
     Default: none is excluded.
-d <jdbc_drivers> ... the space separated list of  jdbc (database) drivers to export
     Default: all drivers from application_server/drivers are exported.
     You can use '-d <none>' to avoid exporting drivers.
-excludeDrivers <jdbc_drivers> ... the  list of drivers  to exclude from  the export
     e.g.: -excludeDrivers driver1.jar driver2.zip
     Default: none is excluded.
-pi <plugin_names> ... the list of plugins to export e.g -pi plugin1.jar plugin2.zip
     Default: all plugins from application_server/plugins are exported.
     You can use '-pi <none>' to avoid exporting plugins.
-excludePlugins <plugin_names> ... the  list of plugins  to exclude from  the export
     e.g.: -excludePlugins plugin1.jar plugin2.zip
     Default: none is excluded.
-nas ... space separated list of solutions that must be exported but are not in the
     current solution's modules (for example solutions for batch processors).
     If active is false  no solution will be exported  in war. It might be  wrong to
     use this flag if you also specify multiple solutions in -s.
     Default: only active solution and its modules are exported.
-pluginLocations <ABSOLUTE paths to developer 'plugins' folder> ...  needed in  case
     you don't run the exporter from [servoy_install]/developer/exporter
     Default: '../plugins'.
-crefs ... exports only the components used by the (current if -s has more) solution
-crefs <additional_component_names> ... can be 'all'  (without the ')  or a  list of
     components separated by spaces; exports the  components  used  by  the solution
     (current if -s has more) and the components in this additional components list.
     Default: only the used components exported.
-excludeComponentPkgs ... space separated list of  excluded component packages (from
     all available component packages).
     Default: none is excluded.
-srefs ... exports  only the services used by  the (current if -s has more) solution
-srefs <additional_service_names> ... can  be  'all' (no ')  or  a list  of services
     separated by spaces; exports the services used by the  (current if -s has more)
     solution  and the services in this additional services list.
     Default: only the used services are exported.
-excludeServicePkgs ... space separated list of  excluded service packages (from all
     available service packages).
     Default: none is excluded.
-md ... export metadata tables (tables marked as metadata); uses the metadata stored
     in workspace files for each metadata table.
-checkmd ... check that metadata for metadata tables is the same for each table both
     in the according workspace file and in the actual table in the  database before
     exporting; this only makes sense if -md was specified
     Default: false
-sd ... exports sample data. IMPORTANT all needed DB servers must already be started
-sdcount <count> ... number of rows to  export per table. Only  makes sense when -sd
     is also present. Can be 'all' (without the ') in which  case  it will  still be
     limited but to a very high number: 49999999
     Default: 5000
-i18n ... exports i18n data
-users ... exports users
-tables ... export  all table  information  about  tables from  referenced  servers.
     IMPORTANT: all needed DB servers must already be started
```

