****

**HUSACCT**

**System documentation**

Team 1:

Erik Verhoef 1572690  
René van Aerle 1556647  
Tim Linschoten 1548807  
Niels Nijveldt 1562854  
Stefan Collette 1575795

Teachers: Leo Pruijt, Christian Köppe, Michiel Borkent.

Date: 9-6-2012

Contents

[1 Introduction 3](#_Toc327302896)

[2 Functionality 4](#_Toc327302897)

[2.1 Use case model 4](#_Toc327302898)

[2.1.1 Workspace 4](#_Toc327302899)

[2.1.2 Sub GUIs 7](#_Toc327302900)

[2.1.3 Import architecture 8](#_Toc327302901)

[2.1.4 Export architecture 8](#_Toc327302902)

[2.1.5 Export violation report 8](#_Toc327302903)

[2.1.6 Application properties 8](#_Toc327302904)

[2.1.7 Change language 10](#_Toc327302905)

[2.1.8 Toolbar 10](#_Toc327302906)

[2.1.9 Taskbar 11](#_Toc327302907)

[3 Decisions and justification 12](#_Toc327302908)

[3.1 Functional Requirements 12](#_Toc327302909)

[3.2 Non-functional requirements 12](#_Toc327302910)

[3.3 Decisions and justification 12](#_Toc327302911)

[4 Software partitioning 14](#_Toc327302912)

[4.1 Physical class diagram 14](#_Toc327302913)

[4.2 Physical software partitioning model 15](#_Toc327302914)

[4.3 Plugin component diagram 16](#_Toc327302915)

[4.4 Architectural Rules 16](#_Toc327302916)

[5 Subsystem specification 17](#_Toc327302917)

[5.1 Start up 17](#_Toc327302918)

[5.2 ServiceProvider 17](#_Toc327302919)

[5.3 StateController 18](#_Toc327302920)

[5.4 Loading dialog 19](#_Toc327302921)

[5.5 Error/info message 20](#_Toc327302922)

[6 Build 20](#_Toc327302923)

[7 Maven plugin 21](#_Toc327302924)

[8 Testing 21](#_Toc327302925)

[9 Future work 21](#_Toc327302926)

[9.1 Known bug list 21](#_Toc327302927)

[9.2 Ideas for improvement 21](#_Toc327302928)

# Introduction

HUSACCT is an acronym for “Hogeschool Utrecht Software Architecture Conformance Checking Tool”. As the name suggest HUSACCT is a software architecture compliance checking tool. These tools are used to see if a software project is built according to the architecture of an application. HUSACCT has several core functions: defining a logical architecture, analyzing an existing application, defining architectural rules, validate the code according to the defined rules.

The development of HUSACCT was split in 6 components. Each component had a team assigned. The teams consisted of 4 to 5 students from Utrecht University of Applied Sciences. The system components were:

* Control & GUI   
  Responsible for managing the workflow of the entire application as well as the main GUI. Another responsibility was the development of an Eclipse plugin, a Maven plugin and if possible a build server.
* Analyse  
  Responsible for the analysis of an existing application. It was also necessary to give a graphical overview of the application. This component was split into 2 parts, Analyse Java and Analyse C#.
* Define  
  Responsible for the definition of the logical architecture and the mapping of the architecture to an existing application.
* Validate  
  Responsible for the validation of the defined architectural rules.
* Graphics  
  Responsible for giving a graphical representation of the defined and analysed architecture.

This document will elaborate on the Control & GUI component. In chapter 2 we will describe the core functionalities based on the use cases. Chapter 3 will describe all major decisions and justifications derived from the functional and non-functional requirements. Chapter 4 contains the technical diagrams and architectural rules. Other major functionalities to support the use cases are described in Chapter 5. Chapter 6 elaborates on the deployment mechanism. Chapter 7 elaborates on the Maven plugin. Information on the test suite is given in Chapter 8. And finally chapter 9 details the known bugs and ideas for improvement of HUSACCT.

