HCL

What’s New in HCL RTist 11.1

updated for release 2022.21

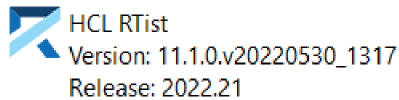
HCL SOFTWARE

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Overview

- RTist 11.1 is based on Eclipse 2020.06 (4.16)

- HCL RTist is 100% compatible with IBM RSARTE. All features in IBM RSARTE are also present in HCL RTist. However, HCL RTist contains some features that do not exist in IBM RSARTE.
  - Those features are marked in this presentation by



(c) Copyright IBM Corporation 2004, 2016. All rights reserved.
(c) Copyright HCL Technologies Ltd. 2016, 2022. All rights reserved.
Visit https://RTist.hcldoc.com/help/topic/com.ibm.xtools.rsarte.webdoc/users-guide/overview.html

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

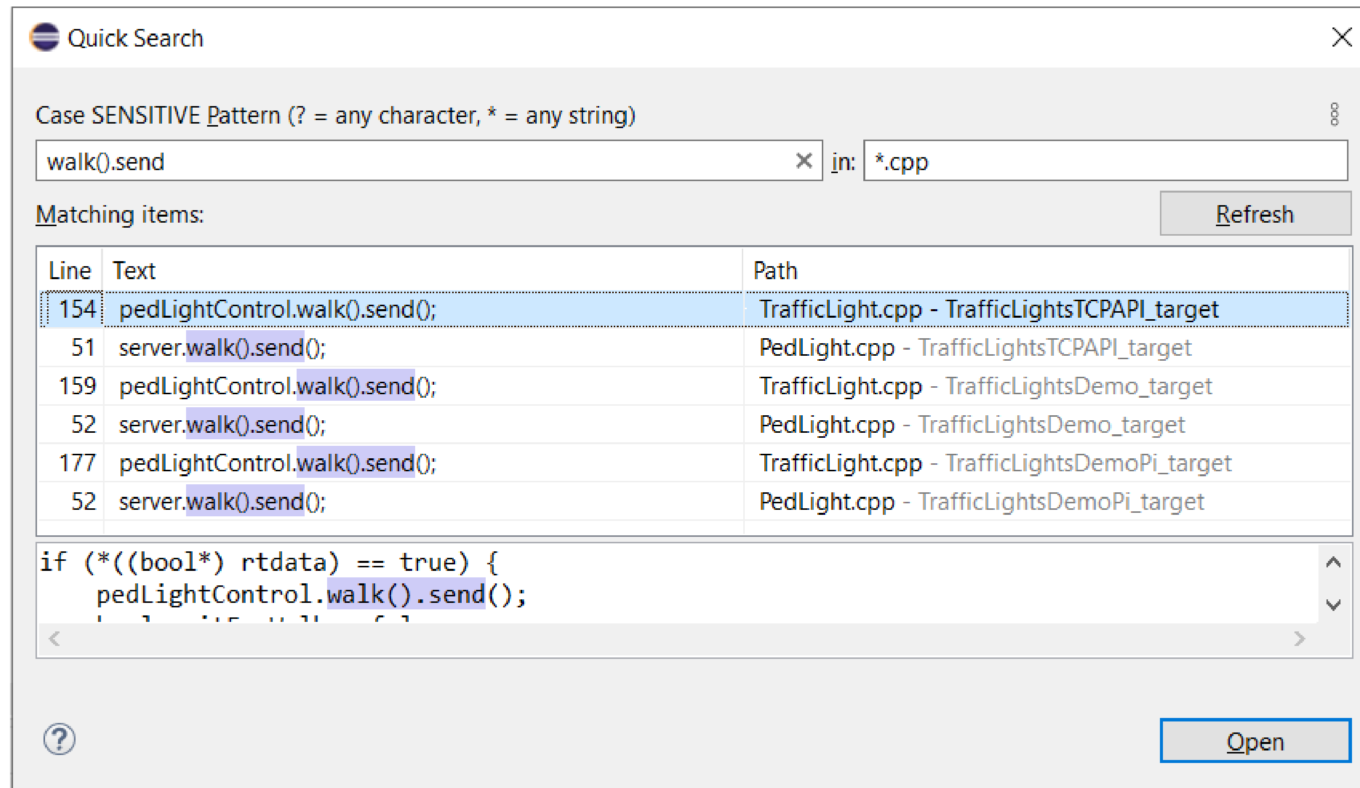
Eclipse 4.16 (2020.06)

- Compared to RTist 11.0, RTist 11.1 includes new features from 4 quarterly Eclipse releases:
  - 2019.09 (https://www.eclipse.org/eclipse/news/4.13/platform.php)
  - 2019.12 (https://www.eclipse.org/eclipse/news/4.14/platform.php)
  - 2020.03 (https://www.eclipse.org/eclipse/news/4.15/platform.php)
  - 2020.06 (https://www.eclipse.org/eclipse/news/4.16/platform.php)
- For full information about all improvements and changes in these Eclipse releases see the links above
  - Some highlights are listed in the next few slides...

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Eclipse 4.16 (2020.06)

- A new Quick Search dialog allows you to search the files of your workspace faster (“as-you-type”)
  - For a similar search experience in model files, use the Find Named Element command instead

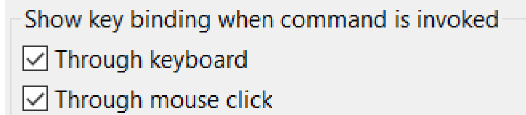


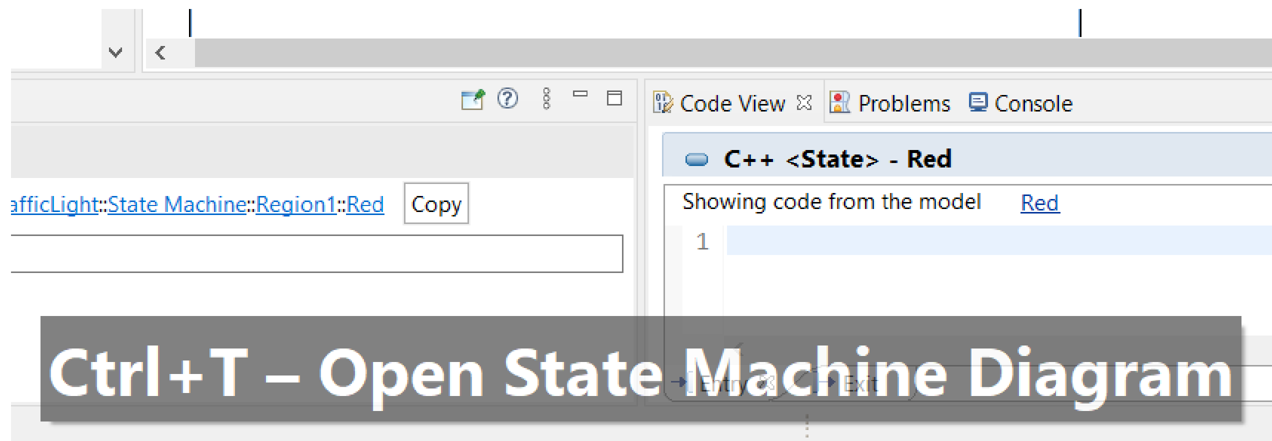
Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Eclipse 4.16 (2020.06)

- By default at most 99 editors can now be open at the same time
  - Helps keeping the performance good when working with Eclipse for a long time
  - This can be controlled by the preference **General – Editors – Close editors automatically**
- Showing key bindings when performing commands
  - New preferences in **General – Keys**
  - This is a good way to learn about key bindings for the commands that are used, and can also help in presentations

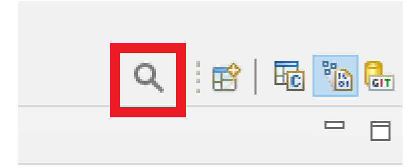




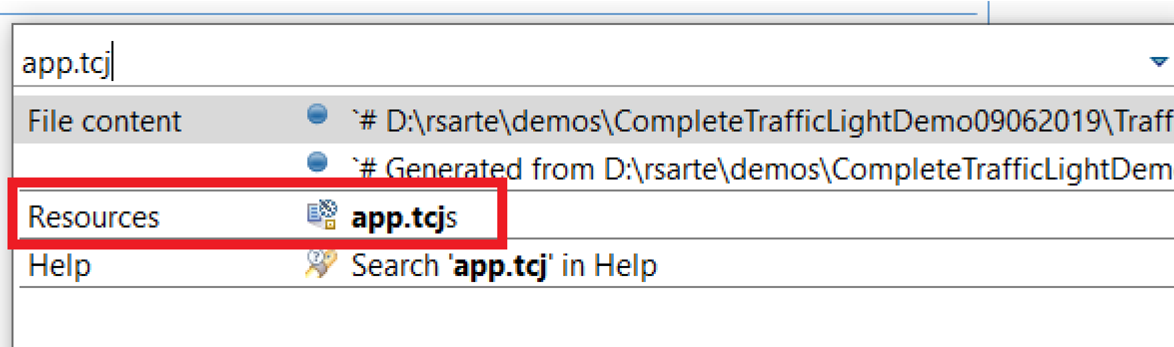
Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

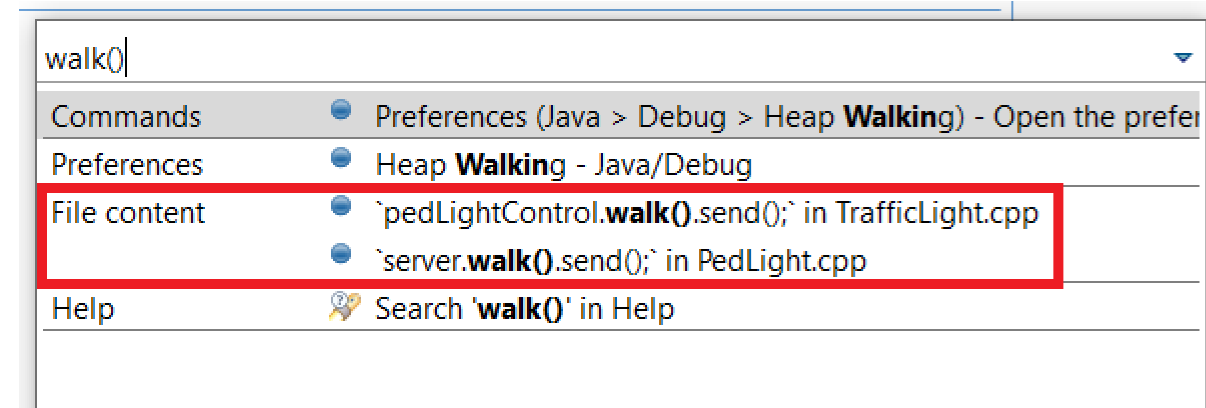
HCL SOFTWARE

Eclipse 4.16 (2020.06)



- Quick Access field replaced with toolbar button
  - Takes less space in the toolbar, and instead uses a normal dialog for typing an showing the results
  - Same key binding as before (Ctrl + 3) but the command is now called “Find Actions”
  - The results now also include matching files in the workspace, and text matches in files (requires that Quick Search has been used at least once)





Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Eclipse 4.16 (2020.06)

- Show code problems inline
  - Makes errors/warnings more visible and lets you apply quick fixes without having to go to the Problems view
  - Enable this feature in preferences at **General – Editors – Text Editors – Show code minings for problem annotations**

