VERSION 2.0

NOVEMBER 1, 2020

This document will be updated
annually from the above date

GETTING STARTED WITH VASL DEVELOPMENT

A STEP-BY-STEP GUIDE

PREPARED BY DOUG RIMMER

CONTACT FOR QUESTIONS ABOUT THIS DOCUMENT:
DOUGRIMMER@ROGERS.COM

GETTING STARTED WITH VASL DEVELOPMENT

TABLE OF CONTENTS

Update History

September, 2016	Minor text edits
March, 2017	Minor text edits related to Idea IDE changes
March, 2018	Minor text edits related to Idea IDE changes
October, 2020	Major rewrite to reflect changed processes and tools

DOCUMENT OVERVIEW

This document provides instructions on obtaining, installing, and using a number of tools necessary to enable VASL development.

This is a multi-step process involving a number of separate tools and products and can appear quite complex. This document tries to provide sufficient detail to allow those interested to get up and running on their own. In the event of problems requiring assistance, contact information is provided.

The document is written to enable those unfamiliar with such tools to create an environment in which they can contribute to further development of VASL. Experienced users will find that they can skip some of the introductory text in each section and proceed directly to the detailed instructions.

For those interested in creating VASL Boards, see separate instructions on the VASL wiki within GitHub (see Section 3 below for information on GitHub).

GETTING STARTED OVERVIEW

In order to work on VASL source code, a number of steps must be completed. The VASL source code is a library, not a complete standalone application. The library is used by VASSAL (http://www.vassalengine.org), which is a standalone application. Development will require the VASL source code along with VASSAL.

The source code is written in Java. To enable access to this, it is necessary to install a version of the Java Development Kit (JDK), which includes tools for developing, debugging, and monitoring Java applications.

To obtain the source code for VASL and to manage changes made to the code, it is necessary to use the GitHub Repository. This requires creating a Git user account.

To actually develop new code requires an Integrated Development Environment (IDE). Several IDE's can be used to provide and this document provides instructions for one of them: IntelliJ.

In order to integrate certain changes, it is necessary to 'build' the VASL module. This document provides instructions for doing so using a tool named Maven which has to be obtained and installed.

While many of the above tools can be purchased, all have "community" or free-access versions that can be used for VASL development. No purchases are necessary.

Finally, this document provides more detailed instructions on how to obtain, install and use all of the above-mentioned tools.

1.0 VASL AND VASSAL

1.1 THE VASL MODULE

The VASL source code is a library, not a complete standalone application. The VASL source code is compiled into the VASL module, which is used by VASSAL to provide the ASL game interface via internet. Usage of the source code is subject to the terms of the Library Gnu Public License (LGPL), described in the LICENSE.txt file and also available from http://www.opensource.org.

1.2 THE VASSAL ENGINE

VASSAL is a standalone application which is used in conjunction with modules such as VASL to provide game interfaces. Neither VASSAL nor any of its modules is a complete game in itself. Both VASSAL and a game module are required in order to play.

1.2.1 VASSAL under the covers: why you need it and what it does

VASL is one of many game modules developed for the VASSAL game engine. The VASSAL module editor is used to add and update VASSAL capabilities, examples of which would include counters, map window options (e.g. zoom), etc.

VASL is one of a small number of modules that contain a lot of custom Java code to enhance and augment standard VASSAL features. For example, concealing a stack is performed via custom VASL code.

VASL "development" could involve either of the above.

1.2.2 VASSAL Module documentation

If you wish to know more about VASSAL functionality and how it works, please consult the VASSAL documentation. There are also a series of YouTube videos about VASSAL, some quite detailed. A solid understanding of VASSAL module development should be considered a prerequisite to VASL development.

1.2.3 Obtaining VASSAL and using it from within the VASL source code

The VASSAL files necessary for VASL code development are included with the VASL source code library and will be copied to your computer with the rest of the source code library (see Section 3.1.3). Strictly speaking, it is not necessary to have VASSAL installed on your computer to edit VASL by changing the source code. However, it is highly recommended.

1.2.4 Using the VASSAL Module Editor

The VASSAL module editor comes with VASSAL program that you install to play VASL. See the VASSAL documentation for more details.