# Functionality

## Use case model



### Workspace

The workspace is a container that stores all the data that’s inputted into the tool. This data is the defined architecture, application properties (application name, programming language, paths and version number) and the mapping of the defined architecture to the source code. The workspace was meant to contain the analysed application, the current violations and the violation history; however this was not possible due to technical difficulties (see 7.1 Known bugs list).

The following use cases are derived from the workspace implementation.

#### New workspace

A workspace is needed to store all the data. A new workspace makes sure there is a container to store all the data.

**Packages/classes**

* Husacct.control.presentation.workspace.CreateWorkspaceDialog
* Husacct.control.presentation.menubar.FileMenu
* Husacct.control.task.WorkspaceController
* Husacct.control.domain.Workspace

#### Open workspace

Open a workspace so the user can continue with a previously saved workspace.

There are different methods for opening a workspace. Each of these methods has its own loaderPanel. These panels are dynamically created via the LoaderPanelFactory.

Once the user input is validated the workspace data will be opened using the ResourceFactory. The workspace data will be loaded into a JDom2 model using the specific resources (HusacctResource, XmlResource).

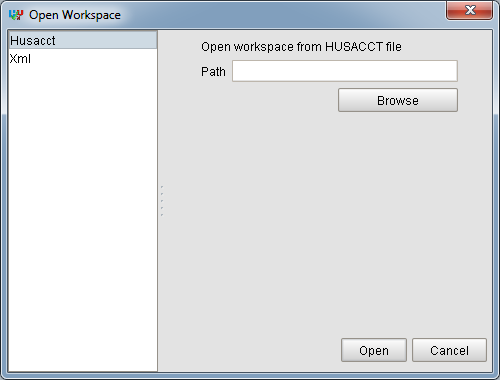


Figure . Open Workspace dialog

**Sequence diagram**



**Packages/classes**

* Husacct.control.presentation.workspace.OpenWorkspaceDialog
* Husacct.control.presentation.menubar.FileMenu
* Husacct.control.presentation.loaders.LoaderPanelFactory
* Husacct.control.presentation.loaders.LoaderPanel
* Husacct.control.presentation.loaders.HusacctLoadPanel
* Husacct.control.presentation.loaders.XmlLoadPanel
* Husacct.control.task.WorkspaceController
* Husacct.control.task.resources.ResourceFactory
* Husacct.control.task.resources.IResource
* Husacct.control.task.resources.HusacctResource
* Husacct.control.task.resources.XmlResource

#### Save workspace

A workspace is saved to file (or other methods) so the user can continue with his work later on.

There are different methods for saving a workspace. Each of these methods has its own SaverPanel. These panels are dynamically created via the SaverPanelFactory.

Once the user input is validated the workspace data will be saved using the ResourceFactory. The workspace data will be transformed from JDom2 to the required format by the selected resource (HusacctResource, XmlResource).

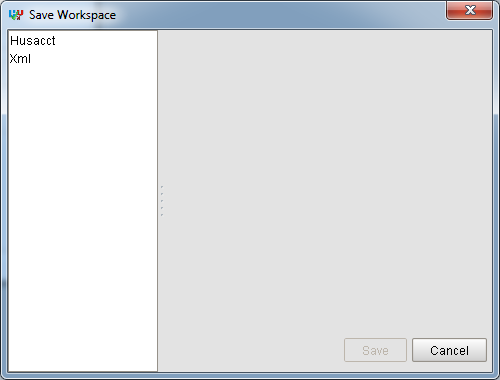


Figure 2. Save Workspace dialog

**Packages/classes**

* Husacct.control.presentation.workspace.OpenWorkspaceDialog
* Husacct.control.presentation.menubar.FileMenu
* Husacct.control.presentation.savers.SaverPanelFactory
* Husacct.control.presentation.savers.SaverPanel
* Husacct.control.presentation.savers.HusacctLoadPanel
* Husacct.control.presentation.savers.XmlLoadPanel
* Husacct.control.task.WorkspaceController
* Husacct.control.task.resources.ResourceFactory
* Husacct.control.task.resources.IResource
* Husacct.control.task.resources.HusacctResource
* Husacct.control.task.resources.XmlResource

#### Close workspace

A workspace will be closed so a new workspace can be opened or created.