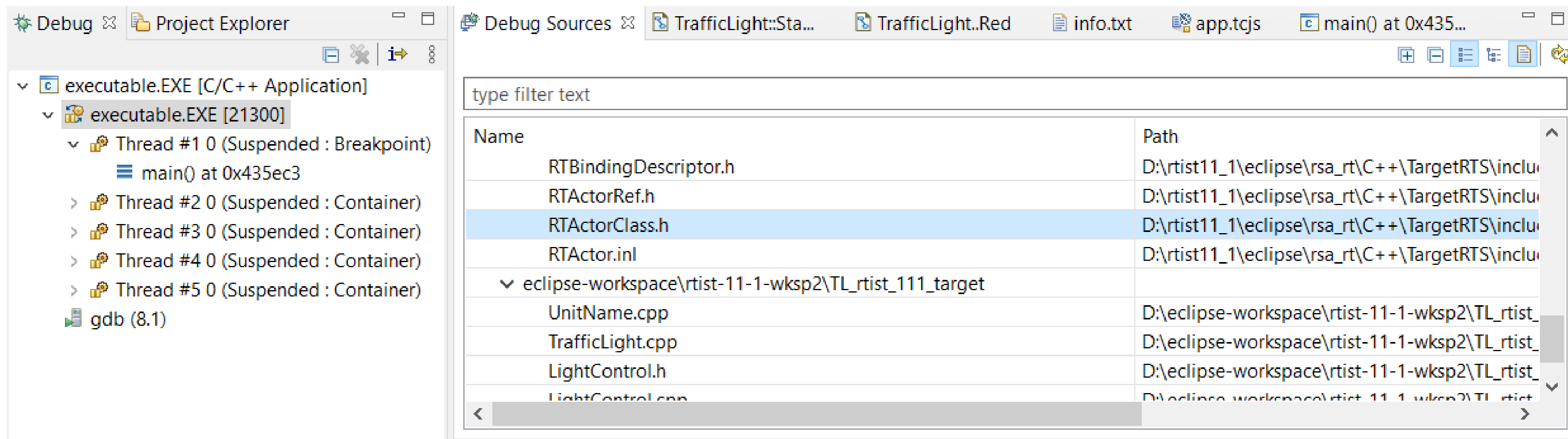
```
34 {
35 //{{{USR platform:/resource/TL_rtist_111/Tr
36 log.log("Red -> Green");
   'cout' is not a member of 'std'
37 std::cout << "test";
38 //}}}USR
39 }
```

- There were several improvements in SWT and GTK
  - The minimal supported GTK version is now 3.20

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com
HCL SOFTWARE

CDT 9.11 (included as part of Eclipse 2020.06)

- New Debug Sources view
  - Shows source files the C++ debugger knows about when debugging an application
  - Useful in particular when the application contains source files that are not present in the Eclipse workspace
  - Source files can be found by searching (filtering) and opened by double-click



Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

CDT 9.11 (included as part of Eclipse 2020.06)

- CODAN improvements
  - Several additional checks implemented
- For more information about CDT improvements see
  https://wiki.eclipse.org/CDT/User/NewIn99
  https://wiki.eclipse.org/CDT/User/NewIn910
  https://wiki.eclipse.org/CDT/User/NewIn911

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Newer EGit Version in the EGit Integration

- The EGit integration in RTist has upgraded EGit from 5.4 to 5.8
  - This is the recommended and latest version for Eclipse 2020.06
- This upgrade provides several new features, performance improvements and bug fixes
  - For detailed information about the changes see
    https://wiki.eclipse.org/EGit/New_and_Noteworthy/5.5
    https://wiki.eclipse.org/EGit/New_and_Noteworthy/5.6
    https://wiki.eclipse.org/EGit/New_and_Noteworthy/5.7
    https://wiki.eclipse.org/EGit/New_and_Noteworthy/5.8

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

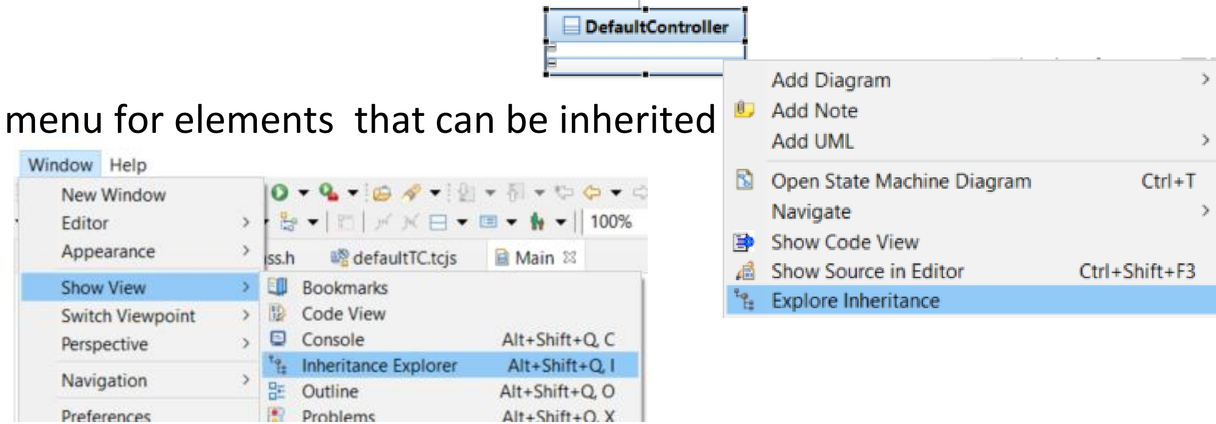
Installation Script

- A bash script is now available which helps automating the installation of RTist
  - Download it from the Info Center
  - Works on both Windows and Linux
- In particular useful for installing RTist 11.1 (due to the requirement of using Java 11 for the installation)
  - Choose whether you want to then run RTist with either Java 8 or Java 11
- For documentation on how to configure and use the script see the Info Center.

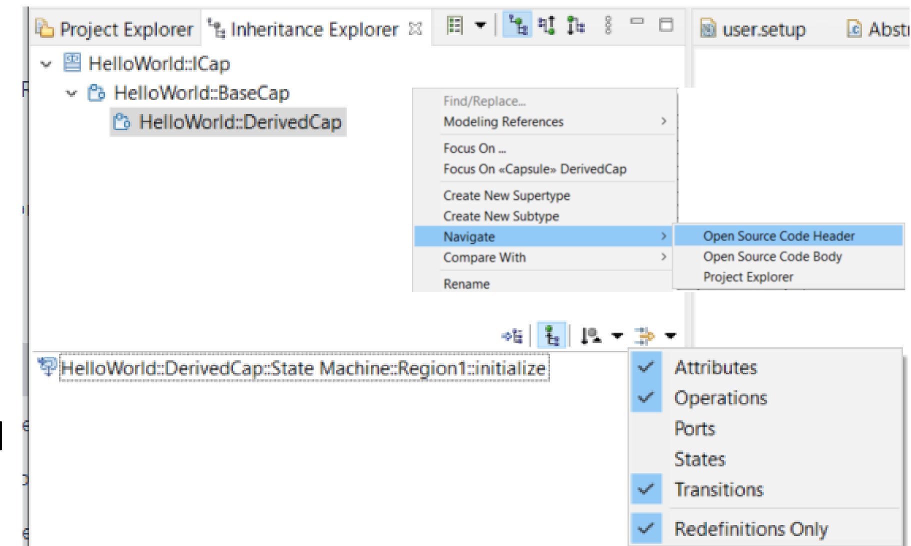
Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Improved Inheritance Explorer

- The Inheritance Explorer is now easier to open
  - A new **Explore Inheritance** command is available in the context menu for elements that can be inherited
  - A new keyboard shortcut has been assigned: Alt + Shift + Q , I



- Both super types (direct and indirect) and sub types (only direct) are now shown initially
  - No longer necessary to manually expand the element to see the sub types
- Support for navigating from Inheritance Explorer to generated C++ code
- Interfaces are now supported in the same way as other types
- Improved the Members view
  - For a capsule, its ports, states and transitions can now be shown
  - Possible to only see the redefinitions (e.g. useful to understand how an inherited capsule or class has been modified)
  - Improved sorting of members according to different criteria

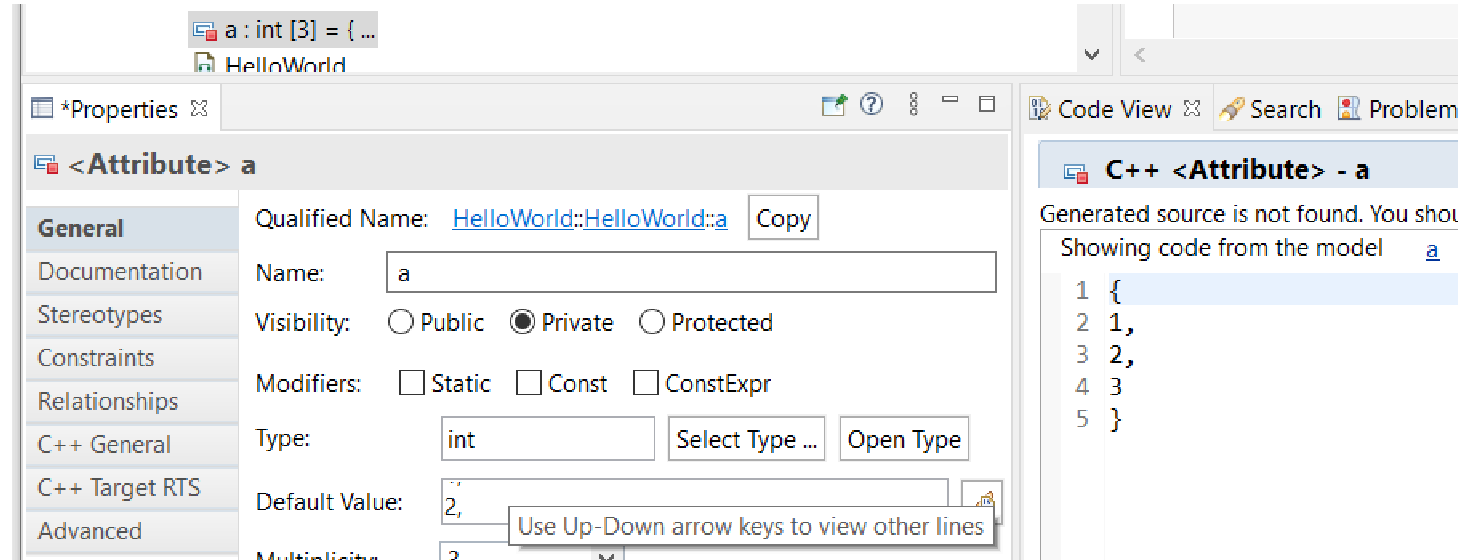


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Properties View Improvements

- The Default Value field now supports multi-line values
  - To create a multi-line default value you still need to use the Code View or Code Editor
  - For editing a multi-line default value you can now use the Properties view, but it’s still often more convenient with the Code View or Code Editor



  - For quickly viewing a multi-line default value the Properties view can be handy