```
        -warFileName ... the name of the  war file; do *NOT* use this if  multiple solutions
             were given in -s argument (it does not make sense then as multiple wars will be
             generated).
             Default: the (current if -s has more) solution name
        -overwriteGroups ...  overwrites Groups
        -allowSQLKeywords ... allows SQLKeywords
        -stopOnDataModelChanges ... stops import if data model changes.
             This option is ignored if allowDataModelChanges is present.
        -allowDataModelChanges ... (optionally) a space separated list of server names  that
             allow data model changes.  If the list is missing,  then data model changes are
             allowed on all servers.
        -skipDatabaseViewsUpdate... skips database views update
        -overrideSequenceTypes ... overrides Sequence Types
        -overrideDefaultValues ... overrides Default Values
        -insertNewI18NKeysOnly ... inserts NewI18NKeysOnly
        -importUserPolicy ... 0/1/2 where:
             don't = 0
             create users & update groups = 1 (default)
             overwrite completely = 2
        -addUsersToAdminGroup ... adds Users To Admin Group
        -updateSequences ... updates Sequences
        -upgradeRepository ... automatically upgrade repository if needed
        -contextFileName ... a path to a tomcat context.xml  that should be included
             into the WAR/META-INF/context.xml
        -useAsRealAdminUser ... the  default admin user login  given via   -defaultAdminUser
             above will be available as a normal admin user in solutions as well.
        -license.company_name OR license.<i>.company_name,
             The name of the company that has the license,  where <i> is used when there are
             multiple licenses:
             -license.1.company_name name1 -license.2.company_name name2
        -license.code OR license.<i>.code,
             The license code, where <i> is used when there are multiple licenses:
             -license.1.code XXXX-XXXX-XXXX -license.2.code  XXXX-XXXX-XXXX
        -license.licenses OR license.<i>.licenses,
             The number of licenses, where <i> is used when there are multiple licenses:
             -license.1.licenses SERVER -license.2.licenses 1000
        -userHomeDirectory <user_home_directory> ... this must be a writable directory where
             Servoy application  related files  will be stored;  if not set, then the system
             user home directory will be used.
        -doNotOverwriteDBServerProperties ... SKIP overwrite of old DBserver properties - if
             they were  stored separately  by  a previously deployed war  (due to changes to
             properties via admin page) - with ones from the war export's servoy.properties.
             If -overwriteAllProperties below is set then this flag has no effect.  Prior to
             Servoy 2019.09,  '-overwriteDBServerProperties'  was used instead but it is now
             removed in order to have the same default value as in UI export wizard.
        -overwriteAllProperties ... overwrite  all  (potentially  changed  via  admin  page)
             properties  of a previously deployed war  application  with the values from the
             servoy.properties of this war export.
        -log4jConfigurationFile ... a  path  to a log4j  configuration file  that  should be
             included nstead of the default one.
        -webXmlFileName ... a path to a web.xml  that should be included instead  of default
             one; it should be a web.xml file previously generated via a Servoy WAR export.
        -ng2 true / false / sourcemaps ... export Titanium NG2 binaries.  If 'sourcemaps' is
             given, sourcemaps will be generated for .ts files - useful for debugging.
             Default: true
        -ng1 ... export NG1 client resources; not exported by default.

EXIT codes: 0 - normal, 1 - export stopped by user, 2 - export failed, 3 - invalid arguments
Export DONE.
```

Exporting to ".servoy" or to mobile solution ".war" files

You can see a list of available command line arguments in a console window by executing `{servoyInstall}/developer/exporter/export.bat -help` and `{servoyInstall}/developer/exporter/mobile_export.bat -help`;

```
Workspace exporter. Exports workspace solutions into .servoy files.
USAGE:

   -help or -? or /? or no arguments ... shows current help message.

                  OR

   -s <solutions_separated_by_comma> -o <out_dir> -data <workspace_location> [optional_args]

        Optional arguments:

        -verbose ... prints more info to console
        -p <properties_file> ... path and name of properties file used to start exporter.
             Default: the 'servoy.properties' file  from 'application_server'  will be used.
        -as <app_server_dir> ... specifies where to find the 'application_server' directory.
             Default: '../../application_server'.
        -pl alternate project locations; solution, resources and  other needed projects will
             be searched for in subfolders (deep) of the given 'workspace_location' as well.
                                                                    For example: if the
             workspace needs to contain projects from different git repositories,  those can
             be checked out in '<workspace_loc>', '<workspace_loc>/a', '<workspace_loc>/b/c'
             and so on.
        -ie ignore build errors.  CAUTION! the use of this flag is discouraged; it can cause
             invalid solutions to be exported.
                -sb skip build. No build markers will be generated. This can greatly decrease
                         export time. CAUTION! the use of this flag is discouraged;
             it can cause invalid solutions to be exported (added in 2019.12).
        -dbi ... export based on dbi files (even if database servers are available)
        -md ws|db|none|both ... take table  metadata from workspace / database / both+check.
             Usually you will want to use 'ws'.
        -sd ... exports sample data. IMPORTANT: all needed DB
             servers must already be started
        -sdcount <count> ... number of rows to  export per table. Only  makes sense when -sd
             is also present. Can be 'all' (without the '). Default: 10000
        -i18n ... exports i18n data
        -users ... exports users
        -tables ... export  all table  information  about  tables from  referenced  servers.
             IMPORTANT: all needed DB servers must already be started
        -pwd <protection_password> ... protect  the exported  solution with given  password.
        -modules [<module1_name> <module2_name> ... <moduleN_name>]
             argument  specified in command line. Includes all or part of referenced modules
             in export. If only '-modules' is used,  it will export all  referenced modules.
             If a list of  modules is also included, it  will export only  modules from this
             list, provided they are referenced by exported solution.
        -isf <import_options_file> ... path  to import options file.  Default value is null.
             If present, will  be added to  export  file as  import_settings.json. This file
             should be  taken from a  developer export  (import_settings.json inside .servoy
             file).