When a workspace is closed, all the services will be reset.

**Packages/classes**

* Husacct.control.presentation.menubar.FileMenu
* Husacct.control.task.WorkspaceController
* Husacct.control.domain.Workspace

### Sub GUIs

Other services in the tool provide a GUI (as a JInternalFrame) for their specific functionality. These provided GUIs are managed via the Viewcontroller and put into the main GUI. The frames are initiated by the user via the menubar or toolbar (more on the toolbar in section 2.1.3 Toolbar). Once the frame is opened it can be managed via the taskbar (more on the taskbar in section 2.1.4 Taskbar) The underlying mechanism that shows the JInternalFrames is the same for the following use cases:

* Define architecture
* Define architecture diagram
* Analysed application overview
* Analysed application diagram
* Validate
* Validate configuration

Each of these use cases use husacct.control.task.ViewController and husacct.ServiceProvider.

Because these use cases are so similar only one sequence diagram will be provided.

**Sequence diagram**



### Import architecture

A defined architecture can be imported.

The defined architecture data will be loaded into a JDom2 model and sent to the DefineService with the following method: loadLogicalArchitectureData(logicalData).

**Packages/classes**

* Husacct.control.presentation.menubar.DefineMenu
* Husacct.presentation.util.ImportArchitectureDialog
* Husacct.control.task.ImportController
* Husacct.ServiceProvider

### Export architecture

A defined architecture can be exported so it can be reused later.

The defined architecture data will be requested from the DefineService with the method getLogicalArchitectureData() and transformed from JDom2 to an xml-file.

**Packages/classes**

* Husacct.control.presentation.menubar.DefineMenu
* Husacct.presentation.util.ExportArchitectureDialog
* Husacct.control.task.ExportController
* Husacct.ServiceProvider

### Export violation report

A violation report can be exported.

The possible extensions are requested from the ValidateService with the method getExportExtentions(). A Java.io.File and file extension will be sent back with the method exportViolations(file, filename.getExtension()) to the ValidationService once a file location is selected. The ValidationService will create the report.

**Packages/classes**

* Husacct.control.presentation.menubar.ValidateMenu
* Husacct.control.presentation.util.ExportViolationsReportDialog
* Husacct.control.task.ExportController
* Husacct.ServiceProvider

### Application properties

It is possible to save the application properties. An application can be analysed based on the properties. The properties are application name, programming language, version number and paths.

The application properties dialog is made generic so the properties can be set while creating a new workspace and later on via the menubar.

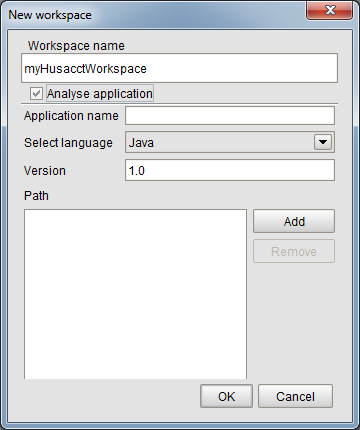


Figure 3. Analyse application from New workspace dialog

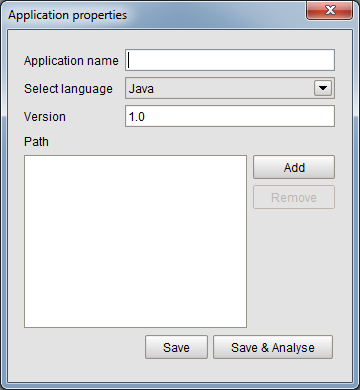


Figure 4. Application properties dialog

The saved application properties are sent to the DefineService with the method createApplication(applicationDTO.name,

applicationDTO.paths,

applicationDTO.programmingLanguage,

applicationDTO.version)

Based on the properties the application can be analysed. In the New workspace dialog an application will be analysed when the analyse application checkbox is checked and all fields are validated. In the application properties dialog can be analysed by pushing the ‘save & analyse’-button. The AnalyseService is requested to analyse with the method: analyseApplication(). The actual analysis is put in a different thread to prevent the application from freezing (more on threading in 5.1.4 Threading)

**Packages/classes**

* Husacct.presentation.util.SetApplicationDialog
* Husacct.presentation.util.SetApplicationPanel
* Husacct.presentation.workspace.CreateWorkspaceDialog
* Husacct.task.ApplicationController

### Change language

It is possible to change the language of HUSACCT.