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Brace Initialization

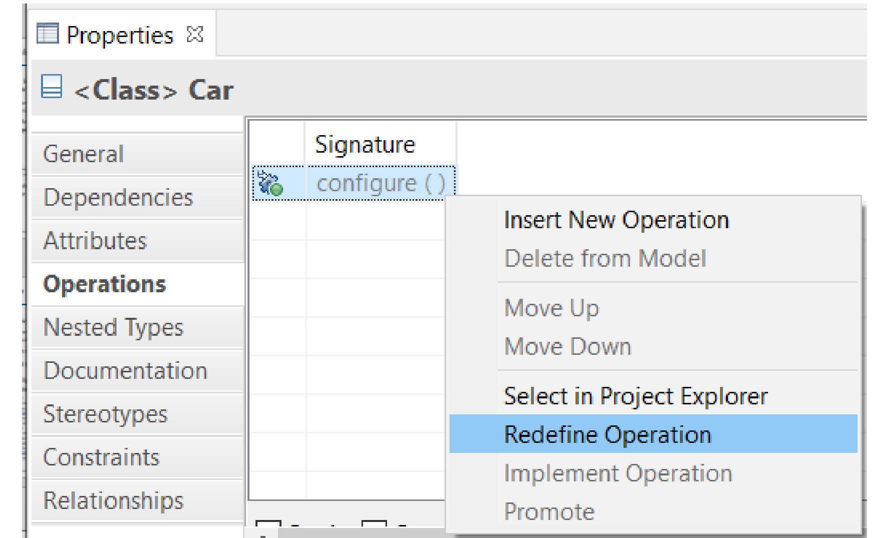
- For newly created attributes, brace initialization will be selected by default
  - But only if using C++ 11 as code standard
  - Existing attributes are not affected by this change
- The Project Explorer now shows if an attribute uses brace initialization, or some other form of initialization

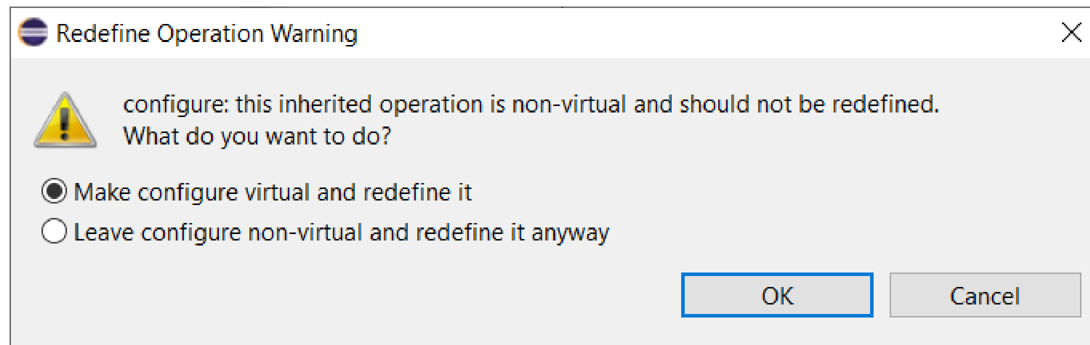
- When you create a new attribute, and C++ 11 is used, you can now choose if you want to use brace or equal syntax for initialization by directly typing in the Project Explorer
  - When the code standard is older than C++ 11, only the equal syntax is available, and it will map to constructor intialization as before

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Redefining Non-Virtual Operations

- When redefining a non-virtual operation in the UI, a warning dialog now appears
- By default the dialog suggests to make the inherited operation virtual, so the model (and generated C++) will become correct





Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

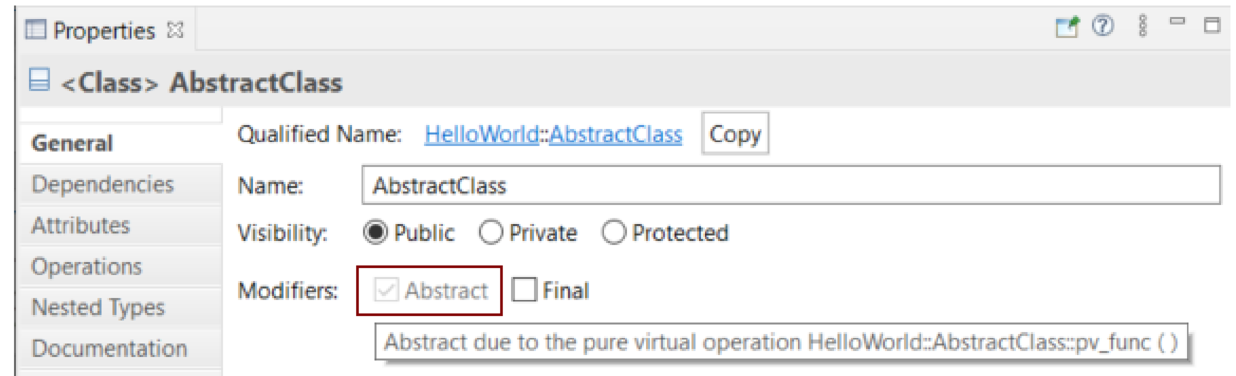
Improvements related to Abstract Classes and Pure Virtual Operations (1/2)

- Previously it was necessary to mark a class (or capsule) as Abstract for the model compiler to treat it like an abstract class
  - A warning would be printed for non-abstract classes containing pure virtual operations

```
14:08:08 : WARNING : HelloWorld::AbstractClass::pv_func : This operation cannot be pure-virtual because the owning classifier is not abstract.
```

- Now the model compiler automatically treats classes with pure virtual operations as abstract

- The Properties View implements the same condition
  - If a class (or capsule) is not marked as abstract, but has pure virtual or interface operations (locally defined or inherited), the Abstract checkbox is automatically set (and made read-only)



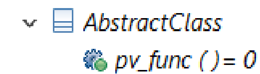
  - A tooltip explains why the class (or capsule) is considered abstract (useful in case it's abstract because you forgot to redefine or implement an operation)

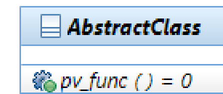
Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Improvements related to Abstract Classes and Pure Virtual Operations (2/2)

- Note that automatic computation of the Abstract property is based on comparing the signature of operations. If operations in the inheritance hierarchy have parameters with inconsistent use of types (e.g. mixed use of regular types, typedefs, type aliases or macros) then the automatic computation may fail.
- A new preference **RealTime Development – Build/Transformations – C++ – Automatically compute if a class or capsule is abstract** controls if the Abstract property should be automatically computed (by default it's not)
  - Note that this preference applies both for code generation and the Properties view
- The Project Explorer and diagrams now show all abstract classes and pure virtual operations in italics





Signature: pv_func ( )= 0

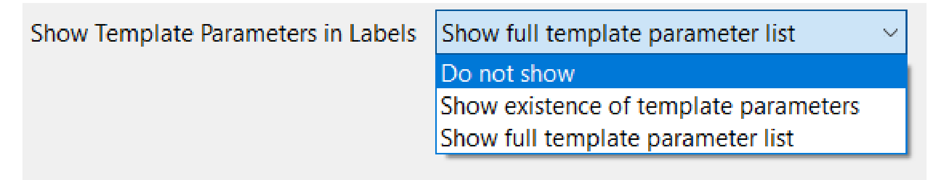
- Also, pure virtual operations now have ”= 0” appended to their signatures

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Project Explorer Improvements

- The Project Explorer can now show template information after the name of an element that has template parameters
  - Makes it easier to see if an element is a template without having to expand it in the Project Explorer, or look in the Properties view
  - A new preference **RealTime Development – Project Explorer – Show Template Parameters in Labels** controls what to show



List
typename Element
size : unsigned int

Do not show template parameters

List<T>
typename Element
size : unsigned int

Show existance of template parameters

List<typename Element, size : unsigned int>
typename Element
size : unsigned int

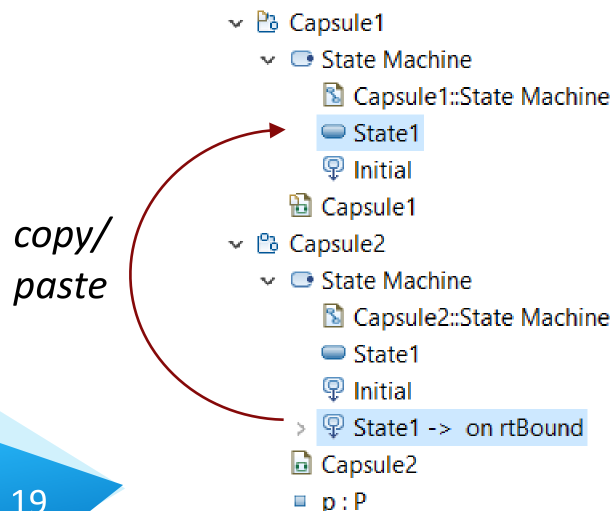
Show full template parameter list

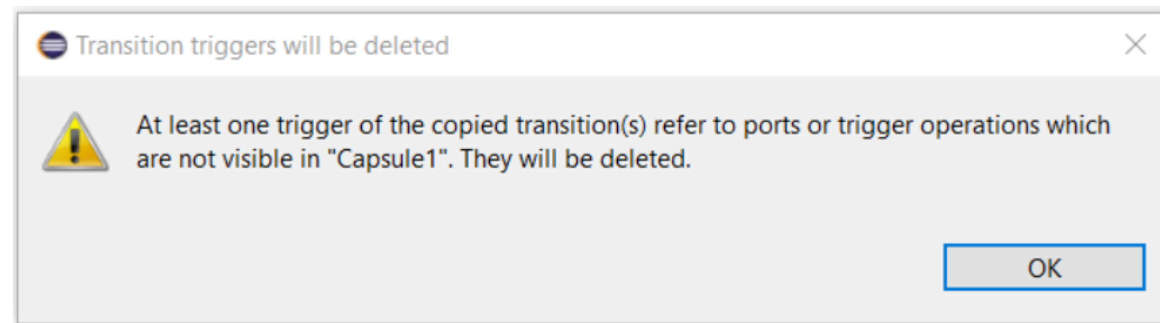
Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Copy/Paste of Transitions in the Project Explorer

- You can now copy a transition and paste it on a target state using the Project Explorer
  - More convenient than creating a new transition and then copy/paste the effect and guard code (and possibly other transition properties) separately
  - Works for transitions in both capsule and passive class state machines
  - The pasted transition will initially become a self-transition and can be rerouted later if needed
- If ports or trigger operations referenced by triggers of the copied transition are not available in the target context, a dialog will inform that such triggers will be deleted



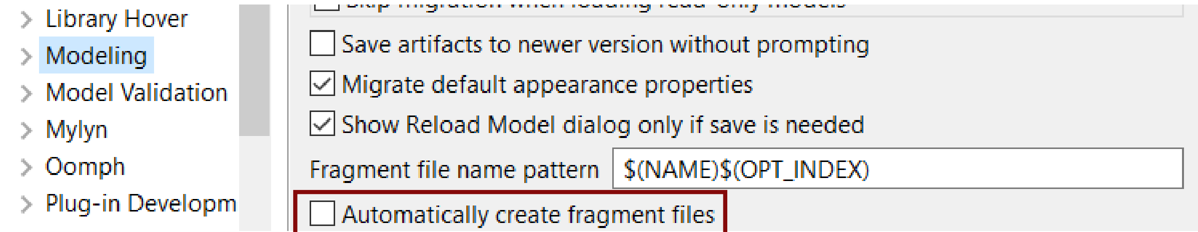


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Automatic Creation of Fragment Files

- A new preference was added for automatically creating fragment files for newly created model elements
  - *Modeling – Automatically create fragment files*



- Setting this preference can be useful if you prefer to always create fully fragmented models