EXIT codes: 0 - normal, 1 - export stopped by user, 2 - export failed, 3 - invalid arguments
Export DONE.
```

Exported solution versions

Starting with the Servoy 2020.06 release, it is good practice to have versions for your solutions and modules.

Before exporting solutions as files, make sure you set the solution versions in the solution properties view in the developer.

This is used to avoid problems caused by incompatible solution/modules versions in the workspace where the solutions will be imported.

```
Workspace exporter for mobile solutions. Exports mobile workspace solutions in .war format.
USAGE:

   -help or -? or /? or no arguments ... shows current help message.

                  OR

   -s <solutions_separated_by_comma> -o <out_dir> -data <workspace_location> [optional_args]

        Optional arguments:

        -verbose ... prints more info to console
        -p <properties_file> ... path and name of properties file used to start exporter.
             Default: the 'servoy.properties' file  from 'application_server'  will be used.
        -as <app_server_dir> ... specifies where to find the 'application_server' directory.
             Default: '../../application_server'.
        -pl alternate project locations; solution, resources and  other needed projects will
             be searched for in subfolders (deep) of the given 'workspace_location' as well.
                                                                    For example: if the
             workspace needs to contain projects from different git repositories,  those can
             be checked out in '<workspace_loc>', '<workspace_loc>/a', '<workspace_loc>/b/c'
             and so on.
        -ie ignore build errors.  CAUTION! the use of this flag is discouraged; it can cause
             invalid solutions to be exported.
        -sb skip build. No build markers will be generated. This can greatly decrease
             export time. CAUTION! the use of this flag is discouraged;
             it can cause invalid solutions to be exported.
        -production ... export normal mobile client. Default: false (exports unit test war).
        -server_url <url> ... application server URL.  Used to find mobile service solution.
             Default: http://localhost:[detectedInstallationPortNumber]
        -service_solution ... name    of    the   service   solution    (default   will   be
                               mySolutionName_service).
        -sync_timeout <seconds> ... client sync call timeout. Default: 30 sec.
        -long_test_names ... only if '-production'  is not set; it will generate 'long' test
                           method names that include solution and form/scope name;  this can
                           help with  dumb junit tools  that ignore junit test suite nesting
                           when showing test results - thus loosing that information.
                           (starting with 2020.09)

EXIT codes: 0 - normal, 1 - export stopped by user, 2 - export failed, 3 - invalid arguments
Export DONE.
```

(Import) Test suite runner

Arguments are shown below. This is a JUnit suite that actually will import a .servoy file and run the JSUnit tests on them. It's not an actual command line tool but it can be used directly from command line, or from several build tools. See Servoy Software Factory Example using Jenkins as an example.

```
This is the smart import-test-client test-suite runner HELP message.

This suite is able to import one or more .servoy solutions and run jsunit tests on them.  If you did
not expect to see this message but still got it, then something is probably misconfigured...

The working directory is assumed  to be [path_to_your_servoy_install]/application_server. If this is
not the case please define system property 'servoy.application_server.dir'.