The LocaleController automatically detects available languages according to the resource bundles (properties files) located within the husacct.common.locale package. It is possible to add a new language within this package with a filename “husacct\_<xx>.properties” where “xx” is the language. For example, husacct\_en.properties contains the english translation. If the naming convention is honored, the language will be automatically available at startup.

When creating a new language resource bundle, it is required to check the bundle for correctness. This resourcebundle check is integrated within the build process (section 6 Build).

The ControlService provides the method getTranslatedString(stringIdentifier). This method returns the value of the given stringIdentifier within the currently selected language resource bundle. Below is the sequence diagram that shows the mechanism of changing the language.



**Packages/classes**

* Husacct.control.task.LocaleController
* Husacct.control.ILocaleChangeListener
* Husacct.control.IControlService
* Husacct.ServiceProvider

### Toolbar

A toolbar is implemented to increase the ease of use of the HUSACCT-tool. All the buttons in the toolbar are directly linked to the items in the menubar.



Figure 5. Toolbar

The buttons are from left to right:

* New workspace
* Open workspace
* Save workspace
* Define architecture
* Defined architecture diagram
* Application properties
* Analysed application overview
* Analysed application diagram
* Validate

**Packages/classes**

* Husacct.presentation.toolbar.ToolBar
* Husacct.presentation.toolbar.ToolBarItem

### Taskbar

The taskbar is shown when an internalframe is opened. The taskbar is implemented to give a clear overview of all the opened internalframes.

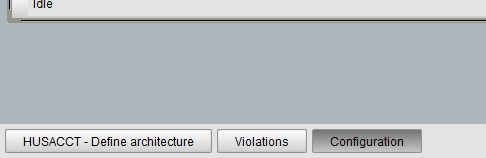


Figure 6. Taskbar

Left-click on a button will put the corresponding frame to the front.

Right-click on a button will open a contextmenu.

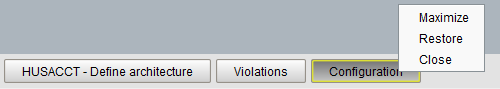


Figure 7. Contextmenu

Maximize, maximizes the frame.  
Restore, sets the size of the frame to the default and centers the frame.  
Close, close the frame.

**Packages/classes**

* Husacct.presentation.taskbar.TaskBar
* Husacct.presentation.taskbar.ContextMenu
* Husacct.presentation.taskbar.InternalFrameAdapter
* Husacct.presentation.taskbar.ContextClickListener

# Decisions and justification

## Functional Requirements

|  |  |  |
| --- | --- | --- |
| **CFR** | **Description** | **Priority** |
| 1 | Ability to call the define service. This service is responsible to handle several operations which has to do with defining architecture rules. | Must have |
| 2 | Ability to call the analyse service. This service is responsible to handle several operations which has to do with analyzing. | Must have |
| 3 | Ability to call the validate service. This service is responsible to handle several operations which has to do with validating. | Must have |
| 4 | Ability to save project. | Must have |
| 5 | Ability to close project. | Should have |
| 6 | Ability to open project. | Must have |
| 7 | Ability to open a specific view | Should have |
| 8 | Ability to close a specific view. | Should have |
| 9 | Ability to create a new project. | Must have |
| 10 | Ability to show the architectural graphics. | Must have |