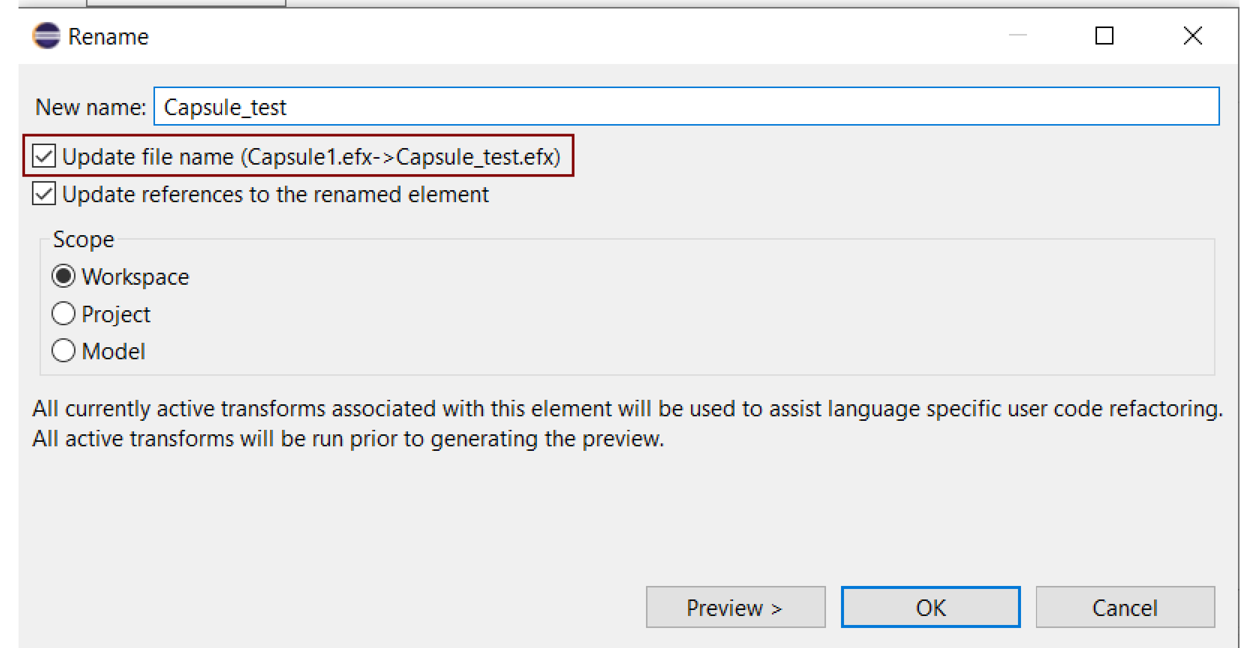
- Note that
  - for state machines no fragment files will be automatically created
  - fragment file creation cannot be undone

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Automatic Rename of Model Files

- A new checkbox was added in the Rename dialog, for automatically renaming the model file when its root element is renamed (triggered by the **Refactor – Rename** command)
- Helps ensure consistency between model element names and model file names
  - Note that like other refactorings, the file renaming is not undoable
  - Remember to use **Refactor – Rename** (as opposed to just renaming the element) if you want the model file name to update too

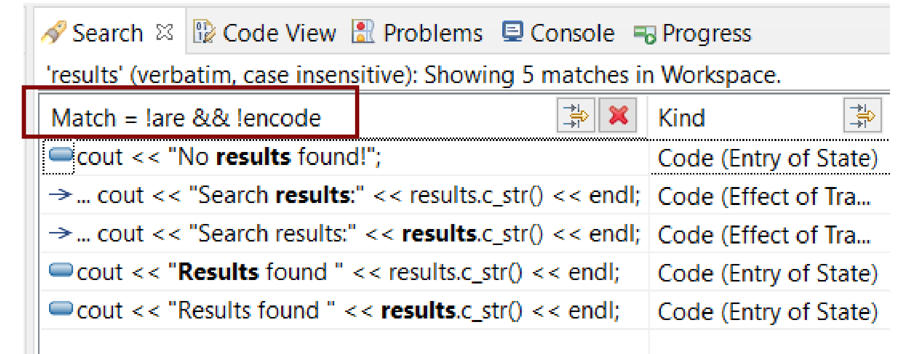


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

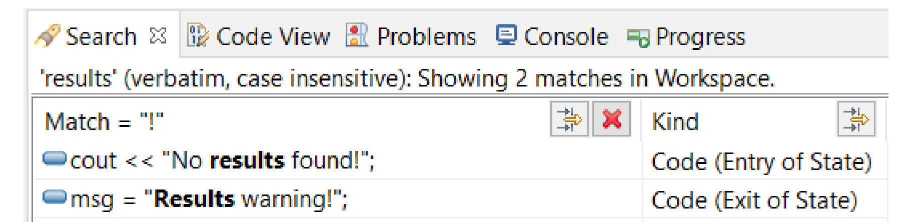
HCL SOFTWARE

Search Filtering

- It’s now possible to filter search results using Boolean operators NOT (!) and AND (&&)
  - Useful if a search returns too many matches
  - Use a filter on the form
    `!A && !B && ... !X` to hide matches where certain words are not present
  - Use a filter on the form
    `A && B && ... X` to only show matches where certain words are present
  - ...or any combination, where some words are present and others not



- Enclose the filter string in double quotes to apply the filter verbatimly
  - Needed if the filter string contains the characters ! or &&

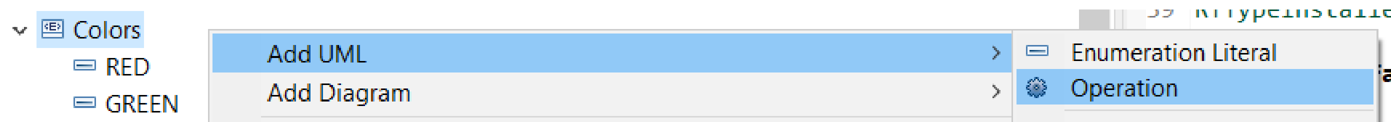


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

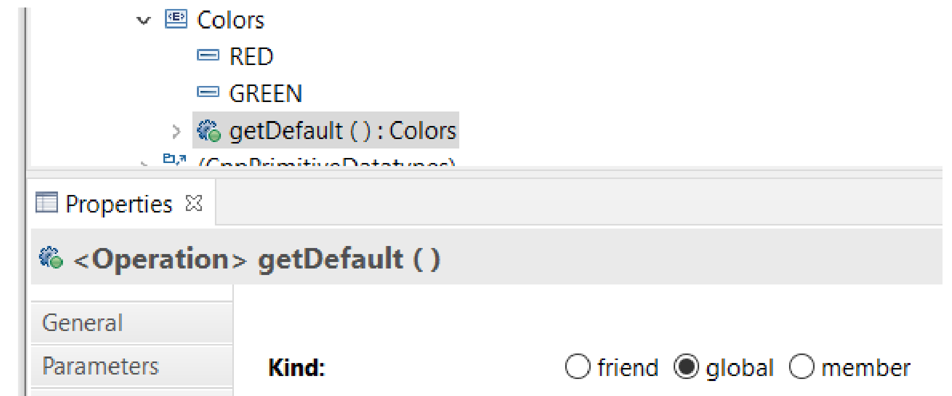
HCL SOFTWARE

Enums with Operations

- Enumerations can now have operations
  - Create them as usual with **Add UML - Operation**



- Such operations will be translated to global functions
  - C++ enums cannot have member functions, but it's sometimes useful to have functions that operate on or return enum literals
  - Using global functions can then be an alternative to wrapping the enum inside a class



- This works the same both for scoped and non-scoped enumerations

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Generic Type Descriptors

- The model compiler now supports generating type descriptors for type aliases with template parameters
  - For example: `template<typename T, unsigned int N > using StdArray = std::array<T, N>;`
  - If type descriptor functions are defined for the type alias, they will be generated as template functions with the same template parameters
  - Allows to implement generic type descriptors that work for all (or many) instantiations of the template
  - A new `RTObject_class::fromType<T>()` template function can be used for looking up the type descriptor of a type at compile time. Useful for example when implementing generic encode or decode functions. Specialize it for the types that you use (specializations for built-in types are available in the TargetRTS). For example:

```
template <> inline const RTObject_class* RTObject_class::fromType<RTString>() {
    return &RTType_RTString;
}
```

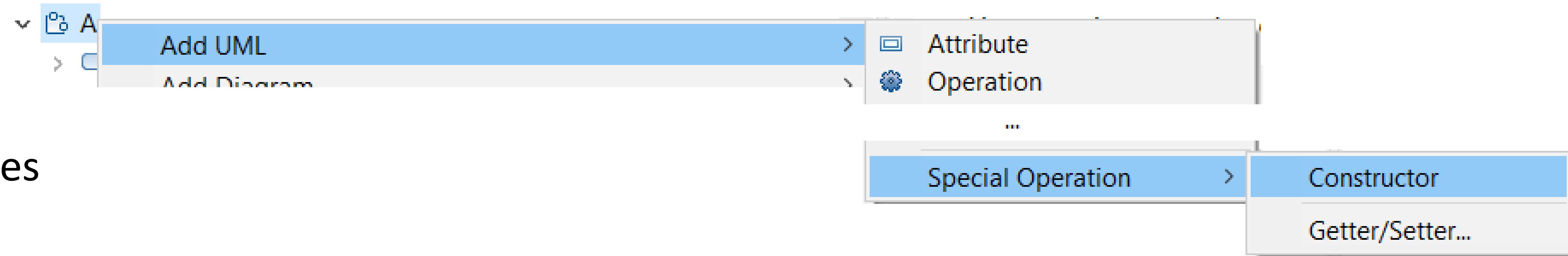
- You can specify a unique name for the type descriptor of a specific template instantiation
  - For example: `template <> const char* RTName_StdArray<StdString, 4>::name = "StdArray<StdString, 4>";`
  - The TargetRTS now prints a warning if two type descriptors with the same name exists. Helps troubleshooting missing template specializations for the name attribute.

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

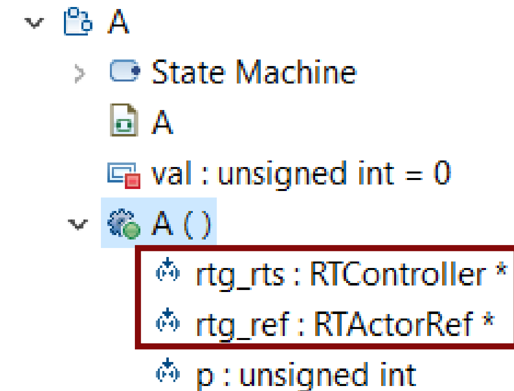
HCL SOFTWARE

Custom Capsule Constructors

- It's now possible to create custom constructors for capsules



- Each capsule constructor has two mandatory parameters:
  - **rtg_rts** Controller (i.e. thread) that will run the created capsule instance
  - **rtg_ref** Capsule part where the created capsule instance will be inserted

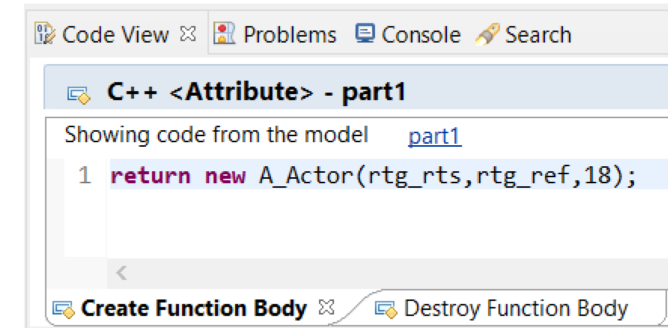


- In addition you can add any number of user-defined parameters
- This feature makes it possible to pass initialization data to a capsule instance already when it's created
  - Previously this could only be done by sending such data with the initialization event (which is not possible for fixed capsule parts)
  - Custom capsule constructors work for all capsules regardless of the capsule part they are incarnated into

