Using AUTA

Contents

[General AUTA Information 2](#_Toc458679916)

[AUTA Command Lines 2](#_Toc458679917)

[AUTA Identifiers 2](#_Toc458679918)

[AUTA Delimiting 2](#_Toc458679919)

[AUTA Parse Order 2](#_Toc458679920)

[AUTA Command Blocks 2](#_Toc458679921)

[AUTA Extra Commands 3](#_Toc458679922)

[AUTA Caching and Output Commands 3](#_Toc458679923)

[AUTA Command Block Types 4](#_Toc458679924)

[AUTA Import/Export Command Block 4](#_Toc458679925)

[Cache Commands 4](#_Toc458679926)

[Output Commands 4](#_Toc458679927)

[AUTA Import/Export Group Command Block 5](#_Toc458679928)

[Cache Commands 5](#_Toc458679929)

[Output Commands 5](#_Toc458679930)

[AUTA Types Command lines 6](#_Toc458679931)

[Cache Commands 6](#_Toc458679932)

[AUTA Labels Command block 7](#_Toc458679933)

[Cache Commands 7](#_Toc458679934)

[AUTA Scope Command block 8](#_Toc458679935)

[Cache Commands 8](#_Toc458679936)

[Extra Commands 8](#_Toc458679937)

[AUTA Code Replication Command block 9](#_Toc458679938)

[Cache Commands 9](#_Toc458679939)

[Output Commands 9](#_Toc458679940)

[AUTA Function Command block 11](#_Toc458679941)

[Cache Commands 11](#_Toc458679942)

[Output Commands 11](#_Toc458679943)

# General AUTA Information

AUTA is a Code processing tool. Broadly it supplies expanded and superior macro functionality. One of the major benefits of AUTA is that code is expanded and inserted into the code. This makes it clear to the programmer, what the macro has done and allows clear straightforward debugging of macro’d code. The functionality is easy to expand and will be growing in the future.

# Using AUTA

AUTA generates an exe, which should be passed a directory path as its first argument. It will recursively pass all folders, processing all files ending in **.h**,**.cpp** or **.cu**. AUTA can be easily expanded to work on other languages (including other file extensions), it will just needs to be updated to interpret the appropriate comment syntax.

# AUTA Command Lines

An AUTA command line should be within a comment. An AUTA Command line should begin with #AUTA and an AUTA command line should contain nothing else but whitespace and comment controllers (// and /\* \*/).  
As such it is of the form:

* #AUTA CommandBlockType CommandSyntax
* // #AUTA CommandBlockType CommandSyntax
* /\*#AUTA CommandBlockType CommandSyntax \*/
* /\*#AUTA CommandBlockType CommandSyntax
* #AUTA CommandBlockType CommandSyntax\*/

# AUTA Identifiers

AUTA Identifiers are identical to C++ identifiers. They are case sensitive and must begin with either an alphabet character or an underscore (\_). After that they can contain any alphanumeric character or underscore. Identifiers cannot be immediately preceded or followed by any valid Identifier character.

# AUTA Delimiting

AUTA uses the \ character as a delimiting character. This tells AUTA that the following ‘character’ is not to be interpreted as AUTA syntax. The \ will be stripped when AUTA parses the code. This is used for delimiting commas in comma separated lists (\,) and delimiting AUTA concatanation markers (\##)

# AUTA Concatenation

AUTA allows for concatenation as per C++ macros. This allows tokens to be identifiable as tokens, but once processed end up densely packed with items around them. ## and its surrounding whitespace will be entirely removed. If you wish the ## to NOT be concatenated by AUTA then it should be proceeded by an AUTA delimiting character (i.e. \##)

# AUTA Parse Order

AUTA guarantees that all blocks which supply information to AUTA will be parsed before any block can request information from AUTA. This means that declaration order of AUTA blocks is irrelevant.

# AUTA Command Blocks

AUTA is entirely controlled through Command blocks. An AUTA command block begins with a block declaration line and ends with a block end line. All the lines between the begin command line and the end command line will be processed by AUTA. Nothing outside an AUTA Command block will be processed.

#AUTA CommandBlockType BlockIdentifier ExtraCommands  
#AUTA end BlockIdentifier

CommandBlockType and BlockIdentifier must be valid AUTA Identifiers. ComandBlockType must also be a specified AUTA command block keyword.

