IBM Security AppScan Source Utilities

Version 8.8.0.0

User Guide

IBM Security AppScan Source Utilities

Version 8.8.0.0

User Guide

Contents

Chapter 1. The Ounce/Make build utility

Ounce/Make is a tool that automates the importing of configuration information into AppScan® Source from build environments that use `makefile`. Ounce/Make eliminates the need to import configuration information from `makefiles` manually.

The Ounce/Make utility provides a command line interface for generating AppScan Source project files (`.ppf`) from `makefiles` (a `makefile` provides a means to build an executable application or library from source files). The generated `ppf` files contain all of the information needed by AppScan Source to assess the source code that the corresponding makefile is responsible for compiling. Once Ounce/Make generates the `.ppf` files, you can import or add the project files into AppScan Source.

To launch the utility:

- On Windows systems, run `<install_dir>\bin\ouncemake.exe` (where `<install_dir>` is the location of your AppScan Source installation), for example on Windows (32-bit):

  ```
  C:\Program Files\IBM\AppScan Source\bin\ouncemake.exe
  ```

- On Linux systems, run `<install_dir>/bin/ouncemake`, for example:

  ```
  /opt/ibm/appscansource/bin/ouncemake
  ```

`Make` is the tool that automates compiling, linking (and so forth), of programs, taking into account the interdependencies of modules and their modification times. `Make` reads instructions from a `makefile`, which specifies a set of targets to build, the files on which they depend, and the commands to execute to produce them.

Requirements

To create AppScan Source project files successfully, you must run Ounce/Make in a suitable environment. The following list itemizes the requirements for Ounce/Make to run successfully. If you do not meet all of these requirements, Ounce/Make fails.

- The directory from which Ounce/Make runs must contain a valid makefile.
- The build environment must be able to issue a make command that will succeed.
- Run the make clean command before running Ounce/Make. You can explicitly run make clean before running Ounce/Make or include it with Ounce/Make by specifying the `- clean` option.
- Makefiles that Ounce/Make encounters cannot contain hardcoded absolute paths under the following circumstances:
  - To the make executable when calling another makefile:

    For example, do not reference the path `/usr/bin/make -f makefile.mk`. In the makefile, reference make through the make executable or a variable. The variable may be the *make macro*, `${MAKE}`, or another variable that you specify in the Properties file.
  - To the compiler executable when compiling source code:

    For example, `/usr/bin/gcc -I.. -DFOO -o myfile.o myfile.cpp`
  - To the linker executable when linking object files

For example, `/usr/bin/ld file1.o file2.o`

- In `#include` statements.

  To use a `#include` statement, add the following flag to the project file as an option of the configuration:

  `--remote_root <remote dir>`

  where `<remote dir>` defines the mount point of the remote directory.

  Note: You can only specify a single `remote_root`. All hard coded paths to `#include` statements must resolve to a single mount point.
- Do not specify macros for the make, compiler, and linker executable on the command line when invoking make (for example, `make CC=gcc LD=ld`).

What Ounce/Make supports

Ounce/Make supports multiple computer platforms, compilers, and `make` versions.

Platforms

You can run Ounce/Make on all platforms supported by AppScan Source.

Make Versions

Ounce/Make supports the following make versions:

- make: The UNIX make utility is a software engineering tool for managing and maintaining computer programs
- gmake: GNU Make is a tool that controls the generation of executables and other non-source files of a program from the program's source files
- nmake: `NMAKE.exe` (the Microsoft Program Maintenance Utility) is a 32-bit tool that builds projects based on commands within a description file

Compilers

Ounce/Make supports the following compilers:

- GCC: The GNU Compiler Collection. AppScan Source supports the C and C++ front ends
- cl: `cl.exe` is the compiler to Microsoft Visual C++

Operation

Ounce/Make operates with your `Make` to extract the necessary configuration information from your makefiles and create appropriate AppScan Source project files (`ppf`). To accomplish this, Ounce/Make requires that you provide certain information about the build environment, such as the make version used.

Use the Properties file to describe the build environment (see “Ounce/Make properties file” on page 7).

Running Ounce/Make

You can run Ounce/Make from any directory where a make command executes successfully. In most cases, the `ouncemake` executable is not in the directory where you invoke it. Therefore, it is recommended that you add the directory that contains `ouncemake` to the `PATH` environment variable to invoke `ouncemake` easily.

For example, if your source code and/or makefile reside in the directory `\development\srcdir`, to run `ouncemake` you must first change to that directory:

```
%cd \development\srcdir
```

and then run `ouncemake`:

```
%ouncemake <options>
```

Note: For backward compatibility, the `pmake` command is still accepted. However, `pmake` may be removed in a future release.

Naming project files

Ounce/Make uses conventions outlined in this topic to name AppScan Source project files.

- When creating AppScan Source project files, Ounce/Make uses the relative directory path from the directory in which you invoked `ouncemake` to the directory where `ouncemake` creates the project file.
- The directory where you invoke `ouncemake` becomes the first component in the path name.
- Underscores replace all path separators such as backslash (\) on Windows and slash (/) on Linux.
- When the filename length exceeds operating system limitations, Ounce/Make strips components of the path, starting on the left, until the filename length complies with the system naming convention.
- When you run `ouncemake` at the root of a file system, such as `/` or `c:\`, `ouncemake` creates an AppScan Source project file named `root.ppf`.

AppScan Source saves the created `.ppf` in the location next to the makefile that the `ppf` represents. For example, if you run Ounce/Make, creating a single project file, AppScan Source saves the `ppf` in the directory from which you invoked Ounce/Make. Refer to "Example 2: Ounce/Make with recursive option" on page 15 to see the `ppf` files created in multi-project mode.

Note: If there is more than one makefile in a directory, Ounce/Make creates only one `.ppf` file in the directory.

Example 1

This example illustrates a `ppf` file created with the path separators replaced by underscores.

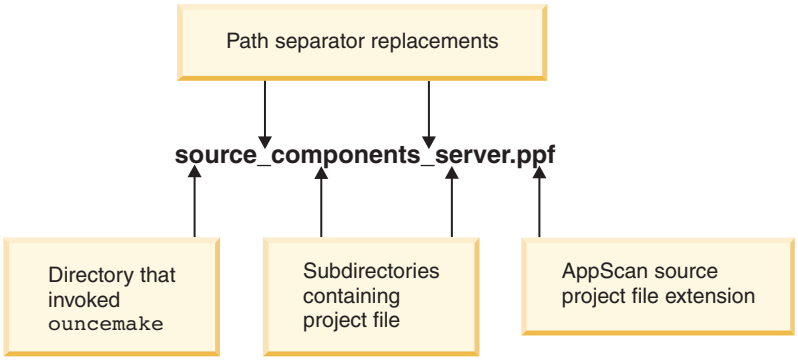
Invoke `ouncemake` from the following directory:

```
C:\development\source
```

During execution, Ounce/Make creates an AppScan Source project file in:

```
C:\development\source\components\server
```

The name of the `ppf` is `source_components_server.ppf`:



Example 2

Microsoft Windows and Linux limit path and file names. These operating systems limit the number of characters to 255. Example 2 shows the case of a filename exceeding path length limitations.

User invokes Ounce/Make from the following directory:

```
C:\path1\path2\path3\path4\path5\development\source
```

During execution, Ounce/Make creates an AppScan Source project in the following directory:

```
C:\path1\path2\path3\path4\path5\development\source\components\server
```

If the filename can be a maximum of 25 characters, due to path limitations, the resulting filename is:

```
components_server.ppf
```

Explicitly naming a project

You can optionally use the `OUNCE_PROJ_NAME` environment variable to explicitly specify a name for a newly created project. Define `OUNCE_PROJ_NAME=value` environment variable as with any other environment variable.

If not set, the name is generated by the existing project naming convention.

For example, to specify the project name as `MyMakeProject` from the command line:

```
$(MAKE) $(MAKE_ARGS) all OUNCE_PROJ_NAME=MyMakeProject
```

Ounce/Make output

While running, Ounce/Make generates output informing you of certain events.

The following events trigger output messages:

- Project creation
- Invoking make
- Error

Project creation

When Ounce/Make creates an AppScan Source project file, it outputs a message that includes the `ppf` name. The following example is an Ounce/Make project creation output message:

```
Created AppScan Source project <project_name>.ppf in directory <directory>.
```

Invoking make

When Ounce/Make invokes the make executable, the executable name and the options print. The following sample shows output generated when Ounce/Make invokes your make.

```
/usr/bin/gmake -f makefile.mk release
```

Error

If an error occurs during execution, Ounce/Make prints an error message that describes the error. If the error occurred when the make executable ran, then the error message displayed is the make executable error. If the error was specific to Ounce/Make, an appropriate error message describing the Ounce/Make error condition displays.

Ounce/Make command syntax and make options

Ounce/Make supports multiple options that can change the way it behaves when it runs.

You can set these options in the Properties file (see "Ounce/Make properties file" on page 7 for more details), or include them on the command line. If you set the options in the Properties file, it is not necessary to specify them on the command line every time you run Ounce/Make.

Synopsis

Ounce/Make supports the following syntax:

```
ouncemake [options] [-- make_options]
```

Options

A hyphen (**-**) must precede all options. You must specify options separately; you cannot concatenate them after a single hyphen. For example, the command:

```
ouncemake -sr
```

is not supported syntax, but you can run:

```
ouncemake -s -r
```

Note: Each option must be separated by a space.

When you run Ounce/Make, you can use the abbreviated option or the complete word.

The following table includes columns describing each option.

- **Option**: Identifies the option that Ounce/Make will understand when invoked.
- **Default**: If applicable, explains how Ounce/Make operates by default if you do not specify the option.
- **Description**: Ounce/Make behavior when using this option.

Option	Default when option not specified	Description
`-a <application_name>` `-application <application_name>`	Off	When specified, Ounce/Make creates an application file named `<application_name>.paf` that contains all projects created by Ounce/Make. The file is created in the directory in which `ouncemake` ran.
`-b` `-build`	Off	Perform the build while gathering the make options. This option is incompatible with Cygwin.
`-r` `-recursive`	Non-recursive	Ounce/Make recursively follows all calls to other makefiles. For example, if a makefile exists at the root of a source code tree to call all of the makefiles in the subdirectories, then invoking `ouncemake -r` in the directory containing the root makefile causes Ounce/Make to follow the calls to the subdirectory makefiles.
`-nr` `-non_recursive`	Non-recursive	Ounce/Make does not recursively follow calls to other makefiles.
`-s` `-single_project`	Multiple-project mode	Single-project mode. When in single-project mode, Ounce/Make generates only a single project file in the directory from which it is invoked. If not specified, Ounce/Make is in multiple-project mode.

`-ns`
`-non_single_project`
`-m`
`-multiple_project`
	Multiple-project mode	Multiple-project mode. In this mode, Ounce/Make generates an AppScan Source project file in each directory for every makefile that it encounters.

`-nv`
`-non_verbose`
`-q`
`-quiet`
	Non-verbose mode	Non-verbose mode. Ounce/Make only outputs its own output messages. Ounce/Make suppresses output from the make.

`-v` `-verbose`	Verbose mode	Verbose mode. Ounce/Make outputs the make's output, as well as its own, to standard output.
`-l` `log_level`	1 (off)	1 through 10. 10 provides the most logging.

Option	Default when option not specified	Description
-c <clean_command> -clean <clean_command>	Off	When specified, Ounce/Make interprets <clean_command> as a command, and executes it. <clean_command> should be the command that the user would normally execute to clean. For example, make clean is a common command for executing a clean. Note that you must enclose the command in quotation marks. Since Ounce/Make requires a clean before running, if you do not specify this option, a prompt appears asking if you want to continue.
-nc -no_clean	Off	Tell Ounce/Make not to run clean and do not display a prompt reminding that a clean is **not** going to be run.
-p <properties_file>	n/a	Allows users to specify a properties file for Ounce/Make to use. <properties_file> must be an absolute path to the properties file that Ounce/Make should use.
-? -h -help	n/a	Help for Ounce/Make options.

Ounce/Make properties file

The Ounce/Make Properties file (ouncemake_properties.xml) is an XML file that contains details about your build environment. When Ounce/Make runs, it finds the properties file based on the search path or the -p option. The properties file must specify the compiler used in your build environment, the linker, and make elements.

To learn about the required and optional elements that are specified in the properties file, see "Ouncemake properties file elements" on page 8.

When you run Ounce/Make, if you do not specify the -p option on the command line, Ounce/Make looks through its search path looking for the ouncemake_properties.xml file (see "Ounce/Make command syntax and make options" on page 5 to learn about specifying options on the command line). The Ounce/Make search path is platform-dependent.

The Microsoft Windows search path is:

- Current working directory
- <data_dir>\config (where <data_dir> is the location of your AppScan Source program data, as described in Chapter 9, "Installation and user data file locations," on page 149)

The Linux search path is:

- Current working directory
- <data_dir>/config (where <data_dir> is the location of your AppScan Source program data, as described in Chapter 9, "Installation and user data file locations," on page 149)
- User's home directory
- /etc

Note:

- On Linux, if you do not specify the -p option and the properties file does not exist in the search path, AppScan Source will automatically create one in the current working directory.
- For backwards compatibility, Ounce/Make recognizes pmake_properties.xml. If both pmake_properties.xml and ouncemake_properties.xml exist in the same directory, ouncemake_properties.xml takes precedence.

Ouncemake properties file elements

OuncemakeProperties, the root element of the Properties file, contains the elements that are listed in this topic.

Required elements:

- Compiler
- Linker
- Make

Optional elements:

- MakeOptions
- Options
- GlobalProjectOptions
- FileOptions
- Executable
- MountRoot

Compiler

The Compiler element specifies the compiler executable used in your build environment. The value of this element must be an absolute path to an executable. This element consists of one optional attribute, macro, which specifies the name of the variable that stores the compiler executable when building and a value specific to the path to the compiler. If you do not specify the macro attribute, Ounce/Make sets the macro attribute to CC by default. The Properties File can contain multiple Compiler elements, but at least one is required.

The value for the macro attribute must be unique across all Compiler elements. For example, you cannot list CC as a macro attribute more than once.

Example

```
<Compiler macro=CXX>/usr/bin/gcc</Compiler>
```

Description

`CXX`: Your makefiles reference this compiler with the `CXX` macro.

`/usr/bin/gcc`: Tells Ounce/Make that you use the `/usr/bin/gcc` compiler.

Linker

The `Linker` element specifies a linker executable used in the build environment. The value of this element must be an absolute path to an executable. The Properties File can contain multiple `Linker` elements, but at least one is required.

This element consists of one attribute, `macro`, which specifies the name of the variable that stores the `linker` executable when building. If you do not specify the `macro` attribute, `LD` is the default `linker macro`.

The value for the `macro` attribute must be unique across all `Linker` elements. For example, you cannot list `LD` as a `Linker` attribute more than once.

Example

```
<Linker macro=LD>/usr/bin/ld</Linker>
```

Description

`LD`: Tells Ounce/Make that you use the `LD` macro

`/usr/bin/ld`: Tells Ounce/Make that you use the `/usr/bin/ld` linker

Make

The `Make` element specifies a make executable used in your build environment. The value of this element must be an absolute path to an executable. The Properties File can contain multiple `Make` elements, but at least one is required.

This element consists of one attribute, `macro`, which specifies the name of the variable that stores the make executable when building. If you do not specify the `macro` attribute, Ounce/Make sets the attribute to `MAKE` by default.

The value for the make `macro` attribute must be unique across all `Make` elements.

Example

```
<Make macro=MAKE>/usr/bin/make</Make>
```

Description

- `MAKE`: Tells Ounce/Make that you use the `MAKE` macro.
- `/usr/bin/make`: Tells Ounce/Make that you use the `/usr/bin/make` make

MakeOptions

The `MakeOptions` element specifies the options to pass to the make executable. This element is optional and cannot appear more than once in the Properties File.

Example

```
<MakeOptions>-f makefile.mk release</MakeOptions>
```

Description

The options:

```
-f makefile.mk release
```

pass to the make executable.

Options

The `Options` element specifies options to pass to Ounce/Make upon invocation.

This element provides an alternative to the use of certain Ounce/Make command line options, as described in "Ounce/Make command syntax and make options" on page 5.

Attributes use the syntax:

```
<option> = <true | false>
```

The `Options` element and its attributes are not required.

The `Options` element may include the following attributes:

Attribute	Description
`recursive`	Boolean value. True or false. When true, implies `-r` command line option.
`single_project`	Boolean value. True or false. When true, implies `-s` command line option.
`verbose`	Boolean value. True or false. When true, implies `-v` command line option.
`clean`	String value, enclosed in quotation marks (`"`), such as `"make clean"`. When set, implies `-c` command line option. The value should be the command to execute to perform the clean. For example, `gmake clean`.
`build`	Boolean value. True or false. Perform the build while gathering the make options. **Note:** Incompatible with Cygwin.
`application`	String value. When set, implies `-a` option. The value specified should be the application name that you want.
`no_clean`	Boolean value. True or false. Tell Ounce/Make not to run clean and do not display a prompt reminding that a clean is not going to be run.

Note: Ounce/Make uses the options set in the Properties file. However, if you run Ounce/Make with options included on the command line, these options take precedence over options set in the Properties file. If you run Ounce/Make without options on the command line, Ounce/Make applies the Properties file options.

Example

The following is an example line from the Properties file that makes use of all attributes:

```
<Options recursive="true" single_project="false" verbose="false"
clean="nmake.exe clean" no_clean="false"></Options>
```

Description

- `recursive="true"`
- `single_project="false"` directs Ounce/Make to operate in multiple-project mode. When operating in multiple-project mode, the recursive attribute should also be set to `true`.
- `verbose="false"` turns off verbosity.
- `clean="nmake.exe clean"` cleans the build environment.
- `no_clean= "false"` tells Ounce/Make to run a clean command, suppressing the message stating that the clean will run.

`GlobalProjectOptions`

The `GlobalProjectOptions` element specifies Project level options for all files contained within the project. The `GlobalProjectOptions` element and its attributes are not required.

The following list describes the attributes for the `GlobalProjectOptions` element:

- `include_paths`: String value. Semicolon-separated list of include paths to apply to all files in the project.
- `macros`: String value. Semicolon-separated list of macros to apply to all files in the project.
- `compiler_options`: String value. List of compiler options, separated by a space, to apply to all files in the project. Do not specify `include` paths and macros here.

Example

The following is an example line from the Properties file that makes use of all attributes:

```
<GlobalProjectOptions include_paths="/usr/include;/usr/local/include"
macros="DEBUG;WIN32" compiler_options="-f non-const-strings"/>
```

Description

- `include_paths="/usr/include;/usr/local/include"` adds the path to the Includes section of the project configuration.
- `macros= "DEBUG;WIN32"` adds `DEBUG` and `WIN32` macros to the Macros section of the project configuration.
- `compiler_options=` adds the compiler options to the Options/Command Line section of the project Configuration.

`FileOptions`

The `FileOptions` element allows include paths, macros, and other compiler options to be specified for files with a particular extension. You can use `FileOptions` multiple times to specify different options for files with different extensions. For example, as shown below, if you have a project that contains both C and C++ files, create two `FileOptions` elements, one for each file type.

The following list describes the attributes for the `FileOptions` element:

- `extensions`: String value. Semicolon-separated list of file extensions. Every file with an extension that matches an extension in this list acquires the options specified by this property. If a file extension applies to more than one occurrence of the `FileOptions` property, then the first occurrence in the Ounce Make properties file takes precedence.
- `compiler_options`: String value. List of compiler options, separated by a space, to apply to all files with the specified extension. Do not specify `include` paths and macros here.
- `include_paths`: String value. Semicolon-separated list of `include` paths to apply to all files with the specified extension.
- `macros`: String value. Semicolon-separated list of macros to apply to all files with the specified extension.

Examples

The following `FileOptions` examples show how to configure the Ounce Make properties file to apply the correct options to both C and C++ files.

The `FileOptions` element with `extensions="c"` applies its other attribute values only to files with a `c` extension `<filename.c>`. The `FileOptions` element with `extensions="cpp;cxx"` will apply its other attribute values only to files with `cpp` (`<filename.cpp>`) or `cxx` (`<filename.cxx>`) extensions.

```
<!-- g++ options for C files -->
<FileOptions
 extensions="c"
 compiler_options="-gcc_linux_i386"
 include_paths="/usr/local/include;
 /usr/lib/gcc-lib/i386-redhat-linux/3.2.3/include;
 /usr/include"
 macros=""/>

<!-- g++ options for C++ files -->
 <FileOptions
 extensions="cpp;cxx"
 compiler_options="-g++_linux_i386"
 include_paths="/usr/include/c++/3.2.3;
 /usr/include/c++/3.2.3/i386-redhat-linux;
 /usr/include/c++/3.2.3/backward;/usr/local/include;
 /usr/lib/gcc-lib/i386-redhat-linux/3.2.3/include;
 /usr/include"
 macros="__GNUG__=3" />
```

Description

`extensions="c"` and `extensions="cpp;cxx"`

specifies the file extensions for which these file options apply.

Executable

The `Executable` element allows use of any executable (other than `compiler`, `linker`, or `make`) in the build environment. The value of this element must be an absolute path to an executable. The `Executable` element contains one attribute, `macro`, which specifies the name of the variable used to store the executable name when building. The value for the macro attribute must be unique across all `Executable` elements.

The Properties file may contain any number of Executable elements. The `Executable` element is optional and is only required if you use other executables in

your build environment that may cause Ounce/Make to fail. By identifying these executables, Ounce/Make can prevent the failure.

Example

The following is an example line from the Properties file:

```
<Executable macro=ARCHIVE>/usr/bin/ar<Executable>
```

Description

This example identifies to Ounce/Make that the build uses the `/usr/bin/ar` executable and that the makefiles reference this executable with the `ARCHIVE` macro.

`MountRoot`

The `MountRoot` element is required if you are assessing source code from a remote system. `MountRoot` tells Ounce/Make to pre-pend the specified mount point to all absolute `include` paths that it encounters.

By specifying a `MountRoot`, Ounce/Make implicitly adds the `--remote_root` option to the list of compiler options in the resulting `ppf`.

`MountRoot` is valid on Linux only.

Example

The following is an example line from the Properties file:

```
<MountRoot>/mnt/mount_point/usr/include</MountRoot>
```

Description

If one of the specified include paths is `/usr/include` (such as `gcc -I/usr/include ...`), Ounce/Make pre-pends `/usr/include` with the mount point so that the path stored in the resulting `ppf` file contains the correct path, such as `/mnt/mount_point/usr/include`.

Sample properties files

The AppScan Source installation includes these sample properties files in the `<data_dir>\config` directory (where `<data_dir>` is the location of your AppScan Source program data, as described in Chapter 9, "Installation and user data file locations," on page 149):

- `SampleOuncemakeProperties-Windows.xml`
- `SampleOuncemakeProperties-Linux.xml`

You can modify these files to create a custom properties file. If you are not familiar with XML, use these samples as templates to create your file.

Examples

This section describes three ways to use Ounce/Make.

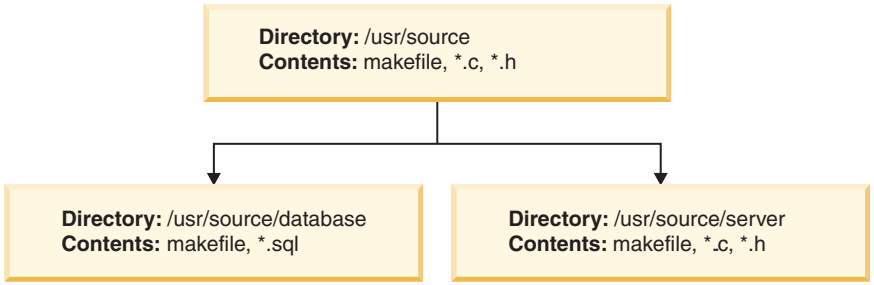
"Example 1: Ounce/Make without options" on page 14 illustrates Ounce/Make without options, creating one AppScan Source project file based only on the makefile in the directory in which you invoke Ounce/Make.

"Example 2: Ounce/Make with recursive option" on page 15 uses Ounce/Make with the `-r` (recursive) option, telling Ounce/Make to operate recursively and follow all calls to other makefiles.

In "Example 3: Ounce/Make with single-project and recursive option" on page 16, Ounce/Make uses both the `-r` (recursive) and `-s` (single-project) options, to create a single AppScan Source project file based on recursive processing of all makefiles that Ounce/Make encounters.

Directory structure and files

All three examples use the same directory structure and files:



This diagram shows a root directory (`/usr/source`) that contains a makefile and source files. The `/usr/source` directory contains two subdirectories, `/usr/source/database` and `/usr/source/server`. The `/usr/source/database` directory contains a makefile and SQL files. The `/usr/source/server` directory contains a makefile and source files.

This example makes the following assumptions about the three makefiles:

- The makefile in `/usr/source` builds the source files in `/usr/source` and calls the makefiles in `/usr/source/database` and `/usr/source/server`.
- The makefile in `/usr/source/database` imports the SQL files into a database.
- The makefile in `/usr/source/server` builds the source files in `/usr/source/server`.

Example 1: Ounce/Make without options

This example illustrates using Ounce/Make non-recursively. In non-recursive mode, Ounce/Make only looks at the makefile in the directory from which it is invoked. If the original makefile contains calls to other makefiles, Ounce/Make ignores them.

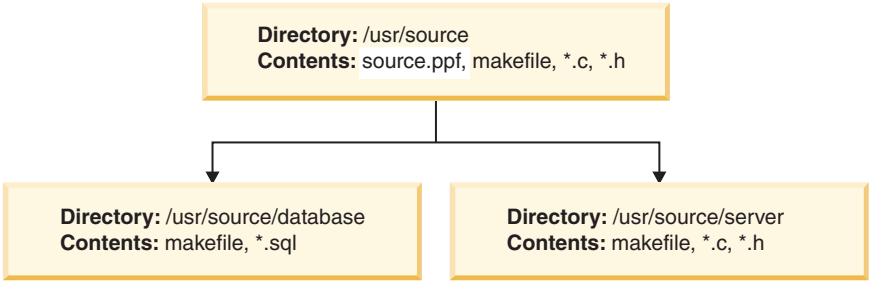
This example runs Ounce/Make from `/usr/source`.

See "Directory structure and files" for a graphical explanation of the directory structure and files that are assumed by this example.

Command

```
ouncemake
```

The following diagram shows the contents of the directories after Ounce/Make runs:



After Ounce/Make runs, `/usr/source` now contains an AppScan Source project file named `source.ppf`. This project file contains all necessary information to assess all source files in `/usr/source`. Ounce/Make in non-recursive mode ignores the calls in the makefile in `/usr/source` calling the makefiles in `/usr/source/database` and `/usr/source/server`.

Example 2: Ounce/Make with recursive option

With the `-r` option, Ounce/Make operates recursively (follows calls to other makefiles contained inside a makefile). Since multiple-project file mode is the default mode, when used with the `-r` option, Ounce/Make creates an AppScan Source project file for every makefile encountered that compiles source code.

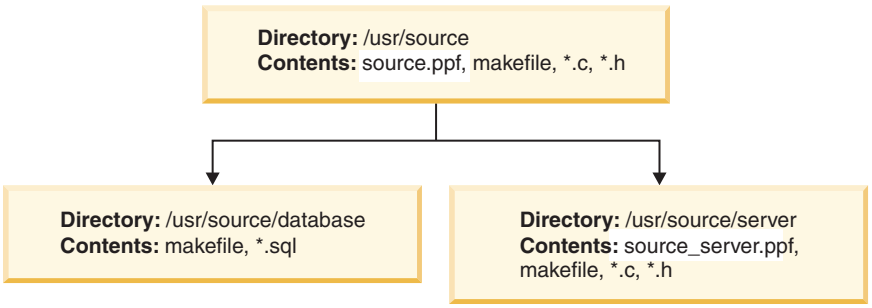
See "Directory structure and files" on page 14 for a graphical explanation of the directory structure and files that are assumed by this example.

Command

```
ouncemake -r
```

The `-r` (recursive) option instructs Ounce/Make to follow makefile calls to other makefiles. For a more detailed description of the recursive option, refer to the table in "Ounce/Make command syntax and make options" on page 5.

The following diagram shows the contents of the directories after Ounce/Make runs:



In this example, Ounce/Make creates an AppScan Source project file in `/usr/source` and `/usr/source/server`. Since the makefile in `/usr/source` called the makefiles in `/usr/source/database` and `/usr/source/server`, Ounce/Make checked to see if those makefiles compiled source code.

In the case of the makefile in `/usr/source/database`, Ounce/Make determined that this makefile does not compile source code; therefore, it did not create an AppScan Source project file. However, Ounce/Make determined that the makefile in

/usr/source/server compiled the source files in that directory, and thus generated an AppScan Source project file for the source files in /usr/source/server.

Example 3: Ounce/Make with single-project and recursive option

Example 3 illustrates using Ounce/Make recursively in single-project mode. Ounce/Make generates a single AppScan Source project file for a combination of the source code compiled by all of the makefiles encountered.

See "Directory structure and files" on page 14 for a graphical explanation of the directory structure and files that are assumed by this example.

Run the following command from the /usr/source directory:

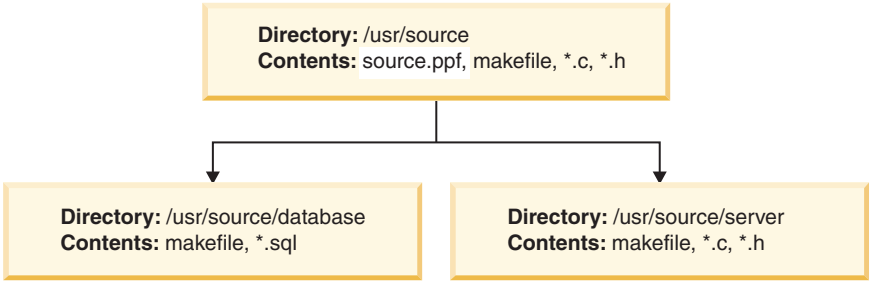
Command

```
ouncemake -r -s
```

The -r (recursive) option instructs Ounce/Make to follow makefile calls to other makefiles. For a more detailed description of the recursive option, see the table in "Ounce/Make command syntax and make options" on page 5.

The -s option instructs Ounce/Make to generate only one AppScan Source project file in the directory from which it is invoked, as opposed to creating a new project for every makefile encountered.

The following diagram shows the contents of the directories after Ounce/Make runs.



A single AppScan Source project file exists in /usr/source. This AppScan Source project file contains the configuration information for the entire source code in /usr/source and /usr/source/server.