## Non-functional requirements

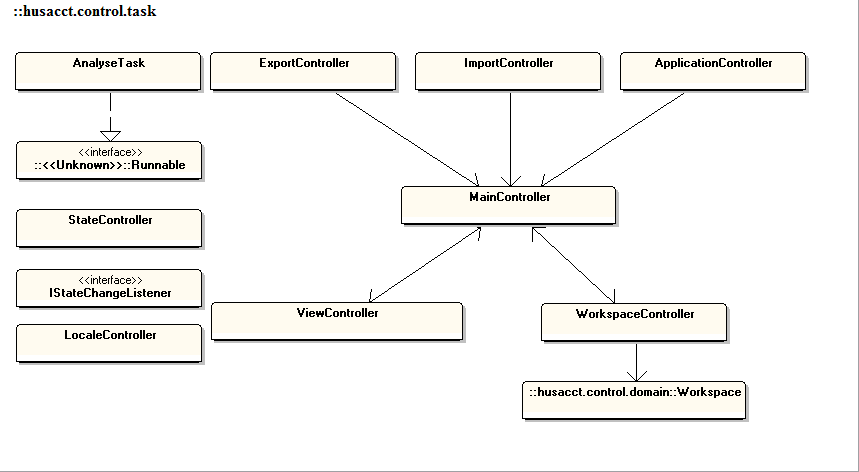
|  |  |  |
| --- | --- | --- |
| **NFR** | **ISO 9123 attr.** | **Requirement** |
| **1.** | **Functionality** |  |
| 1.1 | Suitability | Configuration of the user language must be possible. |
| 1.2 | Interoperability | Communication with other applications (Maven, Sonar, Ant ...) must be supported. E.g. activate, import; export. |
|  | **Reliability** |  |
|  | Maturity | The tool must not go down in case of a failure, but generate a meaningful error message. |
|  | **Usability** |  |
|  | Understandability | The division in required steps (manual and automated) must be clear to the user of the tool. |
|  | Understandability | It must be clear to the user what he/she is working on |
|  | **Maintainability** |  |
|  | Analyseability  Testability | Taking over the development of the tool by other development teams must be unproblematic. |
|  | **Portability** |  |
|  | Adaptability | Plug-ins for IDE’s (Eclipse, Sonar ...) must be provided. |
|  | Installability | The tool must be easy to install. |
|  | Installability | The tool must provide a complete installation package (no extra downloads needed). |
| 4.4 | Adaptability | The tool must be usable in several operating systems. |

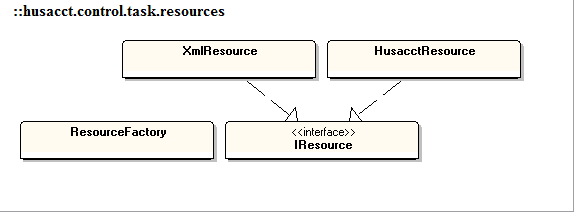
## Decisions and justification

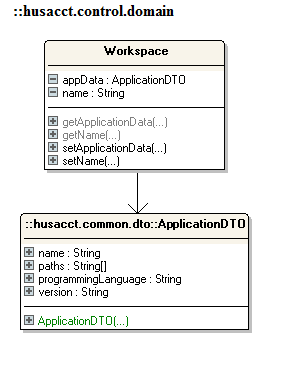
|  |  |
| --- | --- |
| **Decision** | **Justification** |
| In order to ensure the user language of the SACC-tool is independent, we will  use ResourceBundles in Java. We will implement the observer pattern to notify the other services of a language change. | NF1.1 |
| In order to ensure other development teams can easily maintain the source code of the SACC-tool, we make use of GIT combined with GITHUB. | NF4.1 |
| In order to ensure the SACC-tool is operating system independent, we develop  the GUI in Java Swing. | NF5.4 |
| To support plugin development based on the tool, the presentation and task layer are separated. | NF1.2 |
| Services can call a method to display error and/or info messages on top of the main gui | NF2.1 |
| The system keeps track of the state of the workspace, making sure that buttons are enabled/disabled at the appropriate state | NF3.1 |
| Libraries are included within the project, groups/services and testsuites are separated in packages, build scripts are all relative. This ensures that when importing the project in eclipse that all dependencies are met. | NF4.1 |
| The tool is packaged including all dependencies | NF5.2 NF5.3 |
| It was decided not to use a tabbed panes because it wasn't possible to show several windows at once. Instead a taskbar was implemented to give an overview of the opened frames. | NF3.2 |
| JDOM2 was chosen for its ease in use and the ability to merge multiple elements into one document when saving the workspace. | FR 4 FR 6 |