It is an error if AUTA cannot find a matching end of an AUTA block.  
It is also an error if a line which contains #AUTA is not an interpretable AUTA line.

## AUTA Extra Commands

AUTA extra commands are appended after the BlockIdentifier and are of the form   
Command = CommandSyntax  
  
Multiple extra commands can be called from a single Command Block. Currently there is no enforcement of the order that extra commands are applied.

## AUTA Caching and Output Commands

AUTA Command blocks have two types of Commands Cache and Output:

* Cache Commands will cache data in AUTA ready for processing. Cache Command Blocks will not change any file contents.
* Output Commands will replace any contents between the block start line and block end line with the new generated contents. If the contents of a file has not been altered by AUTA the file will not be written to minimise recompiling.

# AUTA Command Block Types

## AUTA Import/Export Command Block

Import Export is a very simple functionality to allow copying code from one place to another. This Allows the programmer to duplicate or move snippets and blocks of code automatically from one place to another.  
  
Exporting multiple blocks with the same BlockIdentifier is an error.

### Cache Commands

#### export

* This will export the entire contents of the Command Block to AUTA.
* The block will be tagged with the BlockIdentifier Tag.
* AUTA will process concatenation symbols.

### Output Commands

#### import

* This will import the code exported with the matching BlockIdentifier Tag.

/\*

#AUTA export Group1

TypeProxyOf<Object1>::GetTypeProxy();

#AUTA end Group1

\*/

//#AUTA import Group1

TypeProxyOf<Object1>::GetTypeProxy();

//#AUTA end Group1

## AUTA Import/Export Group Command Block

Import Export is a very simple functionality to allow copying code from many places and pulling them together automatically. This Allows the programmer to duplicate or move multiple snippets and blocks of code automatically from many places to another.  
  
Exporting multiple blocks with the same BlockIdentifier is NOT an error.

### Cache Commands

#### exportgroup

* This will export the entire contents of the Command Block to AUTA.
* The block will be tagged with the BlockIdentifier Tag.
* AUTA will process concatenation symbols.

### Output Commands

#### importgroup

* This will import the code exported with the matching BlockIdentifier Tag.

/\*

#AUTA exportgroup TypeProxyInitialistation

TypeProxyOf<Object1>::GetTypeProxy();

#AUTA end TypeProxyInitialistation

\*/

/\*

#AUTA exportgroup TypeProxyInitialistation

TypeProxyOf<Object2>::GetTypeProxy();

#AUTA end TypeProxyInitialistation

\*/

/\*

#AUTA exportgroup TypeProxyInitialistation

TypeProxyOf<Object3>::GetTypeProxy();

#AUTA end TypeProxyInitialistation

\*/

//#AUTA importgroup TypeProxyInitialistation

TypeProxyOf<Object1>::GetTypeProxy();

TypeProxyOf<Object2>::GetTypeProxy();

TypeProxyOf<Object3>::GetTypeProxy();

//#AUTA end TypeProxyInitialistation

## AUTA Types Command lines

AUTA Types command block is not globally accessible. It can only be used via another AUTA command block.  
  
AUTA types line is a list of type identifiers, each with a set of different strings to use for each type. when code is passed to the Types Command Block it will produce every permutation of the supplied type identifiers.  
  
This also allows specific permutations to be blacklisted and not generated.

### Cache Commands

#### type

* Defines a type identifier and the list of strings that should replace it

#### blacklisttypes

* Defines a specific combination of TypeIdentifiers and strings that should NOT be produced
* Any TypeIdentifiers not supplied to the blacklist are ignored when determining if a permutation is blacklisted.

#AUTA type R Float64, Int64

#AUTA type TA Float64 , Int64

#AUTA type TB Float64 , Int64

#AUTA blacklisttypes R == Float64 , TA == Int64 , TB == Int64

#AUTA blacklisttypes R == Int64 , TA == Float64

#AUTA blacklisttypes R == Int64 , TB == Float64

## AUTA Labels Command block

AUTA Labels command block is not globally accessible. It can only be used via another AUTA command block.  
  
AUTA labels block is a list of label identifiers. It also contains a list of possible contents for those labels. The Code supplied to AUTA is replicated once for each label set, replacing all labels with the contents of each set. This does not perform permutations.