Chapter 2. AppScan Source command line interface (CLI)

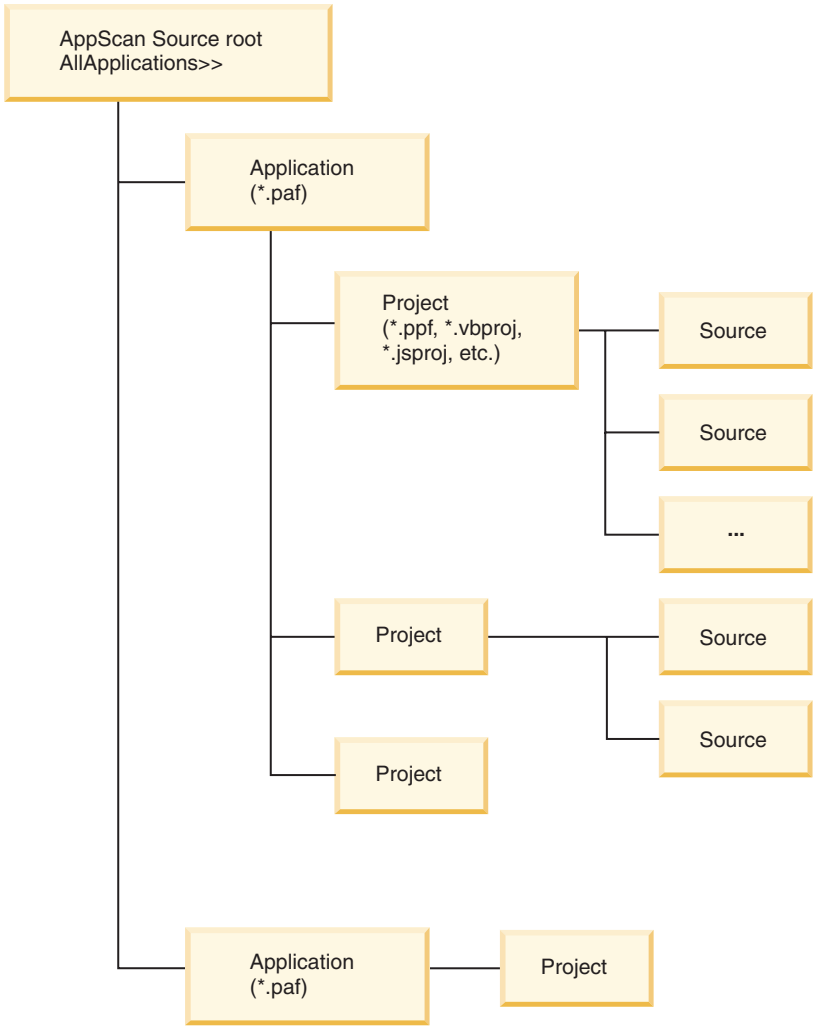
The CLI is an interface to core AppScan Source functionality.

The CLI supports performing AppScan Source functions from the command line or running script files to record commands issued during an interactive session. When running, the CLI provides user feedback as output to the console or optionally to a log file.

Objects and context

The AppScan Source object tree

The object tree shows the basic structure of AppScan Source objects. You can scan any object (applications, projects, or source files) within the tree structure.



Active objects and current context

Many AppScan Source command line interface (CLI) commands act only within the current context; that is, within the current directory or on a specified file. Use the `SetCurrentObject (SET, CD)` command to move to the object before calling any command that requires a current context.

Additionally, some CLI commands only act on an active object; which may be the currently selected source file, project, or application with an assessment.

AppScan Source command line interface (CLI) permissions

Many CLI commands require that you have the appropriate AppScan Source permissions.

If you do not have permission for an action, an error message appears. Note that the `Scan (SC)`, `Publish`, and `Report (RPT)` CLI commands require an AppScan Source for Automation license.

For more information, see your administrator.

Starting the AppScan Source command line interface (CLI)

- On Windows systems, run `<install_dir>\bin\AppScanSrcCli.exe` (where `<install_dir>` is the location of your AppScan Source installation). For example, on Windows (32-bit):

  ```
  C:\Program Files\IBM\AppScanSource\bin\AppScanSrcCli.exe
  ```

- On Linux systems, run `<install_dir>/bin/appscansrccli`, for example:

  ```
  /opt/ibm/appscansource/bin/appscansrccli
  ```

The CLI allows one valid command to be passed on the command line at application start up.

Example:

The CLI passes the `script` command to execute an entire CLI session automatically.

```
<install_dir>\bin\AppScanSrcCli script myscript.txt
```

Displaying the AppScan Source command line interface (CLI) in national languages

This topic describes the steps required for displaying the CLI in national languages.

Procedure

- Open `<data_dir>\config\cli.cp` (where `<data_dir>` is the location of your AppScan Source program data, as described in Chapter 9, "Installation and user data file locations," on page 149) in a text editor. Add this line to the file:

  ```
  -Duser.language=<language>
  ```

  Where `<language>` is the language that you want to display. These values are supported:

  – `de` for German

  – `es` for Spanish

- fr for French
- it for Italian
- ja for Japanese
- ko for Korean
- pt_BR or pt-BR for Brazilian Portuguese
- zh_CN or zh-CN for Simplified Chinese
- zh_TW or zh-TW for Traditional Chinese

Save the file after you have added this line.

- Open <data_dir>\config\ounce.ozsettings in a text editor. Locate the locale setting in the file. This setting will look similar to:

```
<Setting
   name="locale"
   value=""
   default_value=""
   description="The language (and optionally the Country/Region)
       in which UI and messages should be displayed. E.g. fr for
       French, or pt-BR for Portuguese (Brazil)"
   display_name="Locale"
   type="text"
   available_values=""
   read_only="false"
   hidden="false"
   force_upgrade="false"
 />
```

  In this setting, modify the value attribute so that it is set to the same language setting that was specified in the above step. For example, if in the above step you added -Duser.language=fr, change the value attribute to value="fr".

  Save the file after you have modified this setting.

- Start the CLI (if these changes were made while the CLI was already running, you will need to restart the CLI).

Enabling quality scanning

Quality scanning can be enabled by optional command arguments in the AppScan Source command line interface (CLI) and in AppScan Source for Automation.

Quality scanning is enabled in the command line interface when you use the scan command. See scan (sc) for more details.

If you are using AppScan Source for Automation to automate scanning, quality scanning is enabled via the ScanApplication command. See ScanApplication for more details.

Note: If you are using AppScan Source for Automation on Linux, you may encounter this error in the CLI output log:

```
Xlib: connection to ":1.0" refused by server
Xlib: No protocol specified
(Eclipse:13598): GLib-GObject-WARNING **:
   invalid (NULL) pointer instance
```

This is occurs when the AppScan Source for Automation user account does not have permission to conduct quality scanning. To correct this:

- From a terminal, issue the command: xauth list $DISPLAY

This command will return the cookie that is used for the current display (for example, `my_hostname/unix:1  MIT-MAGIC-COOKIE-1 3968070ad5056a979d8e6a3a81017d5c`).

Copy the cookie to the clipboard.

- Use the `su` command to switch to the AppScan Source for Automation user.
- Issue the command, `xauth add <identifier>`, where `<identifier>` is the value that was copied to the clipboard in Step 1.

scan (sc)

Description

Scans an application (or all applications), project, or file. A valid AppScan Source for Automation license is required for use of this command.

Syntax

```
scan [path][config <proj_config>][-name <assessment_name>]
[-scanconfig <scan_configuration_name>]
[-quality <path_file>]
[-qualityOnly <path_file>]
```

- `path`: Optional. Full path in which the exported assessment should be saved.
- `config <proj_config>`: Optional. This argument is only valid for project-level assessments. If your project has a configuration file, specify it using this argument.
- `-name <assessment_name>`: Optional. Provide a name for the assessment.
- `-scanconfig <scan_configuration_name>`: Optional. Specify the name of a scan configuration to use for the scan. If a scan configuration is not specified, the default scan configuration will be used for the scan.
- `-quality <path_file>`: Optional. To scan for quality issues (in addition to security vulnerabilities), include the `quality` argument. When specified, you must include the path and file name of one of these files:
  - Quality rule (`.quality`) file:

    Quality rule files are automatically created when you configure quality scanning in the AppScan Source for Development Eclipse Plug-in. When you create a quality rule configuration, the quality rule file for it is saved to `<user_home>\.ouncequality\rulesets\<quality_rule_config_name>.quality`, where:

    - `<user_home>` is your operating system home directory (for example, on Windows, the directory might be `C:\Documents and Settings\Administrator\`).
    - `<quality_rule_config_name>` is the name that you gave the quality rule configuration when creating it in the AppScan Source for Development Quality Rule Configuration dialog box.

    AppScan Source includes three default quality rule files:

    - `codereview_basic_cpp.quality`: Quality rule configuration that includes recommended C/C++ Code Review rules.
    - `codereview_basic_java.quality`: Quality rule configuration that includes recommended Java™ Code Review rules (designed to generate only high-severity findings).
    - `codereview_extended_java.quality`: This rule file includes the same rules that are found in`codereview_basic_java.quality`, along with:

- Additional Java code review rules, including some that produce findings of lower severity.
- Java data flow analysis rules.

The default rule files are located in `<install_dir>\quality\rulesets` (where `<install_dir>` is the location of your AppScan Source installation). The provided default rule files cannot be modified.

To learn more about quality rule files, see the AppScan Source for Development Eclipse Plug-in infocenter.

– Quality rule argument script file: Text file that contains "Quality scan script arguments" on page 23. This option allows you to fine tune your scan by including/excluding files - and specifying C/C++ headers, macros, and other arguments.

- `qualityonly <path_file>`: Optional. To scan for quality issues only, include the `qualityonly` argument. When specified, you must include the path and file name of one of these files:
  – Quality rule (`.quality`) file:

    Quality rule files are automatically created when you configure quality scanning in the AppScan Source for Development Eclipse Plug-in. When you create a quality rule configuration, the quality rule file for it is saved to `<user_home>\.ouncequality\rulesets\<quality_rule_config_name>.quality`, where:

    - `<user_home>` is your operating system home directory (for example, on Windows, the directory might be `C:\Documents and Settings\Administrator\`).
    - `<quality_rule_config_name>` is the name that you gave the quality rule configuration when creating it in the AppScan Source for Development Quality Rule Configuration dialog box.

    AppScan Source includes three default quality rule files:

    - `codereview_basic_cpp.quality`: Quality rule configuration that includes recommended C/C++ Code Review rules.
    - `codereview_basic_java.quality`: Quality rule configuration that includes recommended Java Code Review rules (designed to generate only high-severity findings).
    - `codereview_extended_java.quality`: This rule file includes the same rules that are found in`codereview_basic_java.quality`, along with:
      - Additional Java code review rules, including some that produce findings of lower severity.
      - Java data flow analysis rules.

    The default rule files are located in `<install_dir>\quality\rulesets` (where `<install_dir>` is the location of your AppScan Source installation). The provided default rule files cannot be modified.

    To learn more about quality rule files, see the AppScan Source for Development Eclipse Plug-in infocenter.

  – Quality rule argument script file: Text file that contains "Quality scan script arguments" on page 23. This option allows you to fine tune your scan by including/excluding files - and specifying C/C++ headers, macros, and other arguments.

Examples

- To scan the default configurations of projects in all applications:

```
AllApplications>> Scan
```

The results appear as:

```
New Scan started
  .
  .
Preparing for Vulnerability Analysis...
Performing Vulnerability Analysis...
Generating Findings...
Preparing project for scan...
  .
  .
Scanned Project:
  Total files: 15
  Total findings: 167
  Total lines: 385
 vkloc: 0.44448395412925595
 v-Density: 22.446439683527426
Scanned Application:
  Total files: 15
  Total findings: 167
  Total lines: 385
 vkloc: 0.44448395412925595
 v-Density: 22.446439683527426
Scan completed:
  Total files: 15
  Total findings: 167
  Total lines: 385
 vkloc: 0.44448395412925595
 v-Density: 22.446439683527426
Elapsed Time - 18 Seconds

New Scan started. Please wait...
Assessment complete
-------------------
Total Call Sites: 75
Total Definitive Security Findings with High Severity: 25
Total Definitive Security Findings with Medium Severity: 37
Total Definitive Security Findings with Low Severity: 9
Total Suspect Security Findings with High Severity: 20
Total Suspect Security Findings with Medium Severity: 80
Total Suspect Security Findings with Low Severity: 60
Total Scan Coverage Findings with High Severity: 50
Total Scan Coverage Findings with Medium Severity: 33
Total Scan Coverage Findings with Low Severity: 17
Total Lines: 3000
...
```

- To scan the debug configuration of `Prj1`:

  ```
  AllApplications\Prj1>> SC config debug
  ```

- To conduct a scan that only looks for quality issues in `App1` (using the `C:\shared\Java Globalization.quality` quality rule file) and sends the scan results to `C:\myAssessment.ozasmt`:

  ```
  AllApplications\App1>> scan "C:\myAssessment.ozasmt"
  -qualityonly "C:\shared\Java Globalization.quality"
  ```

  The results appear as:

  ```
  New scan started

  Importing rules for quality scanning from: C:\shared\Java Globalization.quality
  Quality rules have been imported: 47
  Preparing resources for quality scanning.
  Resources have been prepared.
  Quality scanning...
  Quality scanning is complete, generating findings...
  Quality findings have been generated.

  Scanned application App1: Total files: 1 Total findings: 1
  ```

```
Total lines: 65 vkloc: 0.000000 v-density: 0.000000
Scan completed: Total files: 1 Total findings: 1
Total lines: 65 vkloc: 0.000000 v-density: 0.000000
Elapsed Time - 0 Hour(s) 0 Minute(s) 14 Second(s)

 -------------------
 Total Call Sites: 1
 Total Definitive Security Findings with High Severity: 0
 Total Definitive Security Findings with Medium Severity: 0
 Total Definitive Security Findings with Low Severity: 0
 Total Suspect Security Findings with High Severity: 0
 Total Suspect Security Findings with Medium Severity: 0
 Total Suspect Security Findings with Low Severity: 0
 Total Scan Coverage Findings with High Severity: 0
 Total Scan Coverage Findings with Medium Severity: 0
 Total Scan Coverage Findings with Low Severity: 0
 Total Lines: 65
 Max V-Density: 0
 Max V/kloc: 0
 V-Density: 0
 V/kloc: 0
```

Quality scan script arguments

When launching a quality scan via the command line interface `scan` command or the AppScan Source for Automation `ScanApplication` command, you can pass quality rules to the command using a text file script that contains arguments listed in this help topic.

The script can be placed in a flat text file with any file extension except `.quality`. Valid arguments include:

```
-directory <path to source directory>
-includefile <files in directory to include in the scan>
-excludefile <files in directory to exclude from the scan>
-rulefile <path to rules file>
-bindirectory <path to the binary directory that contains the .class files>
-javacp <fully-qualified build classpath>
-cppheaders <path to C++ headers file>
-cppmacros <path to C++ macros file>
```

See the following table for details about each of these arguments:

Table 1. Quality scan script arguments

Parameter	Classification	Description
`-directory`	Mandatory	Specify the fully-qualified path to the directory or directories that contain the source for the application or project that is being scanned. Use a comma-separated list to specify multiple directories.
`-includefile`	Optional for the `-directory` option	Specify the fully-qualified path to a text file that contains a line-separated list of source files to include in the scan. Instead of scanning all the source files in the source directory (`-directory`), this parameter enables you to scan only those files that are listed in the text file. In the text file, the file name and path for each source file must be relative to the source directory (not fully-qualified). Enter each source file name and path on a separate line in the file.

Table 1. Quality scan script arguments (continued)

Parameter	Classification	Description
`-excludefile`	Optional for the `-directory` option	Specify the fully-qualified path to a text file that contains a line-separated list of source files to exclude from the scan. Instead of scanning all the source files in the source directory (`-directory`), this parameter enables you to identify the files in the source directory that you do not want to scan. In the text file, the file name and path for each source file must be relative to the source directory (not fully-qualified). Enter each source file name and path on a separate line in the file.
`-rulefile`	Mandatory	Specify the fully-qualified path to the `.quality` rule file that contains the rule set to use to scan the source files. These are automatically created when you configure quality scanning in the AppScan Source for Development Eclipse Plug-in. When you create a quality rule configuration, the quality rule file for it is saved to `<user_home>\.ouncequality\rulesets\<quality_rule_config_name>.quality`, where: • `<user_home>` is your operating system home directory (for example, on Windows, the directory might be `C:\Documents and Settings\Administrator\`). • `<quality_rule_config_name>` is the name that you gave the quality rule configuration when creating it in the AppScan Source for Development Quality Rule Configuration dialog box. AppScan Source includes two default quality rule files: • `critical_rules_cpp.quality`: Quality rule configuration that includes recommended C/C++ Code Review rules. • `critical_rules_java.quality`: Quality rule configuration that includes recommended Java code review rules. The default rule files are located in `<install_dir>\quality\rulesets` (where `<install_dir>` is the location of your AppScan Source installation). The provided default rule files cannot be modified. To learn more about quality rule files, see the AppScan Source for Development Eclipse Plug-in infocenter.
`-bindirectory`	Mandatory for Java data flow analysis only	Specify the fully-qualified path to the binary directory that contains the `.class` files for the project or application that you are scanning. This parameter only applies to Java data flow analysis.

Table 1. Quality scan script arguments (continued)

Parameter	Classification	Description
`-javacp`	Mandatory for Java data flow analysis only	Specify the fully-qualified build class path for your project or application. A build class path is specified for each project or application. To scan source code in multiple projects, separate jar files or directories with semicolons as shown in these examples. One project: `-javacp C:\codeAnalysis\jdk6_02_backup\jre\lib` Two projects:`-javacp C:\codeAnalysis\jdk6_02_backup\jre\lib;c:\swt` This parameter only applies to Java data flow analysis.
`-cppheaders`	Optional	Specify the fully-qualified path to a text file that contains custom C/C++ headers. In this file, each header should be on its own line. This parameter only applies to C/C++ data flow analysis.
`-cppmacros`	Optional	Specify the fully-qualified path to a text file that contains custom C/C++ definitions. In this file, each macro definition should be on its own line. This parameter only applies to C/C++ data flow analysis.

Command syntax

AppScan Source command line interface (CLI) commands conform to a usage template with required and optional arguments, similar to a command shell. CLI commands are not case sensitive and do not require switches for different arguments.

The syntax for all CLI commands is:

```
command required_args [optional_args]
```

The command descriptions and usage adhere to the following conventions:

- **`command`**: Actual command, identified in a **`bold Courier`** font.
- Required arguments: Identified in `courier font`.
- Literals: Identified by *`italics`*.
- Optional arguments: Enclosed in brackets `[ ]`.
- Required arguments that have options: Enclosed in braces {}.
- Each command has a long name and an abbreviation. Find the abbreviations in parentheses in the command heading.
- All command names are one word with no spaces or tabs.
- Arguments do not require command switches but are **order dependent**.

  Important: It is important to remember that arguments are order dependent.

Arguments may contain spaces and tabs. Enclose arguments that contain spaces or tabs with double quotation marks (`" "`), otherwise they are parsed as multiple arguments.

AppScan Source command line interface (CLI) commands

This section describes all available CLI commands, required and optional arguments, and examples. Each command corresponds to a specific command category as noted in "AppScan Source command line interface (CLI) command summary."

Each command consists of a long name and an abbreviation. For example, the command to scan an application, project, or file and create an assessment is `scan` with `sc` as the abbreviation.

AppScan Source command line interface (CLI) command summary

The following table lists CLI commands, a description of each command, and whether login is required.

Table 2. CLI commands

Command and abbreviation	Description	Login required
`about (a)`	Display the AppScan Source Command Line Interface version and copyright information.	
`delete (del)`	Deletes a child object from the current object.	Yes
`deleteassess (da)`	This command has been renamed. See `removeassess (da)`.	
`deleteuser (du)`	Delete a user from the AppScan Source database.	Yes
`delvar (dv)`	Delete a single variable.	Yes
`details (det)`	List the assessment details for the current object.	Yes
`echo`	Echoes all input and output to the screen.	
`getaseinfo (gase)`	Print AppScan Enterprise Server settings.	Yes
`help (?)`	Lists help for all commands or a single command.	
`import (im)`	Use the `import` command to add projects (such as `.ppf`) to an existing application.	Yes
`info (i)`	Lists information about the current object's properties and values.	Yes
`list (ls, dir)`	List all objects under the current object in the object tree. The tree displays as a graphical representation of the object ID, name, and type.	Yes

Table 2. CLI commands (continued)

Command and abbreviation	Description	Login required
`listassess (la)`	List the object ID and assessment date/time for the current object in the object tree. Use `listassess` to obtain an ID for use with the `details` command.	Yes
`listgroups (lgrp)`	Lists all of the groups, their permissions, and a description of each.	Yes
`listusers (lu)`	Lists all AppScan Source users.	Yes
`log`	Switch message logging on or off.	
`login (in)`	Log in to the AppScan Enterprise Server (replaces `login_local (local)`).	
`logout (out)`	Log out of AppScan Source and terminate the command line interface AppScan Source command line interface (CLI) session.	Yes
`moduser (mu)`	Modify user information such as product access, permissions, user ID, and email for an AppScan Source or AppScan Enterprise Server user.	Yes
`newuser (nu)`	Create a new AppScan Source user (a valid user name, password, full name, and e-mail address are required). AppScan Source users can exist in the AppScan Enterprise Server user repository and in the AppScan Source database - or, if you have cause to have users that cannot access the server, they can be created locally as AppScan Source users. You can also create a new AppScan Source user that already exists on the AppScan Enterprise Server.	Yes
`openapplication (oa)`	Open an application. This command can also be used to create a new AppScan Source application file.	Yes
`openassessmentfile (oaf)`	Opens an AppScan Source assessment file (`file_name.ozasmt`).	Yes

Table 2. CLI commands (continued)

Command and abbreviation	Description	Login required
`password (passwd)`	The `password` command allows you to change your password or, if you have administrator permissions, to change another user's password.	Yes
`printuser (pu)`	The `printuser (pu)` command displays information about a single user on the screen.	Yes
`publishassess (pa)`	Publish the current assessment or a selected assessment to AppScan Enterprise Server. When this command is used, the assessment is made available to an AppScan Source client such as AppScan Source for Analysis - but it is not made available to the AppScan Enterprise Console (use the "`publishassessase (pase)`" on page 46 command to publish to the AppScan Enterprise Console).	Yes
`publishassessase (pase)`	Publish the current assessment or a selected assessment to the AppScan Enterprise Console. When this command is used, the assessment is not available to AppScan Source clients such as AppScan Source for Analysis (use the "`publishassess (pa)`" on page 45 command to publish to AppScan Source clients).	Yes
`quit`	Ends and closes the AppScan Source command line interface session. Issues a logout if you are logged in.	
`record (rc)`	Turns command recording on or off.	
`refresh (rf)`	`Refresh` restores the current project or application from disk.	Yes
`register (reg)`	Register projects and applications with the AppScan Source database.	Yes
`removeassess (da)`	Remove the selected or current assessment from memory.	Yes

Table 2. CLI commands (continued)

Command and abbreviation	Description	Login required
`report (rpt)`	`Report` generates an AppScan Source report of the specified type, including findings reports and AppScan Source reports. A valid AppScan Source for Automation license is required for use of this command.	Yes
`scan (sc)`	Scans an application (or all applications), project, or file. A valid AppScan Source for Automation license is required for use of this command.	Yes
`script (scr)`	Run a script of commands.	
`setaseinfo (sase)`	Specify Enterprise Console settings.	Yes
`setcurrentobject (set, cd)`	Use `setcurrentobject` to navigate the object tree.	Yes
`setvar (sv)`	Creates a new variable or modifies an existing variable.	Yes
`unregister (unreg)`	This command is used to unregister a previously registered application or project from the current node.	Yes

The following commands have been removed from the CLI or deprecated:

- `add`: Deprecated. Do not use.
- `listproducts (lprod)`: Deprecated. Do not use.
- `liststopobject (lstop)`: Deprecated. Do not use.
- `login_admin`: Removed. Do not use.
- `login_local (local)`: Deprecated. Use `login (in)`.
- `new`: Deprecated. Use `openapplication (oa)`.
- `reset (r)`: Removed. Do not use.
- `runassess (ra)`: Deprecated. Use `scan (sc)`.

about (a)

Description

Display the AppScan Source Command Line Interface version and copyright information.

Syntax

```
about
```

Example

```
AllApplications>> About
--------------------------------------------------------------------------------
-
|       Security AppScan Source 8.5.0.0 Build 175.
|
|       Licensed Materials - Property of IBM Corp. (C) Copyright IBM"
|       Corp. and its licensors 2003, 2011.  All Rights Reserved.  IBM,
|       the IBM logo, ibm.com, Rational, AppScan and ClearQuest are
|       trademarks or registered trademarks of International Business
|       Machines Corp., registered in many jurisdictions worldwide.
|       Other product and service names might be trademarks of IBM or
|       other companies. A current list of IBM trademarks is available
|       on the Web at "Copyright and trademark information" at
|       www.ibm.com/legal/copytrade.shtml. Linux is a registered
|       trademark of Linus Torvalds in the United States, other
|       countries, or both. Microsoft, Windows, Windows NT and the
|       Windows logo are trademarks of Microsoft Corporation in the
|       United States, other countries, or both. UNIX is a registered
|       trademark of The Open Group in the United States and other
|       countries.  Java and all Java-based trademarks and logos are
|       trademarks of Sun Microsystems, Inc. in the United States, other
|       countries, or both.
|
|       This program includes: Jacorb 2.3.0, Copyright 1997-2006 The
|       JacORB project; Jericho HTML Parser 2.1, Copyright 2005 Martin
|       Jericho; and XOM 1.0d22, Copyright 2003 Elliotte Rusty Harold,
|       each of which is available under the Gnu Library General Public
|       License (LGPL), a copy of which is available in the Notices file
|       that accompanied this program.
--------------------------------------------------------------------------------
```

delete (del)

Description

Deletes a child object from the current object.

Use the `list` (`ls`, `dir`) command to view the children of the current object.

Identify the child object to delete by name or ID.

Syntax

del {name|*id* object_id}

- `name`: Child object name. To view the children of the current object, issue a `list` command.
- *id*: Literal. Indicates deleting by object ID.
- `object_id`: Required argument when deleting by ID. The `object_id` is the numerical ID of the child to delete.

Examples

- To delete an application named `App1`:

  ```
  AllApplications>> Delete App1
  ```

- Delete a project with ID 40:

  ```
  AllApplications\App1>> Delete id 40
  ```

deleteassess (da)

This command has been renamed. See `removeassess (da)`.

deleteuser (du)

Description

Delete a user from the AppScan Source database.

Syntax

du username

username: The AppScan Source user to delete.

Example

To delete the user, agraham:

```
AllApplications>> deleteuser agraham
AllApplications>> Deleted user 'agraham'.
```

delvar (dv)

Description

Delete a single variable.

Syntax

delvar <variable name>

variable name: The name of the variable to delete. If a variable of that name exists, it is immediately deleted, followed by a success message.

Example

To delete the myvar variable:

```
AllApplications>> delvar myvar
Deleted variable '%myvar%'.
```

details (det)

Description

List the assessment details for the current object.

Retrieve details about a current assessment or an assessment selected by Assessment ID. Use `listassess (la)` to get a list of assessments and their IDs.

Tip: Use the `log` command to turn on logging before using the `details` command. You can generate a log file in comma-delimited text (`.csv`) format that Microsoft Excel or another program that accepts CSV data can open.

Syntax

details [id]

`id`: Object ID for an assessment from the current session. If an ID is not specified, details are displayed (or sent to a log file) for the most recent assessment.

Example

To display the details of the most recent assessment:

```
AllApplications\Myapps>> details
File, line, col, name, type, severity, confidence
C:\MyApps\WebGoat\webgoat\src\lessons\CommandInjection.java
, 119
, 0
, java.lang.Throwable.printStackTrace
, Vulnerability.Info
, 3-SeverityType_info
, 3-ConfidenceType_low
  .
  .
  .
C:\MyApps\WebGoat\webgoat\src\session\ECSFactory.java
, 625
, 0
, java.lang.String.equalsIgnoreCase
, Vulnerability.Info
, 3-SeverityType_info
, 3-ConfidenceType_low

 -------------------
 Total Call Sites: 1283
 Total Definitive Security Findings with High Severity: 10
 Total Definitive Security Findings with Medium Severity: 15
 Total Definitive Security Findings with Low Severity: 53
 Total Suspect Security Findings with High Severity: 24
 Total Suspect Security Findings with Medium Severity: 25
 Total Suspect Security Findings with Low Severity: 6
 Total Scan Coverage Findings with High Severity: 123
 Total Scan Coverage Findings with Medium Severity: 69
 Total Scan Coverage Findings with Low Severity: 56
 Total Lines: 17197
 Max V-Density: 929.8482293423273
 Max V/kloc: 22.155027039599933
 V-Density: 929.8482293423273
 V/kloc: 22.155027039599933
```

echo

Description

Echoes all input and output to the screen.

This command is usually used within a script.

Syntax

echo

Example

To output comments:

```
3/11/08 2:15 PM  AllApplications>>
3/11/08 2:16 PM  echo "this is a test"
3/11/08 2:16 PM  this is a test
```

getaseinfo (gase)

Description

Print AppScan Enterprise Server settings.

This command displays the AppScan Enterprise Server configuration that has been set by the `setaseinfo (sase)` command.

Syntax

getaseinfo

Note:

- You must be signed in to the AppScan Source command line interface (CLI) with `Manage AppScan Enterprise Settings` permissions to be able to set the `url`. For information about user accounts and permissions, see the *Administering AppScan Source* section of the *IBM Security AppScan Source Installation and Administration Guide*.
- The `userid` and `password` are stored on the machine that is running the AppScan Source client (for example, AppScan Source for Analysis) - while the `url` is stored in the Enterprise Server (which may be located on a remote machine). You cannot access `userid` and `password` information from the remote machine (for example, by issuing the `getaseinfo` command from it).

Example

```
AllApplications>> getaseinfo

Username:        my_domain\my_username
URL:       http://my_aseserver/ase
```

help (?)

Description

Lists help for all commands or a single command.

The AppScan Source command line interface (CLI) help displays the command syntax, aliases, and sample usage for each command. Help is available for individual commands, or you may request a list of all commands with a brief overview of each command.

Syntax

help [command]

`command`: CLI command for which to display online help. You must be at a position in the tree where help is available for the specific object.

Examples

- To list help for all valid commands:

  ```
  AllApplications>> ?
  ```

- To list Help for the `register (reg)` command:

  ```
  AllApplications>> Help register
  ```

import (im)

Description

Use the `import` command to add projects (such as `.ppf`) to an existing application.

When using this command to add a project, these project files are supported:

- AppScan Source project files (`.ppf`).
- Microsoft Visual Studio (version 6) (`.dsp`)
- Microsoft Visual Studio C/C++ (`.vcproj`)
- Microsoft Visual Studio C# (`.csproj`)
- Microsoft Visual Studio Visual Basic (`.vbproj`)
- Eclipse project file (`.epf`)

When importing projects, wildcard characters may be used in place of actual file names and/or file extensions, and you may specify an additional argument to indicate if all projects under the specified directory should be imported (recursive import).

Syntax

`import` `path [true|false]`

- The number of arguments varies depending on the type of the imported object.
- `path`: Complete path of the imported object.
- `true` or `false`: Optional. Include or omit subdirectories. The default is `false` (omit subdirectories).

Example

```
AllApplications\testit>> import c:\testapps\java\webgoat\*.ppf
AllApplications\testit>> ls
23942: webgoat (Project [local])
```

info (i)

Description

Lists information about the current object's properties and values.