2.0 GETTING THE UNDERLYING TOOLS: SDK

2.1 SETTING UP THE JAVA SOFTWARE DEVELOPMENT KIT (SDK)

Section 1.2.1 describes how Java code is used in the VASL module to enhance standard VASSAL features. In order to be able to use Java code from within VASL, a Java Software Development Kit (SDK) must be installed and then referenced. Details on how to reference the SDK in VASL projects will be included in Section 4.1.3.

2.1.1 The SDK: why you need it and what it does

The SDK works primarily behind the scenes in VASL development to enable the use of Java code in the VASL module. The SDK includes tools for developing, debugging, and monitoring Java applications which are exposed to the developer via their Integrated Development Environment (see Section 4).

2.1.2 Getting the SDK

The JDK can be found at https://www.oracle.com/java/technologies/javase-downloads.html

If this link does not work, search for the Java Software Development Kit. As of the date of this documents, the version of VASSAL used to create the current VASL module, is VASSAL3.4.6 and this requires Java 11 JDK. This is likely to change in the future and should be checked before download/installation.

2.1.3 Installing the SDK

Java offers several versions of the SDK. Java SE 11.0.8 is the latest version to be successfully used with VASL. When newer versions appear, feel free to try them. If you encounter problems, install and reference the version cited here.

Complete the download and installation of the SDK by following the screen prompts as you would for any software install. It may be helpful to note the directory in which the SDK is installed as you will need to be able to find it when adding Java references in Section 4.

In order to validate the SDK install and to ensure that Java is in the path (this will be needed in Section 6), go to the command line and type “java -version” without the quotation marks. If you don’t know how to find the command line, please Google it as there are many ways to get there depending on computer types and OS versions. If typing “java –version” does not return a reference to Java then you will need to add it to your path manually. Since methods of doing so can vary by OS, please check the internet for details.

Finally, verify that the Environment Variable JAVA_HOME exists on your computer and points to the installation directory. Since methods of doing so can vary by OS, please check the internet for details.

3.0 GETTING THE CODE: GITHUB

The VASL source code is stored in a code repository called Git.

3.1 THE GIT CODE REPOSITORY

A Code Repository is essentially a web-based file storage location that includes tools for managing access to files and updates to them, and is specially designed to support software development by teams of people.

3.1.1 Code Repository: why you need it and what it does

Git is used to store the VASL source code. In addition, Git is a version control system (VCS) that records changes to a file or set of files over time so that specific versions can be recalled later. A VCS allows developers to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if someone screws things up or loses files, it is possible to easily recover.

A short primer on Git is provided at Annex A.

3.1.2 How to install and access GitHub

3.1.2.1 GitHub

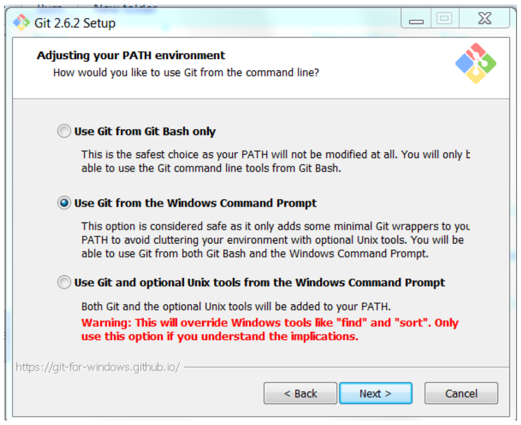
Create a Github account at https://github.com/.

3.1.2.2 Git

Git must be installed on your computer. It can be found at http://www.git-scm.com/ or by googling 'git-scm' if the link does not work. Git-scm is available for most major operating systems. Choose the latest version for your system.

You may see references to the GitHub desktop or GitHub GUI. For the purposes of VASL development, it is NOT recommended that you install this software.

When installing git-scm on Windows use the default options with the exception of allowing git commands from the command prompt.



Git 2.6.2 Setup

Adjusting your PATH environment

How would you like to use Git from the command line?

( ) **Use Git from Git Bash only**

This is the safest choice as your PATH will not be modified at all. You will only b able to use the Git command line tools from Git Bash.

[✓ **Use Git from the Windows Command Prompt**]

This option is considered safe as it only adds some minimal Git wrappers to you PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

( ) **Use Git and optional Unix tools from the Windows Command Prompt**

Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

3.1.3 Cloning the VASL code repository for local development

1. In order to update the VASL code repository, it is necessary to join the vasl-developers team on GitHub. To do so, message the contact email address on the cover page of this

document. Joining vasl-developers is only necessary if you wish to have "write" access to the repository so that you can contribute code. If you just want to clone a repository and code and build separately, you don't need to join the team.