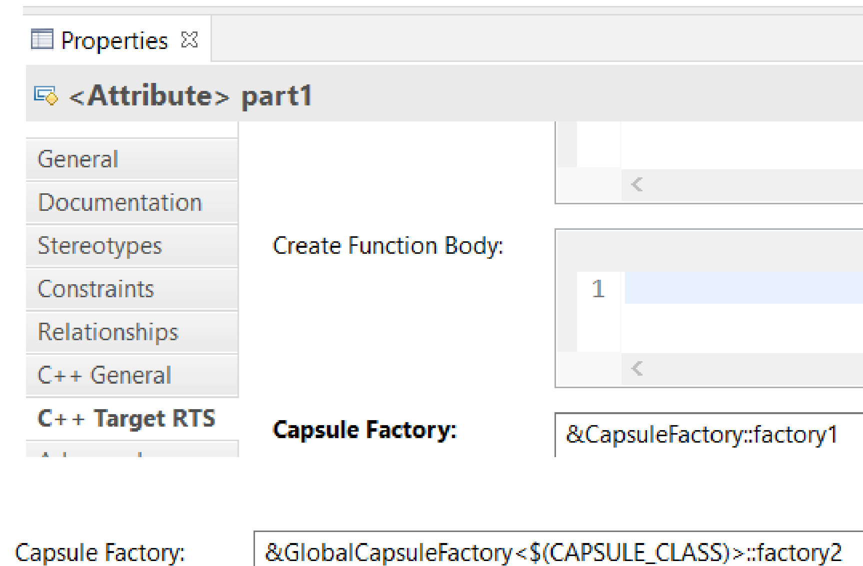
Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Capsule Factories (1/2)

- The concept of a capsule factory was introduced to allow incarnating capsules with custom constructors
  - Specifies how a capsule instance is created and destroyed
  - Can be provided in various ways (in a hierarchical manner)



- New capsule factory code snippets for capsule parts
  - All capsule instances incarnated in that capsule part will use the specified Create/Destroy code
- New capsule factory property for capsule parts
  - Will be used if no Create/Destroy code is provided for that capsule part



- New capsule factory property in the TC
  - Will be used if none of the above are provided
  - Allows specifying a default (global) capsule factory
  - A variable $(CAPSULE_CLASS) can be used in this TC property (expands to the name of the class that is generated from the type of the capsule part)

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Capsule Factories (2/2)

- For optional capsule parts, it's also possible to provide the capsule factory using a new TargetRTS function `RTFrame::incarnateCustom()`

```
RTActorId     incarnateCustom( RTActorRef & cp,
                               RTActorFactory& factory,
                               int index = -1);
```

  - In this case, only the Create code can be provided (the regular delete operator will be used for destroying such capsule instances)
  - Example usage:

```
RTActorId id = frame.incarnateCustom(part1,
RTActorFactory([this](RTController * c, RTActorRef * a, int index) {
  return new A_Actor(c, a, 444); // User-defined constructor
})
);
```

- If multiple capsule factories are provided, they will be picked in this priority order:

1. The capsule factory provided in a call to `RTFrame::incarnateCustom()`
2. The capsule factory specified by means of Create and/or Destroy code snippets on a capsule part
3. The capsule factory specified by the "Capsule Factory" property on a capsule part
4. The capsule factory specified in the "Capsule Factory" property on the TC

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com
HCL SOFTWARE

Dependency Injection

- A capsule normally depends on many things at run-time for its execution
  - Examples: Other capsules typing its capsule parts, the thread that will run the capsule, initialization data to pass to the capsule constructor, etc.
- Spreading out such dependencies in a hard-coded way in an application can make it hard to change them to configure different variants of an application
  - E.g. mocking out dependent capsules when unit testing a capsule
- The TargetRTS now provides a new dependency injection service realized by the RTInjector class
  - Register the dependencies to configure the application (typically early, e.g. in the top capsule constructor)
  - A create function can be registered for a capsule part (identified by its qualified path name)
  - A capsule factory can delegate to `RTInjector::create()` for creating capsule instances
  - If necessary, registered dependencies can be changed at run-time

C++ <Operation> TOP (rtg_rts : RTController *, rtg_ref : RTActorRef *)

Showing code from the model TOP (...)

```
1 RTInjector::getInstance().registerCreateFunction("/logger:0/logger",
2         [this](RTController * c, RTActorRef * a, int index) {
3             //return new SimpleLogger_Actor(c, a);
4             return new TimestampLogger_Actor(LoggerThread, a);
5         }
6 );
```


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Moving Event Data (1/2)

- The data of an event can now be moved instead of copied when sent between two capsules

```
MyClass mc;
thePort.theEvent(mc).send(); // Send by copy
thePort.theEvent(std::move(mc)).send(); // Send by move
```

- This requires that the event data type is movable, which can be accomplished
  - by having a move constructor, and/or
  - by having a move function defined in the type descriptor

- The move function is a new type descriptor function (describing how to move data from a source to a target object)
  - If the target object has a move constructor, a typical implementation is to invoke it (the model compiler can automatically generate such an implementation)
  - Contrary to other type descriptor functions, the move function is optional (you only need to implement it if the type needs to be movable)
  - If no move function is defined, and an attempt is made to move an object, it will instead be copied

Properties

<Class> StdString (typedef of std::string)

Operations
Nested Types
Documentation
Stereotypes
Constraints
Relationships
C++ General
C++ Target RTS
Advanced

Copy Function Body: Edit

```
1   target = new (target) std::string(*source);
```

Move Function Body: Edit

```
1   target = new (target) std::string(std::move(*source));
2
```


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

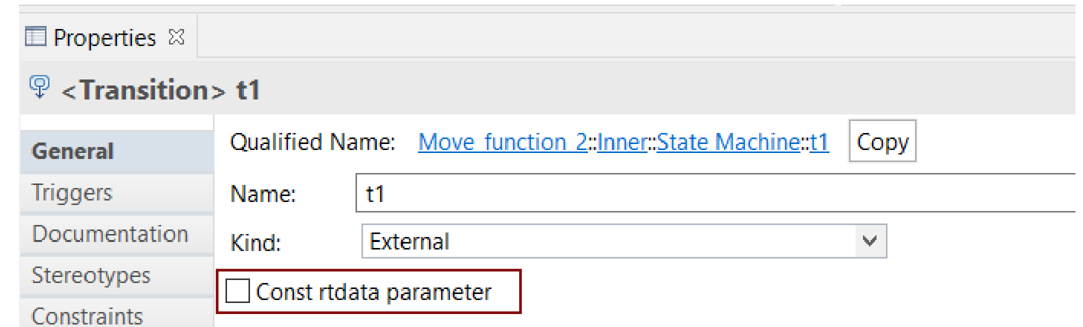
HCL SOFTWARE

Moving Event Data (2/2)

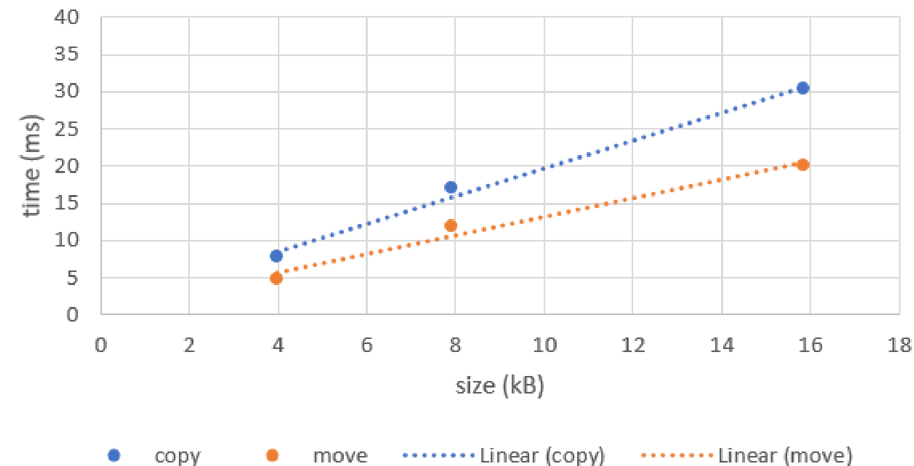
- You can also move the data from a received message into, for example, a capsule attribute

```
someAttr = std::move(*rtdata); // Avoid copying the message data object
```

- This requires that rtdata is declared as non-const (so the move constructor or move assignment operator will be invoked)
  - Can be accomplished by a new transition property



- Moving instead of copying event data can improve application performance if
  - the data object is big, and/or
  - the data object is sent many times

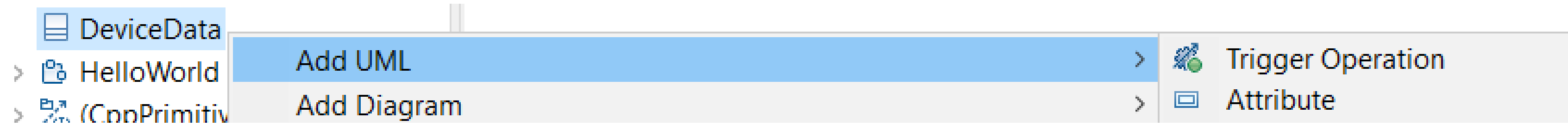


~35% faster

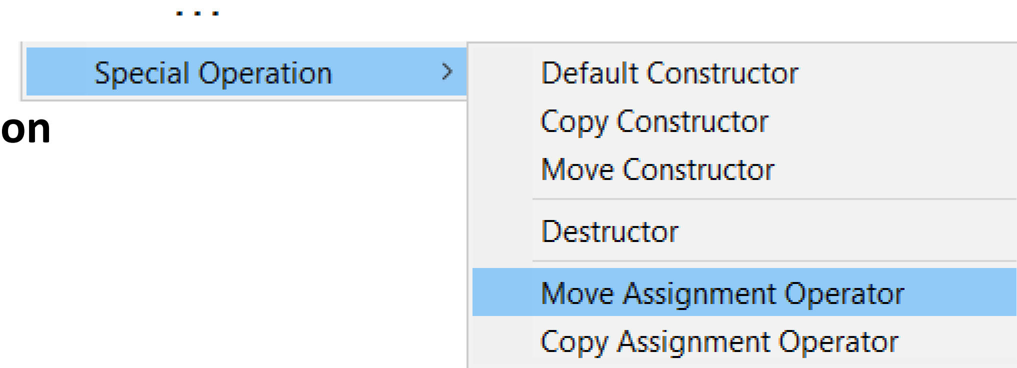
Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Creation of Assignment Operators



- It's now easier to create assignment operators for a class
  - Use new commands in context menu **Add UML – Special Operation**



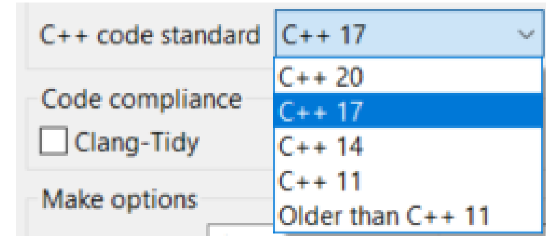
- Both Move and Copy assignment operators can be created

DeviceData
Operation1 ( )
operator= (source : const DeviceData&) : DeviceData& ← *copy assignment operator*
operator= (source : DeviceData&&) : DeviceData& ← *move assignment operator*

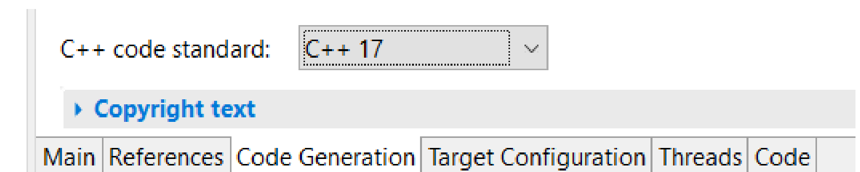
Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