Syntax

`Info`

Example

To list the ID and name of the current application:

```
AllApplications\Myapp>> Info
```

The output is similar to:

```
object: Myapp(Application)
id: 1

...
```

list (ls, dir)

Description

List all objects under the current object in the object tree. The tree displays as a graphical representation of the object ID, name, and type.

Syntax

list [*all*]

- all: Optional. Displays all objects in the object tree.
- Passing no arguments lists only the immediate children of the current object.

Examples

- To list the objects in the current object tree:

```
AllApplications>> List
1: WebGoat_4_0 (Application [local] [registered])
2: test_application (Application [local] [registered])
```

- To list all objects in the Object Tree:

```
AllApplications>> LS all

1: WebGoat_4_0 (Application [local] [registered])
|----198: compile (Project [local] [registered])
|----|----470: WebContent\lp\JavaScriptValidation.html (Source)
|----|----475: WebContent\lp\RemoteAdminFlaw.html (Source)
|----|----483: WebContent\lp\WelcomeScreeen.html (Source)
|----|----474: WebContent\lp\ReflectedXSS.html (Source)
|----|----477: WebContent\lp\SqlInjection.html (Source)
. . .
```

listassess (la)

Description

List the object ID and assessment date/time for the current object in the object tree. Use listassess to obtain an ID for use with the details command.

Syntax

listassess [*all*]

all: Optional. Lists all assessments in memory.

Examples

- To list the current assessment object information:

```
AllApplications\WebGoat_4_0>> listassess

AllApplications\WebGoat_4_0>> 4762: WebGoat_4_0 (Application, Tue Mar 11 14:20:23
EDT 2008)
AllApplications\WebGoat_4_0>> 9033: WebGoat_4_0 (Application, Tue Mar 11 14:43:31
EDT 2008)
```

- To list all assessment object information:

```
AllApplications\Applications>> ListAssess all
542: putty (Application, Fri Aug 04 08:38:18 EDT 2008)
1804: JavaAny (Application, Tue Apr 01 13:17:33 EDT 2008)
2971: snare (Application, Tue Apr 01 08:42:53 EDT 2008)
4773: SimpleIOT (Project, Tue Apr 01 07:31:11 EDT 2008)
4874: JSPWiki (Application, Tue Apr 01 13:22:08 EDT 2008)
```

listgroups (lgrp)

Description

Lists all of the groups, their permissions, and a description of each.

Syntax

listgroups

Example

```
AllApplications>> ListGroups

----------------------------------------
| Group: APPS
| Description: Application and Project Management
| Permissions:
|  REGISTER (Register)
|  ATTRAPPLY (Apply Attributes)
|  ATTRMODIFY (Manage Attributes)
|  SCAN (Scan)
|  VIEWREGISTER (View Registered)
| Group: ASSESSMENTS
| Description: Assessment Management
| Permissions:
|  ASMNTDELETE (Delete Published Assessments)
|  ASMNTPUBLISH (Publish Assessments)
|  ASMNTSAVE (Save Assessments)
|  ASMNTVIEWPUBLISH (View Published Assessments)
| Group: KB
| Description: Knowledgebase Management
| Permissions:
|  CUSTOM (Manage Custom Rules)
|  PATTERN (Manage Patterns)
| Group: ADMIN
| Description: Administration
| Permissions:
|  ASE (Manage AppScan Enterprise Settings)
|  LDAP (Manage LDAP Settings)
|  USER (Manage Users)
----------------------------------------
```

listusers (lu)

Description

Lists all AppScan Source users.

If available, the output includes the first name, last name, and email address.

Syntax

listusers

Example

To list all AppScan Source users:

```
AllApplications>> Listusers
All Users
  admin (admin admin)
  jsmith(John Smith)
  joandarcy(Joan Darcy)
AllApplications>>
```

log

Description

Switch message logging on or off.

Log a session to a specified file or view current logging status. The entire session, including output, is logged to the specified file.

Tip: To log only command input, use the `record (rc)` command.

Syntax

log {*on* file|*off*}

- The `on` or `off` argument is required to turn logging on or off. `Log` without arguments identifies the current state.
- `on`: Required to turn logging on. `On` requires a `file` name.
- `file`: Name of the log file. You can specify the type of file to generate, such as `.txt` or `.csv`.
- `off`: Turn off the current logging session.

Examples

- To log to a file named `MyLogFile.txt`:

  ```
  AllApplications>> Log on c:\MyLogFile.txt
  ```

- To turn logging off:

  ```
  AllApplications>> Log off
  ```

login (in)

Description

Log in to the AppScan Enterprise Server (replaces `login_local (local)`).

If the login is successful, the prompt changes to `AllApplications>>` to indicate that you are logged in.

Syntax

login server user_id password [-persist] [-acceptssl]

- `server`: Required. Specify the URL for your Enterprise Server instance.
- `user_id`: Specify your Enterprise Server user ID. Required unless the persist option has been previously used to save a `cli.token` file. The `cli.token` file resides in the `.ounce` folder of the users `home` directory. If you delete `cli.token`, a user name and password are required to log in.
- `password`: Specify the password for your Enterprise Server user ID. Required unless the persist option has been previously used to save a `cli.token` file.

- `-persist`: Optional. Persist login credentials through an encrypted key. If the persist argument is used, `login` saves the user name and password to an encrypted key file called `cli.token`.
- `-acceptssl`: Optional. Automatically accept SSL certificates. For more information, see "AppScan Enterprise Server SSL certificates"

  Important: If this argument is not included, logging in or publishing to the AppScan Enterprise Console will fail with error if an invalid certificate is encountered (if you have not already permanently accepted the certificate while logging in via another AppScan Source client product).

Example

User John Smith logs into the AppScan Enterprise Server that has been set up by his administrator:

```
>> Login https://myserver:9443/asc/ johnsmith mypassword
Login successful.
AllApplications>>
```

AppScan Enterprise Server SSL certificates

When the AppScan Enterprise Server is installed, it should be configured to use a valid SSL certificate. If this is not done, you will receive an untrusted connection message when logging in to the server from AppScan Source for Analysis or the AppScan Source command line interface (CLI) - or AppScan Source for Development on Windows and Linux.

SSL certificate storage location

Certificates that have been permanently accepted are stored in `<data_dir>\config\cacertspersonal` and `<data_dir>\config\cacertspersonal.pem` (where `<data_dir>` is the location of your AppScan Source program data, as described in Chapter 9, "Installation and user data file locations," on page 149). Remove these two files if you no longer want the certificates permanently stored.

AppScan Source for Automation and SSL certificate validation

By default, certificates are automatically accepted when using AppScan Source for Automation. This behavior is determined by the `ounceautod_accept_ssl` setting in the Automation Server configuration file (`<data_dir>\config\ounceautod.ozsettings` (where `<data_dir>` is the location of your AppScan Source program data, as described in Chapter 9, "Installation and user data file locations," on page 149)). If this setting is edited so that `value="true"` is set to `value="false"`, SSL validation will be attempted and logging in or publishing to AppScan Enterprise Console will fail with error if an invalid certificate is encountered.

AppScan Source command line interface (CLI) and SSL certificate validation

By default, when using the CLI `login` command, SSL validation will be attempted and logging in or publishing to AppScan Enterprise Console will fail with error if an invalid certificate is encountered (if you have not already permanently accepted the certificate while logging in via another AppScan Source client product). This behavior can be modified by using the option `-acceptssl` parameter when issuing the `login` command. When this parameter is used, SSL certificates are automatically accepted.

login_local (local)

Description

Deprecated in AppScan Source Version 6. Use `login (in)`.

logout (out)

Description

Log out of AppScan Source and terminate the command line interface AppScan Source command line interface (CLI) session.

If you are not logged in, `logout` is an invalid command.

Syntax

```
logout
```

Example

To log out:

```
AllApplications>> Logout
>> Logged Out.
>> Exiting Security AppScan Source Command Line Interface...
```

moduser (mu)

Description

Modify user information such as product access, permissions, user ID, and email for an AppScan Source or AppScan Enterprise Server user.

Syntax

```
moduser --userid|-u <user id>
[--fullname|-f <user first and last name>]
[--email|-e <email address>]
[--local {true|false}]
[--group [group[:permission[;permission...]
 [--group...]]
[--removegroup [group[:permission
 [;permission...] [--removegroup...]]
```

User name

`--username|-u`: Required. A valid AppScan Source or AppScan Enterprise Server user name.

Modifying AppScan Enterprise Server users and AppScan Source users

Optional.

When you modify a user without specifying the `local` argument, the `local` argument value that was used to create the user is assumed. To change the location of users, modify the `local` setting:

- If the user ID is currently in the AppScan Enterprise Server user repository (meaning it exists on both the server and locally in the AppScan Source database), set `moduser -u <user id> -local true` to change the ID so that the user gets authenticated using the password provided instead of getting authenticated by the AppScan Enterprise Server.
- If the user ID is currently in the AppScan Source user repository and does not exist on the server, set `moduser -u <user id> -local false` to change the ID so that the user will be authenticated via the AppScan Enterprise Server.

Identifying Information

Values attached to these arguments will be modified in the user record:

- `--fullname|-f`: Optional. Full name of the user. If the entry includes spaces, enclose it with " symbols (for example, `-f "Joe Smith"`).
- `--email|-e`: Optional. Email address of the user.

Groups and Permissions

Optional.

Groups and permissions identify the allowable AppScan Source tasks for that user. Tasks not specifically identified as part of a permission are available to all users:

`--group`: The groups and group permissions to add for this user. Specifying a group without any permissions grants the user all permissions in that group.

or

`--removegroup`: The groups and group permissions to remove from this user. Specifying a group without any permissions removes all permissions in that group.

The groups and permissions are:

- `ASSESSMENTS`: Assessment level permissions.
  - `ASMNTDELETE`: Delete published assessments.
  - `ASMNTPUBLISH`: Publish assessments.
  - `ASMNTSAVE`: Save assessments.
  - `ASMNTVIEWPUBLISH`: View published assessments.
- `ADMIN`: Administrative permissions.
  - `ASE`: Manage AppScan Enterprise settings
  - `USER`: Manage user settings including adding and deleting users and changing user permissions.
- `APPS`: Application and Project level permissions
  - `ATTRAPPLY`: Apply attributes to applications.
  - `ATTRMODIFY`: Create, delete, and modify attributes.
  - `VIEWREGISTER`: View registered applications and projects.
  - `REGISTER`: Register/unregister applications and projects. Implies `VIEWREGISTER` permission.
  - `SCAN`: Scan applications and projects.
- `KB`: Knowledgebase management permissions.
  - `CUSTOM`: Manage custom rules.
  - `PATTERN`: Create, edit, or delete patterns.

- FILTER: Filter management
  - SHAREDFILTERS: Manage shared filters.
- SCANCONFIG: Scan configuration management
  - SHAREDCONFIGS: Manage shared scan configurations.

Example

After Joan Darcy's user credentials were created (with the newuser (nu) command), the System Administrator determined that she only needs save, publish, and view permissions; but not delete permission. Additionally, Joan needs Knowledgebase Patterns permissions:

```
moduser --userid joandarcy --removegroup
ASSESSMENTS:ASMNTDELETE --group KB:PATTERN
```

newuser (nu)

Description

Create a new AppScan Source user (a valid user name, password, full name, and e-mail address are required). AppScan Source users can exist in the AppScan Enterprise Server user repository and in the AppScan Source database - or, if you have cause to have users that cannot access the server, they can be created locally as AppScan Source users. You can also create a new AppScan Source user that already exists on the AppScan Enterprise Server.

Syntax

```
newuser --userid|-u <user id>
--password|-p <password>
--fullname|-f <user first and last name>
--email|-e <email address>
[--local {true|false}]
[--group [group[:permission[;permission...]
 [--group...]]
```

Identifying Information

- --userid|-u: Required. User ID. No spaces are allowed.
- --password|-p: User password. This parameter is required when -local is set to true - and ignored when -local is set to false.
- --fullname|-f: Full name of the user. If the entry includes spaces, enclose it with " symbols (for example, -f "Joe Smith"). This parameter is required when -local is set to false - and optional when -local is set to true.
- --email|-e: Email address of the user. This parameter is required when -local is set to false - and optional when -local is set to true.

Creating AppScan Enterprise Server users and AppScan Source users

Optional. The newuser command always creates a user in the AppScan Source database - and, by default, the command will also create the user on the AppScan Enterprise Server. To set the location of the user, use the --local argument. By default, the argument is set to false, which means that users will be created in both the AppScan Enterprise Server and AppScan Source user repositories. If the argument is set to true, the user will only be created locally as an AppScan Source user.

Local users cannot publish scan results to the AppScan Enterprise Console - however, they can publish to the AppScan Source database. When logging in to AppScan Source, local users must specify a valid AppScan Enterprise Server, along with their AppScan Source user credentials (a server must be provided for access to the AppScan Source database).

Groups and Permissions

Optional. Groups and permissions identify the allowable AppScan Source tasks for that user. Tasks not specifically identified as part of a permission are available to all users:

`--group`: The groups and group permissions for this user. Specifying a group without any permissions grants the user all permissions within that group. The groups and their permissions are:

- `ASSESSMENTS`: Assessment level permissions.
  - `ASMNTDELETE`: Delete published assessments.
  - `ASMNTPUBLISH`: Publish assessments.
  - `ASMNTSAVE`: Save assessments.
  - `ASMNTVIEWPUBLISH`: View published assessments.
- `ADMIN`: Administrative permissions.
  - `ASE`: Manage AppScan Enterprise settings
  - `USER`: Manage user settings including adding and deleting users and changing user permissions.
- `APPS`: Application and Project level permissions
  - `ATTRAPPLY`: Apply attributes to applications.
  - `ATTRMODIFY`: Create, delete, and modify attributes.
  - `VIEWREGISTER`: View registered applications and projects.
  - `REGISTER`: Register/unregister applications and projects. Implies `VIEWREGISTER` permission.
  - `SCAN`: Scan applications and projects.
- `KB`: Knowledgebase management permissions.
  - `CUSTOM`: Manage custom rules.
  - `PATTERN`: Create, edit, or delete patterns.
- `FILTER`: Filter management
  - `SHAREDFILTERS`: Manage shared filters.
- `SCANCONFIG`: Scan configuration management
  - `SHAREDCONFIGS`: Manage shared scan configurations.

LDAP authentication

You cannot add LDAP users to the AppScan Source user repository if they are not already in the AppScan Enterprise Server user repository. To add an AppScan Source user that will be authenticated via LDAP, you must have configured the AppScan Enterprise Server user repository to use an LDAP repository. For information about this, see the AppScan Enterprise Server *Planning & Installation Guide*.

If you are using LDAP authentication and want to add an AppScan Source user that is not part of an LDAP user group, issue the `newuser` command with a `-local true` option to create the user locally in the AppScan Source user repository.

Example

Create a user named Joan Darcy on the AppScan Enterprise Server. Joan's email address is `jdarcy@example.com`. Her user name is `joandarcy` and her password is `123456`. Joan can use AppScan Source with all permissions in the APPS and ASSESSMENTS groups, as well as custom rules permission within the KB group:

```
AllApplications>> newuser --userid joandarcy --password 123456
--fullname "Joan Darcy" --email jdarcy@example.com
--group APPS --group ASSESSMENTS --group KB:CUSTOM
AllApplications>> Created user 'joandarcy'. User ID: 888
```

openapplication (oa)

Description

Open an application. This command can also be used to create a new AppScan Source application file.

When using this command to open an application, these application files are supported:

- AppScan Source application files (`.paf`).
- Microsoft Visual C++ Workspace files (`.dsw`)
- Microsoft Visual Studio.NET solution files (`.sln`)
- Eclipse or Rational® Application Developer for WebSphere® Software (RAD) workspaces (`.ewf`)

Syntax

openapplication path_file

`path_file`

- If you are using the command to open an application, specify the path and file name of the existing application.
- If you are using the command to create an application, specify a valid path and a new file name (ensure that the new file name does not already exist).

Example

To open a Microsoft Visual Studio .NET solution file:

```
AllApplications>> OpenApplication c:\mysln.sln
```

openassessmentfile (oaf)

Description

Opens an AppScan Source assessment file (`file_name.ozasmt`).

An opened assessment becomes the current assessment.

Syntax

openassessmentfile path_file

`path_file`: Path and file name of the existing assessment file.

Example

To open a previously saved assessment:

```
AllApplications>> OpenAssessmentFile c:\priority.ozasmt
```

password (passwd)

Description

The password command allows you to change your password or, if you have administrator permissions, to change another user's password.

To change a password, type the new password twice. Both entries of the new password must match for the change to take effect. When changing your own password, you must type your current password first.

Syntax

```
password [user name]
```

username: Name of the user whose password is being changed. If you do not specify a user name, you are changing your own password.

Examples

- To change your own password:

```
AllApplications>> Password
Type current password: ******
Type new password: *******
Verify new password: *******
Password changed.
AllApplications>>
```

- To change another user's password (if you have administrator permissions):

```
AllApplications>> passwd johnsmith
Changing password for user johnsmith
Type new password: *******
Verify new password: *******
Password changed.
AllApplications>>
```

printuser (pu)

Description

The printuser (pu) command displays information about a single user on the screen.

Syntax

```
printuser username
```

username: AppScan Source user name.

Example

Display user Joan Darcy's original user data:

```
printuser joandarcy
User: joandarcy
First Name: Joan
```

```
Last Name: Darcy
e-mail Address: jdarcy@example.com
Groups:
 - APPS     (Application and Project Management)
  REGISTER     (Register)
  ATTRAPPLY     (Apply Attributes)
  ATTRMODIFY     (Manage Attributes)
  SCAN       (Scan)
  VIEWREGISTER    (View Registered)
 -(ASSESSMENTS) (Assessment Management)
  ASMTDELETE     (Delete Published Assessments)
  ASMNTPUBLISH   (Publish Assessments)
  ASMNTSAVE    (Save Assessments)
  ASMNTVIEWPUBLISH (View Published Assessments)
 -[KB]     (Knowledgebase Management)
  CUSTOM      (Manage Custom Rules)
 -[ADMIN]    (Administration)
```

Note: Square brackets [ ] indicate that this user does not have all of the permissions in this group.

publishassess (pa)

Description

Publish the current assessment or a selected assessment to AppScan Enterprise Server. When this command is used, the assessment is made available to an AppScan Source client such as AppScan Source for Analysis - but it is not made available to the AppScan Enterprise Console (use the "publishassessase (pase)" on page 46 command to publish to the AppScan Enterprise Console).

A valid AppScan Source for Automation license is required for use of this command.

When an assessment is published with this command, the assessment is automatically registered with AppScan Enterprise Server. You do not need to manually register the assessment when using this command.

Syntax

pa [*id*]

id: Optional literal. Identifies the assessment ID. If no ID is included, the command applies to the most recent scan. You can use the listassess (la) command to find the assessment ID.

Example

In the following example, we set the current object to the application pebble, scan the pebble project, and then publish the assessment (which is registered automatically at the same time).

```
AllApplications>> cd pebble
AllApplications\pebble>> scan pebble
New Scan started

Scanning Project pebble (1 of 1)
Preparing project for scan...
  .
  .
Preparing for Vulnerability Analysis...
Performing Vulnerability Analysis...
```

```
Generating Findings...
Preparing project for scan...
  .
  .
Scanned Project pebble:
  Total files: 15
  Total findings: 167
  Total lines: 385
 vkloc: 0.44448395412925595
 v-Density: 22.446439683527426
Scanned Application pebble:
  Total files: 15
  Total findings: 167
  Total lines: 385
 vkloc: 0.44448395412925595
 v-Density: 22.446439683527426
Scan completed:
  Total files: 15
  Total findings: 167
  Total lines: 385
 vkloc: 0.44448395412925595
 v-Density: 22.446439683527426
Elapsed Time - 18 Seconds

AllApplications\pebble>> publishassess

Assessment Successfully Published.
```

publishassessase (pase)

Description

Publish the current assessment or a selected assessment to the AppScan Enterprise Console. When this command is used, the assessment is not available to AppScan Source clients such as AppScan Source for Analysis (use the "publishassess (pa)" on page 45 command to publish to AppScan Source clients).

Syntax

publishassessase [*id*] [*path*] [-folderid <server_folder>]

- *id*: Optional literal. Identifies the assessment ID. If no ID is included, the command applies to the most recent scan. You can use the `listassess (la)` command to find the assessment ID.
- *path*: Optional literal. Path to the assessment file.
- `-folderid <server_folder>`: Optional. Enterprise Console folder to publish to. If this argument is not used, the assessment will be published to your default Enterprise Console folder.

When the optional argument is an integer, the command assumes it is the assessment ID. When it is not an integer, the command assumes it is a path to a saved assessment file.

quit

Description

Ends and closes the AppScan Source command line interface session. Issues a logout if you are logged in.

Syntax

quit

Example

```
AllApplications>> Quit

>> Logged Out.
>> Exiting Security AppScan Source Command Line Interface...
```

record (rc)

Description

Turns command recording on or off.

`Record` captures only commands and arguments issued, but does not capture output.

Tip: To capture output, use the `log` command.

Syntax

record {*on* `file`|*off*}

- `on` or `off`: Required to turn recording on or off.
- `file`: Name of the recorded file. `Record On` requires a `file` argument that includes the valid file path. If the file does not already exist, the AppScan Source command line interface (CLI) creates it. If the file already exists, log data appends to the end of the file.
- `Record` with no arguments identifies whether recording is on or off.

Example

- To log to a file named `MyCommands.txt`:

  ```
  AllApplications>> Record on C:\MyCommands.txt
  ```

- To turn command recording off:

  ```
  AllApplications>> RC off
  ```

refresh (rf)

Description

`Refresh` restores the current project or application from disk.

Use `refresh` when you want to reload a project or application that might have changed.

Syntax

refresh

Example

To refresh `MyProject`:

```
AllApplications\MyApplication\MyProject>> Refresh
```

register (reg)

Description

Register projects and applications with the AppScan Source database.

`Register` allows you to register the application or project on the current node with AppScan Source. Registration makes AppScan Source aware of the applications and projects. Registered applications and projects can be published (using `publishassess (pa)`).

Once registered, you can also unregister applications or projects using `unregister (unreg)`.

Syntax

register

Example

```
AllApplications>> cd pebble
AllApplications\pebble>> register
AllApplications\pebble>> 'pebble' registered successfully.
```

removeassess (da)

Description

Remove the selected or current assessment from memory.

Syntax

removeassess [*id*]

id: Optional literal. Identifies the assessment ID. If no ID is included, the command applies to the most recent assessment. Use `listassess (la)` to find an assessment ID.

Example

To remove an assessment, first get the assessment ID:

```
AllApplications\Applications>> ListAssess
542: putty (Application, Tue Apr 01 08:38:18 EDT 2008)
180: JavaAny (Application, Tue Apr 01 13:17:33 EDT 2008)
```

Then remove assessment with ID 180:

```
AllApplications\Applications>> RemoveAssess 180
```

report (rpt)

Description

`Report` generates an AppScan Source report of the specified type, including findings reports and AppScan Source reports. A valid AppScan Source for Automation license is required for use of this command.

Available report output formats are HTML, PDF, and zip.

Syntax

```
report <"report type"> <output format> <output location>
[<assessment id>] [-includeSrcBefore:<n>] [-includeSrcAfter:<n>]
```

- report type: The name of the report to generate. Specify one of the following:
  - A Findings report:
    - Findings
    - Findings by Type
    - Findings by Classification
    - Findings by File
    - Findings by API
    - Findings by Bundle
    - Findings by CWE
    - DTS Activity
  - An AppScan Source report:
    - DISA Application Security and Development STIG V3 R5
    - OWASP Top 10 2010
    - OWASP Top 10 2007
    - Software Security Profile
    - PCI Data Security Standard V2.0
    - PCI Data Security Standard V1.1
  - A custom report, if available.
- output format : Specify one of the following formats for this report:
  - html: Generates the report as HTML and displays it online.
  - zip: Creates a ZIP file that contains all HTML report components
  - For reports in PDF format, you can specify the level of detail:
    - pdf-summary: Contains counts for each custom report group
    - pdf-detailed: Contains counts for each API for each vulnerability property
    - pdf-comprehensive: Contains tables consisting of every finding for every API
    - pdf-annotated: Contains all findings, any notes included with the findings, and designated code snippets
    - output location: The file path to write the report.
- output location: Specify the absolute path and file name to which you want to save the report.
- assessment id: Optional. The assessment ID, which you obtain from the listassess (la) command. If you omit assessment ID, the report is generated from the most recent scan.
- -includeSrcBefore:<n>: Optional. The number of lines of source code to include in a report before each finding.
- -includeSrcAfter:<n>: Optional. The number of lines of source code to include in a report after each finding.

Examples

- Request a *Findings by API* report written to HTML:

  ```
  AllApplications>> report "Findings by API" html  C:\reports\findings.html
  ```

- Request an *OWASP Top 10* AppScan Source report written to PDF with comprehensive detail using existing assessment 542:

```
AllApplications>> report "OWASP Top 10" pdf-comprehensive
/reports/webgoat_OWASP_10_comp.pdf 542
```

scan (sc)

Description

Scans an application (or all applications), project, or file. A valid AppScan Source for Automation license is required for use of this command.

Syntax

```
scan [path][config <proj_config>][-name <assessment_name>]
[-scanconfig <scan_configuration_name>]
[-quality <path_file>]
[-qualityOnly <path_file>]
```

- `path`: Optional. Full path in which the exported assessment should be saved.
- `config <proj_config>`: Optional. This argument is only valid for project-level assessments. If your project has a configuration file, specify it using this argument.
- `-name <assessment_name>`: Optional. Provide a name for the assessment.
- `-scanconfig <scan_configuration_name>`: Optional. Specify the name of a scan configuration to use for the scan. If a scan configuration is not specified, the default scan configuration will be used for the scan.
- `-quality <path_file>`: Optional. To scan for quality issues (in addition to security vulnerabilities), include the `quality` argument. When specified, you must include the path and file name of one of these files:
  - Quality rule (`.quality`) file:

    Quality rule files are automatically created when you configure quality scanning in the AppScan Source for Development Eclipse Plug-in. When you create a quality rule configuration, the quality rule file for it is saved to `<user_home>\.ouncequality\rulesets\<quality_rule_config_name>.quality`, where:

    - `<user_home>` is your operating system home directory (for example, on Windows, the directory might be `C:\Documents and Settings\Administrator\`).
    - `<quality_rule_config_name>` is the name that you gave the quality rule configuration when creating it in the AppScan Source for Development Quality Rule Configuration dialog box.

    AppScan Source includes three default quality rule files:

    - `codereview_basic_cpp.quality`: Quality rule configuration that includes recommended C/C++ Code Review rules.
    - `codereview_basic_java.quality`: Quality rule configuration that includes recommended Java Code Review rules (designed to generate only high-severity findings).
    - `codereview_extended_java.quality`: This rule file includes the same rules that are found in`codereview_basic_java.quality`, along with:
      - Additional Java code review rules, including some that produce findings of lower severity.
      - Java data flow analysis rules.

    The default rule files are located in `<install_dir>\quality\rulesets` (where `<install_dir>` is the location of your AppScan Source installation). The provided default rule files cannot be modified.

To learn more about quality rule files, see the AppScan Source for Development Eclipse Plug-in infocenter.

– Quality rule argument script file: Text file that contains "Quality scan script arguments" on page 23. This option allows you to fine tune your scan by including/excluding files - and specifying C/C++ headers, macros, and other arguments.

- `qualityonly <path_file>`: Optional. To scan for quality issues only, include the `qualityonly` argument. When specified, you must include the path and file name of one of these files:
  – Quality rule (`.quality`) file:

    Quality rule files are automatically created when you configure quality scanning in the AppScan Source for Development Eclipse Plug-in. When you create a quality rule configuration, the quality rule file for it is saved to `<user_home>\.ouncequality\rulesets\<quality_rule_config_name>.quality`, where:

    - `<user_home>` is your operating system home directory (for example, on Windows, the directory might be `C:\Documents and Settings\Administrator\`).
    - `<quality_rule_config_name>` is the name that you gave the quality rule configuration when creating it in the AppScan Source for Development Quality Rule Configuration dialog box.

    AppScan Source includes three default quality rule files:

    - `codereview_basic_cpp.quality`: Quality rule configuration that includes recommended C/C++ Code Review rules.
    - `codereview_basic_java.quality`: Quality rule configuration that includes recommended Java Code Review rules (designed to generate only high-severity findings).
    - `codereview_extended_java.quality`: This rule file includes the same rules that are found in`codereview_basic_java.quality`, along with:
      - Additional Java code review rules, including some that produce findings of lower severity.
      - Java data flow analysis rules.

    The default rule files are located in `<install_dir>\quality\rulesets` (where `<install_dir>` is the location of your AppScan Source installation). The provided default rule files cannot be modified.

    To learn more about quality rule files, see the AppScan Source for Development Eclipse Plug-in infocenter.

  – Quality rule argument script file: Text file that contains "Quality scan script arguments" on page 23. This option allows you to fine tune your scan by including/excluding files - and specifying C/C++ headers, macros, and other arguments.

Examples

- To scan the default configurations of projects in all applications:

  ```
  AllApplications>> Scan
  ```

  The results appear as:

  ```
  New Scan started
    .
    .
  Preparing for Vulnerability Analysis...
  Performing Vulnerability Analysis...
  Generating Findings...
  ```

```
Preparing project for scan...
  .
  .
Scanned Project:
  Total files: 15
  Total findings: 167
  Total lines: 385
 vkloc: 0.44448395412925595
 v-Density: 22.446439683527426
Scanned Application:
  Total files: 15
  Total findings: 167
  Total lines: 385
 vkloc: 0.44448395412925595
 v-Density: 22.446439683527426
Scan completed:
  Total files: 15
  Total findings: 167
  Total lines: 385
 vkloc: 0.44448395412925595
 v-Density: 22.446439683527426
Elapsed Time - 18 Seconds

New Scan started. Please wait...
Assessment complete
------------------
Total Call Sites: 75
Total Definitive Security Findings with High Severity: 25
Total Definitive Security Findings with Medium Severity: 37
Total Definitive Security Findings with Low Severity: 9
Total Suspect Security Findings with High Severity: 20
Total Suspect Security Findings with Medium Severity: 80
Total Suspect Security Findings with Low Severity: 60
Total Scan Coverage Findings with High Severity: 50
Total Scan Coverage Findings with Medium Severity: 33
Total Scan Coverage Findings with Low Severity: 17
Total Lines: 3000
...
```

- To scan the debug configuration of `Prj1`:

  ```
  AllApplications\Prj1>> SC config debug
  ```

- To conduct a scan that only looks for quality issues in `App1` (using the `C:\shared\Java Globalization.quality` quality rule file) and sends the scan results to `C:\myAssessment.ozasmt`:

  ```
  AllApplications\App1>> scan "C:\myAssessment.ozasmt"
  -qualityonly "C:\shared\Java Globalization.quality"
  ```

  The results appear as:

  ```
  New scan started

  Importing rules for quality scanning from: C:\shared\Java Globalization.quality
  Quality rules have been imported: 47
  Preparing resources for quality scanning.
  Resources have been prepared.
  Quality scanning...
  Quality scanning is complete, generating findings...
  Quality findings have been generated.

  Scanned application App1: Total files: 1 Total findings: 1
  Total lines: 65 vkloc: 0.000000 v-density: 0.000000
  Scan completed: Total files: 1 Total findings: 1
  Total lines: 65 vkloc: 0.000000 v-density: 0.000000
  Elapsed Time - 0 Hour(s) 0 Minute(s) 14 Second(s)

   ------------------
   Total Call Sites: 1
   Total Definitive Security Findings with High Severity: 0
  ```

```
Total Definitive Security Findings with Medium Severity: 0
Total Definitive Security Findings with Low Severity: 0
Total Suspect Security Findings with High Severity: 0
Total Suspect Security Findings with Medium Severity: 0
Total Suspect Security Findings with Low Severity: 0
Total Scan Coverage Findings with High Severity: 0
Total Scan Coverage Findings with Medium Severity: 0
Total Scan Coverage Findings with Low Severity: 0
Total Lines: 65
Max V-Density: 0
Max V/kloc: 0
V-Density: 0
V/kloc: 0
```

script (scr)

Description

Run a script of commands.