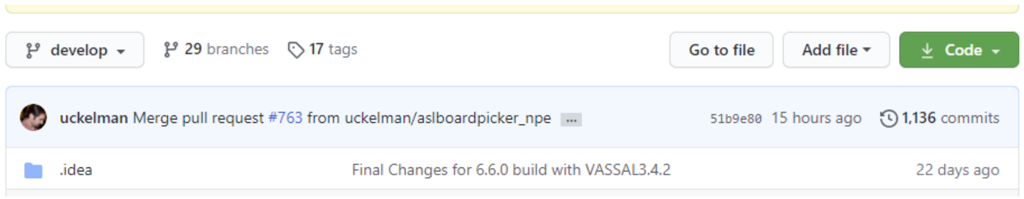
2. Once, you have received confirmation that you are a member of the group, sign in to GitHub.

3. Search for "VASL".

4. Select "vasl-developers/vasl" (click on the 'vasl' at the end as it is a separate hyperlink).

5. Click on Branch-develop and type a name for a new branch in the textbox then click on "Create Branch:". If "Create Branch:" does not appear, check that you are a member of the vasl-developers group. For those wishing to develop and contribute code, please start your branch name with the prefix "feature/".

6. Copy the HTTPS Clone URL for your branch to the Clipboard by clicking on the "Code" button on GitHub and copying the URL for the text box that will display.



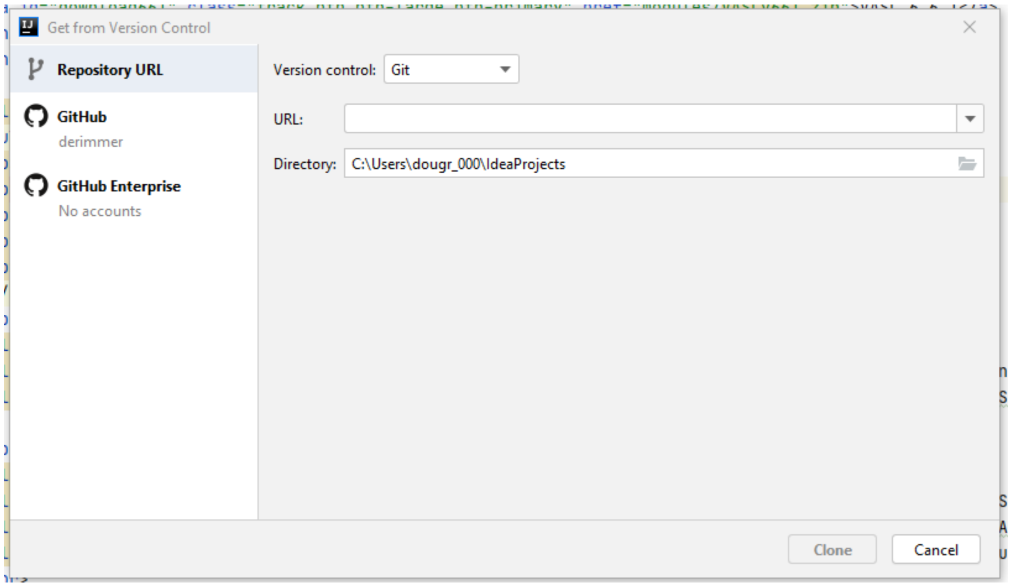
uckelman Merge pull request #763 from uckelman/aslboardpicker_npe ... 51b9e80 15 hours ago 1,136 commits

.idea Final Changes for 6.6.0 build with VASSAL3.4.2 22 days ago

7. While other software can be used, it is possible to clone the VASL code repository from within the IDE, depending on the IDE.

8. In the case of IntelliJ, which is the recommended IDE, click the File menu and then select New -> Project From Version Control.

9. Paste the HTTPS Clone URL for your branch from Step 6 into the "URL" textbox.



Version control: Git

URL:

Directory: C:\Users\dougr_000\IdeaProjects

10. Select a destination location by typing in the Directory text box or clicking on the Folder icon (which will allow you to create a new folder if desired). It is recommended that you use a directory where your IDE will store projects.

