

# Minimal Value Types Command-line Options

This page describes the command line for the experimental java compiler, the java launcher and HotSpot for the support of Minimal Value Types in the valhalla project.

## Minimal Value Types flags:

	Flag	Description
<b>hotspot</b>		
<b>REQUIRED</b>	<b>-XX:EnableMVT</b>	Boolean value, default value is `false` If true enable support for minimal value types with a value capable class and an associated derived value type.
<b>REQUIRED</b>	<b>-XVerify:none</b>	Disable verification. Verification is not yet supported.
	<b>-XX:ValueArrayFlatten</b>	Boolean value, default value is `true` Flatten arrays of values, if possible
	<b>-XX:ValueArrayElemMaxFlatSize</b>	Integer value, default is -1 Max size for flattening an array of values, < 0 means no limit
	<b>-XX:ValueArrayElemMaxFlatOops</b>	Integer value, default is 4 Max number of embedded object references in value type to flatten in an array, < 0 means no limit
	<b>-XX:BigValueTypeThreshold</b>	size_t value in bytes, default is 4 * BytesPerLong  Max value type size for interpreter buffering of local variable table entries
	<b>-XX:ValueTypesBufferMaxMemory</b>	Integer value, default 128 (pages)  Max memory used for value type buffers
	<b>-XX:ValueTypesThreadLocalRecycling</b>	Boolean value, default value is 'true'  Enable thread local recycling of buffered values in the interpreter
	<b>-XX:MinimumVTBufferChunkPerFrame</b>	Integer value, default 2  Minimum number of VT buffer chunks allowed per frame
	<b>-XX:ReportVTBufferRecyclingTimes</b>	Boolean value, default false  Print duration of each VTBuffer recycling for the interpreter
	<b>-XX:ValueTypePassFieldsAsArgs</b>	Non-product: Boolean value, default value is `true` Pass each value type field as an argument to a method call instead of a value type reference
	<b>-XX:ValueTypeReturnedAsFields</b>	Non-product: Boolean value, default value is `true` Return value type fields instead of a value type reference
	<b>-XX:ValueArrayAtomicAccess</b>	Boolean value, default value is `false` Enable atomic access to values in an array, by treating value types as references.
	<b>-XX:-UseTLAB</b>	To see heap allocations which do not use optimized Thread Local Allocation Buffers
	<b>-XX:+PrintEliminateAllocations</b>	Non-product builds: Print out when allocations are eliminated
	<b>-XX:+PrintEscapeAnalysis</b>	Non-product builds: Print results of escape analysis (e.g. if you believe boxing was not eliminated)
<b>java</b>		
	<b>-Djava.lang.invoke.MethodHandle.DUMP_CLASS_FILES</b>	Boolean value, default value is `false` If true dump class files generated for lambda forms and derived value types  into a directory named DUMP_CLASS_FILES under the current directory.
	<b>-Dvalhalla.enableValueLambdaForms</b>	Boolean value, default value is `true` If true then lambda forms with value types in their signature will use the Q type __Value  and value-type specific byte code will be generated.

	-Dvalhalla.dumpProxyClasses	Boolean value, default value is `false` If true, dumps MVT lambda forms.
	-Dvalhalla.enablePoolPatches	Boolean value, default value is `false` If true and if "valhalla.enableValueLambdaForms" is true then lambda forms with value types in their signature  will generate byte code with constant pool patching where appropriate.
<b>javac</b>		
	-Xlint:values	Lint category for warnings associated to bad usage of value capable classes - examples: <ul style="list-style-type: none"> <li>• VCC not final</li> <li>• VCC cannot 'extend' another class</li> <li>• overrides of 'bad' Object methods (such as wait/notify)</li> <li>• illegal modifiers on VCC such as 'synchronized'</li> <li>• non-final instance fields in VCC</li> <li>• cannot assign 'null' to variable of VCC type</li> </ul>