The script may contain any valid AppScan Source command line interface (CLI) commands with arguments. The `record (rc)` command can generate the script file.

Scripts can reference environmental variables, using the `${VAR}` notation (where `VAR` is the name of an environmental variable).

When the `script` command is passed to `AppScanSrcCli` from the command line, it automatically exits `AppScanSrcCli` at the end of the script.

Syntax

script script_file

`script_file`: Full path and file name of the script file that you want to run.

Comments in Scripts

A line preceded by a number sign (#) is a comment. The CLI skips comments when run interactively, or from within a script file.

The following example script contains comments:

```
login localhost User 123456
#navigate to my workspace
cd MyApplication
#scan my application
scan
logout
```

Examples

A user who typically logs in and uses the CLI interactively may want to record a script to execute repeatedly while running from the console. You can script an entire session using the `record (rc)` command, and run it repeatedly or at scheduled intervals. Use a script to automate an entire session by allowing a single batch file or script to log in, create projects, scan, copy files, and so forth.

- To run a script, named `myscript.txt`:

  ```
  AllApplications>> Script c:\myscript.txt
  ```

- The following example script logs in, turns on logging, creates an application, imports all `ppf` projects under a given directory, runs an assessment, and exits:

```
log on C:\mylogfile.log
new MyApplication C:\myAppFolder
cd MyApplication
im c:\Projects\*.ppf
scan
logout
```

Note that `logout` is not necessary if the script always runs from the `AppScanSrcCli` command line.

setaseinfo (sase)

If your AppScan Enterprise Server has been installed with the AppScan Enterprise Console option, you can publish assessments to it. The Enterprise Console offers a variety of tools for working with your assessments - such as reporting features, issue management, trend analysis, and dashboards.

Description

Specify Enterprise Console settings.

If you intend to publish assessments from AppScan Source to the Enterprise Console, use the `setaseinfo` command to specify Enterprise Console connection settings.

Syntax

```
setaseinfo --userid|-u <user name>
--password|-p <password>
--url|-l <url>
[--domain|-d <domain>]
```

- `userid`: The user name that you use to connect to the Enterprise Console. At a minimum, you must be a QuickScan user with your own user folder on the Enterprise Server.
- `password`: The password used to log in to AppScan Enterprise (the password for the user name that was entered).
- `url`: The URL used to access the Enterprise Console web application.

  The format of this URL is:

  `http(s)://<hostname>:<port>/ase/`

  Where `<hostname>` is the name of the machine on which the Enterprise Console has been installed and `<port>` is the port on which the console is running. An example of this URL is `http://myhost.mydomain.ibm.com/ase`.

  Note: This argument is optional if the Enterprise Console URL has already been set.
- `domain`: The domain where AppScan Enterprise is installed.

Note:

- You must be signed in to the AppScan Source command line interface (CLI) with `Manage AppScan Enterprise Settings` permissions to be able to set the `url`. For information about user accounts and permissions, see the *Administering AppScan Source* section of the *IBM Security AppScan Source Installation and Administration Guide*.
- The `userid` and `password` are stored on the machine that is running the AppScan Source client (for example, AppScan Source for Analysis) - while the `url` is stored in the Enterprise Server (which may be located on a remote machine).

You cannot access `userid` and `password` information from the remote machine (for example, by issuing the `getaseinfo` command from it).

Example

```
AllApplications>> setaseinfo --userid my_username --password my_password
 --url http://my_aseserver/ase --domain my_domain
AppScan Enterprise Integration Information Configured.
AllApplications>>
```

setcurrentobject (set, cd)

Description

Use `setcurrentobject` to navigate the object tree.

This command sets the currently selected object in the object tree. The object must be a valid child of the current object. To view the children of the current object, use the `list (ls, dir)` command. Select the object by name or ID.

Syntax

setcurrentobject {name|*id* object id}

- `name`: Name of the child to set. Required when navigating the tree by object name.
- *id*: Literal. Select by object ID.
- `object id`: Numerical ID of the child to set. Required when selecting by ID.

Examples

- To navigate from the root to the Application whose ID is equal to 1:

  ```
  AllApplications>> SetCurrentObject applications
  AllApplications>> CD id 1
  AllApplications\Applications>>
  ```

- To back up one level:

  ```
  AllApplications\Applications>> CD ..
  ```

- To return to the top (root) of the Object Tree:

  ```
  AllApplications\Applications>> Set /
  ```

  or

  ```
  AllApplications\Applications>> Set \
  ```

setvar (sv)

Description

Creates a new variable or modifies an existing variable.

Syntax

setvar <name> <path>

- `name`: The name of the variable. If the variable already exists, then the AppScan Source command line interface (CLI) updates the path. If the variable does not exist, the CLI creates the variable with value of the named path.
- `path`: An existing path for the variable. If the path does not exist, the variable is not created.

Examples

- To create the AppScan Source variable `Myvar` with the path `C:\Myapps`:

```
AllApplications>> SetVar Myvar C:\Myapp
AllApplications>> Created new variable '%Myvar%'
```

- To create an AppScan Source variable named `SRC_ROOT` based on the value of the `SRC_ROOT` system environment variable:

```
# login
login localhost admin
# create the variable SRC_ROOT with the value
# of the environment variable SRC_ROOT
setvar SRC_ROOT ${SRC_ROOT}
```

unregister (unreg)

Description

This command is used to unregister a previously registered application or project from the current node.

Once unregistered, the application or project can not be published. For more information, see the `register (reg)` command.

Syntax

unregister

Example

```
AllApplications\Myapp>> Unregister
AllApplications\Myapp>> 'Myapp' unregistered successfully.
```

Running automated assessments

The AppScan Source command line interface (CLI) enables you to automatically import an AppScan Source project file (`.ppf`) and scan your source code. From the command line, you can run a script, such as the following sample, `Run_Assessments.txt`.

```
AppScanSrcCli scr c:\<install_dir>\bin\Run_Assessments.txt
```

Sample Run_Assessments.txt

```
# Log in.
Login <hostname> <username> <password>
# Turn on logging.
log on c:\myLogFile.log
# Create a new application named "testit."
new testit c:\AppTest
# Navigate to the newly created application.
cd testit
# Import the Project files (.ppfs) under c:\projects\joans.
im c:\projects\joans\*.ppf
# Refresh the Project.
refresh

# Run an assessment.
scan
# Register the assessment
register
```

```
# Publish the assessment
publishassess
# Log out and end the CLI session.
quit
```

Output

```
Logging to 'c:\mylogfile.log'...

AllApplications>> new testit c:\AppTest
AllApplications>> cd testit
AllApplications\testit>> import c:\TestApps\testproj\*.ppf
AllApplications\testit>> refresh
AllApplications\testit>> ls

214: testproj (Project [local])

AllApplications\testit>> la

testit has no current assessments.

scan
New Scan started at 15:41:55
Scanning Project testproj (1 of 1)
Preparing project for scan...
.
.
.

Searching File C:\TestApps\\testproj\src\se\bluefish\blueblog\metarepository\Meta
Category.java (21 of 33)
Searching File C:\TestApps\testproj\src\se\bluefish\blueblog\metarepository\Meta
Repository.java (22 of 33)

-------------------
Total Call Sites: 348
Total Definitive Security Findings with High Severity: 5
Total Definitive Security Findings with Medium Severity: 1
Total Definitive Security Findings with Low Severity: 4
Total Suspect Security Findings with High Severity: 0
Total Suspect Security Findings with Medium Severity: 8
Total Suspect Security Findings with Low Severity: 0
Total Scan Coverage Findings with High Severity: 16
Total Scan Coverage Findings with Medium Severity: 27
Total Scan Coverage Findings with Low Severity: 16
Total Lines: 7386
Max V-Density: 732.2772813430815
Max V/kloc: 10.42512862171676
V-Density: 732.2772813430815
V/kloc: 10.42512862171676

AllApplications\testit>> register
AllApplications\testit>> 'testit' registered successfully.
AllApplications\testit>> pa

Assessment Successfully Published.

AllApplications\testit>> la

AllApplications\testit>> 27001: testit (Application, Fri Mar 14 15:41:55 EDT 2008)

AllApplications\testit>>
```

Chapter 3. The Ounce/Ant build tool

This section describes how to use Ounce/Ant, an AppScan Source build utility that integrates AppScan Source and Apache Ant. Integrating Ounce/Ant with your Ant environment helps you automate builds and code assessments.

Ounce/Ant is an AppScan Source build tool that integrates into your Apache Ant build environment. Rather than create and configure AppScan Source project and application files manually in AppScan Source for Analysis, Ounce/Ant automatically generates AppScan Source projects and applications that contain all necessary information, such as source files and class path. AppScan Source can then scan the project and application files.

The following versions of Apache Ant and the Java JDK are required for use with Ounce/Ant:

- Apache Ant Version 1.7 or later
- Java JDK Version 1.3.1 or later
- Java JDK Version 1.4 required for use with the `var` task

Ounce/Ant and Apache/Ant integration

The steps outlined in this task topic are necessary to integrate Ounce/Ant into the Apache/Ant build environment.

Procedure

1. Copy the `ounceant.jar` file and, optionally, `ant-contrib-1.0b3.jar` to the `ant lib` directory.

   The AppScan Source installation places `ounceant.jar` and `ant-contrib-1.0b3.jar` in `<install_dir>\lib` (where `<install_dir>` is the location of your AppScan Source installation).
2. Optional: Override the `build.compiler` property.

   See “Creating projects” on page 61 for more information about overriding `build.compiler`.)

   Override this property using one of the following methods:

   - Use a property tag in the `build.xml` file.

     ```
     <property name="build.compiler"
     value="com.ouncelabs.ounceant.OunceCompilerAdapter"/>
     ```

   - Specify the value for `build.compiler` on the command line when invoking Ant using the `-D` option.

     ```
     ant -Dbuild.compiler=
     com.ouncelabs.ounceant.OunceCompilerAdapter
     ```

   - Include the value for `build.compiler` in a text file and instruct Ant to load the properties in that file using the `properties` option as described in the Ant documentation.
3. Create `taskdefs`.

   To use Ounce/Ant tasks, you must reference `ounceant.jar` in a `taskdef` task. For example,

   ```
   <taskdef resource="com/ouncelabs/ounceant/task/ounce.xml"
   classpath="ounceant.jar"/>
   ```

To use the `var` task, `ant-contrib-1.0b3.jar` must reference the `taskdef` task as follows:

```
<taskdef resource="net/sf/antcontrib/antlib.xml"/>
```

Ounce/Ant properties

The table in this topic describes the optional Ounce/Ant properties.

Property	Valid values / Default value	Description
`ounce.default.configuration_name`	`<config_name>`	Configuration name created in the AppScan Source project. If not set, uses Configuration 1.
`ounce.build.compiler`	compiler name	The name of the compiler executable to use when the Ant `build.compiler` property is configured to use `OunceCompilerAdapter`. If not set, uses `javac`.
`ounce.project_name`	name	Project name. If not set, Ounce/Ant generates a name. See "Naming projects" on page 62 for more information.
`ounce.project_dir` (Required to set `ounce.project_name`)	`<project_dir>`	Project working directory. If not set, uses the current build file directory.
`ounce.application_name`	`<app_name>`	Application name. See "Creating and naming applications" on page 63.
`ounce.application_dir` (Required to set `ounce.application_name`)	`<app_dir>`	Application working directory. You must set `ounce.application_name` and `ounce.application_dir` to add a project to an application.
`ounce.projects.store_full_paths` (optional)	`true` or `false`	Store absolute paths created in applications and projects. If not set, uses relative paths.

Setting properties

You can use the `var` task to set properties any number of times. When invoked, either explicitly or implicitly, `ounceCreateProject` uses the current property values. You can set properties by one of the methods outlined in this task topic.

Procedure

- Use an attribute to an Ounce/Ant task if the attribute exists.

- Use a `var` task before the appropriate AppScan Source task. Properties set with `var` remain in scope only for the file in which they are set. To use the `var` task, the Ant contribution library must be in the Ant `lib` directory, and the `taskdef` task must reference it.

  ```
  <taskdef resource="net/sf/antcontrib/antlib.xml"/>
  ```
- Use `-D` on the Ant command line.
- Place the property in a build properties file (typically called `build.properties`).

Creating projects

The `ounceCreateProject` task generates AppScan Source projects.

Use `ounceCreateProject` explicitly or implicitly:

- Explicitly call `ounceCreateProject` in an Ant build file.
- Implicitly call `ounceCreateProject` every time the `javac` task is called by overriding the `build.compiler` property.

Projects can be grouped into applications. You can use the optional application attribute or the appropriate Ounce/Ant properties to place the created project into an application.

Projects can also be augmented using Ounce/Ant. Project attributes from multiple projects can be merged into one AppScan Source project. If more than one `javac` or `ounceCreateProject` is to augment the same project, retain the same name and directory.

Note: If you override `build.compiler` to create projects, run `ant clean` first.

Refer to "ounceCreateProject task example" on page 62 for usage of properties and their nested attributes.

ounceCreateProject

`ounceCreateProject` accepts the following parameters and nested elements:

Parameters specified as attributes	Description
`name`	Name for the project.
`workingDir`	Working directory for the project.
`classpath`	Class path for the project.
`sourcepath`	Source path for the project source dependencies. These files are not scanned for vulnerabilities.
`jdkName`	AppScan Source JDK name to use when scanning project. (Must be created in AppScan Source for Analysis)
`appName`	Application that contains this project (optional).
`appDir`	Application directory (optional)

Parameters specified as nested elements	Description
`ounceSourceRoot`	Specify the project source roots

Parameters specified as nested elements	Description
ounceWeb	Specify the project web content
ounceExclude	Allows the exclusion of certain files from the file set specified in the parent element.

ounceCreateProject task example

```
<ounceCreateProject
 name="myProjectName"
 workingDir="${sandbox}/myProject"
 classpath="${my.class.path}"
 sourcepath="${my.source.path}"
 jdkName="jdk15"
 appName="myApp"
 appDir="${sandbox}>
<ounceSourceRoot dir="src"/>
<ounceExclude dir="src/test"/>
<ounceExclude file="src/mydir/Bad.java"/>
<ounceSourceRoot dir="src2"/>
<ounceSourceRoot file="src3/mydir.java"/>
<ounceWeb webContextRoot="web/myProject.war"/>
<ounceExclude dir="web/test"/>
<ounceExclude file="web/partial.jsp"/>
</ounceCreateProject>
```

ounceSourceRoot

ounceSourceRoot accepts the following parameters specified as attributes:

Parameters Specified as Attributes	Description
dir	Path to a valid Java source root
file	Path to an individual file

ounceWeb

ounceWeb accepts the following parameters specified as attributes.

Parameters Specified as Attributes	Description
webContextRoot	Path to a valid web context root or war file.

ounceExclude

ounceExclude accepts the following parameters specified as attributes:

Parameters Specified as Attributes	Description
dir	Path to a directory to exclude
file	Path to an individual file to exclude

Naming projects

You can name a project using the default naming convention or by creating a project name manually.

Default project naming

The Ounce/Ant default naming scheme allows for the generation of unique project names.

```
<directory>_<target_name>[_<number>]
```

- <directory>: Name of the project directory (does not include full path)
- <target name>: Name of the current target
- <number>: An integer that begins at 1 and increments as necessary to guarantee that it is unique. Note that <number> appears only if there is a conflict.

Example

With the following parameters, the project name becomes test_compile. If two projects are created in the compile target, the second project name becomes test_compile_1.

```
<working_directory> = C:\mydir\test
<directory> = test
<target_name> = compile
```

Naming projects manually

If you create a project with ounceCreateProject, you can use the name attribute to specify a project name. If the name attribute is not specified, or you create projects using javac, the AppScan Source task looks at the ounce.project_name property, and if set, it uses the value as the project name. Therefore, the ounce.project_name property can be set using var before each call to the javac task, providing a unique name to every project created.

Creating and naming applications

Ounce/Ant can create AppScan Source applications during operation and group the created AppScan Source projects into the applications.

You create these applications with two Ant properties:

- ounce.application_name
- ounce.application_dir

Important: To create the application, you must either set both properties or set the appName and AppDir attributes of the ounceCreateProject task.

If these properties are set during project creation, the project is added to an application with the name specified by ounce.application_name and a working directory specified by ounce.application_dir.

You can set these properties multiple times using the var task to group projects into the applications that you want. An appropriate application name indicates the combination of projects into a single application.

Tip: If you use ounceCreateProject, you may also use the appName and appDir attributes to specify the application in which to include the project.

Build integration

The `ounceCli` task allows access to the AppScan Source command line interface (CLI) to scan during a build. The `ounceCli` task can invoke `cli` from Ant.

`ounceCli` requires the following parameters specified as attributes:

- `dir`: Location of the AppScan Source installation directory
- `script`: Location of `cli` script file to run
- `output`: Location of `cli` output file

Example

```
<ounceCli
dir="${installDir}"
 script="${scripts}/cli_script.txt"
 output="log.txt"/>
```

Chapter 4. AppScan Source Data Access API

The Data Access API provides access to AppScan Source-generated assessment results, including findings and finding details. It also provides access to assessment metrics such as analysis date and time, lines of code, V-density, and number of findings.

The Data Access API is included with the installation of these AppScan Source components:

- AppScan Source for Analysis
- AppScan Source command line interface (CLI)

The Data Access API is installed as `<install_dir>\sdk\ouncesdk.jar` (where `<install_dir>` is the location of your AppScan Source installation).

When running a program that uses the SDK, specify these Java Virtual Machine (JVM) arguments on the command line:

```
java -classpath
<install_dir>\lib\avalon-framework-4.1.5.jar;
<install_dir>\lib\icu4j-4_8.jar;
<install_dir>\lib\jacorb.jar;
<install_dir>\lib\log4j-1.2.8.jar;
<install_dir>\lib\logkit-1.2.jar;
<install_dir>\sdk\ouncesdk.jar;
<install_dir>\lib\xml-apis.jar;
<install_dir>\lib\saxon9.jar
... com.company.product.ClassName
```

The Data Access API requires JDK 1.5 or later.

Data Access API object model

Diagram 1 - UML (Unified Modeling Language) diagram detailing the object model of assessments

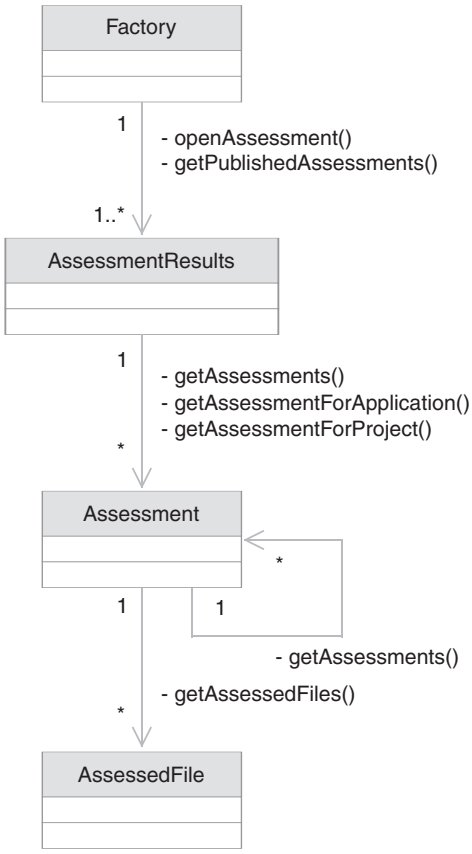


Diagram 2 - UML (Unified Modeling Language) diagram detailing the object model of findings

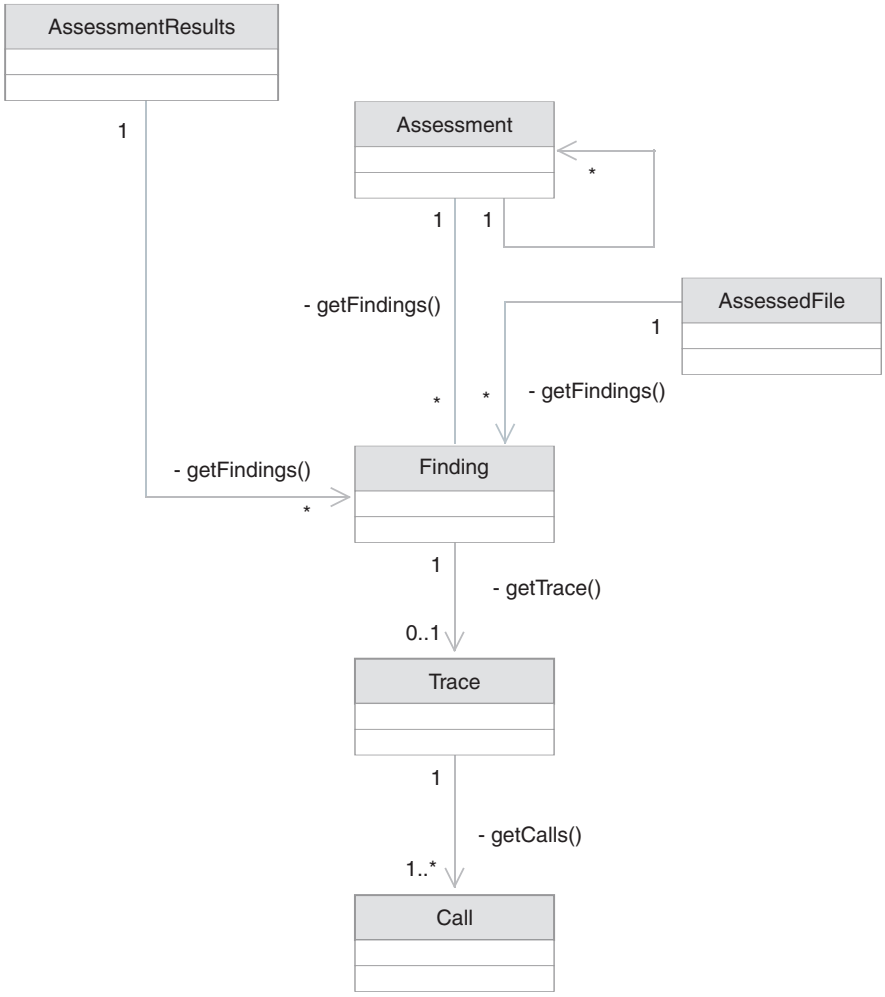


Diagram 1 - UML (Unified Modeling Language) diagram detailing the object model of assessments

At the top of the diagram is the `Factory` class. The `Factory` class is shown as being related to the `AssessmentResults` class via two methods in the `Factory` class: `openAssessment()` and `getPublishedAssessments()`. The relationship between these two classes is one `Factory` class to one or more `AssessmentResults` classes.

Under the `Factory` class, the diagram depicts the `AssessmentResults` class. The `AssessmentResults` class is shown as being related to the `Assessment` class via three methods: `getAssessments()`, `getAssessmentForApplication()`, and `getAssessmentForProject()`. The relationship between these two classes is one `AssessmentResults` class to zero or more `Assessment` classes.

Under the `AssessmentResults` class, the diagram depicts the `Assessment` class. One `Assessment` class is shown as being related to zero or more `AssessedFile` classes via the `getAssessedFiles()` method. The `Assessment` class also depicts a relationship with itself via the `getAssessments()` method. This relationship is shown as one `Assessment` class to zero or more child `Assessment` classes.

Under the Assessment class, the diagram depicts the AssessedFile class. One Assessment class is related to zero or more AssessedFile classes via the getAssessedFiles() method.

Diagram 2 - UML (Unified Modeling Language) diagram detailing the object model of findings

At the top of the diagram are three classes: AssessmentResults, Assessment, and AssessFile. Each of these three classes are depicted as being related to the Finding class via a getFindings() method. The relationship is shown as one to zero or more Finding classes. The Assessment class also depicts a relationship with itself of one Assessment class to zero or more child Assessment classes.

Under the Finding class, the diagram depicts the Trace class. One Finding is related to zero or one Trace classes via the getTrace() method.

Under the Trace class, the diagram depicts the Call class. One Trace class is related to one or many Call classes via the getCalls() method.

Using the Data Access API

This section contains a number of scenarios for which you might want to use the Data Access API. You can find more complete examples in the SamplePublished.java and SampleSdk.java files included in <install_dir>\sdk\sample\com\ouncelabs\sdk\sample (where <install_dir> is the location of your AppScan Source installation).

Opening an assessment from a file

This sample code is provided as an example for opening an assessment from a file.

```
Factory factory = new Factory();
try {
 factory.init(install_dir);
 factory.login("admin", "123456", "localhost");
 AssessmentResults assessmentResults =
  factory.openAssessment("myApp.ozasmt");
 printAssessmentResults(assessmentResults);
} finally {
 factory.shutdown();
}
```

Printing a list of findings

This sample code is provided as an example for printing a list of findings.

```
private void printAssessment(AssessmentResults results)
  throws OunceException {
 Assessment childAssessment = assessmentResults.
 getAssessmentForApplication("myApp");
 Finding[] findings = childAssessment.getFindings();
 for (int i = 0; i < findings.length; ++i) {
  Finding finding = findings[i];
  Object[] data = new Object[] {
   finding.getApiName(),
   finding.getVulnerabilityType(),
   finding.getClassification(),
   finding.getSeverity(),
   finding.getFilename(),
   new Integer(finding.getLineNumber()),
   finding.getSrcContext(),
   Boolean.valueOf(finding.isExcluded()),
  };
```

```
  printData(i,data);
  Trace trace = finding.getTrace();
  if (trace != null) {
   printTrace(trace);
  }
 }
}
```

Getting a list of published assessments

This sample code is provided as an example for getting a list of published assessments.

```
Factory factory = new Factory();
try {
 factory.init(install_dir);
 factory.login("admin", "123456","localhost");

 AssessmentFilter filter = new AssessmentFilter(20);
 filter.setUserName("joe");
 filter.setDateProximity(2.5, DateProximityUnit.DAY);
 AssessmentResults[] publishedAssessments =
 factory.getPublishedAssessments(filter);
 printAssessmentList(publishedAssessments);
} finally {
 factory.shutdown();
}
```

Printing a trace

This sample code is provided as an example for printing a trace.

```
private void printTrace(Trace trace) throws OunceException {
 Call[] calls = trace.getCalls();
 for (int i = 0; i < calls.length; ++i) {
  printCall(calls[i], 1);
 }
}
 private void printCall(Call call, int indentLevel) throws
 OunceException {
 printSpaces(indentLevel * 4);
 Object data[] = new Object[] {
  call.getFilename(),
  call.getClassName(),
  call.getMethodName(),
  call.getSrcContext()
 };
 printData(data);
 Call[] calls = call.getCalls();
 for (int i = 0; i < calls.length; ++i) {
 printCall(calls[i], indentLevel + 1);
 }
}
```

Data Access API classes and methods

This section describes the AppScan Source Data Access classes and methods in detail. The classes are:

- `com.ouncelabs.sdk.Factory`
- `com.ouncelabs.sdk.assessment.AssessedFile`
- `com.ouncelabs.sdk.assessment.Assessment`
- `com.ouncelabs.sdk.assessment.AssessmentDiff`
- `com.ouncelabs.sdk.assessment.AssessmentResults`

- com.ouncelabs.sdk.assessment.Finding
- com.ouncelabs.sdk.assessment.Trace
- com.ouncelabs.sdk.assessment.Call
- com.ouncelabs.sdk.assessment.SeverityType
- com.ouncelabs.sdk.assessment.ClassificationType
- com.ouncelabs.sdk.assessment.AssessmentFilter
- com.ouncelabs.sdk.assessment.OunceException

AssessedFile

The com.ouncelabs.sdk.assessment.AssessedFile class represents an assessment of an individual file. It provides access to assessment data for a file, such as its findings and statistics.

AssessedFile.getFindings

Signature

```
public Finding[] getFindings()
```

Description

Provides access to all of the findings for this AssessedFile.

Returns

An array of finding objects for this assessed file.

Throws

OunceException, if the Data Access API cannot retrieve the findings.

AssessedFile.getStats

Signature

```
public AssessmentStats getStats()
```

Description

Provides access to rolled-up statistics for this file, such as total number of findings, total lines, and so on.

Returns

AssessmentStats object containing the statistics for this assessment.

Throws

OunceException, if the Data Access API cannot retrieve the statistics.

AssessedFile.getFilename

Signature

```
public String getFilename()
```

Description

Provides access to the absolute path to the file represented by this `AssessedFile`.

Returns

String containing the absolute path to the file represented by this `AssessedFile`.

Assessment

The `com.ouncelabs.sdk.assessment.Assessment` class represents an assessment of an application or a project.

Assessment.getFindings

Signature

```
public Finding[] getFindings()
```

Description

Provides access to all of the findings for this Assessment.

Returns

An array of finding objects for this assessment.

Assessment.getStats

Signature

```
public AssessmentStats getStats()
```

Description

A complete list of statistics including: the total number of files scanned, total number of vulnerabilities found, time of scan, and vulnerability density.

Returns

`AssessmentStats` object containing the statistics for this assessment

Assessment.getAssessments

Signature

```
public Assessment[] getAssessments()
```

Description

Returns the child `Assessment` objects for this assessment. Since project assessments do not have assessment children, assessment child objects are only returned for application assessment objects.

Returns

- For application assessments, returns an array of `Assessment` objects, one for each scanned project in the application.
- For project assessments, returns an empty array.

Throws

`OunceException`, if the Data Access API cannot retrieve the assessments.

`Assessment.getFiles`

Signature

```
public AssessedFile[] getFiles()
```

Description

Provides access to each file that was part of this assessment.

Returns

An array of `AssessedFile` objects. Each object represents a file that was part of this assessment.

Throws

`OunceException`, if the Data Access API cannot retrieve the assessed files.

`Assessment.getFileByPath`

Signature

```
public AssessedFile getFileByPath(filePath)
```

Description

Provides access to an assessed file by path.

Returns

`AssessmentFile` object corresponding to specified file path or null if the specified file path does not correspond to an assessed file.

Throws

`OunceException`, if the Data Access API cannot retrieve the assessed files.

`Assessment.getAssesseeName`

Signature

```
public String getAssesseeName()
```

Description

Provides the name of the application or project from which this assessment was generated.

Returns

Name of an application or project.

AssessmentDiff

The com.ouncelabs.sdk.assessment.AssessmentDiff class holds the difference between two assessments, providing the delta between the two assessments .