11. Click Clone.

The cloning process will take several minutes depending on your computer.

4.0 WORKING WITH THE CODE: INTELLIJ

4.1 THE INTELLIJ IDE

Integrated Development Environments (IDE’s) provide a means of linking a number of tools together to allow developers to write software code and then manage their work by linking to project management tools and repositories.

4.1.1 IDE’s: why you need one and what they do

Above all, the IDE is the place where code gets written. IDE’s link to underlying software development tools (such as the JDK) that then provide developers with access to coding languages, structures and libraries. They also integrate tools that allow developers to link their work to larger teams and projects.

A number of IDE’s can be used to write VASL source code in Java, for example, Eclipse. The following instructions apply specifically to another one, IntelliJ. They could be used as a guide to working with Eclipse or other IDE’s. It is however recommended that IntelliJ be used.

4.1.2 Installing an IDE: IntelliJ

To install IntelliJ:

1. Go to https://www.jetbrains.com/idea/download/

2. Download the Community Edition version for appropriate platform (Windows, Mac, etc.)

3. Run install as normal for the platform (i.e. on Windows, click “Open” or “Run” in the dialog boxes)

4.1.3 Setting up IntelliJ for development

1. Create the VASL project

Clone the VASL code repository using IntelliJ as per Steps 7-11 of Section 3.1.3.

The project will open in IntelliJ. Before anything else, you should create a link to the branch you created in Github in Section 3.1.3 Step 5.

In the bottom right corner of IntelliJ, click on “develop”. This is the Github master directory from which you cloned the code. From the list that appears, under Remote Branches, click on the branch that you created. From the menu that appears, select Checkout.

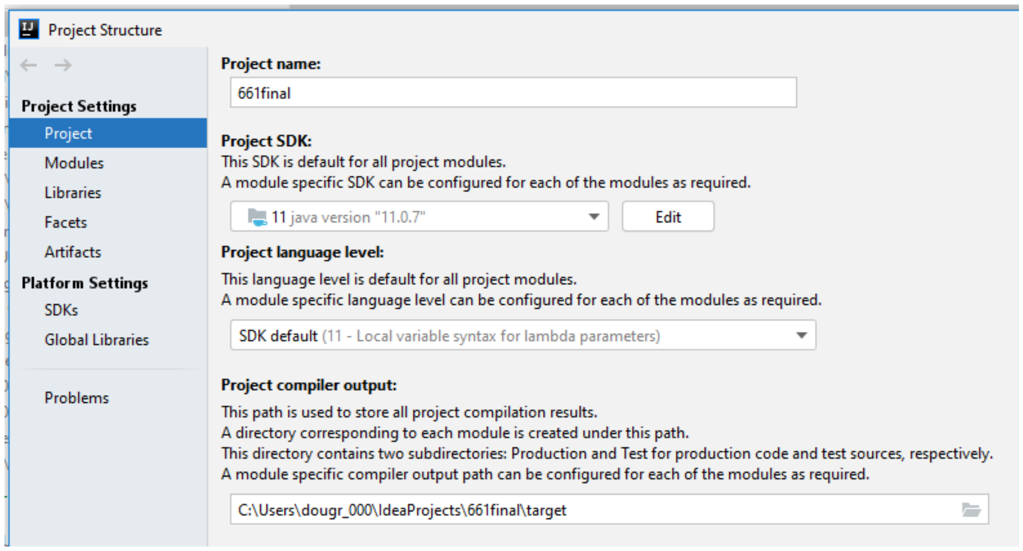
This means that when you commit changes back to Github they will be sent to your branch and must then be merged into develop on Github. This is a good practice when managing a code base with multiple contributors. Commits should never be pushed directly to develop.

2. Configure the VASL project

You will need to configure the project in order to enable tools such as the JDK and VASSAL code to be accessed when coding.

Select Menu File →Project Structure. Select Project Settings and check that Project SDK is the one that you installed (see Section 2.1.3). If not, click the dropdown or "New" button to select the Java JDK.

Check that the Project language level dropdown is set to "SDK default".



Project name:

661final

Project SDK:

This SDK is default for all project modules.
A module specific SDK can be configured for each of the modules as required.

11 java version "11.0.7"

Project language level:

This language level is default for all project modules.
A module specific language level can be configured for each of the modules as required.