More C++ Language Standards



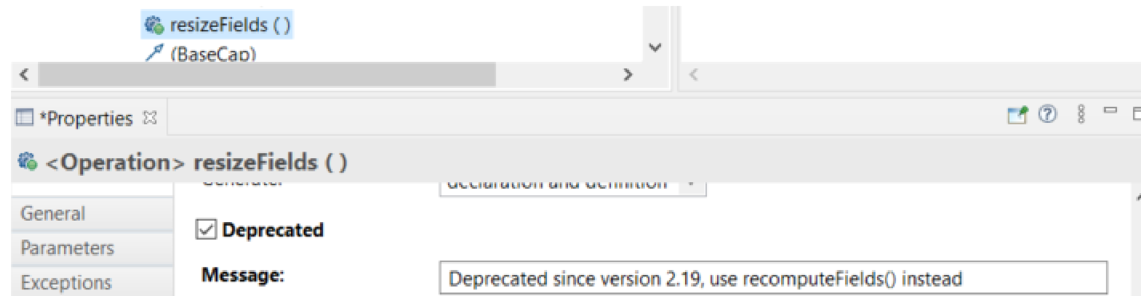
- The preference **RealTime Development – Build/Transformations – C++ – C++ Code Standard** can now be set to more C++ language standards
- The corresponding model compiler argument `--codeStandard` was extended too
- The default language standard is now set to C++ 17
  - Matches the compilers used for building the precompiled versions of the TargetRTS
- Explicitly setting which language standard to use makes it possible for the model compiler to issue a warning if a C++ language construct is used that is not supported in the selected language version
- It is now possible to also set the C++ language standard in the TC
  - Makes it possible to override the C++ language standard specified in the workspace when building a certain TC



Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Marking Elements as Deprecated

- Many elements can now be marked as deprecated using new widgets in the C++ General property tab
  - Optionally, a deprecation message can also be provided (for example to suggest an alternative element to use instead)



- A deprecated element is allowed to be used, but its usage is discouraged
  - The C++ compiler will print the deprecation message if it detects that the deprecated element is used

```
../DerivedCap.cpp:36:14: warning: 'void DerivedCap_Actor::resizeFields()' is deprecated: Deprecated since version 2.19, use recomputeFields() instead [-Wdeprecated-declarations]
 resizeFields();
```

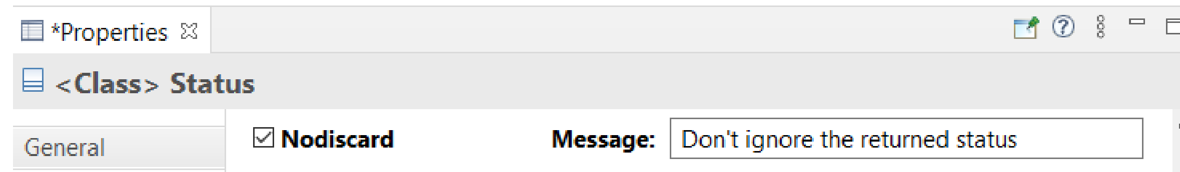
- This feature requires C++ 14 or later
  - If the **C++ Code Standard** preference is older than C++ 14, the model compiler will print a warning

```
10:01:25 : WARNING : CPPModel::DerivedCap::resizeFields : Marking an element as deprecated requires C++ 14 or later.
```


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Marking Elements as Nodiscard

- Operations and types (operation return types) can now be marked as nodiscard using new widgets in the C++ General property tab
  - Optionally, a message can also be provided



- If the C++ compiler detects that a nodiscard operation returns a value that is discarded by the caller, it will print a message (either a predefined message, or the custom message specified)

```
..\HelloWorld.cpp(30): warning C4858: discarding return value: Don't ignore the returned status
```

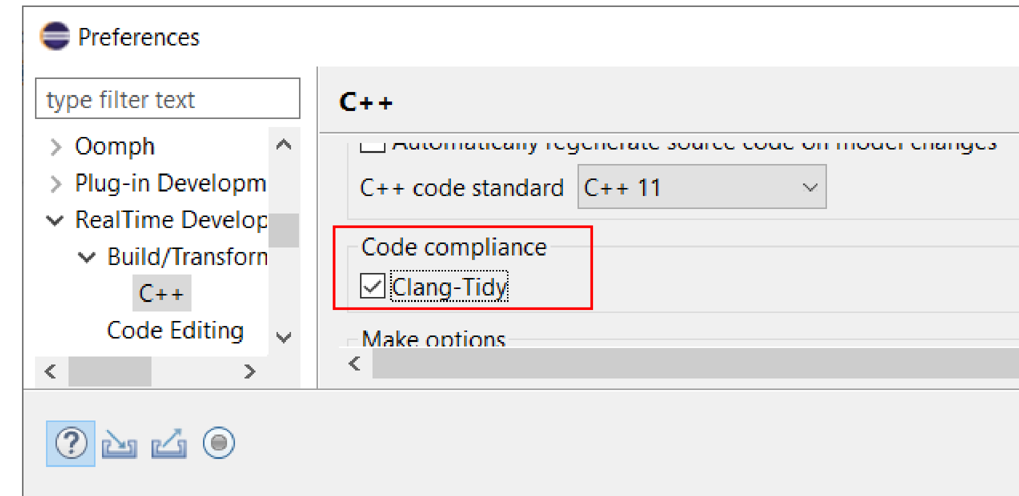
- This feature requires C++ 17 or later (C++ 20 if a custom message is provided)
  - If the **C++ Code Standard** specifies a too old code standard, the model compiler will print a warning

```
14:42:09 : WARNING : HelloWorld::Status : Marking an element as nodiscard requires C++ 17 or later.
14:46:20 : WARNING : HelloWorld::Status : Marking an element as nodiscard with a message requires C++ 20.
```


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Code Compliance

- A new preference was introduced to let the model compiler generate code according to certain code compliance rules
- Support for these Clang-Tidy rules are implemented:
  - **cppcoreguidelines-pro-type-static-cast-downcast**
    Suppress warnings for use of static_cast to downcast event data in transition functions




```
transition2_t1( static_cast< const bool * > ( msg->data ), static_cast< P::Base * > ( msg->sap()
/* NOLINT(cppcoreguidelines-pro-type-static-cast-downcast) */ ) );
```

  - **misc-unused-parameters**
    Suppress warnings for named function parameters that are not used in the function body

```
static void rtg_B_init(const RTObject_class * type /* NOLINT(misc-unused-parameters) */, B * target );
```

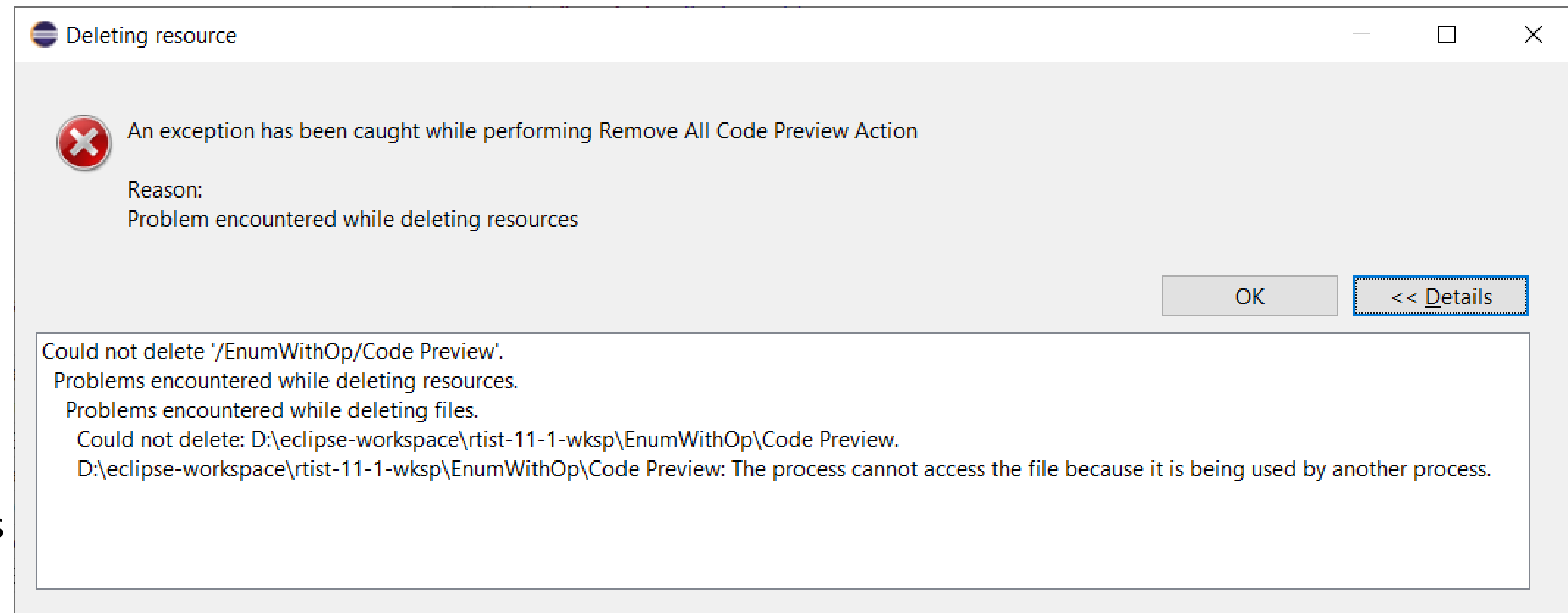
  - **bugprone-sizeof-expression**
    Suppress warnings for computing the size of a pointer type using sizeof

```
, sizeof( SomeClassPtr ) /* NOLINT(bugprone-sizeof-expression) */
```


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Error Message when Failing to Delete Files or Folders

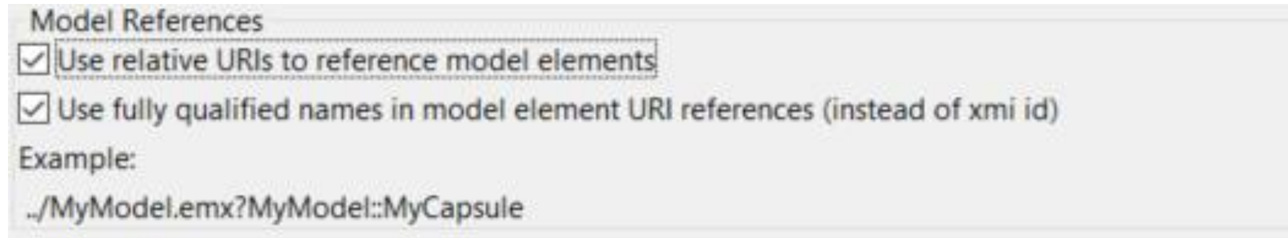
- Certain commands in RTist involve deletion of files and/or folders
  - Cleaning a TC
  - Removing code preview
  - ...etc
- Now, if the required files or folders cannot be deleted, a clear error message is shown
  - Previously there would be a silent failure in such situations which could be hard to understand the reason for
  - The new message is identical to what Eclipse would show if you directly try to remove the files/folders from the Project Explorer. Click the **Details** button to see exactly which file or folder that couldn't be deleted, and why.



Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

More Flexible Model References in Transformation Configurations

- A TC references model elements by means of URIs (e.g. list of source elements, top capsule etc)
- Such URIs can now be relative, and use qualified names instead of unique IDs to identify the element
  - Makes it easier to reuse a TC (e.g. by copy/paste) in different projects
- New preferences control how new URIs will created:
  RealTime Development – Transformation Configuration Editor – Model References

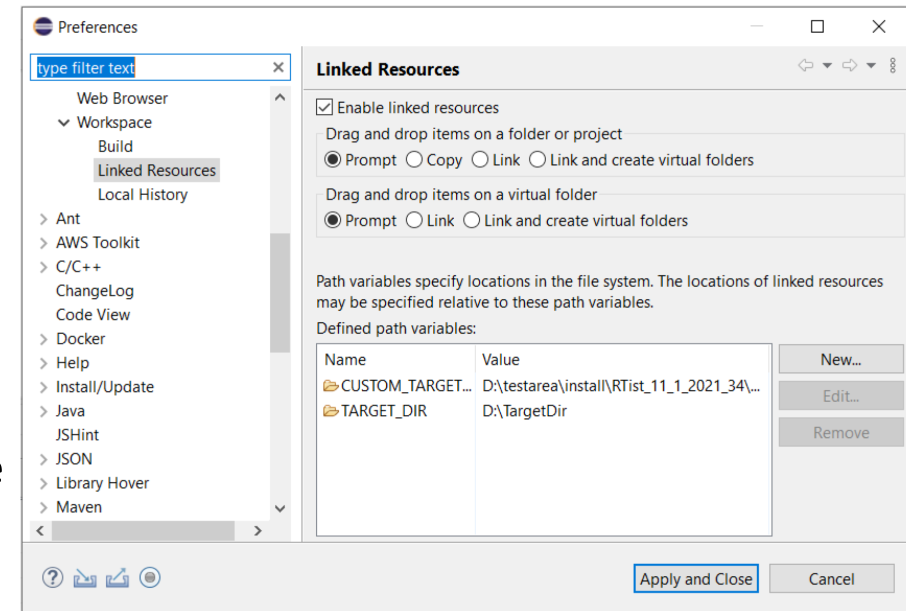


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

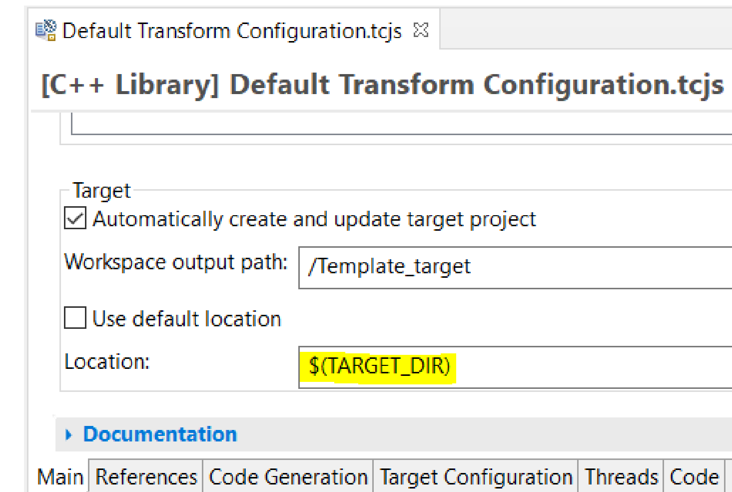
Support for Path Variables in Transformation Configurations

- Path variables can now be used in certain TC properties
  - Useful for those TC properties that specify a path
  - Define path variables in Preferences at **General – Workspace – Linked Resources**
  - This can be an alternative to using string substitutions (**Run/Debug – String Substitutions**) or environment variables in order to have a more generic TC (a path variable takes precedence over other kinds of variables, if the same variable name is used).



- The model compiler now prints a warning if a variable used in a TC property cannot be resolved

```
WARNING : Cannot resolve variable '$(TARGET_DIR)' in
'Location' property:'$(TARGET_DIR)'
```



Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Using std::chrono Library Types when Setting Timers

- The TargetRTS was updated to allow timers to be set using types from the std::chrono library
  - New overloads of Timing::Base functions `informAt`, `informIn` and `informEvery`
- Makes it easier to integrate with other code using std::chrono for time management
- No need to always specify fractions of seconds in nanoseconds (as with `RTTimespec`)

Example: Setting a timer to expire in 2.5 seconds

Using RTTimespec (the only option previously)

```
timer.informIn(RTTimespec(2, 500000000));
```

Using std::chrono::milliseconds

```
timer.informIn(std::chrono::milliseconds(2500));
```

Using operator""ms (requires C++ 14)

```
timer.informIn(2500ms);
```


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

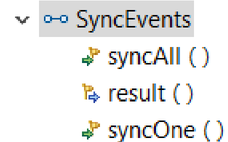
New TargetRTS Flag for Faster Plugin Capsule Part Imports

- When importing a capsule instance into a plugin capsule part a run-time check `RTActor::isReferencedBy()` is performed to ensure there are no cycles in the reference graph
- This run-time check can sometimes take too much time
- The TargetRTS now provides a new compile flag `RTIMPORT_ISREFERENCEDBY_CHECK` for disabling this run-time check
  - Set it to 0 in `RTLibSet.h` or `RTTarget.h` to disable the check

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com
HCL SOFTWARE

Named Event Enumerations in Generated Code

- Previously the enums generated for protocol classes were anonymous
- Now they are named ”RTInEvents” and ”RTOutEvents” respectively
  - Makes it possible to reference them from other code




```
 9 struct SyncEvents
10 {
11     class Base : public RTRootProtocol
12     {
13     public:
14         inline Base( void );
15         inline ~Base( void );
16         enum RTInEvents
17         {
18             rti_syncOne = rtiLast_RTRootProtocol + 1
19             , rti_syncAll
20         };
```

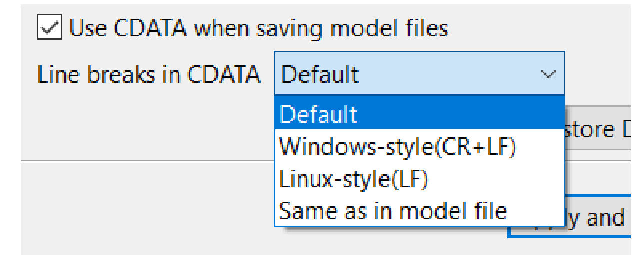
```
34     class Conjugate : public RTRootProtocol
35     {
36     public:
37         inline Conjugate( void );
38         inline ~Conjugate( void );
39         enum RTOutEvents
40         {
41             rti_result = rtiLast_RTRootProtocol + 1
42         };
```


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

Configurable Line Endings for CDATA Sections

- It’s now possible to control which kind of line breaks to be used when saving model files containing CDATA sections
  - A new preference was added: **Modeling – Line breaks in CDATA**



- By default, any line break is accepted. If you want to ensure consistent line breaks in all CDATA sections, use one of the other three options
  - Windows-style (CR+LF)
  - Linux-style (LF)
  - Same as in model file (read the first line break from the model file and use that in CDATA sections in that file)
- Having consistent line endings in CDATA sections can simplify textual comparison of model files

Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

Unit Testing of Capsules using the Mocha Framework (1/3)

RTist only

- Mocha is a popular JavaScript framework for testing asynchronous applications

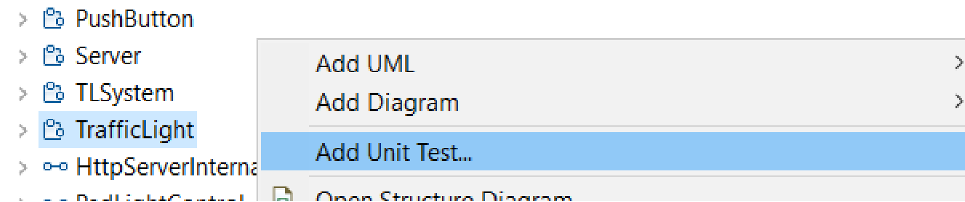
- It's now possible to use Mocha also for unit testing capsules

simple, flexible, fun

  - Provided by a new component that can be selected when installing

  - This feature integrates with NodePlus if it also is installed. However, it can also be used with any other JavaScript development environment.

- To create a Mocha unit test for a capsule, invoke the new context menu command **Add Unit Test**



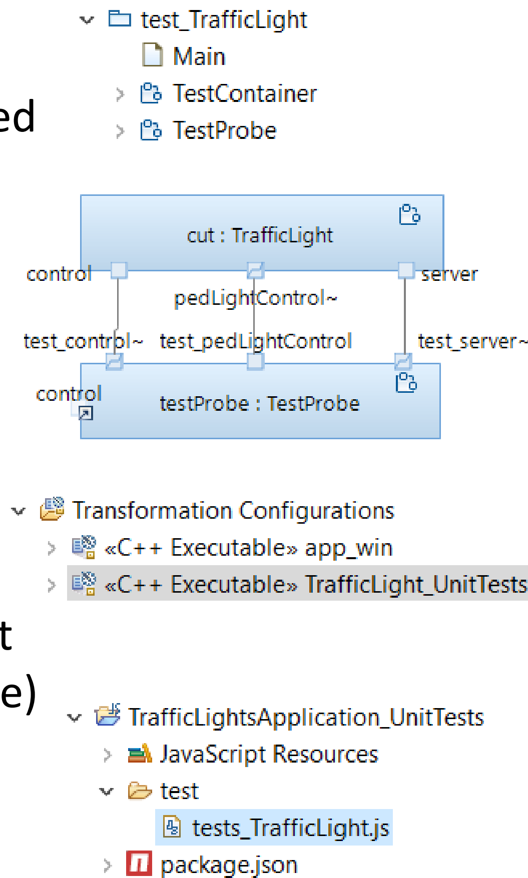
Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

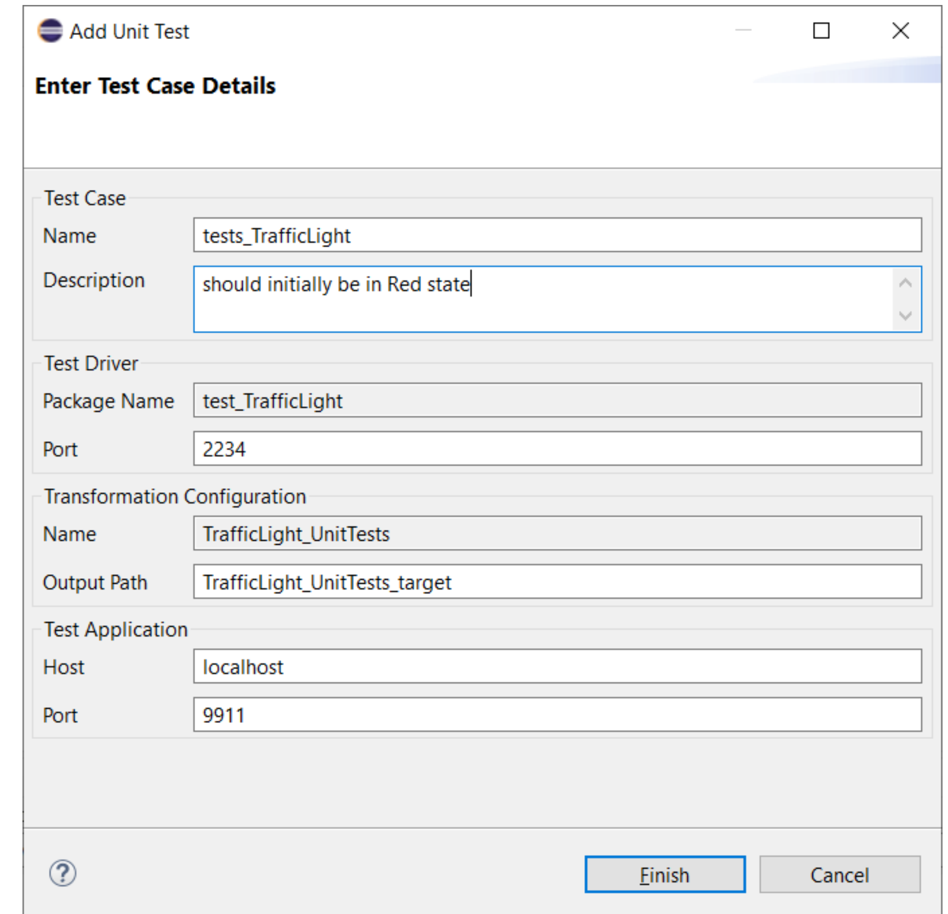
HCL SOFTWARE

Unit Testing of Capsules using the Mocha Framework (2/3)