AssessmentDiff.getCommonFindings

Signature

```
public Finding[] getCommonFindings()
```

Description

Get the findings that were common to both assessments.

AssessmentDiff.getLostFindings

Signature

```
public Finding[] getLostFindings()
```

Description

Get the findings that were in the first assessment, but not in the second.

AssessmentDiff.getNewFindings

Signature

```
public Finding[] getNewFindings()
```

Description

Get the findings that were in the second assessment, but not in the first.

AssessmentFilter

Use the com.ouncelabs.sdk.assessment.AssessmentFilter class to specify filtering criteria when retrieving published assessments.

AssessmentFilter.AssessmentFilter

Signature

```
public AssessmentFilter(int maxResults)
```

Description

Constructor that only specifies the maximum number of published assessments to retrieve. You can specify additional filtering options by calling the set methods in the class (for example, setUserName() to filter by published user).

Parameters

- maxResults: The maximum number of results to return.

AssessmentFilter.AssessmentFilter

Signature

```
public AssessmentFilter(String appName,
String userName,
Double dateProximityDuration,
int DateProximityUnit,
Long dateRangeStart, Long dateRangeEnd,
int maxResults)
```

Description

Constructor taking all criteria options as arguments.

Parameters

- `appName`: The application name. (Null if not filtered by `appName`.)
- `userName`: The user who published the application. (Null if not filtered by user name.)
- `dateProximityDuration`: When paired with `dateProximityUnit`, the number of units (days, weeks, and so on) to filter starting with the current date. (Null if not filtered by date proximity.)
- `dateProximityUnit`: When paired with `dateProximityDuration`, the unit, such as days, weeks, and so on, by which to count. Required if `dateProximityDuration` is specified. See "`DateProximityUnit.value`" on page 80 for valid units.
- `dateRangeStart`: The start of a date range. (Null if not filtering by date range.)
- `dateRangeEnd`: The end of the date range (null if not filtering by date range).
- `maxResults`: The maximum number of results to return.

AssessmentResults

The `com.ouncelabs.sdk.assessment.AssessmentResults` class represents the entire assessment.

AssessmentResults.getFindings

Signature

```
public Finding[] getFindings()
```

Description

Provides access to all of the findings for this `AssessmentResults`.

Returns

An array of finding objects for this assessment.

Throws

`OunceException`, if the Data Access API cannot retrieve the findings.

AssessmentResults.getAssessments

Signature

```
public Assessment[] getAssessments()
```

Description

Provides access to the assessment data for each application that was assessed as part of this `AssessmentResults`.

Returns

An array of Assessment objects, one for each application that was part of the `AssessmentResults`.

Throws

`OunceException`, if the Data Access API cannot retrieve the assessments.

AssessmentResults.getStats

Signature

```
public AssessmentStats getStats()
```

Description

Provides access to rolled-up statistics for this assessment, such as total files assessed, total lines, and so forth.

Returns

`AssessmentStats` object containing the statistics for this assessment.

Throws

`OunceException`, if the Data Access API cannot retrieve the assessment statistics.

AssessmentResults.getAssessmentForApplication

Signature

```
public Assessment[] getAssessmentForApplication(String applicationName)
```

Description

Provides access to the assessment data for the specified application.

Parameters

`applicationName`: The application name.

Returns

An array of Assessment objects, one for each application that matches the specified name.

Throws

`OunceException`, if the Data Access API cannot retrieve the assessments.

Compatibility Note

This method is only valid for assessments run with AppScan Source (Ounce) Version 6.0 or later. When called on assessments generated in an earlier version of AppScan Source, it always returns an empty array.

AssessmentResults.getAssessmentForProject

Signature

```
public Assessment[]
getAssessmentForProject(String projectName, String applicationName)
```

Description

Provides access to the assessment data for the specified project in the specified application.

Parameters

`projectName`: The project name.

`applicationName`: The name of the application in which the project resides.

Returns

An array of Assessment objects, one for each project that matches the specified name.

Throws

`OunceException`, if the Data Access API cannot retrieve the assessments.

Compatibility Note

This method is only valid for assessments run with AppScan Source (formerly Ounce) Version 6.0 or later. When called on assessments generated in an earlier version of AppScan Source, it always returns an empty array.

AssessmentResults.getName

Signature

```
public String getName()
```

Description

Returns the name of the `AssessmentResults`.

Returns

String containing the name of the `AssessmentResults`.

AssessmentResults.getMessages

Signature

```
public Message[] getMessages()
```

Description

Provides access to all status messages that appeared in the console when running the assessment in AppScan Source for Analysis.

Returns

An array of Message objects.

Throws

`OunceException`, if the Data Access API cannot retrieve the messages.

Call

The `com.ouncelabs.sdk.assessment.Call` class represents a node in an AppScan Source trace.

Call.getCalls

Signature

```
public Call[] getCalls()
```

Description

Provides access to the calls that were made in the context of this call.

Returns

An array of Call objects, in the order that they were invoked

Throws

`OunceException`, if the Data Access API cannot retrieve the calls.

Call.getFilename

Signature

```
public String getFilename()
```

Description

Provides access to the filename in which this call occurred.

Returns

The filename for this call.

Call.getLineNumber

Signature

```
public int getLineNumber()
```

Description

Provides access to the line number on which the call occurred.

Returns

The line number of this call.

Call.getColumnNumber

Signature

```
public int getColumnNumber()
```

Description

This call provides access to the column number on which the call occurred.

Returns

The column number the call originated on.

Call.getSignature

Signature

```
public String getSignature()
```

Description

Provides access to the signature of the API that was called.

Returns

The signature of this call.

Call.getSrcContext

Signature

```
public String getSrcContext()
```

Description

Provides access to the source context of the call.

Returns

The source context of this call.

Call.getMethodName

Signature

```
public String getMethodName()
```

Description

Provides access to the method name that was called.

Returns

The method name for this call.

Call.getClassName

Signature

```
public String getClassName()
```

Description

Provides access to the class name to which the called method belongs.

Returns

The class name for this call method.

Call.getTraceType

Signature

```
public TraceType getTraceType()
```

Description

Provides access to the trace type for this call. The trace type, for example, can specify that a call is either a source or a sink.

Returns

`TraceType` object that represents the trace type for the call.

ClassificationType

The `com.ouncelabs.sdk.assessment.ClassificationType` class represents the classification of a finding.

Static members

- `ClassificationType.Vulnerability`
- `ClassificationType.Type1`
- `ClassificationType.Type2`
- `ClassificationType.Unknown`

ClassificationType.value

Signature

```
public int value()
```

Description

Provides access to the value of this `ClassificationType`. The value returned corresponds to one of the values of the static members.

Returns

The classification value for this `ClassificationType` object.

DateProximityUnit

When com.ouncelabs.sdk.assessment.DateProximityUnit is paired with dateProximityDuration, the unit, such as days, weeks, and so forth, by which to count. Required when dateProximityDuration is specified. Valid units are described in this section.

Static members

- DateProximityUnit.Hour
- DateProximityUnit.Day
- DateProximityUnit.Week
- DateProximityUnit.Month
- DateProximityUnit.Year

DateProximityUnit.value

Signature

```
public EnumType value()
```

Description

Provides access to the value of this DatProximityUnit. The value returned corresponds to one of the values of the static members.

Returns

The date value for this DateProximityUnit object.

Factory

com.ouncelabs.sdk.Factory provides methods for initialization, logging in, and opening assessments. This class is the entry point into the Data Access API.

Factory.init

Signature

```
public void init(String installDir, boolean acceptInvalidSSLCerts)
```

Description

Initializes the Data Access API. You must call Factory.init before you call any other method.

Parameters

- installDir: Path to the AppScan Source installation directory (see "Default installation location" on page 149).
- acceptInvalidSSLCerts: Set the boolean to true if you want to accept invalid SSL certificates (SSL certificates will not require validation).

Throws

OunceException, if the initialization fails. Generally, this only occurs if you specify the wrong directory.

Factory.shutdown

Signature

```
public void shutdown()
```

Description

Exits the Data Access API. If `Factory.init` is called, you must call `Factory.shutdown` for the Data Access API to shut down correctly. If not called, `OunceScanner.exe` may continue to run as a rogue process.

Factory.clearCache

Signature

```
public void clearCache()
```

Description

This call clears the cache for the Data Access API. It should be used periodically when processing many findings results in excessive memory usage.

Factory.login

Signature

```
public void login(String user name, String password, String host name[int :port])
```

Description

Logs in to the AppScan Enterprise Server running at the specified URL or on the specified host name. You must log in before you can get assessments published to the AppScan Source Database.

Parameters

- `username`: The user name of the user logging in.
- `password`: The password of the user logging in.
- `hostname`: The URL or host name host name of the AppScan Enterprise Server to which you want to log in. `hostname` can be either an IP address or a domain name.

If necessary, you can specify the AppScan Enterprise Server port number by appending `:<port>`. For example, `public void login("hwall", "shhh2008", "MyHost:2880")`.

Throws

`OunceException`, if the login fails.

Factory.openAssessment

Signature

```
public AssessmentResults openAssessment(String pathname)
```

Description

Opens the assessment specified by pathname.

Parameters

`pathname`: The file path to an assessment file.

Returns

The `AssessmentResults` object representing the assessment specified by `pathname`.

Throws

- `FileNotFoundException`, if the specified filename does not exist, or cannot be opened.
- `OunceException`, if the Data Access API cannot load the assessment file.

`Factory.getPublishedAssessments`

Signature

```
public AssessmentResults[] getPublishedAssessments(AssessmentFilter filter)
```

Description

Provides access to published assessments. Takes an `AssessmentFilter` object that you can use to specify the set of published assessments to retrieve.

Parameters

`filter`: An `AssessmentFilter` object that specifies the set of published assessments to retrieve.

Returns

An array of `AssessmentResults` objects matching the specified filter.

Throws

`OunceException`, if the Data Access API cannot retrieve the assessment results.

`Factory.diffAssessments`

Signature

```
public AssessmentDiff diffAssessments
(AssessmentResults result1, AssessmentResults result2)
```

Description

Finds the difference between two assessment results.

Throws

`OunceException`.

`Finding`

The `com.ouncelabs.sdk.assessment.Finding` class represents an individual finding in an assessment. It provides access to all of the data associated with a finding, such as classification and severity.

Finding.getFilename

Signature

```
public String getFilename()
```

Description

Provides access to the filename in which this finding was found.

Returns

A string containing the absolute path to the file in which this finding was found.

Finding.getLineNumber

Signature

```
public int getLineNumber()
```

Description

Provides access to the line number on which the finding was found.

Returns

The line number for this finding.

Finding.getApiName

Signature

```
public String getApiName()
```

Description

Provides access to the name of the API for this finding.

Returns

The API name for this finding.

Finding.getCallerName

Signature

```
public String getCallerName()
```

Description

Provides access to the signature of the method that contained the call to the API that this finding represents.

Returns

The signature of the caller.

Finding.getClassification

Signature

```
public ClassificationType getClassification()
```

Description

Provides access to current classification of the finding.

Returns

`ClassificationType` object that represents the classification of the finding

Finding.getSeverity

Signature

```
public SeverityType getSeverity()
```

Description

Get the current value of the severity in the finding.

Returns

`SeverityType` object that represents the severity of the finding. `SeverityType` is described in "`SeverityType.value`" on page 89.

Finding.getVulnerabilityType

Signature

```
public String getVulnerabilityType()
```

Description

Provides access to the current vulnerability type of the finding.

Returns

The vulnerability type for this finding. Vulnerability types, such as Buffer Overflow or SQL Injection, are described in the AppScan Source Security Knowledgebase.

Finding.getOriginalClassification

Signature

```
public ClassificationType getOriginalClassification()
```

Description

Provides access to the original classification of the finding. If the finding has not been modified, `getOriginal` is the same as `getClassification`.

Returns

`ClassificationType` object that represents the original classification of a finding where the classification has been modified.

Finding.getOriginalSeverity

Signature

```
public SeverityType getOriginalSeverity()
```

Description

Gets the original value of the severity of the finding. If the finding has not been modified, `getOriginalSeverity` is the same as `getSeverity`.

Returns

`SeverityType` object that represents the original severity for a finding where the severity has been modified.

Finding.getOriginalVulnerabilityType

Signature

```
public String getOriginalVulnerabilityType()
```

Description

Provides access to the original vulnerability type of the finding. If the finding has not been modified, `getOriginalVulnerabilityType` is the same as `getVulnerability`.

Returns

A `VulnerablilityType` object that represents the original vulnerability type for a finding where the vulnerability has been modified.

Finding.getModifiedClassification

Signature

```
public ClassificationType getModifiedClassification()
```

Description

Provides access to the modified classification of the finding.

Returns

`ClassificationType` object that represents the classification of a finding where the classification has been modified. (Null if not matched.)

Finding.getModifiedSeverity

Signature

```
public SeverityType getModifiedSeverity()
```

Description

Gets the modified value of the severity. Same as `getSeverity()` if modified.

Returns

SeverityType object that represents the severity for a finding where the severity has been modified. (Null if not matched.)

Finding.getModifiedVulnerabilityType

Signature

```
public String getModifiedVulnerabilityType()
```

Description

Provides access to the modified vulnerability type of the finding.

Returns

A VulnerablilityType object that represents the vulnerability type for a finding where the vulnerability has been modified. (Null if not matched.)

Finding.getSrcContext

Signature

```
public String getSrcContext()
```

Description

Provides access to the source context for this finding.

Returns

The source context for this finding.

Finding.getDefectSubmissionUser

Signature

```
public String getDefectSubmissionUser()
```

Description

Provides access to the user name who last submitted this finding to an external defect tracking system. Note, it is the defect tracking system user name, which may be different from the AppScan Source user name.

Returns

The submitting user from the last defect tracking system submission of this finding.

Finding.getDefectSubmissionDate

Signature

```
public String getDefectSubmissionDate()
```

Description

Provides access to the date string from when this finding was last submitted to an external defect tracking system.

Returns

The submission date string from the last defect tracking system submission of this finding.

Finding.getDefectInfo

Signature

```
public String getDefectInfo()
```

Description

Provides access to the external defect tracking system ID from when this finding was last submitted to an external defect tracking system.

Returns

A string with defect IDs associated with this finding.

Finding.getProperties

Signature

```
public String[] getProperties()
```

Description

This call provides access to the properties associated with the finding. These properties provide information about this finding's vulnerability type and about this finding's API.

Returns

Array of properties:

- `Vulnerability.value`: Vulnerability type reported in the Findings views, such as:

  ```
  Vulnerability.BufferOverflow
  Vulnerability.Injection.SQL
  ```

- `Mechanism.value`: API security mechanism; used to define custom report categories, such as:

  ```
  Mechanism.AccessControl
  Mechanism.Cryptography
  ```

- `Technology.value`: API technology; used to define custom report categories, such as:

  ```
  Technology.Communications.HTTP
  Technology.Database
  ```

- `Attribute.value`: API method tag, such as:

  ```
  Attribute.Deprecated
  Attribute.Modifier.Protected
  ```

See the AppScan Source Security Knowledgebase for detailed information about property values.

Throws

`OunceException`, if the Data Access API cannot retrieve the properties.

`Finding.isExcluded`

Signature

```
public boolean isExcluded()
```

Description

Indicates whether this finding is excluded from the assessment. Excluded findings do not contribute to any statistics for the assessment.

Returns

If excluded, `true`, if not excluded, `false`.

`Finding.getTrace`

Signature

```
public Trace getTrace()
```

Description

Provides access to the AppScan Source trace associated with this finding, if any.

Returns

Trace object for the finding if there is one, null if not.

Throws

`OunceException`, if the Data Access API cannot retrieve the trace.

`Finding.getNotes`

Signature

```
public String getNotes()
```

Description

Provides access to the notes on a finding.

Returns

Any notes for the finding, null if there are no notes.

Throws

`OunceException`, if the Data Access API cannot retrieve the notes.

`Trace`

The `com.ouncelabs.sdk.assessment.Trace` class represents the AppScan Source trace for a finding.

Trace.getCalls

Signature

```
public Call[] getCalls()
```

Description

Provides access to the root call of the AppScan Source trace.

Returns

An array of `Call` objects. Currently always contains a single object.

Throws

`OunceException`, if the Data Access API cannot retrieve the calls.

SeverityType

The `com.ouncelabs.sdk.assessment.SeverityType` class represents the severity of a finding.

Static members

Severity types are:

- `SeverityType.High`
- `SeverityType.Medium`
- `SeverityType.Low`
- `SeverityType.Info`
- `SeverityType.Unknown`

SeverityType.value

Signature

```
public int value()
```

Description

Provides access to the value of this `SeverityType`. The value returned corresponds to one of the values of the static members.

Returns

The severity value for this `SeverityType` object.

OunceException

`com.ouncelabs.sdk.assessment.OunceException` is the exception class used by the Data Access API. To find more information about an exception, call `getMessage()` on the exception.

Chapter 5. Ounce/Maven plug-in

This section describes the Ounce/Maven plug-in, which uses Maven, an Apache build tool, to integrate AppScan Source into the Maven workflow.

The Ounce/Maven plugin allows you to create AppScan Source application and project files. If AppScan Source for Automation is also installed, use Ounce/Maven to run source code security scans and generate comprehensive security reports.

For information about AppScan Source for Automation, see Chapter 6, "AppScan Source for Automation," on page 97.

For more information about the AppScan Source family of products, see http://www.ibm.com/software/rational/products/appscan/source/.

Installing Ounce/Maven

Before you begin

The following are prerequisites for installing and running Ounce/Maven:

- Apache Maven Version 2.x: For information about deploying and using Maven plug-ins, see the Apache Maven Project website at http://maven.apache.org/.
- AppScan Source for Automation: For more information, see Chapter 6, "AppScan Source for Automation," on page 97.

About this task

Once you install and configure Maven, Ounce/Maven downloads the first time you reference it.

The Ounce/Maven site documentation includes descriptions of the Ounce/Maven goals, their parameters, examples, usage notes, and detailed examples.

You can find the site documentation at http://mojo.codehaus.org/ounce-maven-plugin.

Procedure

1. If Maven is not already installed, install Maven from http://maven.apache.org/. Follow the directions on the Maven website.
2. Either:
   - Edit a Maven `pom` file to include one or more Ounce/Maven goals, as described in the Ounce/Maven site documentation
   - Call one or more Ounce/Maven goals from the command line, as described in the Ounce/Maven site documentation

Using Ounce/Maven

The Ounce/Maven plug-in allows you to use Ounce/Maven to create AppScan Source projects and applications, scan the applications, publish the resulting assessments, and generate AppScan Source reports. Specify the Ounce/Maven goals and parameters as you would for any other Maven plugin.

You can call Ounce/Maven commands in two ways:

- Using a Maven `pom` (build) file: The `pom` file allows you to create AppScan Source application and project files as part of your build. After installing Ounce/Maven, you can modify a Maven `pom` file to specify the `ounce:application` and `ounce:project-only` goals as needed for your AppScan Source tasks.
- From the command line: Invoke the `ounce:project`, `ounce:scan`, and `ounce:report` goals from the command line to create AppScan Source project files (or override project file parameters from the `pom` file), start AppScan Source scans, publish assessments, and generate AppScan Source reports.

Each of the Ounce/Maven goals includes a number of parameters.

- For information about Ounce/Maven goals, see "Ounce/Maven goals" on page 93.
- For information about the parameters for each goal, see the Ounce/Maven site documentation at http://mojo.codehaus.org/ounce-maven-plugin.

Ounce/Maven scenarios

Ounce/Maven allows you to:

- Create AppScan Source application and project files
- Scan applications and publish the results (assessments) to the AppScan Source database
- Generate reports from assessment results
- Integrate AppScan Source reports into the `site` target

This section describes these tasks in brief. For detailed information about AppScan Source-specific concepts and language, see the *AppScan Source for Analysis User Guide*.

Creating application and project files

AppScan Source uses an application and project model to manage data, as follows:

- **Application**: An application contains configuration data and customizable information, along with a list of the projects in that application.
- **Project**: A project consists of a set of files (including source code) and its related information (such as configuration data). To scan a project, it must be part of an application.

The `ounce:application` goal creates AppScan Source application (`.paf`) files - whereas `ounce:project` and `ounce:project-only` goals create AppScan Source project (`.ppf`) files.

See the *AppScan Source for Analysis User Guide* for information about other application and project file formats.

Scanning applications

An AppScan Source scan analyzes source code for security vulnerabilities. The result of a scan is an assessment, which is an XML file.

Note: For detailed information about AppScan Source capabilities, see the *IBM Security AppScan Source for Analysis User Guide*.

Use the `ounce:scan` goal from the command line to scan the application and its projects and optionally generate a report from the assessment.

After a scan is finished, Ounce/Maven allows you to:

- Publish the assessment to the AppScan Source Database. This makes the assessment results available to other users with access to the database and the necessary privileges.
- Generate a report.

Reporting

For many, the preferred output from AppScan Source is a detailed report that provides data about the assessments The `ounce:report` goal allows you to generate those reports from new or previous assessments. The `ounce:report` goal can, if needed, scan an application, and then publish the assessment or generate reports about the assessment. Unlike `ounce:scan`, however, you can use `ounce:report` to generate reports for existing assessments.

Integrating reports with the site target

To integrate `ounce:report` with the site target, add the Ounce/Maven plug-in to the reporting section of the `pom`. The site goal executes `ounce:report` and the report output becomes part of the site documentation for your application.

When integrating with the site target, you do not need to specify any additional parameters. You can specify the report type as one of the values described in "reportType Values" on page 95. If you do not specify a report type, the default report is `Findings`.

Ounce/Maven goals

Ounce/Maven provides the following goals to perform AppScan Source functions:

- `ounce:application`: Generates an AppScan Source application file that contains references to all child projects as defined by the `pom` file. An application is required for scanning (and therefore reporting).
- `ounce:project`: Creates one or more AppScan Source project files depending on the number of Maven child projects. The `ounce:project` goal is intended to run from the command line and incorporates the Maven build into the goal.
- `ounce:project-only`: Creates one or more AppScan Source project files depending on the number of Maven child projects. The `ounce:project-only` goal is provided to integrate the creation of AppScan Source project files into the Maven build life cycle.
- `ounce:scan`: Scans an application. You can optionally publish the assessment or generate a report. Run the `ounce:scan` goal from the command line.
- `ounce:report`: Generates an AppScan Source report. If needed to refresh results, execute a scan before generating the report. Run the `ounce:report` goal from the command line.

Note:

- To make your application and project files portable, create path variables to map file paths to their locations.
- For examples of how to use Ounce/Maven goals, see the Ounce/Maven site documentation at http://mojo.codehaus.org/ounce-maven-plugin/.

ounce:application

Description

Generates an AppScan Source application file that contains references to all child projects as defined by the pom file. An application is required for scanning and reporting. While `ounce:application` does not create AppScan Source project files, you can run `ounce:project-only` to create the project within the same build file.

ounce:project

Description

Use `ounce:project` from the command line. The `ounce:project` goal creates one or more AppScan Source project files, depending on the number of Maven child projects.

Using ounce:project

When you invoke the `ounce:project` goal from the command line, it automatically builds any required dependencies. If no application file exists, `ounce:project` builds it.

All parameters to `ounce:project` are optional. The project name is determined by the Maven `artifactId` and you cannot override it.

Run Maven with an `ounce:project` goal from the directory that contains the top-level `pom.xml` file (or, if only a subset of the build is required, from a subdirectory that contains a child `pom.xml`).

ounce:project-only

Description

The `ounce:project-only` goal is used from within a `pom` file. The `ounce:project-only` goal creates one or more AppScan Source project files depending on the number of Maven child projects.

Using ounce:project-only

All parameters to `ounce:project-only` are optional. Project name is determined by the Maven `artifactId` and cannot be overridden.

ounce:scan

Description

The `ounce:scan` goal generates a scan of a given application, optionally followed by a report on the resulting assessment. Run `ounce:scan` from the command line.

Note: The `ounce:scan` goal is not intended to be part of the Maven build life cycle.

Using ounce:scan

When you specify ounce:scan, you can request that Ounce/Maven run a report from the assessment immediately following the scan. In this case, specify the report parameters described in "reportType Values" and "reportOutputType Values." If you specify the reportType, you must also specify reportOutputType and reportOutputPath.

Note: The ounce:scan goal creates or updates application and project files, if needed.

ounce:report

Description

The ounce:report goal generates a report from an assessment. If you do not specify an existing assessment, ounce:report runs ounce:scan before generating the report. Run ounce:report from the command line.

Specify the report parameters described in "reportType Values" and "reportOutputType Values." If you specify the reportType, you must also specify reportOutputType and reportOutputPath.

reportType Values

- A Findings report:
  - Findings
  - Findings by Type
  - Findings by Classification
  - Findings by File
  - Findings by API
  - Findings by Bundle
  - Findings by CWE
  - DTS Activity
- An AppScan Source report:
  - DISA Application Security and Development STIG V3 R5
  - OWASP Top 10 2010
  - OWASP Top 10 2007
  - Software Security Profile
  - PCI Data Security Standard V2.0
  - PCI Data Security Standard V1.1
- A custom report, if available.

reportOutputType Values

- Specify one of the following formats for this report:
  - html: Generates the report as HTML and displays it online.
  - zip: Creates a ZIP file that contains all HTML report components.
- For reports in PDF format, you can specify the level of detail:
  - pdf-summary: Contains counts for each custom report group
  - pdf-detailed: Contains counts for each API for each vulnerability property
  - pdf-comprehensive: Contains tables consisting of every finding for every API

- `pdf-annotated`: Contains all findings, any notes included with the findings, and designated code snippets
- `pdf-annotated`: Generates an annotated report as a PDF file.

Chapter 6. AppScan Source for Automation

The Automation Server (`ounceautod`) allows you to automate key aspects of the AppScan Source workflow and integrate security with build environments during the software development life cycle (SDLC). The Automation Server allows you to queue requests to scan and publish assessments, and generate reports on the security of application code.

Through the AppScan Source for Automation client command line executable (`ounceauto`), you submit requests to the server. When processing requests, the Automation Server runs as a client of the associated AppScan Enterprise Server and can connect only to a single AppScan Enterprise Server. It listens on a TCP port (default 13205) for connections from local host only.

- On Windows systems, the Automation Server runs as the **IBM Security AppScan Source Automation** service.
- On Linux systems, it runs as a daemon:
  - To stop the daemon, issue this command: `/etc/init.d/ounceautod stop`
  - To start the daemon, issue this command: `/etc/init.d/ounceautod start`

The Automation Server processes requests as a specified AppScan Source user and thus inherits the permissions of that user. This user ID must have whatever permissions it needs, depending on the commands it needs to run. For example, if the user ID needs to run the `PublishAssessment` command, the user ID can be given publish and register permissions and does not require permission to scan (refer to the *Administering AppScan Source* section of the *AppScan Source Installation and Administration Guide* for more details). Submitting a request to the Automation Server does not require user credentials.

Specifying Automation Server login credentials from the command line

If you did not specify a user name and password during the installation process, as described in the IBM® Security AppScan Source Installation and Administration Guide, you must configure the Automation Server after installation to run as an AppScan Source user.

About this task

If the user specified does not already exist, you will need to specify it in the installation panel or via command line - and then create it manually (post-installation) with AppScan Source for Analysis or the AppScan Source command line interface (CLI). To create the new user by command line, use the "`newuser (nu)`" on page 41 command. When creating the new user, ensure that you specify the same user name and password that was specified for Automation Server login. Other settings, such as permission, can be set according to your needs.

Procedure

- **On Linux systems**:
  1. Change your `LD_LIBRARY_PATH` to include `<install_dir>/bin` (where `<install_dir>` is the location of your AppScan Source installation), for example:

```
export LD_LIBRARY_PATH=/opt/ibm/appscansource/
bin:$LD_LIBRARY_PATH
```

2. Execute <install_dir>/bin/ounceautod with arguments to specify a user name and password:

   ```
   /opt/ibm/appscansource/bin/ounceautod -u <username> -p <password> --persist
   ```

- **On Windows systems**:

  ```
  <install_dir>\bin\ounceautod.exe -u <user name> -p <password> --persist
  ```

  – <install_dir> is the location of your AppScan Source installation
  – -u <user name>: The user with which the Automation Server authenticates when processing a request. The user must be defined in AppScan Source for Analysis with the required permissions.
  – -p <password>: The password for the user. If you specify a user name, you must specify the password.
  – --persist: Preserve the login credentials on disk. Creates an encrypted key file with the specified user name and password.

Results

After you specify a user name and password, the credentials are saved to disk.

The Automation Server configuration file

The ounceautod configuration file, ounceautod.ozsettings, specifies the properties for the Automation Server daemon. The ounceautod.ozsettings file resides in <data_dir>\config (where <data_dir> is the location of your AppScan Source program data, as described in Chapter 9, "Installation and user data file locations," on page 149).

The ounceautod.ozsettings file contains these properties:

Property	Value	Description
ounceautod_max_ concurrent_requests	1 (Default)	Number of concurrent requests allowed.
ounceautod_accept_ssl	true (Default)	Automatically accept SSL certificates. For more information, see "AppScan Enterprise Server SSL certificates" on page 38.
ounceautod_port	13205 (Default)	Port number on which to run ounceautod. **Note:** The port value in this setting must match the port setting in the ounceauto.ozsettings file in <data_dir>\config.
ounceautod_server_hostname	<hostname>:port	Host name of the computer on which AppScan Enterprise Server was installed and the port on which it runs on that computer.

If this file is modified, you must restart the Automation Server service (described in Chapter 6, "AppScan Source for Automation," on page 97).

Automation Server logging

The Automation Server generates a log file, `ounceautod.log`, which logs every request made to the Automation Server as well as the response. Log entries reside in `<data_dir>\logs\ounceautod.log` (where `<data_dir>` is the location of your AppScan Source program data, as described in Chapter 9, "Installation and user data file locations," on page 149).

The Automation Server logs the following events for all requests:

- Request received
- Request started
- Request finished
- Request failed

Each log entry contains:

- Log entry timestamp
- Event type
- Request type
- Request ID
- Caller, if specified
- Request specific information

Enabling quality scanning

Quality scanning can be enabled by optional command arguments in the AppScan Source command line interface (CLI) and in AppScan Source for Automation.

Quality scanning is enabled in the command line interface when you use the `scan` command. See `scan (sc)` for more details.

If you are using AppScan Source for Automation to automate scanning, quality scanning is enabled via the `ScanApplication` command. See `ScanApplication` for more details.

Note: If you are using AppScan Source for Automation on Linux, you may encounter this error in the CLI output log:

```
Xlib: connection to ":1.0" refused by server
Xlib: No protocol specified
(Eclipse:13598): GLib-GObject-WARNING **:
   invalid (NULL) pointer instance
```

This is occurs when the AppScan Source for Automation user account does not have permission to conduct quality scanning. To correct this:

- From a terminal, issue the command: `xauth list $DISPLAY`

  This command will return the cookie that is used for the current display (for example, `my_hostname/unix:1  MIT-MAGIC-COOKIE-1 3968070ad5056a979d8e6a3a81017d5c`).

  Copy the cookie to the clipboard.
- Use the `su` command to switch to the AppScan Source for Automation user.

- Issue the command, `xauth add <identifier>`, where `<identifier>` is the value that was copied to the clipboard in Step 1.