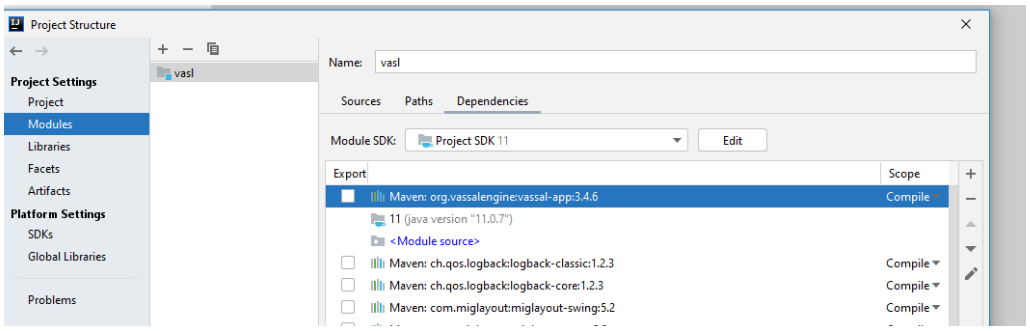
SDK default (11 - Local variable syntax for lambda parameters)

Project compiler output:

This path is used to store all project compilation results.
A directory corresponding to each module is created under this path.
This directory contains two subdirectories: Production and Test for production code and test sources, respectively.
A module specific compiler output path can be configured for each of the modules as required.

C:\Users\dougr_000\IdeaProjects\661final\target

Click the Modules tab and then the Dependencies tab.



Name: vasl

Module SDK: Project SDK 11

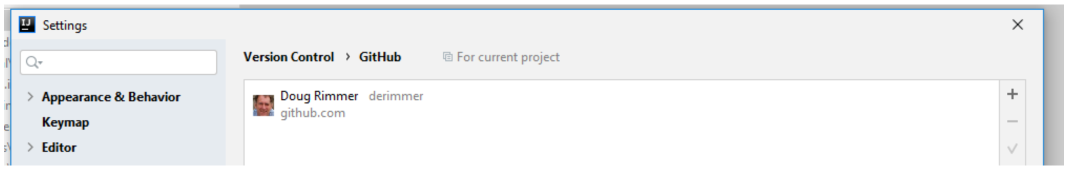
Export		Scope
☐	Maven: org.vassalengine:vassal-app:3.4.6	Compile
	11 (java version "11.0.7")	
	<Module source>	
☐	Maven: ch.qos.logback:logback-classic:1.2.3	Compile
☐	Maven: ch.qos.logback:logback-core:1.2.3	Compile
☐	Maven: com.miglayout:miglayout-swing:5.2	Compile

If the Java item is missing, click the green + sign and select "Library"→ "Java" and navigate to the item, then click OK. If you have trouble finding the SDK, look in directories where software files are usually kept (e.g. directories with "Program Files" in their name in Windows).

Ensure that the Maven: org.vasslaengine:vassal-app:3.4.x item is at the top of the list (it usually initially appears below the <Module Source> item; use the up arrow to move it to the very top of the list. Click OK.

To ensure that the link to Git is in place, select File -> Settings and on the next screen select Version Control and then Git. Verify that the correct path to your installation of Git (see Section 3.1.2.2) is listed in the Path to Git executable textbox. If not add it by typing the path or navigating to it via the Directory box beside the textbox. Click Test to verifying the linkage.

For GitHub, click on the GitHub tab. Make sure that you appear in the window. If not, click on the + sign to add yourself.