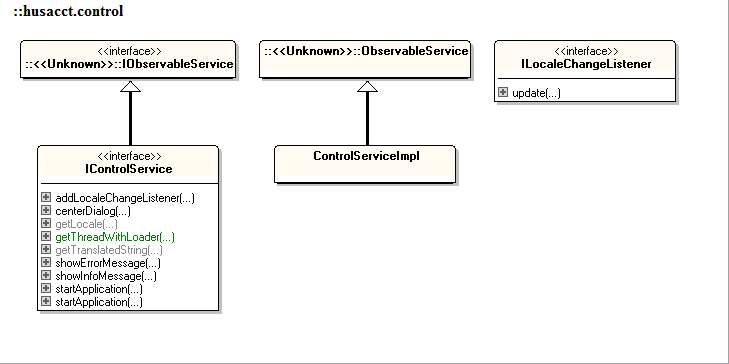
# Software partitioning

## Physical class diagram









## Physical software partitioning model



## Plugin component diagram



## Architectural Rules

|  |  |  |
| --- | --- | --- |
|  | **Rules on the contents of a module** | Restricts the contents of a module |
|  | A module can have only one parent | - |
|  | No loops in the module hierarchy | - |
|  | Unique responsibility | - |
|  | Visibility convention | Only the interfaces are visible to the other services. |
|  | Naming convention | - |
|  | Subclass convention | - |
|  | Interface convention | - |
|  | **Rules on the legality of Dependency** |  |
|  | Is allowed to use | View is allowed to use Controller and Controller is allowed to use View  Controller is allowed to use Define, Analyse, Validate and Architecture Graphics Interfaces |
|  | Is only allowed to use | - |
|  | Is the only module allowed to use | - |
|  | Must use | - |
|  | Is not allowed to | Define, Analyse, Validate and Architecture Graphics are not allowed to use Controller |
|  | … use modules in higher layer | - |
|  | … use modules in a not directly lower layer | - |
|  | Exceptions to all those rules | Define, Analyse, Validate and Architecture Graphics are allowed to give an Exception to Controller or to add an observer to LocaleService. |
|  | **Rules on Dependency Limitation** |  |
|  | Publish Subscribe pattern (or Observer) | Observer pattern is implemented. The LocaleService is the Observerable and the other services have their own observer.  A notify to the observers is send when the user language has been changed in the GUI. |
|  | Facade pattern | The different services(Define, Analyse, Validate and Architecture Graphics) do have a façade(interface) which the controller communicates with. |
|  | Data Transfer Object | - |
|  | Error messages conform X | When an export or import fails an Exception is sent to us by the appropriate service. |
|  | Communication protocol conform Y | When an export or import succeeds we get a message (Boolean) confirming the import/export. |
|  | No cycles in the dependency | - |

# Subsystem specification

## Start up

The following diagram shows the startup sequence of HUSACCT.



## ServiceProvider

The ServiceProvider keeps track of the instances of the services used within the application and is allowed to be used by any other part of the system. The ServiceProvider provides an entrypoint to the other services within the system. To make sure there is only one instance of each service available, the ServiceProvider itself is implemented as a singleton.



## StateController

The StateController is used to ensure the workflow within the application. This is done by listening to any changes within all services. Whenever a service notifies a change, the StateController will determine a new state and notify all listeners, providing the newly determined state.