ScanApplication

Description

Scan the specified application. Optionally save the assessment or generate an assessment report.

Syntax

```
ounceauto ScanApplication
<-application <application_name>|
 -application_file <application file>>
[-name <assessment name>]
[-save <filename>]
[-caller <caller>]
[-publish]
[-report <report type> <output format>
<output location>]
[-includeSrcBefore <n>
[-includeSrcAfter <n>]
[-scanconfig <scan_configuration_name>][-quality <path_file>]
[-qualityonly <path_file>]
```

- `-application <application name>`: The name of the application to scan.
- `-application_file <path to AppScan Source Application file>`: The path to an AppScan Source application file.
- `-name <assessment name>`: Optional. A name for the assessment.
- `-save <filename>`: Optional. Save the assessment results to this file.
- `-caller <caller>`: Optional. Assign a caller to the report generation operation. The caller can be the name of an actual user, but this is not required. The caller name is written to the ounceauto log file.
- `-publish`: Optional. Publish the assessment after scanning.
- `-report`: Optional. Generate a report after the scan.
- `<report type>`: Name of the report type. The report types consist of Findings reports, AppScan Source reports, and custom reports. Refer to the options in "GenerateReport" on page 105.
- `<output format>`: Specify the report format. Refer to the options in "GenerateReport" on page 105.
- `<output location>`: The location to write the report.
- `-includeSrcBefore <n>`: Optional. The number of lines of source code to include before each finding.
- `-includeSrcAfter <n>`: Optional. The number of lines of source code to include after each finding.
- `-scanconfig <scan_configuration_name>`: Optional. Specify the name of a scan configuration to use for the scan. If a scan configuration is not specified, the default scan configuration will be used for the scan.
- `-quality <path_file>`: Optional. To scan for quality issues (in addition to security vulnerabilities), include the `quality` argument. When specified, you must include the path and file name of one of these files:
  - Quality rule (`.quality`) file:

    Quality rule files are automatically created when you configure quality scanning in the AppScan Source for Development Eclipse Plug-in. When you

create a quality rule configuration, the quality rule file for it is saved to <user_home>\.ouncequality\rulesets\<quality_rule_config_name>.quality, where:

- <user_home> is your operating system home directory (for example, on Windows, the directory might be C:\Documents and Settings\Administrator\).
- <quality_rule_config_name> is the name that you gave the quality rule configuration when creating it in the AppScan Source for Development Quality Rule Configuration dialog box.

AppScan Source includes three default quality rule files:

- codereview_basic_cpp.quality: Quality rule configuration that includes recommended C/C++ Code Review rules.
- codereview_basic_java.quality: Quality rule configuration that includes recommended Java Code Review rules (designed to generate only high-severity findings).
- codereview_extended_java.quality: This rule file includes the same rules that are found incodereview_basic_java.quality, along with:
  - Additional Java code review rules, including some that produce findings of lower severity.
  - Java data flow analysis rules.

The default rule files are located in <install_dir>\quality\rulesets (where <install_dir> is the location of your AppScan Source installation). The provided default rule files cannot be modified.

To learn more about quality rule files, see the AppScan Source for Development Eclipse Plug-in infocenter.

– Quality rule argument script file: Text file that contains "Quality scan script arguments" on page 23. This option allows you to fine tune your scan by including/excluding files - and specifying C/C++ headers, macros, and other arguments.

• qualityonly <path_file>: Optional. To scan for quality issues only, include the qualityonly argument. When specified, you must include the path and file name of one of these files:

– Quality rule (.quality) file:

Quality rule files are automatically created when you configure quality scanning in the AppScan Source for Development Eclipse Plug-in. When you create a quality rule configuration, the quality rule file for it is saved to <user_home>\.ouncequality\rulesets\<quality_rule_config_name>.quality, where:

- <user_home> is your operating system home directory (for example, on Windows, the directory might be C:\Documents and Settings\Administrator\).
- <quality_rule_config_name> is the name that you gave the quality rule configuration when creating it in the AppScan Source for Development Quality Rule Configuration dialog box.

AppScan Source includes three default quality rule files:

- codereview_basic_cpp.quality: Quality rule configuration that includes recommended C/C++ Code Review rules.
- codereview_basic_java.quality: Quality rule configuration that includes recommended Java Code Review rules (designed to generate only high-severity findings).

- codereview_extended_java.quality: This rule file includes the same rules that are found incodereview_basic_java.quality, along with:
  - Additional Java code review rules, including some that produce findings of lower severity.
  - Java data flow analysis rules.

  The default rule files are located in <install_dir>\quality\rulesets (where <install_dir> is the location of your AppScan Source installation). The provided default rule files cannot be modified.

  To learn more about quality rule files, see the AppScan Source for Development Eclipse Plug-in infocenter.

– Quality rule argument script file: Text file that contains "Quality scan script arguments" on page 23. This option allows you to fine tune your scan by including/excluding files - and specifying C/C++ headers, macros, and other arguments.

Note: If you are using AppScan Source for Automation on Linux, you may encounter this error in the CLI output log:

```
Xlib: connection to ":1.0" refused by server
Xlib: No protocol specified
(Eclipse:13598): GLib-GObject-WARNING **:
   invalid (NULL) pointer instance
```

This is occurs when the AppScan Source for Automation user account does not have permission to conduct quality scanning. To correct this:

- From a terminal, issue the command: xauth list $DISPLAY

  This command will return the cookie that is used for the current display (for example, my_hostname/unix:1 MIT-MAGIC-COOKIE-1 3968070ad5056a979d8e6a3a81017d5c).

  Copy the cookie to the clipboard.
- Use the su command to switch to the AppScan Source for Automation user.
- Issue the command, xauth add <identifier>, where <identifier> is the value that was copied to the clipboard in Step 1.

Return Value

The Request ID if successful, or -1 if the request submission was unsuccessful.

Examples

- To scan the WebGoat application, publish it, and annotate the log with John Smith as the caller:

  ```
  ounceauto scanapplication -application_file C:\WebGoat\WebGoat.paf
  -publish -caller JohnSmith
  ```
- To scan the WebGoat application and create a Findings report in the C:\WebGoat directory:

  ```
  ounceauto scanapplication -application WebGoat
  -report Findings html C:\WebGoat\MyReport.html
  ```
- To conduct a scan that only looks for quality issues in the WebGoat application (using the C:\shared\rules.quality quality rule file):

  ```
  ounceauto scanapplication -application_file C:\WebGoat\WebGoat.paf
  -qualityonly C:\shared\rules.quality
  ```

Quality scan script arguments

When launching a quality scan via the command line interface `scan` command or the AppScan Source for Automation `ScanApplication` command, you can pass quality rules to the command using a text file script that contains arguments listed in this help topic.

The script can be placed in a flat text file with any file extension except `.quality`. Valid arguments include:

```
-directory <path to source directory>
-includefile <files in directory to include in the scan>
-excludefile <files in directory to exclude from the scan>
-rulefile <path to rules file>
-bindirectory <path to the binary directory that contains the .class files>
-javacp <fully-qualified build classpath>
-cppheaders <path to C++ headers file>
-cppmacros <path to C++ macros file>
```

See the following table for details about each of these arguments:

Table 3. Quality scan script arguments

Parameter	Classification	Description
`-directory`	Mandatory	Specify the fully-qualified path to the directory or directories that contain the source for the application or project that is being scanned. Use a comma-separated list to specify multiple directories.
`-includefile`	Optional for the `-directory` option	Specify the fully-qualified path to a text file that contains a line-separated list of source files to include in the scan. Instead of scanning all the source files in the source directory (`-directory`), this parameter enables you to scan only those files that are listed in the text file. In the text file, the file name and path for each source file must be relative to the source directory (not fully-qualified). Enter each source file name and path on a separate line in the file.
`-excludefile`	Optional for the `-directory` option	Specify the fully-qualified path to a text file that contains a line-separated list of source files to exclude from the scan. Instead of scanning all the source files in the source directory (`-directory`), this parameter enables you to identify the files in the source directory that you do not want to scan. In the text file, the file name and path for each source file must be relative to the source directory (not fully-qualified). Enter each source file name and path on a separate line in the file.

Table 3. Quality scan script arguments (continued)

Parameter	Classification	Description
`-rulefile`	Mandatory	Specify the fully-qualified path to the `.quality` rule file that contains the rule set to use to scan the source files. These are automatically created when you configure quality scanning in the AppScan Source for Development Eclipse Plug-in. When you create a quality rule configuration, the quality rule file for it is saved to `<user_home>\.ouncequality\rulesets\<quality_rule_config_name>.quality`, where: • `<user_home>` is your operating system home directory (for example, on Windows, the directory might be `C:\Documents and Settings\Administrator\`). • `<quality_rule_config_name>` is the name that you gave the quality rule configuration when creating it in the AppScan Source for Development Quality Rule Configuration dialog box. AppScan Source includes two default quality rule files: • `critical_rules_cpp.quality`: Quality rule configuration that includes recommended C/C++ Code Review rules. • `critical_rules_java.quality`: Quality rule configuration that includes recommended Java code review rules. The default rule files are located in `<install_dir>\quality\rulesets` (where `<install_dir>` is the location of your AppScan Source installation). The provided default rule files cannot be modified. To learn more about quality rule files, see the AppScan Source for Development Eclipse Plug-in infocenter.
`-bindirectory`	Mandatory for Java data flow analysis only	Specify the fully-qualified path to the binary directory that contains the `.class` files for the project or application that you are scanning. This parameter only applies to Java data flow analysis.
`-javacp`	Mandatory for Java data flow analysis only	Specify the fully-qualified build class path for your project or application. A build class path is specified for each project or application. To scan source code in multiple projects, separate jar files or directories with semicolons as shown in these examples. One project: `-javacp C:\codeAnalysis\jdk6_02_backup\jre\lib` Two projects:`-javacp C:\codeAnalysis\jdk6_02_backup\jre\lib;c:\swt` This parameter only applies to Java data flow analysis.
`-cppheaders`	Optional	Specify the fully-qualified path to a text file that contains custom C/C++ headers. In this file, each header should be on its own line. This parameter only applies to C/C++ data flow analysis.

Table 3. Quality scan script arguments (continued)

Parameter	Classification	Description
-cppmacros	Optional	Specify the fully-qualified path to a text file that contains custom C/C++ definitions. In this file, each macro definition should be on its own line. This parameter only applies to C/C++ data flow analysis.

Using Ounceauto from the command line

The interface to AppScan Source for Automation, ounceauto, issues commands to the server specified by command line arguments, prints the output, if any, and returns. Each request is assigned a Request ID, and all log entries regarding the request contain the Request ID for identification purposes.

Run ounceauto from the command line as follows:

```
ounceauto <command name> <command arguments>
```

Ounceauto commands include:

- GenerateReport
- PublishAssessment
- PublishAssessmentASE
- ScanApplication
- Wait

GenerateReport

Description

Create a report from an assessment.

Syntax

```
ounceauto GenerateReport
-assessment <assessment path>
-type <report type>
-output <output format>
-file <output location>
[-caller <caller>]
[-includeSrcBefore <n>]
[-includeSrcAfter <n>]
```

- -assessment <assessment path>: Path to the assessment file for which you want to generate the report.
- -type <report type>: Name of the report type. Report types consist of Findings reports, AppScan Source reports, and custom reports.

  AppScan Source report types include:

  – A Findings report:
    - Findings
    - Findings by Type
    - Findings by Classification
    - Findings by File
    - Findings by API
    - Findings by Bundle

- Findings by CWE
- DTS Activity

– An AppScan Source report:
- DISA Application Security and Development STIG V3 R5
- OWASP Top 10 2010
- OWASP Top 10 2007
- Software Security Profile
- PCI Data Security Standard V2.0
- PCI Data Security Standard V1.1

– A custom report, if available.

- `-output <output format>` : Specify one of these formats for this report,
  – `html`: Generates the report as HTML and displays it online.
  – `zip`: Creates a ZIP file that contains all HTML report components
  – For reports in PDF format, you can specify the level of detail:
    - `pdf-summary`: Contains counts for each custom report group
    - `pdf-detailed`: Contains counts for each API for each vulnerability property
    - `pdf-comprehensive`: Contains tables consisting of every finding for every API
    - `pdf-annotated`: Contains all findings, any notes included with the findings, and designated code snippets
    - `output location`: The file path to write the report.
- `-file <output location>`: Specify the path and file name to which you want to save the report.
- `-caller <caller>`: Optional. Assign a caller to the report generation operation. The caller can be the name of an actual user, but this is not required. The caller name is written to the `ounceauto` log file.
- `-includeSrcBefore <n>`: Optional. The number of lines of source code to include before each finding.
- `-includeSrcAfter <n>`: Optional. The number of lines of source code to include after each finding.

Return Value

The Request ID if successful, or -1 if the request submission was unsuccessful.

Examples

- To generate a Findings by API report as an HTML file:

```
ounceauto generatereport -assessment C:\Ounce\Data\Webgoat.ozasmt
-type "Findings by API" -output html
-file C:\reports\Webgoat_Findings.html
```

- To generate an OWASP Top 10 AppScan Source report as a PDF:

```
ounceauto generatereport -assessment C:\Ounce\Data\Webgoat.ozasmt
-type "OWASP Top 10" -output pdf-annotated
-file C:\Reports\Webgoat_OWASP.pdf
```

PublishAssessment

Description

Publish the selected assessment to the AppScan Source database.

Syntax

```
ounceauto PublishAssessment - file <assessment.ozasmt>
[-caller <caller>]
```

- `-file <assessment file>`: Full path to an assessment file.
- `-caller <caller>`: Optional. Assign a caller to the report generation operation. The caller can be the name of an actual user, but this is not required. The caller name is written to the ounceauto log file.

Return Value

The Request ID if successful, or -1 if the request submission was unsuccessful.

Example

To publish the `WebGoat_Internal` assessment:

```
ounceauto publishassessment -file C:\Ounce\Data\WebGoat_Internal.ozasmt
```

PublishAssessmentASE

Description

Publish the selected assessment to AppScan Enterprise.

Syntax

```
ounceauto PublishAssessmentASE -file <assessment.ozasmt>
[-caller <caller>] [-folderid <server_folder>]
```

- `-file <assessment file>`: Full path to an assessment file.
- `-caller <caller>`: Optional. Assign a caller to the report generation operation. The caller can be the name of an actual user, but this is not required. The caller name is written to the `ounceauto` log file.
- `-folderid <server_folder>`: Optional. Enterprise Console folder to publish to. If this argument is not used, the assessment will be published to your default Enterprise Console folder.

Return Value

The Request ID if successful, or -1 if the request submission was unsuccessful.

Example

To publish the `WebGoat_Internal` assessment:

```
ounceauto publishassessmentase -file C:\Ounce\Data\WebGoat_Internal.ozasmt
```

ScanApplication

Description

Scan the specified application. Optionally save the assessment or generate an assessment report.

Syntax

```
ounceauto ScanApplication
<-application <application_name>|
 -application_file <application file>>
```

```
[-name <assessment name>]
[-save <filename>]
[-caller <caller>]
[-publish]
[-report <report type> <output format>
<output location>]
[-includeSrcBefore <n>
[-includeSrcAfter <n>]
[-scanconfig <scan_configuration_name>][-quality <path_file>]
[-qualityonly <path_file>]
```

- `-application <application name>`: The name of the application to scan.
- `-application_file <path to AppScan Source Application file>`: The path to an AppScan Source application file.
- `-name <assessment name>`: Optional. A name for the assessment.
- `-save <filename>`: Optional. Save the assessment results to this file.
- `-caller <caller>`: Optional. Assign a caller to the report generation operation. The caller can be the name of an actual user, but this is not required. The caller name is written to the ounceauto log file.
- `-publish`: Optional. Publish the assessment after scanning.
- `-report`: Optional. Generate a report after the scan.
- `<report type>`: Name of the report type. The report types consist of Findings reports, AppScan Source reports, and custom reports. Refer to the options in "GenerateReport" on page 105.
- `<output format>`: Specify the report format. Refer to the options in "GenerateReport" on page 105.
- `<output location>`: The location to write the report.
- `-includeSrcBefore <n>`: Optional. The number of lines of source code to include before each finding.
- `-includeSrcAfter <n>`: Optional. The number of lines of source code to include after each finding.
- `-scanconfig <scan_configuration_name>`: Optional. Specify the name of a scan configuration to use for the scan. If a scan configuration is not specified, the default scan configuration will be used for the scan.
- `-quality <path_file>`: Optional. To scan for quality issues (in addition to security vulnerabilities), include the `quality` argument. When specified, you must include the path and file name of one of these files:
  - Quality rule (`.quality`) file:

    Quality rule files are automatically created when you configure quality scanning in the AppScan Source for Development Eclipse Plug-in. When you create a quality rule configuration, the quality rule file for it is saved to `<user_home>\.ouncequality\rulesets\<quality_rule_config_name>.quality`, where:

    - `<user_home>` is your operating system home directory (for example, on Windows, the directory might be `C:\Documents and Settings\Administrator\`).
    - `<quality_rule_config_name>` is the name that you gave the quality rule configuration when creating it in the AppScan Source for Development Quality Rule Configuration dialog box.

    AppScan Source includes three default quality rule files:

    - `codereview_basic_cpp.quality`: Quality rule configuration that includes recommended C/C++ Code Review rules.

- codereview_basic_java.quality: Quality rule configuration that includes recommended Java Code Review rules (designed to generate only high-severity findings).
- codereview_extended_java.quality: This rule file includes the same rules that are found incodereview_basic_java.quality, along with:
  - Additional Java code review rules, including some that produce findings of lower severity.
  - Java data flow analysis rules.

The default rule files are located in <install_dir>\quality\rulesets (where <install_dir> is the location of your AppScan Source installation). The provided default rule files cannot be modified.

To learn more about quality rule files, see the AppScan Source for Development Eclipse Plug-in infocenter.

– Quality rule argument script file: Text file that contains "Quality scan script arguments" on page 23. This option allows you to fine tune your scan by including/excluding files - and specifying C/C++ headers, macros, and other arguments.

• qualityonly <path_file>: Optional. To scan for quality issues only, include the qualityonly argument. When specified, you must include the path and file name of one of these files:

– Quality rule (.quality) file:

Quality rule files are automatically created when you configure quality scanning in the AppScan Source for Development Eclipse Plug-in. When you create a quality rule configuration, the quality rule file for it is saved to <user_home>\.ouncequality\rulesets\<quality_rule_config_name>.quality, where:

- <user_home> is your operating system home directory (for example, on Windows, the directory might be C:\Documents and Settings\Administrator\).
- <quality_rule_config_name> is the name that you gave the quality rule configuration when creating it in the AppScan Source for Development Quality Rule Configuration dialog box.

AppScan Source includes three default quality rule files:

- codereview_basic_cpp.quality: Quality rule configuration that includes recommended C/C++ Code Review rules.
- codereview_basic_java.quality: Quality rule configuration that includes recommended Java Code Review rules (designed to generate only high-severity findings).
- codereview_extended_java.quality: This rule file includes the same rules that are found incodereview_basic_java.quality, along with:
  - Additional Java code review rules, including some that produce findings of lower severity.
  - Java data flow analysis rules.

The default rule files are located in <install_dir>\quality\rulesets (where <install_dir> is the location of your AppScan Source installation). The provided default rule files cannot be modified.

To learn more about quality rule files, see the AppScan Source for Development Eclipse Plug-in infocenter.

– Quality rule argument script file: Text file that contains "Quality scan script arguments" on page 23. This option allows you to fine tune your scan by including/excluding files - and specifying C/C++ headers, macros, and other arguments.

Note: If you are using AppScan Source for Automation on Linux, you may encounter this error in the CLI output log:

```
Xlib: connection to ":1.0" refused by server
Xlib: No protocol specified
(Eclipse:13598): GLib-GObject-WARNING **:
   invalid (NULL) pointer instance
```

This is occurs when the AppScan Source for Automation user account does not have permission to conduct quality scanning. To correct this:

- From a terminal, issue the command: `xauth list $DISPLAY`

  This command will return the cookie that is used for the current display (for example, `my_hostname/unix:1  MIT-MAGIC-COOKIE-1 3968070ad5056a979d8e6a3a81017d5c`).

  Copy the cookie to the clipboard.
- Use the `su` command to switch to the AppScan Source for Automation user.
- Issue the command, `xauth add <identifier>`, where `<identifier>` is the value that was copied to the clipboard in Step 1.

Return Value

The Request ID if successful, or -1 if the request submission was unsuccessful.

Examples

- To scan the WebGoat application, publish it, and annotate the log with John Smith as the caller:

  ```
  ounceauto scanapplication -application_file C:\WebGoat\WebGoat.paf
  -publish -caller JohnSmith
  ```
- To scan the WebGoat application and create a Findings report in the `C:\WebGoat directory`:

  ```
  ounceauto scanapplication -application WebGoat
  -report Findings html C:\WebGoat\MyReport.html
  ```
- To conduct a scan that only looks for quality issues in the `WebGoat` application (using the `C:\shared\rules.quality` quality rule file):

  ```
  ounceauto scanapplication -application_file C:\WebGoat\WebGoat.paf
  -qualityonly C:\shared\rules.quality
  ```

Wait

Description

Block until the specified request finishes.

Syntax

ounceauto wait -requestid <request_id>

`-requestid <request_id>`: The ID of the request to wait for.

Return Value

If the request corresponding to <request_id> completes successfully, the return value is 0, otherwise -1.

Example

The following Windows example illustrates a scan of an application file, and then waits for the scan to complete.

```
ounceauto scanapplication -application_file WG0.paf
ounceauto wait -requestid %errorlevel%
```

On Linux, the equivalent of this example is:

```
ounceauto scanapplication -application_file WG0.paf
ounceauto wait -requestid $?
```

Chapter 7. Framework for Frameworks handling APIs

AppScan Source provides a set of Java APIs that allow you to add support for frameworks that are used in your applications. The classes and methods offered in these APIs allow you to account for frameworks for which built-in support is not provided.

Note: AppScan Source includes built-in support for these frameworks:

- Apache Struts 1 and 2
- Spring MVC 2.5 and 3
- ASP .NET MVC (Windows only)
- Enterprise JavaBeans (EJB) 2
- ASP .NET (Windows only)
- J2EE
- JavaServer Faces (JSF) 2
- .NET 4.5 (Windows only)
- Jax - RS (V1.0 and V1.1)
- Jax - WS (V2.2)

Modern frameworks have moved a great deal of information that affects the run time behavior of applications out of normal source code and into configuration files and annotations. In the past, this resulted in blind spots during static analysis. Product teams could create custom rules for individual applications, however, no framework existed that could flexibly describe the activities of these frameworks in an automated way.

By using the Framework for Frameworks APIs, you can quickly and easily add support for new frameworks right in AppScan Source. This is accomplished by processing the frameworks' associated configuration information and providing that data back to AppScan Source through the associated APIs.

The Framework for Frameworks APIs are included with the installation of these products:

- AppScan Source for Automation
- AppScan Source for Analysis
- AppScan Source for Development

The APIs are installed to `<install_dir>\walalib` (where `<install_dir>` is the location of your AppScan Source installation).

An example project archive is installed in `<data_dir>\samples\F4FEjbExample.zip` (where `<data_dir>` is the location of your AppScan Source program data, as described in Chapter 9, “Installation and user data file locations,” on page 149).

Note: Trace nodes with class names that begin with `Appscan.Synthetic`, `Appscan.Synthetic.Validator`, and `AppScan.Synthetic.Replacement` correspond to methods that are synthesized by AppScan Source.

- `AppScan.Synthetic` methods are used to stitch traces together in application code that uses frameworks.

- An `AppScan.Synthetic.Validator` method models the underlying validation performed by the framework runtime. You can select a validator method and mark it as a **Validator** if needed.
- An `AppScan.Synthetic.Replacement` method indicates that a method in the application code was replaced by AppScan Source to capture data flow between disjoint components (such as controllers and views) of the framework.

Main Framework for Frameworks API components

F4FHandler

`F4FHandler` is the abstract class that all new framework handlers must extend. It enforces the workflow of a framework handler through abstract methods that must be overridden. It also provides access to the other two API components, `F4FApp` and `F4FAction`.

F4FApp

Provides information about the application being scanned, including information from its configuration (such as source roots and web root). This component also provides access to every class in the application for the purpose of determining necessary information, such as annotations and superclass.

F4FAction

Provides information to AppScan Source about the application being scanned. Entry points, connections between abstract methods and their implementations, and more can be described to the scanning engine by calling the methods provided in this API.

Using the Framework for Frameworks APIs

This section highlights the steps that are required when using the Framework for Frameworks APIs. Where appropriate, these steps will be related to the provided example, allowing you to follow along.

In this example, we will:

- Create a class that extends `F4FHandler`
- Implement `F4FHandler`'s two abstract methods
- Create a Java archive (`.jar`) with a manifest file that identifies your handler class
- Export the `.jar` to the AppScan Source `waflgens` directory
- "About the example" on page 115
- "Importing the example project into Eclipse or Rational Application Developer for WebSphere Software (RAD)" on page 115
- "Creating a class that implements `F4FHandler`" on page 118
- "Creating a manifest file for your handler" on page 119
- "Creating a JAR for your handler" on page 119
- "Exporting the JAR to `waflgens`" on page 122

About the example

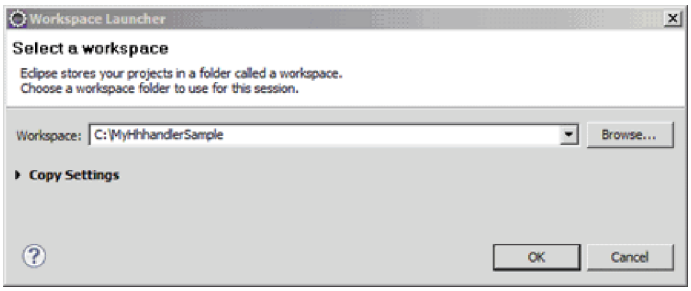
Included with your AppScan Source installation is a sample handler that reads and provides Framework for Frameworks information about Enterprise Java Bean (EJB) 2 applications (also referred to as Enterprise Java Beans, or just Beans).

EJB is a framework that attempts to make it easier to work with and reuse business logic components. Each bean represents a business component and must have associated interface classes that other Beans must use to interact with it. These interface classes are associated with the Bean through the application's EJB configuration file (`EJB-jar.xml`). This creates a blind spot, as the static analysis engine will not pick up those associations in the source, which makes the EJB framework particularly suited to be handled through Framework for Frameworks.

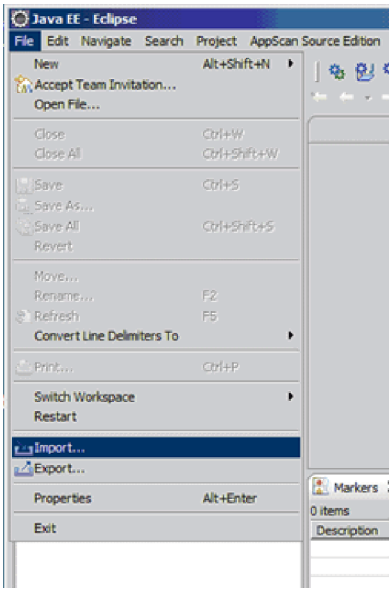
Importing the example project into Eclipse or Rational Application Developer for WebSphere Software (RAD)

Procedure

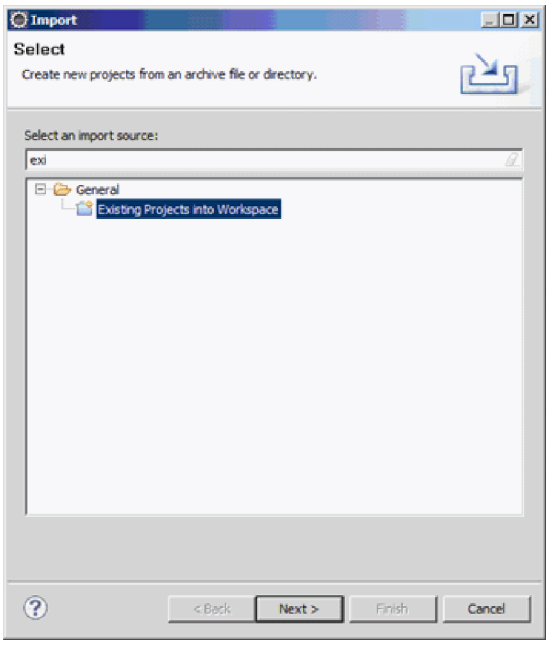
1. Start Eclipse.
2. Create a new workspace for your handler project.



3. Select **File** > **Import** from the main workbench menu.



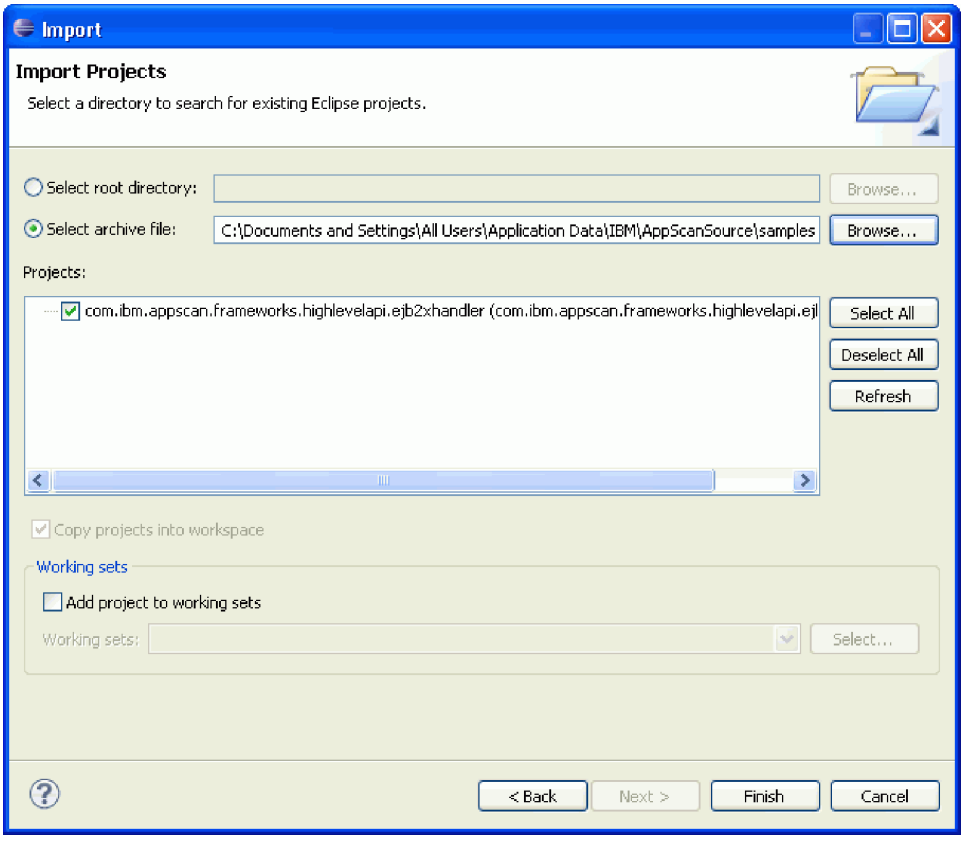
4. In the Import wizard **Select** page, select **Existing Projects Into Workspace** and then click **Next**.