You can give parameters to this test suite using system properties. Configurable system properties:

  servoy.test.help
        Prints this help message to the console when the value is 'true'.

  servoy.test.property-file
        Key  for  the system property  pointing to the  servoy.properties file that  should be  used
        (either absolute  or relative to the working dir). If not specified,  the properties will be
        loaded from '<working dir>/<default properties file name obtained through usual means>'.

  servoy.test.target-exports
        Key for the  system property that  points to the  solutions that should  be  tested. It must
        point to a folder containing exported solutions or to an exported solution file. The name of
```

```
      the export files  must be the same as the solution it contains that must be  tested. If this
      is not specified, the default value is '<working dir>/../../exportedSolutions/'.

servoy.test.solution-load.timeout
      Key for the system property that specifies the maximum number of seconds in which a solution
      should load and be ready for testing. Solutions with long running onOpen handlers might want
      to set this.  If a solution doesn't load in this time frame, the  tests  will fail.  Default
      value: 300 sec. Minimum: 5 sec.

servoy.test.import-config-file
      Key for the system property that points to the import configuration properties file. That is
      a standard .properties file from which import options will be read. If this is not specified
      then a set of default  import options will be used - which will  try to overwrite and import
      as much  as  possible.  Put a 'key=value' pair on each line in the file; here  is a list  of
      properties that can be used in the file (defaults, marked with '*', are used for unspecified
      properties):

          overwriteStyles       [*true/false] Overwrite repository styles with import version
          overwriteUserGroups   [*true/false] Overwrite  repository group  security settings  with
                                              import version
          cleanSolutionName     [newName]     A  clean  import  will be  performed  using the  new
                                              solution name. Only specify this if you want a clean
                                              import. Default: do not use clean import
          overrideSequenceTypes [*true/false] Override  existing  sequence  type  definitions  (in
                                              repository) with the sequence types contained in the
                                              import file
          updateSequences       [*true/false] Update sequences for all tables on  all servers used
                                              by the imported solution and modules
          overrideDefaultValues [*true/false] Override  existing  default  values  (in repository)
                                              with the default values contained in the import file
          allowSQLKeywords      [*true/false] Allow reserved SQL keywords as table or column names
                                              (will fail unless supported by the backend database)
          allowDataModelChanges [*true/false/serverNames] Allow data model (database) changes
          skipDatabaseViewsUpdate
                                [true/*false] Skip database view create/update (table info will be
                                              imported)
          showDataModelChanges  [*true/false] Display data model (database) changes
          importMetaData        [*true/false] Import solution meta data
          importSampleData      [*true/false] Import solution sample data
          importI18NKeys        [*true/false] Import internationalization (i18n) data (inserts and
                                              updates)
          insertNewI18NKeysOnly [true/*false] Insert  new  internationalization (i18n)  keys  only
                                              (inserts only, no updates)
          importUserPolicy      [skip/create_new_users_update_groups/*overwrite_all]
                                              Where:

                                              [skip] Do not import users contained in import
                                              [create_new_users_update_groups] Create non-existing
                                                     users  and   add  existing  users  to  groups
                                                     specified in import
                                              [overwrite_all] Overwrite  existing users completely
                                                     (USE WITH CARE)

          addUsersToAdminGroup  [*true/false] Allow users to be added to the Administrators group.
          solutionPassword_name [password]    Only use this if the  export is password  protected.
                                              'name' is  the protected  solution's name. One  such
                                              property can be set for each protected solution
          importHookUserName    [userName]    Any  import   hooks  will  run  using   this  user's
                                              credentials. Null by default.
servoy.test.long-test-method-names
      Key for the boolean system property that, when 'true', will generate  long test method names
      that contain solution name and form/scope names in them; this can help with dumb junit tools
      that ignore junit test suite nesting when showing test results - and loose that information.
      (starting with 2020.09)
```

Mobile test suite runner

Arguments are shown below. This is a JUnit suite that will run the JSUnit tests on an exported and deployed mobile solution (and it's mobile service solution that is already running or can be imported from .servoy and made available). It's not an actual command line tool but it can be used directly from command line, or from several build tools. See Servoy Software Factory Example using Jenkins as an example.

```
This is the mobile-test-client test-suite runner HELP message.