A state contains a list of any of the following enumerated values located in husacct.control.task.States;

* NONE  
  *No workspace has been created or opened*
* OPENED  
  *A new workspace has been created or opened*
* DEFINED  
  *An logical architecture has been defined within the DefineService*
* APPSET  
  *Application details have been set within the DefineService*
* ANALYSED  
  *An application has been analysed within the AnalyseService*
* MAPPED  
  *An analysed application has been mapped to the logical architecture within the DefineService*
* VALIDATED  
  *A mapped application has been validated within the ValidateService*

|  |  |
| --- | --- |
| When a NONE state is determined, menu items will adjust accordingly:  Same menu items with the OPENED state: |  |

## Loading dialog

A loading dialog is provided. This to make sure that the application does not freeze while performing time consuming tasks.

The ThreadWithLoader prepares a loadingDialog, creates a separate thread for the task and a final MonitorThread. The MonitorThread will wait for the TaskThread to finish so it can close the LoadingDialog. When the user interrupts the task by closing the LoadingDialog, a signal will be sent to the TaskThread. The TaskThread will notify the monitor of any interruptions by throwing an exception. Although the mechanism to interrupt a task exists, it is not yet implemented.



## Error/info message

The ControlService provides the methods showErrorMessage(String message) and showInfoMessage(String message). With these methods generic error and info message can be created.

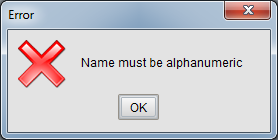


Figure . Error message

# Build

A build script is used to deploy the tool. The build script is located within the build folder and requires Apache Ant (<http://ant.apache.org/>) to run. Ant enables us to easily integrate the build script within popular continuous integration servers like Jenkins or Hudson. By running the build.ant script (using Eclipse, Ant or server), the following targets are executed in order;

* **Clean**Removes any previous binaries, builds and test reports
* **Compile**Compiles the HUSACCT application to the /bin
* **ExtractLib**Extracts the libraries used within the /lib folder to the /bin folder.
* **Jar**Creates a runnable Jar from the binaries. Additionally it creates a runnable jar from the build launcher which allows the tool to be ran with a greater heapsize.
* **Addresources**Any non-java file is ignored by the compiler. This step copies all non-java files from /husacct/common/locale and /husacct/common/resources to the /bin folder. As an additional step it validates the locale resources using the following checks;
  + Cross bundle – Checks if all keys are used in all resource bundles
  + Empty value – Checks if there are no empty values within a single resource bundle
  + Line end – Checks the line for extra whitespaces within a single resource bundle
  + Empty key – Checks if there are no empty keys within a single resource bundle
  + Duplicate key – Checks if there are no duplicate keys within a single resource bundle
* **Test**Runs all JUnit4 test suites and outputs all reports as xml, these xml reports are required to enable easy integration with continuous integration servers. Additionally, if not using a continuous integration server, the reports are transformed to html using an .xsl located in the /build/conf folder.

# Maven plugin

One of the requirements was to provide some of the HUSACCT functionality via a Maven plugin. The plugin does the following:

When using the Maven plugin, it will try to locate a HUSACCT workspace file in the same directory from where the plugin is executed. The plugin will then try to analyse and generate a violation report (xml by default) using the application details, mapping and rules defined within the workspace.

The location of the workspace file, the export method and export location can be defined using several known configuration methods from Maven. For more information about how to configure Maven properly check the official Maven website, http://maven.apache.org/.

# Testing

Tests are made with JUnit 4. The tests are bundled in Test Suites. Each service has its own test suite.

For our subsystem we have the following test classes:

* ImportExportControllerTest
* LocaleControllerTest
* ObservableServiceTest
* ServiceProviderTest
* StateControllerTest
* WorkspaceControllerTest

All of these classes can be tested at once with the test suite ControlTestSuite.

# Future work

## Known bug list

The tool does not allow big files to be opened at the moment. This is due to the fact that JDom2 is used. JDom2 is easy to use but is slow when large files must be opened.

## Ideas for improvement

* There is a lot of duplicated code to request the JInternalFrames from the other services. It might be possible to make this more efficient.
* Change ILocaleChangeListener to IServiceListeners.