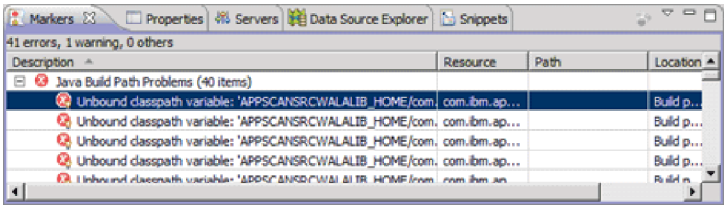
5. Select the **Select archive file** radio button and then click **Browse**. In the **Select archive containing the projects to import** dialog box, locate `<data_dir>\samples\F4FEjbExample.zip` (where `<data_dir>` is the location of your AppScan Source program data, as described in Chapter 9, "Installation and user data file locations," on page 149) and then click **Open**.

   This zip file is an Eclipse archive that contains the configuration and source code for the `Ejb2xHandler` framework handler example.

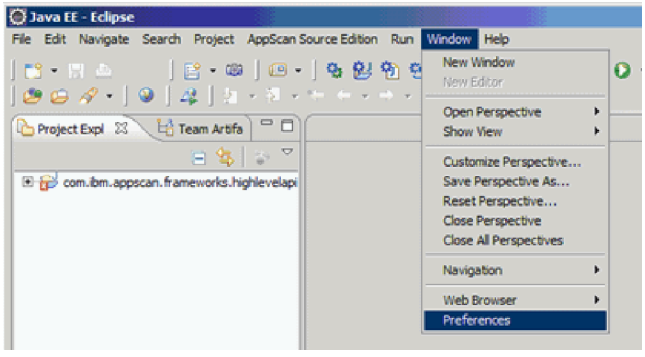
   Click **Finish** to import the archive.



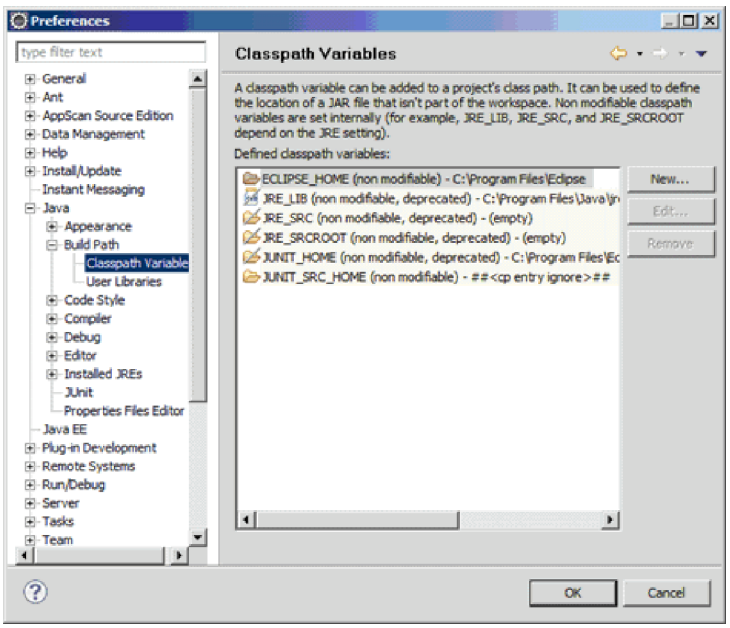
6. After the archive has been imported, you will notice a number of build errors. The example has a class path variable that points to the directory holding its libraries.



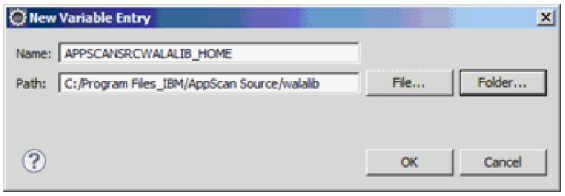
7. Next, you will define a variable. Select **Window** > **Preferences** from the main workbench menu.



8. In the Preferences dialog box, select **Java** > **Build Path** > **Classpath Variables** to open the Classpath Variables preference page. Click **New**.



9. Create a new variable named `APPSCANSRCWALALIB_HOME` and set the **Path** to `<install_dir>\walalib` (where `<install_dir>` is the location of your AppScan Source installation).



After clicking **OK** in all dialog boxes, the sample should build completely with no errors.

Creating a class that implements `F4FHandler`

To create a new framework handler, you must first create a class that implements `F4FHandler`. Two methods must be implemented to support the Framework for Frameworks functionality.

Override `isApplicable`

Purpose: AppScan Source calls `isApplicable` to determine if it should run your handler. If you return `True`, it will run your handler by calling `handleApp`. If you return `False`, nothing further will be called.

Note: `isApplicable` includes a check at the beginning of the method to ensure the application is a Java application before continuing.

Observe in the example: In the `Ejb2xHandler`, `isApplicable` first checks to see if the language is appropriate (as EJB is only present in Java applications). If the application is Java-based, `isApplicable` then searches for any instances of `ejb-jar.xml`, which is the required configuration file for an EJB 2 application. If configuration files are found, they are read into the handler and `True` is returned to let AppScan Source know that it should call `handleApp` to deal with the information contained in the configuration files.

Override `handleApp`

Purpose: AppScan Source calls `handleApp` to allow you to determine and set information about the framework or frameworks in use by the current application being scanned. The parameters `F4FApp` and `F4FAction` are used to get information about the application and set specifics about how to handle the frameworks that are present and that your handler handles.

Creating a manifest file for your handler

Create a file named `Manifest.txt` to contain the information needed for your JAR's manifest when it is exported. Specifically, AppScan Source will look for the `Framework-Handler` entry when loading your handler JAR to determine which class contains the `F4FHandler` functionality.

Place these two lines in the `Manifest.txt` file:

```
Manifest-Version: 1.0
Framework-Handler: package.HandlerClassName
```

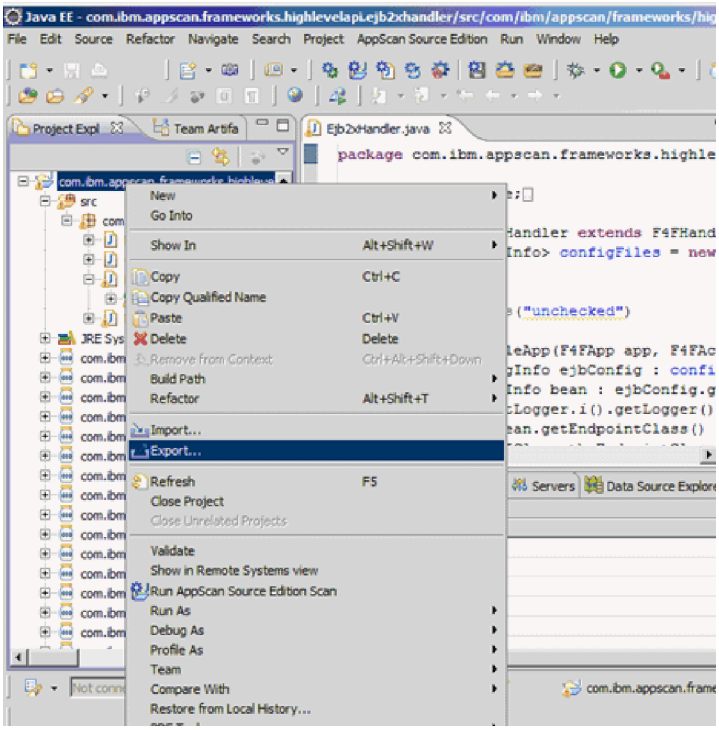
Where `package.HandlerClassName` is the full package and class name for your class that implements the `F4FHandler` interface.

Observe in the example: `Manifest.txt` contains the information that will be placed in the `Ejb2xHandler`'s JAR file manifest when it is created. There are only two lines, one listing the version and the other specifying the framework handling class, `Framework-Handler:`
`com.ibm.appscan.frameworks.highlevelapi.ejb2xhandler.Ejb2xHandler`.

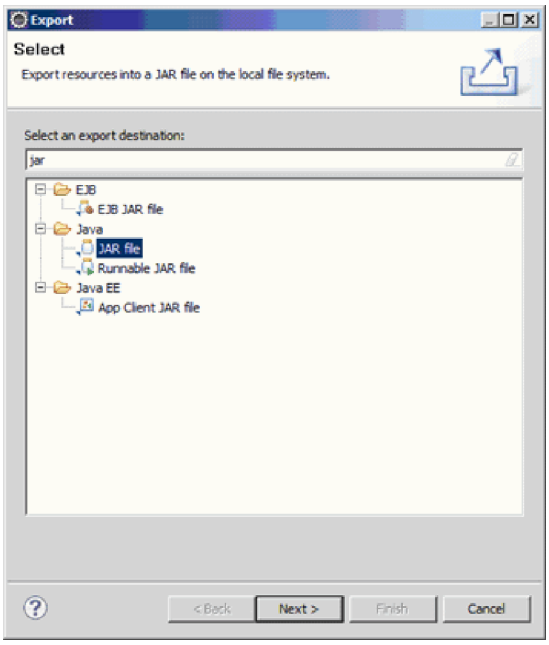
Creating a JAR for your handler

Procedure

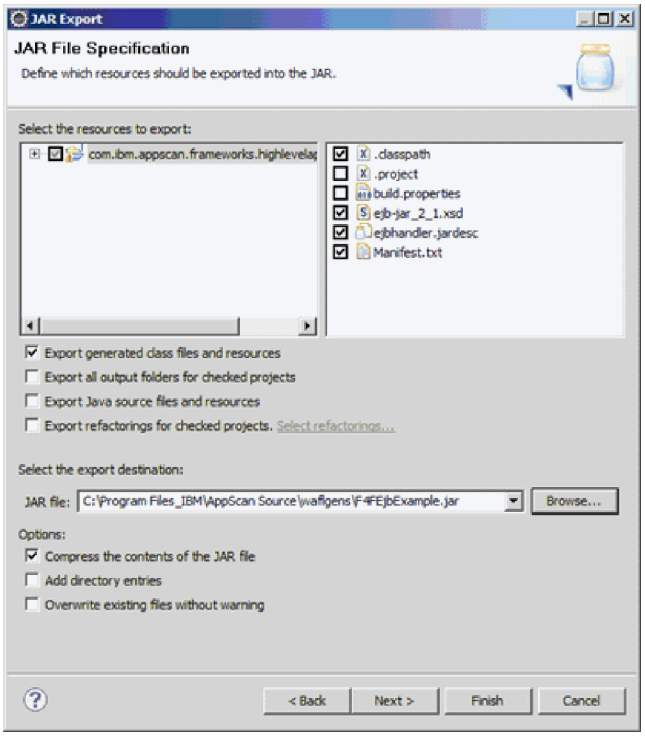
1. Right click your project in the Eclipse Project Explorer view and select **Export**.



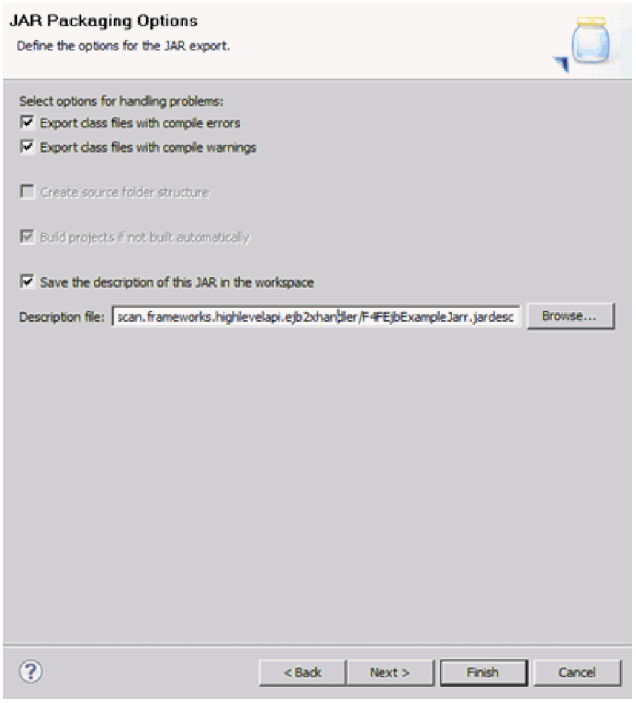
2. In the Export wizard, select **JAR file** and then click **Next**.



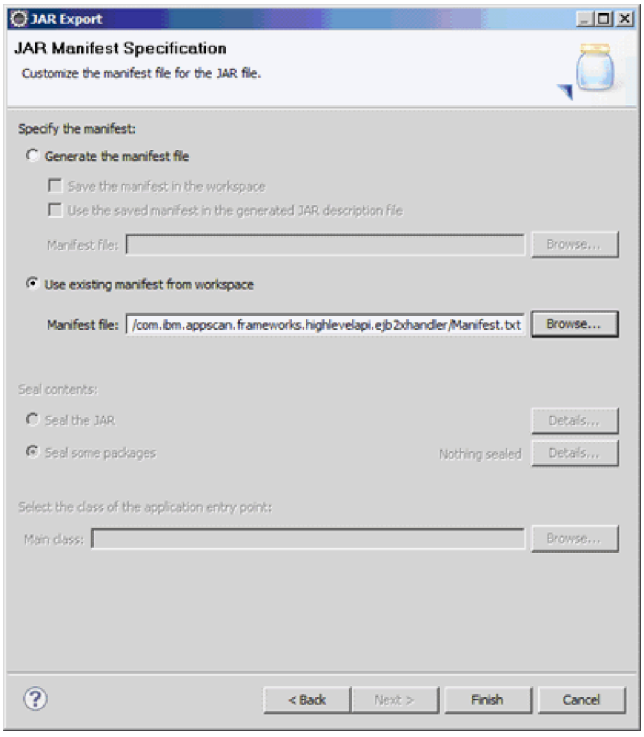
3. In the JAR Export **Select the export destination** section, select a suitable location for your JAR file and provide a name. Click **Next**.



4. Click **Next** again. Optionally, you can also set a location and name for a description file. This will save a `.jardesc` file into your project that will contain all JAR export settings from this export. With this, you can repeat the export without specifying the settings again.



5. Select the **Use existing manifest from workspace** radio button click **Browse**. Select the `Manifest.txt` file you created earlier.



6. Click **Finish**

Exporting the JAR to `waflgens`

If you did not export the handler JAR directly to the `waflgens` directory during its creation, copy it over now. The full path is `<install_dir>\waflgens` (where `<install_dir>` is the location of your AppScan Source installation).

Common actions performed by the handler

Create a web service entry point

Many frameworks provide their own entry points into an application. A common example is to expose web services that are either identified in a configuration file or in annotations in the code. After searching in the application's configuration files or directly in the bytecode for designated entry points, the method `F4FAction.addTaintedCallback` can be used to create a tainted data entry point at the appropriate method.

Observe in the example: In EJB 2, web service entry points are declared by defining *endpoints* in the application's configuration file (`ejb-jar.xml`). Then `handleApp` loops through the beans declared in `ejb-jar.xml` and whenever an endpoint class is defined, it obtains the list of method names. It then declares their implementations as web service entry points using the `addTaintedCallback` method.

Replace a method

Modern frameworks frequently make use of virtual functions and abstraction to more loosely couple business components. While this can be an improvement to the development process, it creates difficulties for static analysis when the connection between virtual function and its implementation is handled in a configuration file or via annotations in the code. `F4FAction.replaceCalls` allows a handler to designate these connections.

Observe in the example: In EJB 2, each bean has a set of interfaces (local and remote) that declare how other beans may interact with it. This means that, wherever a bean's interface `class.method` is called, it is replaced by the framework with the actual `ImplementationClass.method`.

Starting at line 62, our example handler loops through each bean and takes its remote and local interfaces and replaces them with their actual implementations.

Logging

A handler can use the `com.ibm.wala.andromeda.util.logging.TaintLogger` class to log informative messages during execution - and to cause error messages to appear in the AppScan Source user interface. The `TaintLogger` class employs the `log4j` library. To log a message, first obtain a `Logger` object by calling `TaintLogger.i().getLogger()`. Then, invoke logging methods on the `Logger` (for example, `Logger.warn`) to log the messages that you want. Log messages will appear in `<data_dir>\logs\scanner_exceptions.log` (where `<data_dir>` is the location of your AppScan Source program data, as described in Chapter 9, "Installation and user data file locations," on page 149). If `Logger.error` or `Logger.fatal` is used to log a message, the error the message will also appear in the Console view in the AppScan Source user interface.

Framework for Frameworks API classes and methods

This section contains the Framework for Frameworks API classes and methods, abbreviated for Adobe PDF. To access the complete set of API documentation, launch the AppScan Source for Analysis online help and navigate to Reference > Framework for Frameworks handling APIs > Framework for Frameworks API classes and methods.

F4FActions

```
java.lang.Object
  extended by com.ibm.appscan.frameworks.highlevelapi.F4FActions

public class F4FActions
extends java.lang.Object
```

Class for specifying how the application's framework constructs should be modeled. An `F4FHandler` mutates the `F4FAction` object passed to `F4FHandler.handleApp(F4FApp, F4FActions)` as it analyzes the application.

Constructor Detail

`F4FActions`

```
public F4FActions()
```

Create an empty `F4FActions` object. Should not be needed for implementing a new framework handler, as the relevant `F4FActions` object will be passed to `F4FHandler.handleApp(F4FApp, F4FActions)`.

addTaintedCallback

```
public void addTaintedCallback(IMethod method,
                               int numParams)
```

Same as "addTaintedCallback" `(String, int)`, but takes an `IMethod` directly rather than a VDB signature

addTaintedCallback

```
public void addTaintedCallback(java.lang.String vdbMethodSig,
                               int numParams)
```

Make a method a tainted callback, with all parameters tainted.

Note: For .NET apps, we need fully-qualified VDB signatures. So, instead of `int` as a parameter type, we need `System.Int32`, etc. To see the full mapping from fully-qualified names to the names usually used in VDB, see `DotNetVDBUtil.systemName2VDBShortName`.

Parameters:

- `vdbMethodSig` - the signature of the `callback` method
- `numParams` - the number of parameters for the `callback` method, including the `this` parameter

replaceCalls

```
public void replaceCalls(java.lang.String oldVDBSig,
                         java.lang.String newVDBSig)
```

Replace all calls to one method with calls to another method. We require that the descriptors for the old and new method (i.e., the number of arguments, argument type, and return type) are identical.

Note: replacement will only occur when `oldVDBSig` is the _declared_ target at a call site. So, if `oldVDBSig` is `Integer.toString()`, and we see a call to `Object.toString()`, we will _not_ perform a replacement at that call site, even though it might invoke `Integer.toString()`.

Note: for .NET apps, we need fully-qualified VDB signatures. So, instead of `int` as a parameter type, we need `System.Int32`, etc. To see the full mapping from fully-qualified names to the names usually used in VDB, see `DotNetVDBUtil.systemName2VDBShortName`

Parameters:

- `oldVDBSig` - signature of method whose calls should be replaced
- `newVDBSig` - signature of method to replace calls with

replaceCallsWithSyntheticExpr

```
public void replaceCallsWithSyntheticExpr(java.lang.String vdbSig,
    com.ibm.appscan.frameworks.specinfo.SyntheticExpr expr)
```

Replace all calls to a method with an arbitrary WAFL `SyntheticExpr`. For example, one could replace calls with an assignment via an `AssignmentExpr`.

Note: replacement will only occur when `oldVDBSig` is the _declared_ target at a call site. So, if `oldVDBSig` is `Integer.toString()`, and we see a call to `Object.toString()`, we will _not_ perform a replacement at that call site, even though it might invoke `Integer.toString()`.

Note: for .NET apps, we need fully-qualified VDB signatures. So, instead of `int` as a parameter type, we need `System.Int32`, etc. To see the full mapping from fully-qualified names to the names usually used in VDB, see `DotNetVDBUtil.systemName2VDBShortName`

Parameters:

- `vdbSig` - signature of method whose calls should be replaced
- `expr` - synthetic expression to replace calls with

replaceCallsWithParamPattern

```
public void replaceCallsWithParamPattern(java.lang.String oldVDBSig,
                                         java.util.Map<java.lang.String,
                                         java.util.Map<java.lang.Integer,
                                         java.util.regex.Pattern>>
                                         newSig2Pattern)
```

Replace calls to one method with calls to another method only if the parameters of `String` type are constants meeting specified patterns. We require that the descriptors for the old and new method (i.e., the number of arguments, argument type, and return type) are identical.

Note: replacement will only occur when `oldVDBSig` is the _declared_ target at a call site. So, if `oldVDBSig` is `Integer.toString()`, and we see a call to `Object.toString()`, we will _not_ perform a replacement at that call site, even though it might invoke `Integer.toString()`.

Note: for .NET apps, we need fully-qualified VDB signatures. So, instead of `int` as a parameter type, we need `System.Int32`, etc. To see the full mapping from fully-qualified names to the names usually used in VDB, see `DotNetVDBUtil.systemName2VDBShortName`

Parameters:

- `oldVDBSig` - signature of method whose calls should be replaced
- `newSig2Pattern` - maps VDB signature of each possible replacement method `m` to a map `M` from integer parameter positions to `Patterns`. If the string constant parameters in the appropriate positions match the patterns in `M` at some call site, a replacement to `m` will be performed.

addFrameworkInfo

```
public void addFrameworkInfo
(com.ibm.appscan.frameworks.specinfo.IFrameworkInfo info)
```

Add arbitrary additional framework info. This method should only be needed for rare cases where the other APIs provided are insufficient.

addTaintedCallback

```
public void addTaintedCallback(java.lang.String vdbMethodSig,
                               java.util.Collection<TaintedParam>
                               taintedParams)
```

Make some method a tainted callback, with only certain parameter access paths being treated as tainted.

Note: for .NET apps, we need fully-qualified VDB signatures. So, instead of `int` as a parameter type, we need `System.Int32`, etc. To see the full mapping from fully-qualified names to the names usually used in VDB, see `DotNetVDBUtil.systemName2VDBShortName`

Parameters:

- `vdbMethodSig` - the signature of the callback method, in VDB format
- `taintedParams` - information on which parameter access paths should be tainted

addHighLevelSyntheticMethod

```
public void addHighLevelSyntheticMethod(HighLevelSyntheticMethod m)
```

equivalent to `addHighLevelSyntheticMethod(m, true)`

addHighLevelSyntheticMethod

```
public void addHighLevelSyntheticMethod(HighLevelSyntheticMethod m,
                                        boolean isEntrypoint)
```

Add a high-level synthetic method. A corresponding WAFL synthetic method (possibly an entrypoint) will be generated.

Parameters:

- `m` - the method

- `isEntrypoint` - should the method be marked as an entrypoint in WAFL?

createGlobal

```
public Global createGlobal(java.lang.String name,
                           java.lang.String declaredVDBType,
                           boolean isEntrypointScoped)
```

Create a new global that can be accessed from `HighLevelSyntheticMethods`.

Parameters:

- `name` - name for the global
- `declaredVDBType` - the declared type of the global (e.g., `java.lang.String`).

  Note: for .NET apps, we need a fully-qualified VDB type. So, instead of `int` as a parameter type, we need `System.Int32`, etc. To see the full mapping from fully-qualified names to the names usually used in VDB, see `DotNetVDBUtil.systemName2VDBShortName`
- `isEntrypointScoped` - if true, the global is scoped to a single entrypoint (i.e., it is request-scoped). Otherwise, the global is scoped across entrypoints (i.e., it is "session" or "application" scoped)

Returns:

- a Global object, which can be read/written inside a `HighLevelSyntheticMethod`

createGlobal

```
public Global createGlobal(java.lang.String name,
                           IClass declaredClass,
                           boolean isEntrypointScoped)
```

Just like “createGlobal”`(String, String, boolean)`, but takes an `IClass` for the declared type instead of a type name

getGlobals

```
public java.util.Collection<Global> getGlobals()
```

For internal usage.

getAdditionalFrameworkInfo

```
public java.util.Collection
<com.ibm.appscan.frameworks.specinfo.IFrameworkInfo>
getAdditionalFrameworkInfo()
```

For internal usage.

getCallReplacement2SigsInfo

```
public java.util.Map
<java.lang.String,java.util.Map
<java.lang.String,java.util.Map
<java.lang.Integer,java.util.regex.Pattern>>>
getCallReplacement2SigsInfo()
```

For internal usage.

getCallReplacement2ExprInfo

```
public java.util.Map
<java.lang.String,com.ibm.appscan.frameworks.specinfo.SyntheticExpr>
getCallReplacement2ExprInfo()
```

For internal usage.

getCallback2TaintedParams

```
public java.util.Map
<java.lang.String,java.util.Collection<TaintedParam>>
getCallback2TaintedParams()
```

For internal usage.

getHighLevelSyntheticMethods

```
public java.util.List
<com.ibm.wala.util.collections.Pair
<HighLevelSyntheticMethod,java.lang.Boolean>>
getHighLevelSyntheticMethods()
```

For internal usage.

toString

```
public java.lang.String toString()
```

Overrides:

- `toString` in class `java.lang.Object`

F4FApp

```
java.lang.Object
  extended by com.ibm.appscan.frameworks.highlevelapi.F4FApp

public class F4FApp
extends java.lang.Object
```

Representation of an application, with methods to query various properties of classes, methods, etc. Implemented mostly by delegating to methods from the T.J. Watson Libraries for Analysis (WALA); the goal is to consolidate the most useful WALA methods in a single type. See the WALA home page (http://wala.sourceforge.net) for full details on WALA APIs.

Constructor Detail

```
public F4FApp(IClassHierarchy cha)
```

Should not be needed to implement a new handler. The relevant `F4FApp` object will be passed as a parameter to `F4FHandler.handleApp(F4FApp, F4FActions)`

getAppClass

```
@Deprecated
public IClass getAppClass(java.lang.String vdbClassName)
```

Deprecated. Use `getIClass(String)` instead; this method simply delegates to that one.

getIClass

```
public IClass getIClass(java.lang.String vdbClassName)
```

get the IClass for some class in the application, including library jars/DLLs. If no class with the provided name is found, return null

Parameters:

- vdbClassName - class name in VDB format, e.g., java.lang.String

getClassAnnotations

```
public java.util.Collection<Annotation>
getClassAnnotations(IClass klass)
```

Get the annotations/attributes for a class. For .NET, the result will include inherited attributes.

Parameters:

- klass - the class whose annotations are desired

getMethodAnnotations

```
public java.util.Collection<Annotation>
getMethodAnnotations(IMethod method)
```

Get the annotations / attributes for a method. For .NET, these will include inherited attributes.

Parameters:

- method - the method whose annotations are desired

getFieldAnnotations

```
public java.util.Collection<Annotation>
getFieldAnnotations(IField field)
```

Get the annotations / attributes for a field.

Parameters:

- field - the field whose annotations are desired

getMethodParametersAnnotations

```
public java.util.Collection<Annotation>[]
getMethodParametersAnnotations(IMethod method)
```

Get annotations on parameters as an array of Collections, where each array element gives the annotations on the corresponding parameter. Note that the this parameter for an instance method cannot have annotations.

Parameters:

- method - the method whose parameter annotations are desired

getAllApplicationClasses

```
public java.util.Collection<IClass>
getAllApplicationClasses()
```

Get all the classes in the application (i.e., excluding those in library jars).

getClassHierarchy

```
public IClassHierarchy getClassHierarchy()
```

Get the WALA class hierarchy for the application. Most handlers should be able to work via the other methods in this class and shouldn't need to operate directly on the class hierarchy. But, access is provided for advanced use.

getMethodsDeclaredInClass

```
public java.util.Collection<IMethod>
getMethodsDeclaredInClass(IClass klass)
```

Get all the static and instance methods declared in `klass`

getClassMethods

```
public java.util.Collection<IMethod>
getClassMethods(java.lang.String className,
                java.lang.String methodName)
```

Get all the methods in an class with a particular name. If the class cannot be found, returns an empty Collection.

Parameters:

- `className` - the class name in VDB (i.e., source-level) format, e.g., `java.lang.String`
- `methodName` -

getClassMethods

```
public java.util.Collection<IMethod>
getClassMethods(IClass appClass,
                java.lang.String methodName)
```

Get all the methods in an class with a particular name.

Parameters:

- `appClass` - the class
- `methodName` -

getStringConstantsReturnedByMethod

```
public java.util.Collection<java.lang.String>
getStringConstantsReturnedByMethod(IMethod method)
```

Get the possible String constants returned by the method. E.g., if the method has a statement return "result";, then "result" will be in the returned Collection. Throws an `IllegalArgumentException` if the return type of method is not `String`.

F4FHandler

```
java.lang.Object
  com.ibm.appscan.frameworks.highlevelapi.F4FHandler

public abstract class F4FHandler
extends java.lang.Object
```

The abstract class that any new framework handler should extend.

Note: Any sub-class of `F4FHandler` must have a no-argument constructor, for instantiation using reflection

Constructor Detail

```
F4FHandler

public F4FHandler()
```

handleApp

```
public abstract void handleApp(F4FApp app,
                               F4FActions actions)
```

Define what actions should be represented in the generated WAFL specification to handle the given application.

Parameters:

- `app` - the application to be analyzed
- `actions` - the actions to be taken; implementations of `handleApp(F4FApp, F4FActions)` should mutate this parameter and store the desired actions

isApplicable

```
public abstract boolean isApplicable()
```

Is the framework handler applicable for the target application? Implementations should check the language of the app, whether relevant configuration files are present, etc.

setFrameworksInput

```
public void setFrameworksInput(FrameworksInput input)
```

Should not be invoked by a framework handler.

getFrameworksInput

```
protected FrameworksInput getFrameworksInput()
```

Get the parsed representation of the input parameters to the frameworks code.

TaintedParam

```
java.lang.Object
  extended by com.ibm.appscan.frameworks.highlevelapi.TaintedParam

public class TaintedParam
extends java.lang.Object
```

A type used to represented how some method parameter may reference tainted data.

Constructor Detail

```
public TaintedParam(int paramPos,
                    java.lang.String accessPath)
```

Creates a new `TaintedParam` object for a particular parameter position and access path.

Parameters:

- `paramPos` - the position of the tainted parameter, numbered starting from 0 (where the `this` parameter of an instance method is parameter 0)

- `accessPath` - access path of fields from the parameter that should be tainted, e.g., "f.g". Use "" to indicate the parameter itself should be tainted.

getParamPos

```
public int getParamPos()
```

getAccessPath

```
public java.lang.String getAccessPath()
```

hashCode

```
public int hashCode()
```

Overrides:

- `hashCode` in class `java.lang.Object`

equals

```
public boolean equals(java.lang.Object obj)
```

Overrides:

- `equals` in class `java.lang.Object`

toString

```
public java.lang.String toString()
```

Overrides:

- `toString` in class `java.lang.Object`

High-level synthetic methods

Synthetic methods are a useful construct for modeling advanced data flow in frameworks. For example, many standard frameworks (such as Struts and Spring) encourage a *model-view-controller* (MVC)(see http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html) architecture for the application. With an MVC structure, client form submission is often handled in this manner:

1. Based on the URL, determine the application *model* class `M` for holding the submitted form data and the *controller* class `C` containing the business logic.
2. Create an `M` model object and set its properties based on the (untrusted) form data in the HTTP request. The properties are typically set via *setter* JavaBeans (for example, `setName()` or `setAddress()`).
3. Perform some validation on the data in the `M` object.
4. Create a `C` controller object, and pass the `M` object to a method `C.execute()` that performs the business logic. Typically, `execute()` will return the name of a view to render the result.
5. Based on the view name, determine the appropriate view file (for example, a JavaServer page) to show. Often, the data in the `M` object is passed to the view via attributes of the request or session object.