This suite is able to deploy  one mobile test solution  (in Servoy's embedded Tomcat) and run jsunit
tests on it. If you did not  expect to see this message but still got it, then something is probably
misconfigured...

The  test mobile client  sync operation(s) can  either be  done against  an already  running service
solution or by specifying the 'servoy.test.targetServiceSolution' property  (that will use a .servoy
export to start the service solution).

The working directory is assumed  to be [path_to_your_servoy_install]/application_server. If this is
not the case please define system property 'servoy.application_server.dir' as well.

You can give parameters to this test suite using system properties. Configurable system properties:

  servoy.test.help
        Prints this help message to the console when the value is 'true'.

  servoy.test.target
        Key for the system property that points to the exported test .war file. Required.

  servoy.test.username
        Key for the system property that gives the auto-login username.

  servoy.test.password
       Key for the system property that gives the auto-login password.

  servoy.test.clientConnectTimeout
       Key for the system property that specifies the time to wait  (seconds) for mobile test client
       to connect before failing the tests with timeout. Defaults to 30 seconds.

  servoy.test.mobileClientUrl
       Key for the  system property that  specifies  the URL pointing  to the  deployed  mobile test
       client. This is the absolute URL to the deployed war application - up to (excluding)  the war
       file name  (context). It is  useful in  case the  browser/driver used by Selenium for testing
       is on a different PC/device. Example:  http://localhost:8080. If  this is  not specified, the
       mobile client URL will be built as 'http://localhost:[detectedInstallationPortNumber]/'.

  servoy.test.seleniumDriver
       Key for the  system property that  specifies the selenium configuration that will be used  to
       start the mobile test client. This decides  which browser/driver on which device will be used
       for testing. When selenium  is used, 'selenium-server-standalone-x.y.z.jar' needs  to be part
       of the classpath  (or ios-server-x.y.z.jar when using ios-driver).  Can  be  one of (drop the
       quotation):

           - 'localhost_chrome' - run tests on  localhost  using self-started  Chrome Driver server.
                 The 'chromedriver' executable needs to  be in system's PATH or referenced by system
                 property 'webdriver.chrome.driver'.  You can  download the server (currently) from:
                 http://code.google.com/p/chromedriver/downloads/list

           - 'remote [capPairsDelimiter=...]
                     ipad/iphone/android/chrome/safari/internetExplorer/firefox/edge/opera
                     URL
                     [capability_pairs_in_format key=value]'

                 runs tests on localhost  or  remote location using WebDriver's wire protocol. 'URL'
                 specifies the already running remote WebDriver server's location (that supports the
                 chosen capabilities).

                 This can be used to  run ipad/iphone/android/chrome/firefox/internetExplorer/opera/
                 safari tests by connecting to an already started WebDriver server. Use  this option
                 with  pre-started 'ios-driver', 'Chrome Driver', 'Internet Explorer Driver Server',
                 'OperaDriver', 'Firefox Driver', ... driver servers.

                 You can  optionally  specify a number of  key=value pairs  that  will be set in the
                 RemoteWebDriver's  capabilities object. For example,  if you use Sauce Labs service

                             idle-timeout=500 deviceOrientation=portrait

                 The value will  be interpreted as a  String.  You can  specify  multiple capability
```

```
                pairs  separated by spaces or by the chars specified via 'capPairsDelimiter=' above
                if set.

                Firefox Driver server  can be used remotely as well  by starting a firefox instance
                with the Firefox Driver xpi installed and correct system properties configured. For
                more info  on how to start these  remote  WebDriver servers see  (links at the time
                this doc was written):

                    http://code.google.com/p/selenium/wiki/ChromeDriver
                    http://ios-driver.github.io/ios-driver/safari.html
                    http://code.google.com/p/selenium/wiki/FirefoxDriver
                    http://code.google.com/p/selenium/wiki/OperaDriver
                    http://code.google.com/p/selenium/wiki/InternetExplorerDriver
                    http://code.google.com/p/selenium/w/list
                    http://docs.seleniumhq.org/download/

                Example (using ios-driver): 'remote iphone http://mobiletesting_1:4444/wd/hub'

          - 'localhost_firefox' (DEFAULT) - run  tests  on  localhost using self - started  Firefox
                Driver server. The Firefox driver  is available in selenium-server-stanalone.jar as
                an .xpi that gets  added to then installed Firefox's profile when it is started.