### Cache Commands

#### label

* Defines the start of an AUTA labels block.
* Label blocks do not have tags and are assumed to be used within the scope they are supplied in
* An AUTA scope cannot have more than one labels block.
* Should be followed by a comma separated list of label identifiers.

#### labels

* Defines labels for a single label set
* Should be followed by a comma separated list of strings to use for each label.

#AUTA labels OP\_SYMBOL, OP\_NAME

#AUTA label +,Add

#AUTA label -,Subtract

#AUTA label \*,Multiply

#AUTA label /,Divide

#AUTA end labels

## AUTA Scope Command block

AUTA scope block allows a scope to be defined once and used in multiple places.   
An AUTA Scope block can contain

* Label Command Blocks
* Type Command Lines.
* Scope Command Blocks.

### Cache Commands

#### import

* Defines the start of an AUTA Scope Command block.

#### importscopeblock

* Appends the specified scope to this scope.
* Scopes do not interact. All scopes are applied independently and successively.

### Extra Commands

#### scope

* Tells the AUTA Command block calling this extra command to use this scope for replication.
* The calling block will perform all it’s operations, then pass the replicated result to this scope for further processing.

/\* #AUTA scopeblock EACH\_UINT\_TYPE

#AUTA labels T

#AUTA label UInt32

#AUTA label UInt64

#AUTA end labels

#AUTA end EACH\_UINT\_TYPE \*/

/\* #AUTA scopeblock EACH\_INT\_TYPE

#AUTA labels T

#AUTA label Int32

#AUTA label Int64

#AUTA end labels

#AUTA end EACH\_INT\_TYPE\*/

/\* #AUTA scopeblock EACH\_INTEGER\_TYPE

#AUTA importscopeblock EACH\_INT\_TYPE

#AUTA importscopeblock EACH\_UINT\_TYPE

#AUTA end EACH\_INTEGER\_TYPE\*/

/\*

#AUTA export Group1

TypeProxyOf<Object<T>>::GetTypeProxy();

#AUTA end Group1

\*/

//#AUTA import Group1 scope = EACH\_INT\_TYPE

TypeProxyOf<Object<Int32>>::GetTypeProxy();

TypeProxyOf<Object<Int64>>::GetTypeProxy();

TypeProxyOf<Object<UInt32>>::GetTypeProxy();

TypeProxyOf<Object<UInt64>>::GetTypeProxy();

//#AUTA end Group1

## AUTA Code Replication Command block

AUTA Code Replication block behaves almost identically to an Import / Export Command block but can contain

* Label Command Blocks
* Type Command Lines.

### Cache Commands

#### codeblock

* Defines the start of an AUTA Codeblock export Command block.

#### body

* Bounds a code block to be used when replicating.
* Multiple bodies can be defined within a single codeblock (each requires a unique Tag)
* Bodies will be replicated then multiple bodies concatenated
* This allows the user to have some control over the output ordering of the replicated code

### Output Commands

#### replicate

* Replicates each body separately.
* Then concatenates each body
* Fills the replicate Command block with the generated Code.

/\* #AUTA codeblock CPUArithmeticBodies

#AUTA body DeviceFunctionPointersBody

void Device ## OP\_NAME ## TA ## TB(void\* r, void\* a, void\* b, size\_t size)

{

R \*ir = reinterpret\_cast<R\*>(r);

TA \*ia = reinterpret\_cast<TA\*>(a);

TB \*ib = reinterpret\_cast<TB\*>(b);

\*ir = \*ia OP\_SYMBOL \*ib;

}

#AUTA end DeviceFunctionPointersBody

#AUTA end CPUArithmeticBodies \*/

/\* #AUTA scopeblock EACH\_ARITHMETIC\_OPERATOR\_TYPE

#AUTA type R Float64, Int64

#AUTA type TA Float64 , Int64

#AUTA type TB Float64 , Int64

#AUTA blacklisttypes R == Float64 , TA == Int64 , TB == Int64

#AUTA blacklisttypes R == Int64 , TA == Float64

#AUTA blacklisttypes R == Int64 , TB == Float64

#AUTA labels OP\_SYMBOL, OP\_NAME

#AUTA label +,Add

#AUTA label -,Subtract

#AUTA end labels

#AUTA end EACH\_ARITHMETIC\_OPERATOR\_TYPE\*/

//#AUTA replicate CPUArithmeticBodiesTemp scope = EACH\_ARITHMETIC\_OPERATOR\_TYPE

void DeviceAddFloat64Float64 (void\* r, void\* a, void\* b, size\_t size)