All of the above features can be modelled with Framework for Frameworks synthetic methods, thereby exposing the behaviors for analysis by AppScan Source. The Framework for Frameworks API provides high-level synthetic methods to ease generation of synthetic methods.

Note: Trace nodes with class names that begin with `Appscan.Synthetic`, `Appscan.Synthetic.Validator`, and `AppScan.Synthetic.Replacement` correspond to methods that are synthesized by AppScan Source.

- `AppScan.Synthetic` methods are used to stitch traces together in application code that uses frameworks.
- An `AppScan.Synthetic.Validator` method models the underlying validation performed by the framework runtime. You can select a validator method and mark it as a **Validator** if needed.
- An `AppScan.Synthetic.Replacement` method indicates that a method in the application code was replaced by AppScan Source to capture data flow between disjoint components (such as controllers and views) of the framework.

VDB format

Methods in the Framework for Frameworks API require `Strings` representing type names or method signatures to be in VDB format. VDB type names are simply fully-qualified source-level names (for example, `java.lang.String`) - or, in the case of inner classes, represented with a dollar symbol ($)(for example `javax.swing.text.DefaultEditorKit$DefaultKeyTypedAction`). For `.NET`, short names such as `int` and `string` should be avoided when using the Framework for Frameworks API. Instead, use fully-qualified names such as `System.Int32` and `System.String`.

A VDB method signature consists of these two parts:

1. The method name, including the fully-qualified name of the enclosing class (for example, `java.lang.String.substring`).
2. A descriptor for the method, giving the parameter types and return type. The parameter types are enclosed in parentheses and separated by semicolons (`;`). The closing parenthesis is followed by a colon (`:`), and then the return type.

An example VDB method signature is:

```
java.lang.String.substring(int;int):java.lang.String
```

An example VDB method signature from an inner class is:

```
javax.swing.text.DefaultEditorKit$DefaultKeyTypedAction.
    actionPerformed(ActionEvent):void
```

For most API methods requiring a `String` in VDB format, there is often an equivalent method that takes an `IClass` or `IMethod` object instead. For example, method `F4FActions.addTaintedCallback(String,int)` requires its first parameter to be a VDB-formatted method signature - while `F4FActions.addTaintedCallback(IMethod,int)` identifies the method with an `IMethod` object. Using the API methods that take `IClass` and `IMethod` objects may be more convenient for these reasons:

1. The user need not worry about VDB formatting for these methods as this is handled internally.
2. In obtaining the `IClass` or `IMethod` object (for example, via `F4FApp.getClassMethods()`), one ensures that the corresponding class or method actually exists in the application code.

Using high-level synthetic methods

This topic illustrates how to use some of the key features of high-level synthetic methods. For full details, see the Framework for Frameworks API classes and methods Javadoc documentation.

- "Creating a high-level synthetic method" on page 133

- "Creating locals"
- "Adding calls"
- "Adding a return value" on page 134
- "Globals" on page 134

Creating a high-level synthetic method

To create a high-level synthetic method and add it to the `F4FActions` object `actions`, use code such as:

```
HighLevelSyntheticMethod m = HighLevelSyntheticMethod.make();
actions.addHighLevelSyntheticMethod(m);
```

You can specify the name, parameter types, and return type of the synthetic method by passing a VDB method signature to `HighLevelSyntheticMethod.make()`.

Creating locals

A new local variable for a high-level synthetic method `m` can be created in this manner:

```
Local l = m.newLocal("java.lang.String");
```

Constructors need not be invoked in synthetic methods. Given the above code, one can assume that `l` refers to a non-null `String` object when adding further statements to the synthetic method.

Adding calls

Most statements in high-level synthetic methods will be method calls, added via the `addCall()` method. The parameters to `addCall()` represent the method to be invoked, file position information for the invocation, and parameters to be passed at the invocation. This is an example for adding a call to a setter method `sample.BeanType.setName()`, assuming an `F4FApp` object `app`:

```
Collection<IMethod> setterMethods =
   app.getClassMethods("sample.BeanType", "setName");
// assume there is exactly one "setName" method in BeanType
IMethod setter = setterMethods.iterator().next();
HighLevelSyntheticMethod m = HighLevelSyntheticMethod.make();
Local l = m.newLocal("sample.BeanType");
m.addCall(setter, null, l, Taint.taint());
```

Steps 2–5 of handling client form submission in an MVC architecture (as outlined in "High-level synthetic methods" on page 131) can all be modelled to some degree by adding appropriate calls to a synthetic method in the manner shown above.

The parameters for a call are represented by objects of type `Param`, which can be any one of:

- A `Local` object, representing a local
- A `Taint` object, representing untrusted or *tainted* data.
- An `EnclosingFormal` object, representing a formal parameter of the high-level synthetic method. For example, if you have a synthetic method with signature `synthMethod(int):void`, `EnclosingFormal.FIRST` refers to the `int` parameter of the method.
- A `Global` object representing a global (discussed in "Globals" on page 134).

Note: For non-static instance methods, the value to be passed as `this` must be provided to `addCall()`. In the above example, the value in `l` will be passed as `this` to `setName()`.

Adding a return value

A synthetic method can return a value by using the `setReturnedValue()` method. Return values can be useful for generating marker methods to model complex framework validation. For example, if you are using a framework that performs complex validation of a tainted HTTP request value before passing it to the setter method of a model object, you can expose the validation on traces discovered by AppScan Source by using code such as this:

```
String validatorSig =
   "Synth.validatorModel(java.lang.String):
   java.lang.String";
HighLevelSyntheticMethod validator =
   HighLevelSyntheticMethod.make(validatorSig);
// just return the parameter passed in
validator.setReturnedValue(EnclosingFormal.FIRST);
HighLevelSyntheticMethod m = ...;
Local validated = m.addCall(validatorSig, null, Taint.taint());
// now, validated can be passed to a setter
```

The synthetic method `Synth.validatorModel()` simply returns the `String` that is passed to it as a parameter. Then, a call to `Synth.validatorModel()` is added in another synthetic method, passing a tainted value as an argument. The result of this call is stored in `validated`, which can be passed in a subsequent call to a setter method (as illustrated by example in "Adding calls" on page 133). Traces involving this tainted data will include the call to `Synth.validatorModel()`, and an AppScan Source user can choose to filter the traces if the validation is deemed sufficient.

Globals

Globals are an advanced feature that can be used to expose flow between disparate parts of an application. For example, globals can be used to model the flow of data from a controller to a view via request or session attributes. The key operations for creating and accessing globals are:

- To create a `Global` object representing a global, use `F4FActions.createGlobal()`.
- To write to a global, use `HighLevelSyntheticMethod.addGlobalWrite()`.
- To read from a global, pass the `Global` object as a `Param` parameter in a call to `HighLevelSyntheticMethod.addCall()` - or have it returned from a synthetic method via `HighLevelSyntheticMethod.setReturnedValue()`.

Example: A controller class writes a user's first name to the request attribute `firstName`, and then the view reads this request attribute and renders the value to the response. At a high level, one could model this flow as:

```
Global firstNameGlobal = actions.createGlobal
   ("firstName", "java.lang.String", true);
HighLevelSyntheticMethod controller = ...;
Local firstNameLocal = controller.newLocal("java.lang.String");
controller.addGlobalWrite(firstNameGlobal, firstNameLocal, null);
HighLevelSyntheticMethod view = ...;
view.addCall("Response.write(java.lang.String):void",
   null, firstNameGlobal);
```

By adding a write to `firstNameGlobal` in the controller synthetic method and then passing `firstNameGlobal` to `Response.write()` in the `view` synthetic method, the data flow from controller to the view is exposed.

Note: This example contains many simplifications. Among other things, a full version would need to expose the proper flow of data to `firstNameLocal`.

Example: Synthetic method creation

This example demonstrates the creation of a synthetic method. In this synthetic method, we first taint a class field and then add a call to a method in the user's code and finally pass the result of this method call to a sink.

Application scenario

In this example, `createUser` is a REST API implemented in the JAX-RS framework. A user can invoke this API using a URL such as `http://host:port/users/createUser`. The framework runtime delegates this call to `UserRestService.createUser()` since the path matches. Additionally, the framework runtime initializes the `urlInfo` class variable from the client data before calling `createUser()`. Finally, the data returned by `createUser()` is sent back to the client by the runtime.

What is modelled in the synthetic method

To capture the "Application scenario" using a synthetic method, we have these elements:

- The tainting of the `urlInfo` class variable of `UserRestService`.
- The call to `UserRestService.createUser()`.
- The return of data to the client.

This is the code for the `createUser()` method:

```
import java.io.BufferedWriter;

@Path("users")
public class UserRestService {
  @Context
  UriInfo urlInfo;

  @Path("/createUser")
  public User createUser(){
    MultivaluedMap<String, String> queryParams =
      urlInfo.getQueryParameters();
    String name = queryParams.getFirst("name");
    User user = new User(name);
    return user;
  }
}
```

Step 1: Create an empty synthetic method

```
22    public class JAXRSHandler extends F4FHandler{
23    @Override
        public void handleApp(F4FApp app, F4FActions actions) {
24    HighLevelSyntheticMethod synthMethod = HighLevelSyntheticMethod.make();
```

- The synthetic method object is initialized at line 24.

Step 2: Model the tainting of a class field

The code snippet below demonstrates how specific class fields can be tainted. In this example, we wish to taint the class fields that are annotated with the `@Context` annotation.

```
27    // create a local variable of the appropriate type
28    Local userRestServiceClazzInstance =
        synthMethod.newLocal("com.ibm.appscan.UserRestService");
29
30    // get all the class fields
31    for(IField field: app.getIClass
        ("com.ibm.appscan.UserRestService").getDeclaredInstanceFields()){
32
33      //get all the annotations associated with the field
34      Collection<Annotation> fieldAnnotations = app.getFieldAnnotations(field);
35
36      //for each annotation of the field check if it is an @Context annotation
37      for (Annotation annotation : fieldAnnotations) {
38
39        if (annotation.getType().getName().toString().equals
            ("Ljavax/ws/rs/core/Context")) {
40
41          // call the F4F API to assign taint to the field
43          synthMethod.addInstanceVariableWrite(
                userRestServiceClazzInstance /*Variable representing
                class instance*/,
44              field /*field to taint  */,
                Taint.taint() /* taint */,
                null);
45        }
46      }
47    }
```

The Framework for Frameworks API `addInstanceVariableWrite` takes four arguments:

1. The first argument is the class reference whose field we want to taint. In the example, the local variable `userRestServiceClazzInstance` refers to this argument.
2. The second argument is the class field we want to taint.
3. The third argument is the new value that will be assigned to the class variable field. In the example, we want to taint this variable so that a `Taint.taint()` is passed.
4. The last argument is `FilePositionInfo`, which is `null` in this example.

Step 3: Model the calling of the `createUser()` method

In this step, we simulate the call in our synthetic method.

```
50    // call createUser() and store the returned value to a local variable
51    Local returnObj = synthMethod.addCall(
52      "com.ibm.appscan.UserRestService.createUser():com.ibm.appscan.User"
          /* signature of the method to be called */,
          null, /* FilePositionInfo */
53        userRestServiceClazzInstance /* Object on which the method
            will be called */);
```

The `addCall` high-level API takes three arguments:

- The first argument is the signature of the method that we want to invoke. In this case, we want to invoke `User.createUser()`, hence its signature is used.

- The second argument is `FilePositionInfo`. It is passed as `null` as it is not required for this example.
- The last argument represents the list of parameters which are required to invoke the `createUser()` method.

Since `createUser()` takes no arguments, the only argument passed to this call is the `this` object, a `Local` variable (`userRestServiceClazzInstance`) of the type `User`. The method `createUser()` returns a newly-created `User` which is stored in a new `Local` variable called `returnedObj` (as shown on line 51).

Step 4: Model data return to the client

In this final step, we create a sink to model the return of data back to the client.

```
58    // create a PrintWriter object
59    Local printWriterObj = synthMethod.newLocal("java.io.PrintWriter");
60
61    // we want to call the print API on the PrintWriter object.
      // The print API takes a single argument of Object type returns void
62
63    // we create a param list of size 2. The first item in this list is always the
      //  'this' object and rest are actual method arguments.
64    Param[] printWriterObjParam =  new Param[2];
65    printWriterObjParam[0] = printWriterObj;  // The "this" object
      // the value returned by the call to the createUser method
66    printWriterObjParam[1] = returnObj;
67
68    // Now add a call to the print method
69    synthMethod.addCall("java.io.PrintWriter.print(java.lang.Object):void",
        null, printWriterObjParam);
70  }
```

- At line 59, we create a `Local` object for the `PrintWriter` class in the synthetic method.
- From lines 64 to 66, we prepare the parameter list which is required to invoke the `print` method on the `PrintWriter` class instance (`printWriterObj`):
  - The first parameter is the object reference (`printWriterObj`), itself.
  - The second parameter is the argument to the `print` method. We want to pass the return value which is stored in the `returnObj` Local variable.
- At line 69, we add a call to the `PrintWriter.print` method to model the return of data back to the client.

Chapter 8. AppScan Source client component error messages

CRWSA0900E The excluded bundle cannot be renamed

The Excluded Bundle is a built-in bundle that holds all excluded findings. It cannot be renamed.

CRWSA0907E Failed to add <> because it already exists in the database and cannot be overwritten

The custom rule already exists in the AppScan Source database and cannot be recreated or overwritten. This occurs in one of these situations:

- The rule that was shipped in the database is marked as a sink - and you are attempting to create a custom rule that marks the sink as taint propagation.
- The rule that was shipped in the database is marked as a taint propagator - and you are attempting to create a custom rule that changes that rule by adding taint propagation.

CRWSA0920E There were errors importing the signatures into the project

Verify that the project is configured properly before importing signatures.

CRWSA0925E The value entered for "{0}" is not valid.

The value entered is not a valid format. Please enter a valid format.

CRWSA0930E Invalid date format entered for "{0}".

CRWSA0935E Invalid number format entered for "{0}".

CRWSA0940E Unknown user

Ensure that the user exists on the Enterprise Server and that you have entered the user information correctly.

CRWSA0945E The specified User ID is too short. User IDs must be at least 1 character in length.

CRWSA0950E The specified Password is too short. Passwords must be at least 6 characters in length.

CRWSA0955E The specified User ID is too long. User IDs cannot exceed 255 characters in length.

CRWSA0960E The specified User ID contains invalid characters.

CRWSA0965E The specified Password is too long. Passwords cannot exceed 16 characters in length.

CRWSA0970E The specified passwords do not match. Re-enter the passwords.

CRWSA0975E The Admin user cannot be modified.

CRWSA0980E The specified email address is invalid. - The required format is <user>@<domain>, where <domain> cannot exceed 255 characters.

CRWSA0985E The 'Email' is required.

CRWSA0990E The 'Name' is required.

CRWSA0995E The specified User ID exists in the repository, the 'Name' and 'Email' fields have been updated.

CRWSA1010W User {0} could not be added.

Refer to the AppScan Enterprise Server help for information.

CRWSA1050E Unexpected error loading external editor.

Ensure that the external editor is available and working correctly.

CRWSA1055E Error loading external editor. Properties file is corrupt.

Ensure that the external editor is available and working correctly.

CRWSA1060E Error loading external editor. Properties file is missing.

Ensure that the external editor is available and working correctly.

CRWSA1065E Error loading external editor. Unable to load preferred editor

Ensure that the external editor is available and working correctly.

CRWSA1070E Error loading external editor. Editor plug-in returned a failure.

Ensure that the external editor is available and working correctly.

CRWSA1075E Error loading external editor. Unable to create executable instance.

Ensure that the external editor is available and working correctly.

CRWSA1080E Error loading Eclipse. Connection failed.

Ensure that you have properly configured Eclipse to be used as an external editor (use an Eclipse that contains `com.ouncelabs.core.filelauncher.jar` in its `plugins\` directory).

CRWSA1085E Error loading Eclipse. No response from server.

Ensure that you have properly configured Eclipse to be used as an external editor (use an Eclipse that contains `com.ouncelabs.core.filelauncher.jar` in its `plugins\` directory).

CRWSA1090E Error loading Eclipse. Protocol failure.

CRWSA1095E Error loading Eclipse. Path to Eclipse is invalid.

Ensure that you have properly configured Eclipse to be used as an external editor (use an Eclipse that contains `com.ouncelabs.core.filelauncher.jar` in its `plugins\` directory).

CRWSA1100E Error loading custom editor. Unable to execute {0}

CRWSA1120E Scan Failed

CRWSA1125E An error occurred while retrieving the results of the scan

CRWSA1150W Warning: Marking "{0}" as a validation routine will have no effect on the current finding or trace graph. Are you sure you want to add the root node as a validation routine?

CRWSA1155E At least one source, sink, source property, or sink property is required.

CRWSA1160E Unable to launch the custom rules wizard while an assessment is running.

CRWSA1165E Error adding custom rules

CRWSA1170E Failed to add {0}.

CRWSA1175E Failed to add {0} because it already exists in the database and cannot be overwritten.

CRWSA1185E Validate failed due to error in compilation

CRWSA1190E Compilation Failed

CRWSA1195E An error occurred while trying to open the editor.

When the editor fails to open, try restarting AppScan Source.

CRWSA1200E The selected file was not found on this system.

CRWSA1205E An error occurred while creating source code markers for the assessment.

CRWSA1210E One or more source files could not be located for the source code snippet feature. Do you want to be prompted for the location of the missing files?

CRWSA1215E Failed trying to connect to the AppScan Enterprise Console. - Verify that the server is running at the specified location.

CRWSA1220E Login timeout after {0,number} seconds - It is recommended that your restart the AppScan Enterprise Console services.

CRWSA1225E Error storing unsaved filters. Changes to filters will not be saved.

CRWSA1230E Unable to save assessment while scanning.

CRWSA1240E Application not found

CRWSA1245E Could not find {0}

Verify the location of the specified JRE.

CRWSA1250E Unable to find startup.jar or the org.eclipse.equinox.launcher plug-in

Verify the specified Eclipse-based installation.

CRWSA1290W One or more projects was missing a JDK. The default JDK has been set for these projects.

The specified JDK of an imported project was not recognized by AppScan Source. To correct this, add it as a recognized JDK using the Java and JSP preference page.

CRWSA1295W The project was missing a JDK. The default JDK has been set for this project.

The specified JDK of an imported project was not recognized by AppScan Source. To correct this, add it as a recognized JDK using the Java and JSP preference page.

CRWSA1300E One or more projects is missing a JDK and there is no default specified. You will be prompted to choose your default JDK after clicking the OK button. In the future, this JDK will be used for all projects that have not explicitly set a JDK.

CRWSA1305E The application is read-only. Changes that are made to the assessment will not persist in future scans.

It is possible that the application is remote or that it needs to be saved. In either case, changes that are made to the assessment will not be available in future scans.

CRWSA1340E "{0}" failed to register

CRWSA1345E The application file is not writable.

CRWSA1350E Error Deleting Entries

CRWSA1355E Unable to delete an application while an assessment is running.

CRWSA1360E Unable to delete a project while an assessment is running.

CRWSA1365W Any projects to which you want the changes applied will need to be refreshed.

CRWSA1370W Some of the selected findings have existing notes. Are you sure you want to overwrite them?

CRWSA1375W Failed to export to file

CRWSA1380W No findings to export. All selected findings are missing from the assessment results.

CRWSA1383E No findings to submit. All selected findings were previously submitted and the "Submit defects only once" preference is set.

CRWSA1385W These fields must be completed to submit a defect:

CRWSA1395W Bundle successfully saved to {0}.

CRWSA1400E The project file "{0}" could not be found.

Verify that the AppScan Source project file exists.

CRWSA1405E The ASP.NET directory "{0}" could not be found

Verify that the ASP.NET directory exists.

CRWSA1410E Error importing project file "{0}"

CRWSA1415E Error importing ASP.NET directory "{0}"

CRWSA1420E Error creating project

CRWSA1425E The working directory "{0}" does not exist. Select a valid directory.

CRWSA1430E The web context root "{0}" does not exist. Select a valid directory or WAR file.

CRWSA1435E The web context root "{0}" cannot be a parent directory or the same directory as the project directory "{1}".

CRWSA1440E The content root "{0}" does not exist. Select a valid directory.

CRWSA1445E The file or directory "{0}" does not exist. Select a valid file or directory.

CRWSA1450E The content root "{0}" cannot be a parent directory or the same directory as the project directory "{1}".

CRWSA1455E The method {0} already has an ActionObject for type {1}.

CRWSA1460E "{0}" is not a valid directory

CRWSA1465E JDK Missing for project

CRWSA1470E This Java/JSP project references an invalid JDK ({0}). A default JDK will be provided.

CRWSA1475E JDK Name Already Exists

CRWSA1480E A JDK named "{0}" already exists.

Specify a unique JDK name.

CRWSA1485E The JDK Name must have at least one character.

CRWSA1490E A JDK must be selected for this operation

An existing JDK must be selected.

CRWSA1495E Unable to Save

CRWSA1500E Unable to save while an assessment is running

CRWSA1505E The source root "{0}" already exists as a root.

CRWSA1510E The directory "{0}" does not exist. Select a valid directory.

CRWSA1515E The precompile directory "{0}" does not exist. Select a valid directory.

CRWSA1520E The class {0} exists in both the dependency {1} and on the source path.

Verify the specified dependencies.

CRWSA1525E Unexpected file type on class path ({0}).

Ensure that your Java project contains only `.jar`, `.class`, or `.java` files.

CRWSA1530E Projects cannot be validated while a scan is running.

CRWSA1535E

CRWSA1540E The configuration name is empty

CRWSA1545E Invalid Source Root

CRWSA1550E Source root "{0}" is invalid. - Source roots which are files must be under the working directory.

CRWSA1555E A valid web context root could not be found.

CRWSA1565E Successfully created encoding routine for {0}

CRWSA1568E Could not resolve variables in path "{0}"

Add the path variable using the Change Variables preference page.

CRWSA1570E Invalid Attribute name {0}

CRWSA1575E Invalid Attribute value {0}

CRWSA1580E You must create an attribute before adding values.

CRWSA1585E Application must have a non-empty name.

CRWSA1590E Duplicate entry "{0}". Specify a unique value.

CRWSA1595E Name cannot be empty.

CRWSA1600E A view with findings in it must be selected for report to generate.

CRWSA1625E Access is denied. Current account lacks permission to {0}.

CRWSA1630E An error occurred while attempting to import the workspace. Make sure that your importer is properly configured in Preferences > Eclipse Workspace Importers.

CRWSA1640E Unable to add project. {0} is a reference or read-only application.

The application is read-only - or it is a reference to an Eclipse workspace or Microsoft Visual Studio solution. Projects cannot be added.

CRWSA1650E Unable to find the style sheet {0}.

CRWSA1655E A name is required for the Pattern

CRWSA1660E At least one criteria must be specified for the Pattern

CRWSA1665E The Criteria cannot be empty.

CRWSA1670E File is required for the Pattern.

CRWSA1675E A scan rule of the name "{0}" already exists. Choose a unique name.

CRWSA1700E The report layout is currently empty. Add report elements in the Report Layout tab.

CRWSA1705E There are no bundles to export

CRWSA1710E There is no open assessment to export

CRWSA1715E The passwords that you entered in the New Password and Confirm New Password fields do not match

CRWSA1725E The password is too short. Enter a password between 6 and 16 characters.

CRWSA1730E The password is too long. Enter a password between 6 and 16 characters.

CRWSA1735E Settings file "{0}" has no display name and will be skipped.

CRWSA1736E Could not open file <filename>. Make sure the user the automation server is running as has permissions to the file

CRWSA1737E Could not open file <filename>. Make sure the user the automation server is running as has permissions to the file, and that it is a local application

CRWSA1740E Settings file named "{0}" is already open.

CRWSA1745E Settings file "{0}" is not visible to the user.

CRWSA1750I This tab contains some settings that require restarting AppScan Source Services for changes to take effect. Before restarting the AppScan Source services, close all AppScan Source client applications running on or connected to this computer. Do you want to restart the services now?

CRWSA1755E "{0} has a setting that is missing one or more required attributes (name, display_name, or type). This file will not be loaded."

CRWSA1760E A licensing error has occurred:

CRWSA1775E AppScan Enterprise Console connection failed to initialize. Verify the availability of the AppScan Enterprise Console with these settings in an external browser.

CRWSA1780E AppScan Enterprise Console connection failed to initialize. Check logs and/or verify the availability of the AppScan Enterprise Console with these settings in an external browser. - Error details: - {0}

CRWSA1790E Failed to publish assessment to AppScan Enterprise Console. Check logs and/or contact your System Administrator.

CRWSA1795E Failed to initialize publishing service. Verify your AppScan Enterprise Console connection settings.

CRWSA1800E Failed to initialize publishing service. Verify the availability of the AppScan Enterprise Console and your ability to log on.

CRWSA1805E Unable to download file. - An error occurred while attempting to connect to the site.

Ensure that you have a working internet connection and that the site is available.

CRWSA1810E Login required before the bundle that was created in the version 7.0.0 product can be opened

CRWSA9999E AppScan Source for Analysis has encountered a critical error and must close.

Chapter 9. Installation and user data file locations

When you install AppScan Source, user data and configuration files are stored outside of the installation directory.

- "Default installation location"
- "Default AppScan Source data directory"
- "AppScan Source temporary file location" on page 150

Default installation location

When AppScan Source is installed, the software is placed in one of these default locations:

- 32-bit versions of Microsoft Windows 7 and Microsoft Windows Server 2008:
  `<SYSTEMDRIVE>:\Program Files\IBM\AppScanSource`
- 64-bit versions of Microsoft Windows 7 and Microsoft Windows Server 2008:
  `<SYSTEMDRIVE>:\Program Files (x86)\IBM\AppScanSource`
- Linux: If you are the root user, the Installation Wizard installs your software in `/opt/ibm/appscansource`. If you are not the root user, you can install the AppScan Source for Development Eclipse plug-in - which installs to `<home_directory>/AppScan_Source` by default.
- OS X: `/Applications/AppScanSource.app`

Important:

- The installation directory name can only contain English characters. Folders with names containing non-English characters are not permitted.
- If you are installing on Windows, you must have Administrator privileges to install AppScan Source components.
- If you are installing on Linux, you must have root privileges to install AppScan Source server components.

Default AppScan Source data directory

AppScan Source data consists of items such as configuration, sample, and log files. When AppScan Source is installed, data files are placed in these locations by default:

- Microsoft Windows 7, Microsoft Windows Server 2008: `<SYSTEMDRIVE>:\ ProgramData\IBM\AppScanSource`

  Note: `ProgramData\` is a hidden folder, and to see it you must modify your view preferences in **Explorer** to show hidden files and folders.
- Linux: `/var/opt/ibm/appscansource`
- OS X: `/Users/Shared/AppScanSource`

To learn how to change the location of the AppScan Source data directory, see "Changing the AppScan Source data directory" on page 150.

AppScan Source temporary file location

Some AppScan Source operations result in the creation of temporary files, which are stored in these locations by default:

- Microsoft Windows 7, Microsoft Windows Server 2008: `<SYSTEMDRIVE>:\ProgramData\IBM\AppScanSource\temp`

  Note: `ProgramData\` is a hidden folder, and to see it you must modify your view preferences in **Explorer** to show hidden files and folders.
- Linux: `/var/opt/ibm/appscansource/temp`
- OS X: `/Users/Shared/AppScanSource/temp`

The temporary file location is always located in a `temp` directory in the AppScan Source data directory. You can change the temporary file location by changing the data directory, as described in "Changing the AppScan Source data directory." This will cause the `temp` to be located in the data directory that you have chosen.

Changing the AppScan Source data directory

You may want to change the location of the AppScan Source data directory for the purpose of managing hard disk space. You can change the location after AppScan Source installation by following the steps in this topic.

Before you begin

Before completing this task, ensure that all AppScan Source client applications have been exited or shut down. AppScan Source client applications include:

- AppScan Source for Analysis
- AppScan Source for Development (Eclipse or Visual Studio plug-in)(supported only on Windows and Linux)
- AppScan Source command line interface (CLI)
- AppScan Source for Automation

In addition, if you have installed AppScan Source for Automation, ensure that the Automation Server has been shut down:

- On Windows, stop the **IBM Security AppScan Source Automation** service.
- On Linux, issue this command: `/etc/init.d/ounceautod stop`
- On OS X, issue this command: `launchctl stop com.ibm.appscan.autod`

Procedure

1. Define an `APPSCAN_SOURCE_SHARED_DATA=<data_dir>` environment variable, where `<data_dir>` is the location in which you want AppScan Source data to be stored.

   Note:

   - The `<data_dir>` location must be a complete and absolute path that already exists on the same machine as your AppScan Source installation.
   - The `<data_dir>` directory name can only contain English characters. Folders with names containing non-English characters are not permitted.

2. Locate the default data directory that was created when AppScan Source was installed (see "Default AppScan Source data directory" on page 149 to learn about default data directory locations).
3. Copy or move the contents of the default data directory to the <data_dir> location that is specified in the environment variable.
4. **Applies only to AppScan Source for Automation installed on Linux**:
   a. Edit the /etc/init.d/ounceautod file.
   b. Locate this line,

      ```
      su - ounce -c
      'export LD_LIBRARY_PATH="/opt/IBM/AppScan_Source/bin":$LD_LIBRARY_PATH &&
      cd "/opt/IBM/AppScan_Source/bin" &&
      "/opt/IBM/AppScan_Source/bin/ounceautod" -s' >>
      "/var/opt/ibm/appscansource/logs/ounceautod_output.log" 2>&1 &
      ```

      and replace it with this:

      ```
      su - ounce -c
      'export APPSCAN_SOURCE_SHARED_DATA=<new data directory path here> &&
      export LD_LIBRARY_PATH="/opt/IBM/AppScan_Source/bin":$LD_LIBRARY_PATH &&
      cd "/opt/IBM/AppScan_Source/bin" &&
      "/opt/IBM/AppScan_Source/bin/ounceautod" -s' >>
      "<new data directory path here>/logs/ounceautod_output.log" 2>&1 &
      ```

      Note: The above command is one line.
   c. Save the /etc/init.d/ounceautod file.

What to do next

If you have installed AppScan Source for Automation, start the Automation Server:

- On Windows, start the **IBM Security AppScan Source Automation** service.
- On Linux, issue this command: /etc/init.d/unceautod start
- On OS X, issue this command: launchctl start com.ibm.appscan.autod

Legal notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this documentation in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this documentation. The furnishing of this documentation does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be

incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
20 Maguire Road
Lexington, Massachusetts 02421-3112
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this documentation and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Copyright license

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(C) (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. (C) Copyright IBM Corp. 2003, 2011.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks and service marks

See http://www.ibm.com/legal/copytrade.shtml.

Index

Printed in USA