          - 'full.path.to.java.class.that.starts.selenium' for (advanced) users  with knowledge  of
                Java and Selenium usage. This  allows  starting a custom selenium driver  that will
                be used to start the  mobile client. Classes specified here  have  to implement the
                com.servoy.automation.browser.ISeleniumStarter interface.

          - 'none' - it will not start  any browser for testing. The  mobile test client  will have
                to be started by some other/external means.

servoy.test.targetServiceSolution
      Key for the system property that points to the  exported  mobile service solution ('.servoy'
      file). Optional. If this is not specified,  the service solution must already be available /
      running. If this is set, further import options / settings can be configured via properties:
          'servoy.test.import-config-file' and
          'servoy.test.property-file'.
      These two system properties above will only be used when a target service solution export is
      specified.

servoy.test.property-file
      Key  for  the system property  pointing to the  servoy.properties file that  should be  used
      (either absolute  or relative to the working dir). If not specified,  the properties will be
      loaded from '<working dir>/<default properties file name obtained through usual means>'.

servoy.test.import-config-file
      Key for the system property that points to the import configuration properties file. That is
      a standard .properties file from which import options will be read. If this is not specified
      then a set of default  import options will be used - which will  try to overwrite and import
      as much  as  possible.  Put a 'key=value' pair on each line in the file; here  is a list  of
      properties that can be used in the file (defaults, marked with '*', are used for unspecified
      properties):

          overwriteStyles       [*true/false] Overwrite repository styles with import version
          overwriteUserGroups   [*true/false] Overwrite  repository group  security settings  with
                                              import version
          cleanSolutionName     [newName]     A  clean  import  will be  performed  using the  new
                                              solution name. Only specify this if you want a clean
                                              import. Default: do not use clean import
          overrideSequenceTypes [*true/false] Override  existing  sequence  type  definitions  (in
                                              repository) with the sequence types contained in the
                                              import file
          updateSequences       [*true/false] Update sequences for all tables on  all servers used
                                              by the imported solution and modules
          overrideDefaultValues [*true/false] Override  existing  default  values  (in repository)
                                              with the default values contained in the import file
          allowSQLKeywords      [*true/false] Allow reserved SQL keywords as table or column names
                                              (will fail unless supported by the backend database)
          allowDataModelChanges [*true/false] Allow data model (database) changes
          skipDatabaseViewsUpdate
                                [true/*false] Skip database view create/update (table info will be
                                              imported)
          showDataModelChanges  [*true/false] Display data model (database) changes
          importMetaData        [*true/false] Import solution meta data
```

```
          importSampleData      [*true/false] Import solution sample data
          importI18NKeys        [*true/false] Import internationalization (i18n) data (inserts and
                                              updates)
          insertNewI18NKeysOnly [true/*false] Insert  new  internationalization (i18n)  keys  only
                                              (inserts only, no updates)
          importUserPolicy      [skip/create_new_users_update_groups/*overwrite_all]
                                              Where:

                                              [skip] Do not import users contained in import
                                              [create_new_users_update_groups] Create non-existing
                                                     users  and   add  existing  users  to  groups
                                                     specified in import
                                              [overwrite_all] Overwrite  existing users completely
                                                     (USE WITH CARE)

          addUsersToAdminGroup  [*true/false] Allow users to be added to the Administrators group.
          solutionPassword_name [password]    Only use this if the  export is password  protected.
                                              'name' is  the protected  solution's name. One  such
                                              property can be set for each protected solution
          importHookUserName    [userName]    Any  import   hooks  will  run  using   this  user's
                                              credentials. Null by default.
servoy.test.log4jConfigFile
     Key for the system property that points to a  log4j configuration properties file.  Optional.
     This property will be overridden if an exported service solution is specified by the  logging
     specified by the servoy.properties file.
```