{

Float64 \*ir = reinterpret\_cast< Float64 \*>(r);

Float64 \*ia = reinterpret\_cast< Float64 \*>(a);

Float64 \*ib = reinterpret\_cast< Float64 \*>(b);

\*ir = \*ia + \*ib;

}

void DeviceSubtractFloat64Float64 (void\* r, void\* a, void\* b, size\_t size)

{

Float64 \*ir = reinterpret\_cast< Float64 \*>(r);

Float64 \*ia = reinterpret\_cast< Float64 \*>(a);

Float64 \*ib = reinterpret\_cast< Float64 \*>(b);

\*ir = \*ia - \*ib;

}

void DeviceAddInt64Float64 (void\* r, void\* a, void\* b, size\_t size)

{

Float64 \*ir = reinterpret\_cast< Float64 \*>(r);

Int64 \*ia = reinterpret\_cast< Int64 \*>(a);

Float64 \*ib = reinterpret\_cast< Float64 \*>(b);

\*ir = \*ia + \*ib;

}

void DeviceSubtractInt64Float64 (void\* r, void\* a, void\* b, size\_t size)

{

Float64 \*ir = reinterpret\_cast< Float64 \*>(r);

Int64 \*ia = reinterpret\_cast< Int64 \*>(a);

Float64 \*ib = reinterpret\_cast< Float64 \*>(b);

\*ir = \*ia - \*ib;

}

void DeviceAddFloat64Int64 (void\* r, void\* a, void\* b, size\_t size)

{

Float64 \*ir = reinterpret\_cast< Float64 \*>(r);

Float64 \*ia = reinterpret\_cast< Float64 \*>(a);

Int64 \*ib = reinterpret\_cast< Int64 \*>(b);

\*ir = \*ia + \*ib;

}

void DeviceSubtractFloat64Int64 (void\* r, void\* a, void\* b, size\_t size)

{

Float64 \*ir = reinterpret\_cast< Float64 \*>(r);

Float64 \*ia = reinterpret\_cast< Float64 \*>(a);

Int64 \*ib = reinterpret\_cast< Int64 \*>(b);

\*ir = \*ia - \*ib;

}

void DeviceAddInt64Int64 (void\* r, void\* a, void\* b, size\_t size)

{

Int64 \*ir = reinterpret\_cast< Int64 \*>(r);

Int64 \*ia = reinterpret\_cast< Int64 \*>(a);

Int64 \*ib = reinterpret\_cast< Int64 \*>(b);

\*ir = \*ia + \*ib;

}

void DeviceSubtractInt64Int64 (void\* r, void\* a, void\* b, size\_t size)

{

Int64 \*ir = reinterpret\_cast< Int64 \*>(r);

Int64 \*ia = reinterpret\_cast< Int64 \*>(a);

Int64 \*ib = reinterpret\_cast< Int64 \*>(b);

\*ir = \*ia - \*ib;

}

//#AUTA end CPUArithmeticBodiesTemp

## AUTA Function Command block

AUTA Function Command block allows the user to define a function (including templates) and automatically replicate the definitions, declarations, specialisations and instantiations (as appropriate)

The AUTA function Command block can contain

* Label Command Blocks
* Type Command Lines.

### Cache Commands

#### Function

* Defines the start of an AUTA Function Command block.
* Each line is parsed for being a valid AUTA function definition, it should be of the form
  + template<template parameters> PreFunctionOperands ReturnType Namespace::FunctionName(Function Parameters) PostFunctionQualifiers
  + PreFunctionOperands ReturnType Namespace::FunctionName(Function Parameters) PostFunctionQualifiers
* Braces will be counted to find the end of the function.
* A function block can contain multiple functions.

### Output Commands

#### declare

* Produces a function declaration – i.e. without namespace or body.
* Replicates the function for Labels, but not for Types.

#### define

* Produces a function definition – i.e. appending the types to function name.
* Replicates the function for Labels, but not for Types.

#### instance

* Will produce an instance for the template function.
* Will replicate for types and labels - but only types appearing in the function declaration (other TypeIndentifiers are ignored)

#### specialise

* Will automatically generate a specialisation for a template function.
* This will replicate for ALL types and labels. Even Types not appearing in the function declaration.