Settings

Appearance & Behavior
Keymap
Editor

Version Control › GitHub   For current project

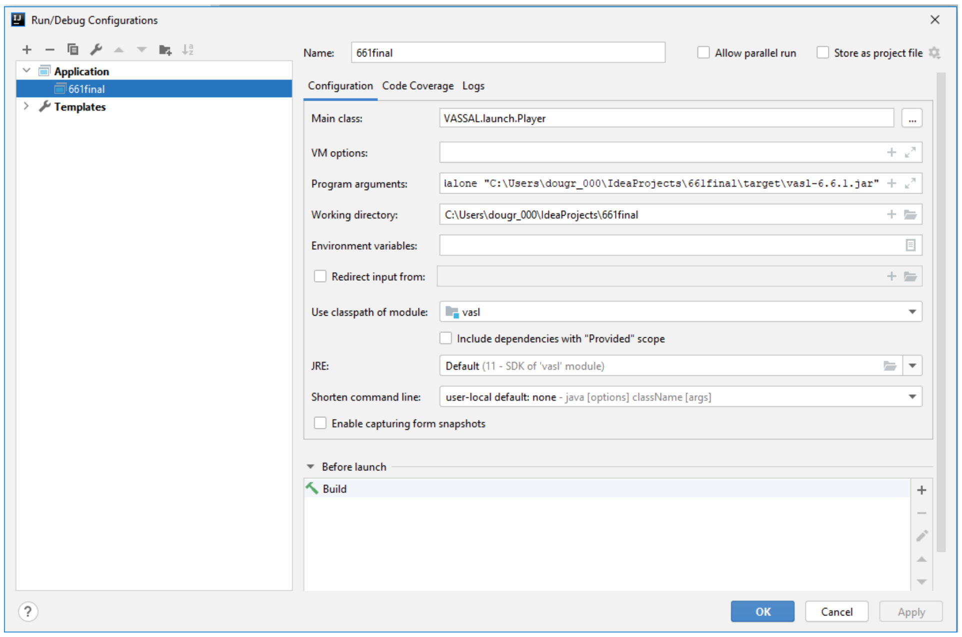
Doug Rimmer   derimmer
github.com

3. Run the VASL Project

To run the project, (to verify that code changes are working), certain Run configurations must be set. Select Menu Run → Edit Configurations. If no item is listed in left-hand listbox, click the green + sign and select “Application”. Enter a name in the Name textbox and “VASSAL.launch.Player” in the Main Class textbox.

Enter --standalone "C:\Users\<your project path>\target\vasl-6.6.x.jar" in the Program Arguments textbox, using the directory where your current module file is located. Note that versions numbers (“6.6.x ”) will change over time. This must point to an actual .jar or .vmod file so when running a module for the first time it is necessary to either build the module first (see Section 6.1) or point to already existing file (such as the one used to play VASL games).

Ensure that the Working Directory textbox refers to the folder where the current project is located. If not, type in the folder name or navigate to it using the Directory button beside the textbox. Select the classpath of module from choices available from the dropdown arrow; do not select “test”. Instead use the option similar to your project name.


Run/Debug Configurations

Application
661final
Templates

Name: 661final   Allow parallel run   Store as project file

Configuration   Code Coverage   Logs

Main class: VASSAL.launch.Player
VM options:
Program arguments: lalone "C:\Users\dougr_000\IdeaProjects\661final\target\vasl-6.6.1.jar"
Working directory: C:\Users\dougr_000\IdeaProjects\661final
Environment variables:
Redirect input from:
Use classpath of module: vasl
Include dependencies with "Provided" scope
JRE: Default (11 - SDK of 'vasl' module)
Shorten command line: user-local default: none - java [options] className [args]
Enable capturing form snapshots

Before launch
Build

OK   Cancel   Apply

Click OK.

Select Menu Run → Run or Debug “Your project name” to run the project.

5.0 MANAGING THE CODE

5.1 USING THE IDE TO CONTROL CODE CHANGES

While writing new code is one thing, ensuring that it integrates properly with the existing code base and new code being developed by other developers on the team is another issue.

5.1.1 Code Management: why we need it and what you must do

Good code management ensures that no work is wasted and no conflicts enter the code base due to uncoordinated efforts of multiple developers. All developers must access the VASL code and commit changes properly in order to maintain an integrated set of code files that will produce the VASL module for use by ASL players.

5.1.1 Creating a branch

Developers should create a branch and use it for specific coding tasks before proceeding to write code. Branches should be created before cloning the source code from Git directly into IntelliJ. See Section 3.1.3 Step 5 for instructions.

5.1.3 Committing changes to your branch

Once you have cloned the code repository and created project within the IDE, you can now make changes to the code.