RTist only

- The **Add Unit Test** command creates everything necessary for writing a unit test for the capsule
  - A test driver model where all service ports of the capsule under test (”cut”) are connected to similar but conjugated ports of a test probe capsule
  - A TC for building the test driver model into an executable that uses the TcpServer library for exposing all test probe ports to the Mocha test script
  - A Node.js project with a Mocha test script ready to implement the unit test (only generated if NodePlus is available)





Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

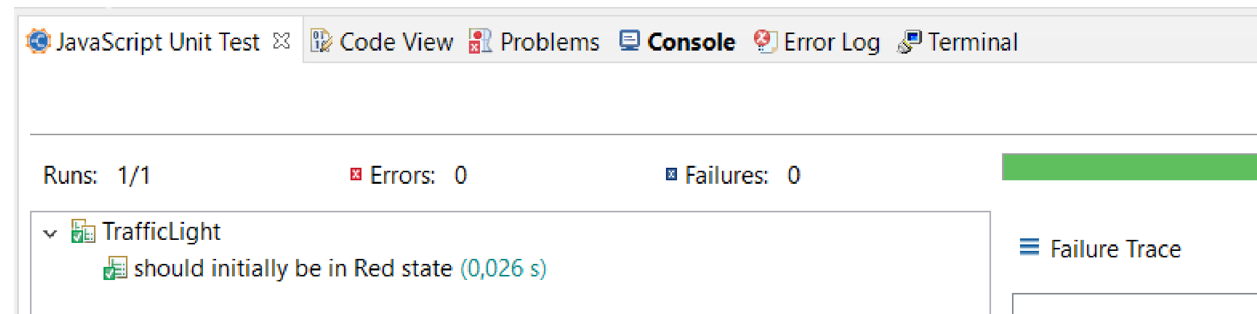
Unit Testing of Capsules using the Mocha Framework (3/3)

RTist only

- The unit test can be executed right away
  - Build the test driver TC (only needed the first time, and whenever you change the capsule under test)
  - Then use your JavaScript IDE to run and debug the test. With NodePlus the steps are:
    - Install the Node.js dependencies for the JavaScript project by right-clicking on the project and do **Run As – npm install**.
      (This is only needed the first time. It is assumed you already have installed Mocha on the machine.)
    - Run the testcase by right-clicking on the testcase .js file and do **Run As – JavaScript Unit Test**
    - If needed you can debug the unit test using **Debug As – JavaScript Unit Test**
    - The test execution result is shown in the **JavaScript Unit Test** view

tests_TrafficLight.js | TestContainer | TrafficLight_UnitTests.tcjs

```
1  var assert = require('assert');
2  describe('TrafficLight', function() {
3      it('should initially be in Red state', function() {
4          this.timeout(15000);
5          const testProbe = require ('rt-test-probe')('localhost', 9911);
6          return testProbe.startListenForEvents(2234)
7          .then((data) => {
8              // TODO: Implement test here
9          })
10         .finally(() => {
11             testProbe.stopListenForEvents();
12         });
13     });
14 });
```

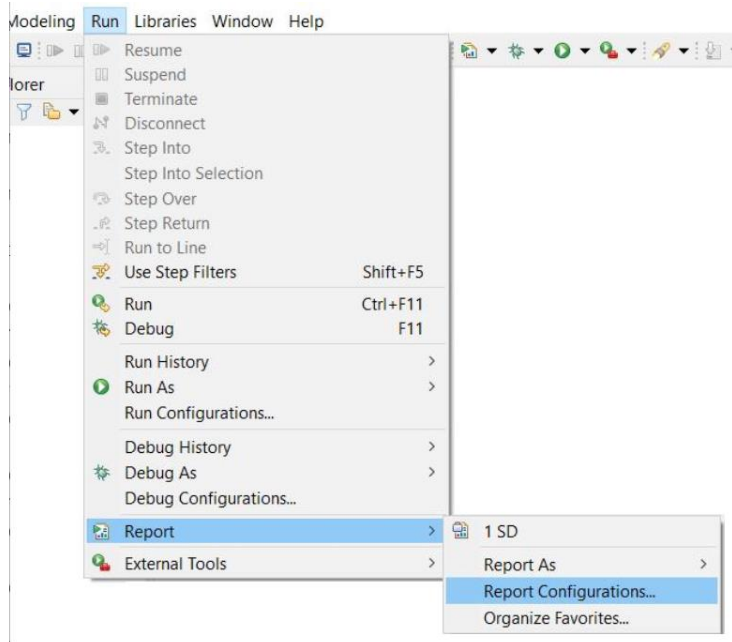


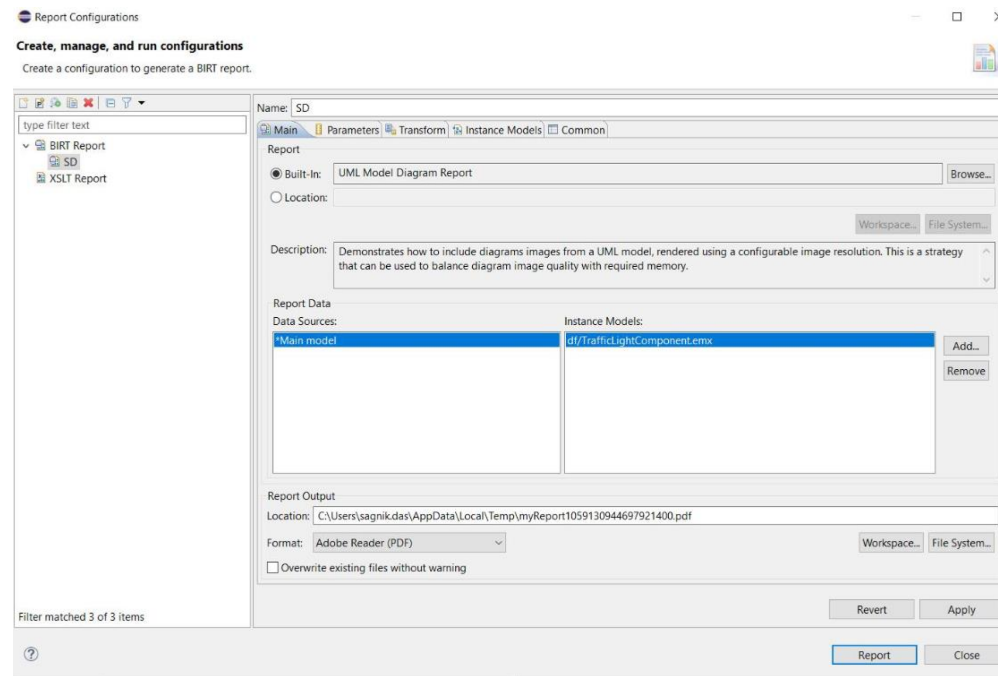
Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

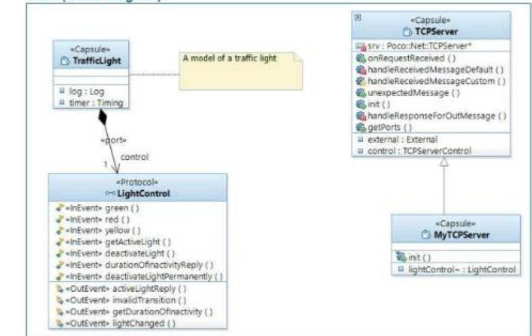
HCL SOFTWARE

Reporting with BIRT

- Create reports that include information from an RTist model
  - Same capabilities as in RTist 10.3, but now adapted for recent Eclipse versions (supports RTist 11.0 and RTist 11.1)
  - Delivered as a separate update site on [our InfoCenter](). Installation instructions are included in the ZIP file.





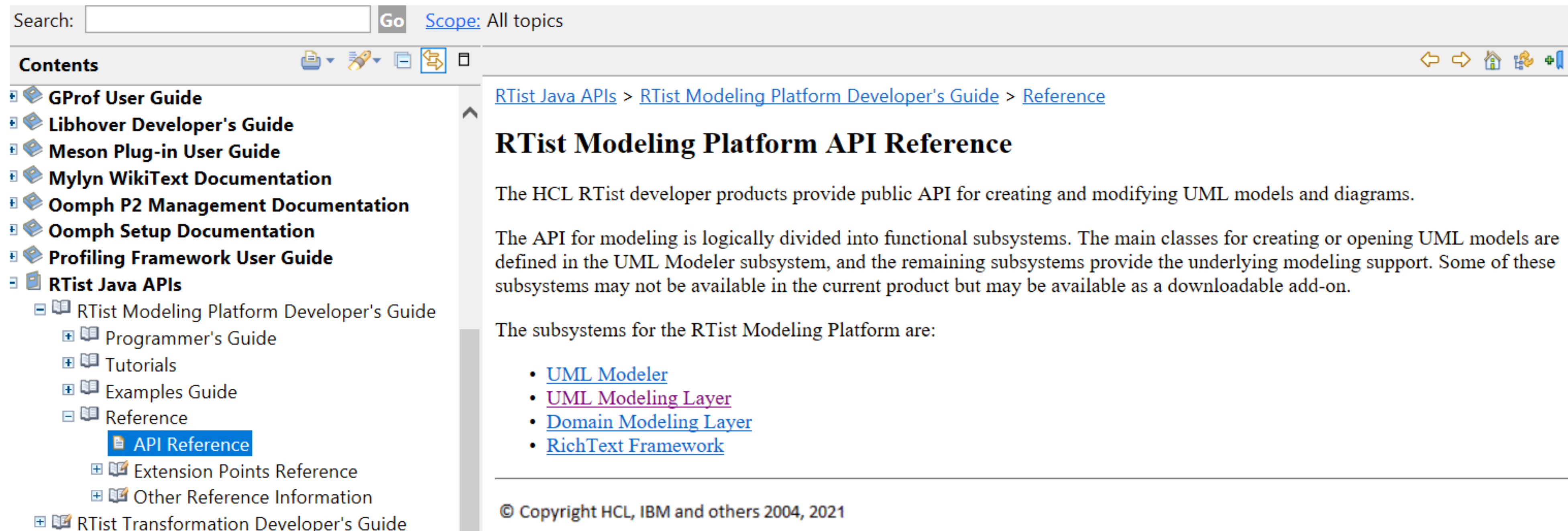


Copyright © 2022 HCL Technologies Limited | [www.hcltechsw.com]()

HCL SOFTWARE

Java API Improvements

- A new method for programmatically redefining an inherited operation was added
  - `com.ibm.xtools.uml.redefinition.RedefFactory.getOperationRedefinition()`
- Read more about this new method in the Help (RTist Java APIs – Reference – API Reference – UML Modeling Layer)

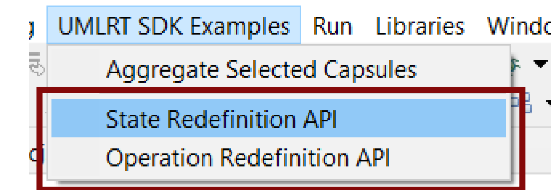


Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL SOFTWARE

UML RT SDK Sample

- The sample was updated with two new commands showing how to redefine states and operations using existing and new Java APIs
  - Also shows how to create generalizations programmatically



Copyright © 2022 HCL Technologies Limited | www.hcltechsw.com

HCL

Relationship™

BEYOND THE CONTRACT

$7 BILLION ENTERPRISE | **110,000** IDEAPRENEURS | **31** COUNTRIES