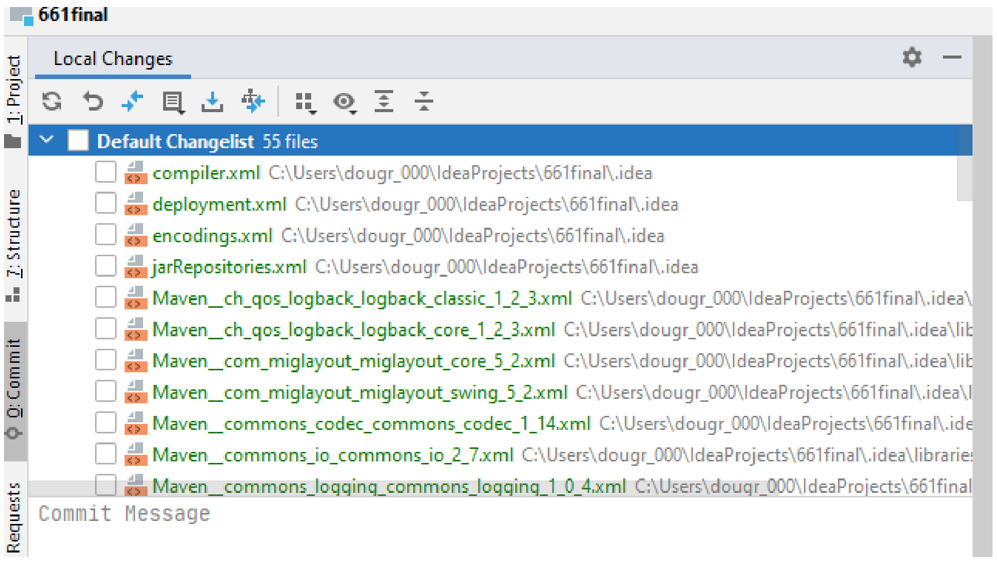
To integrate those changes into your branch in the VASL code repository in Git:

If you wish to add new files to the repository, first right-click on the file in the project pane and use the “Git →Add” menu option.

1. Click Menu VCS → Commit or select the Commit tab from the sidebar.

2. Ensure that the files you have changed and wish to commit show up in the Changelist. Deselect those files that you don’t want to send to GitHub or do not need to be sent (generally, this includes .xml configuration files). Be sure to add a Comment to enable others to understand what changes you are committing. Developers should include the issue number in the commit message.

3. Click Commit and Push from the Commit dropdown button.



```
Default Changelist 55 files
compiler.xml C:\Users\dougr_000\IdeaProjects\661final\.idea
deployment.xml C:\Users\dougr_000\IdeaProjects\661final\.idea
encodings.xml C:\Users\dougr_000\IdeaProjects\661final\.idea
jarRepositories.xml C:\Users\dougr_000\IdeaProjects\661final\.idea
Maven__ch_qos_logback_logback_classic_1_2_3.xml C:\Users\dougr_000\IdeaProjects\661final\.idea\
Maven__ch_qos_logback_logback_core_1_2_3.xml C:\Users\dougr_000\IdeaProjects\661final\.idea\lib
Maven__com_miglayout_miglayout_core_5_2.xml C:\Users\dougr_000\IdeaProjects\661final\.idea\lib
Maven__com_miglayout_miglayout_swing_5_2.xml C:\Users\dougr_000\IdeaProjects\661final\.idea\l
Maven__commons_codec_commons_codec_1_14.xml C:\Users\dougr_000\IdeaProjects\661final\.ide
Maven__commons_io_commons_io_2_7.xml C:\Users\dougr_000\IdeaProjects\661final\.idea\librarie
Maven__commons_logging_commons_logging_1_0_4.xml C:\Users\dougr_000\IdeaProjects\661final
Commit Message
```

4. A warning message may appear if the IDE detects problems with the code changes.

5. An additional screen or two will appear to complete the commit and push.

This process often changes and so may vary from the above steps.

6.0 BUILDING THE VASL MODULE: FROM MAVEN

6.1 BUILDING THE VASL MODULE

Re-building the VASL module is not required to test the impact of your changes on the overall code unless you modified any of the files in the "dist" folder (note there is no code there). If you add counters, the new images must be added to the dist directory. In these cases, you must build the VASL module before these changes will show up when running the project in the IDE.

6.1.1 Building VASL: why you need it and what you need to do

You should build the VASL module if you added or changed files in the “dist” folder. This same build process is used to produce the final .vmod file for a beta or official release. This requires the use of software called Maven, which must be installed and then used to create the VASL module.

6.1.2 Getting and setting up Maven

Maven can be downloaded from https://maven.apache.org/download.cgi. Once downloaded, click on the Install link on the page and then follow the install instructions. While not difficult, there are several particular steps that must be followed and it is preferable to use the instructions on the Maven site.

For convenience add the Maven bin folder to you path. This will allow you to build the VASL module from the command line and the terminal window in IntelliJ. To confirm Maven has been properly installed type “mvn –v” at the command line to display the installed version.

See the pom.xml file in the VASL project for the current version of Maven to install.

6.1.3 Building the VASL module from Maven

1. After installing Maven, copy the "settings.xml.TEMPLATE" document found in the directory to which you cloned your VASL branch in Section 3.1.3 Step 10 to the .m2 directory in your home folder and rename to "settings.xml". Your “home folder” may vary depending on OS. For Windows users, for example, look in the “C:\Users\YourUserName” directory.

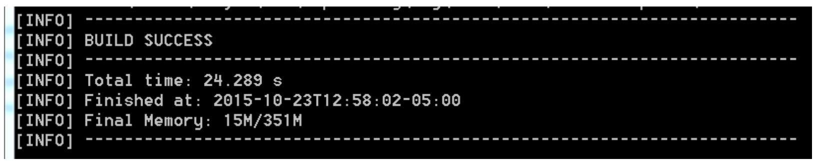
2. Rename the file in the .m2 directory to “Settings.xml”. Open the file, using a text editor such as Notepad on Windows. Edit the file by changing the path to your local JDK installation (see Section 2.1.3).

3. Go to the command line. How you do this will depend on your operating system. Check its help system or the web for details. In Windows, for example, press the Winkey + R then type cmd in the dialog box that appears.

4. At the command line, change the directory to your project directory. To do so, type “chdir DirectoryName” at the command prompt where DirectoryName is the project directory (full path required).

Example: chdir C:\Users\dougr_000\IdeaProjects\STtest

5. To create the VASL module, type "mvnw clean package" at the command prompt. This process may take several minutes. If successful, the final messages displayed will be similar to this:



```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 24.289 s
[INFO] Finished at: 2015-10-23T12:58:02-05:00
[INFO] Final Memory: 15M/351M
[INFO] ------------------------------------------------------------------------
```

6. The build creates a .jar file in the \target directory of the project directory referenced in Step 4. This .jar file can be used to in the “Program arguments” text box referred to in Section 4.1.3.

7. To produce a .vmod file that can be run from the VASSAL application, copy the jar file to a new location and then simply change the .jar extension to .vmod.

Annex A

A Git primer

Below is a short primer on Git, based on “Getting Started with Git”, a more detailed resource found on the Git-scm website at http://git-scm.com/book/en/v2/Getting-Started-About-Version-Control.

As mentioned above, Git is a version control system that allows team development of software code while managing code updates, code conflicts, and rollbacks of changes if required. As a distributed VCS, Git provides each user with a set of code files that fully mirror those in the specific Git repository being used.

As a result, most operations in Git only need local files and resources to operate. Because you have the entire history of the project right there on your local disk, most operations seem almost instantaneous.

Once the VASL source code repository is downloaded to a local computer (see Section 3.1 Step 5), Git has three main states that the local files can reside in: committed, modified, and staged. Committed means that the data is safely stored in the local database. Modified means that the file has been changed but not yet committed to the local database. Staged means that a modified file in its current version has been marked to go into the next commit snapshot.

This leads to the three main sections of a Git project: the Git directory, the working directory, and the staging area.

The Git directory is where Git stores the metadata and object database for a project. This is the most important part of Git, and it is what is copied when a repository is cloned from another computer.

The working directory is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for use or modification.

The staging area is a file, generally contained in the Git directory, that stores information about what will go into the next commit. It’s sometimes referred to as the “index”, but it’s also common to refer to it as the staging area.

The basic Git workflow goes something like this:

1. Modify files in the working directory.
2. Stage the files, adding snapshots of them to the staging area.
3. Do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to the Git directory.

If a particular version of a file is in the Git directory, it’s considered committed. If it has been modified and was added to the staging area, it is staged. And if it was changed since it was checked out but has not been staged, it is modified.

There are a lot of different ways to use Git. There are the original command line tools, and there are many graphical user interfaces of varying capabilities. Some knowledge of Terminal in Mac or Command Prompt or Powershell in Windows would be useful.