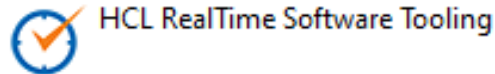


What's New in HCL RTist 10.3

updated for sprint 2020.10

Overview

- ▶ RTist 10.3 is based on Eclipse Photon (4.8)
- ▶ HCL RTist 10.3 is functionally equivalent and 100% compatible with IBM RSARTE 10.3



HCL RealTime Software Tooling

Version: 10.3.0.v20200303_1813

Release: 2020.10

(c) Copyright IBM Corporation 2004, 2016. All rights reserved.

(c) Copyright HCL Technologies Ltd. 2016, 2020. All rights reserved.

Visit <https://RTist.hcldoc.com/help/topic/com.ibm.xtools.rsarte.webdoc/users-guide/overview.html>



Internal Transitions

- ▶ Internal transitions for the enclosing composite state can now be shown in a compartment beside the state chart diagram of the composite state.
- ▶ The transitions can be shown in columns to get a compact presentation



Rulers & Grid

Appearance

Advanced

Filtering

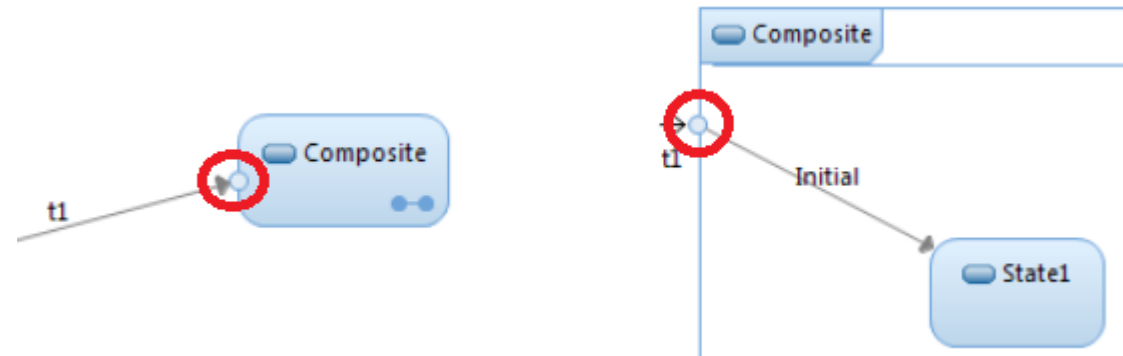
- Show Junction and Entry/Exit Point Names
- Show Choice Point Names
- Show State Names
- Show Transition Names
- Show Internal Transition Comments inside Compartment
- Show Transition Effects
- Show Transition and Trigger Guards
- Show Transition Events
- Show Trigger Parameters
- Show Trigger Ports

Enclosing State Internal Transitions

Show beside state diagram

Navigation in Hierarchical State Machines

- ▶ A transition path can now more easily be followed in a hierarchical state machine by means of the "Go Inside" and "Go Outside" commands
 - Select an entry/exit point before going outside to highlight the same entry/exit point in the enclosing state machine. And in the same way when following a transition path into a composite state.

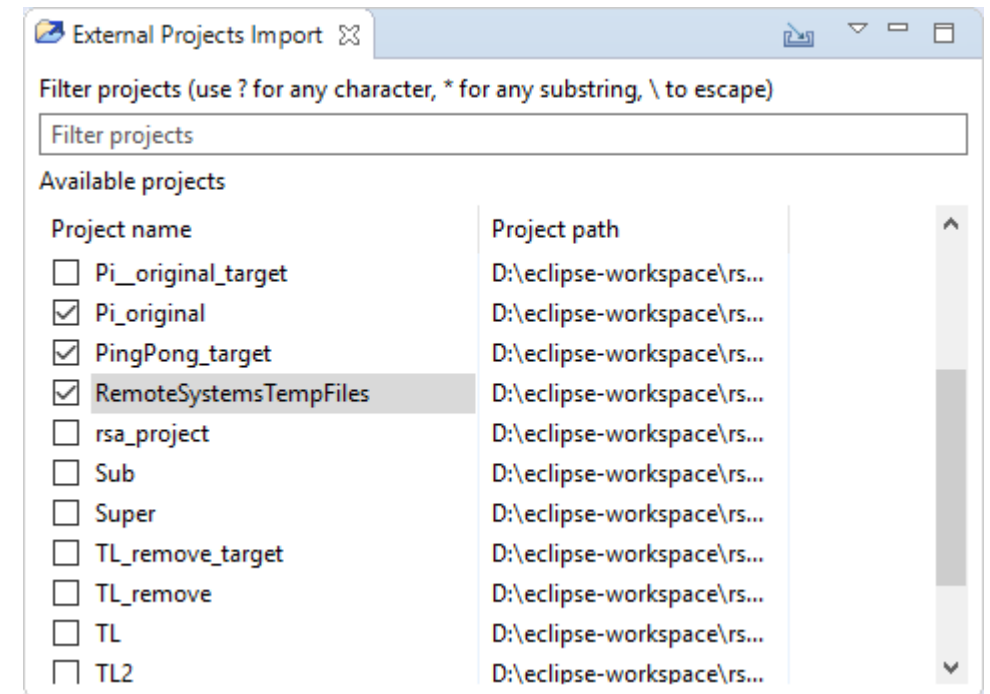
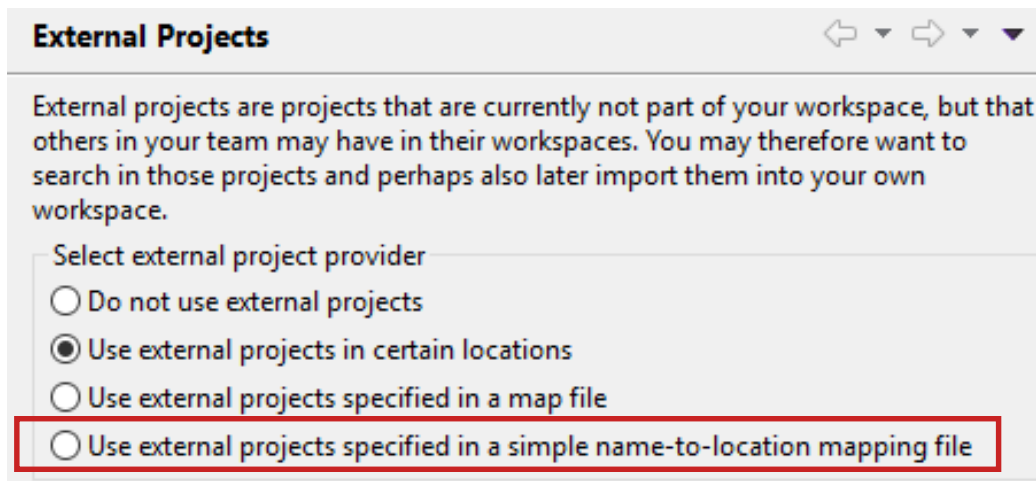


Navigating to Diagrams in the Project Explorer

- ▶ Perform *Navigate – Project Explorer* in the background of a diagram to navigate to the diagram in the Project Explorer
 - Previously it was only possible to navigate to the model element owning the diagram
 - Navigating to diagrams from search results also work now

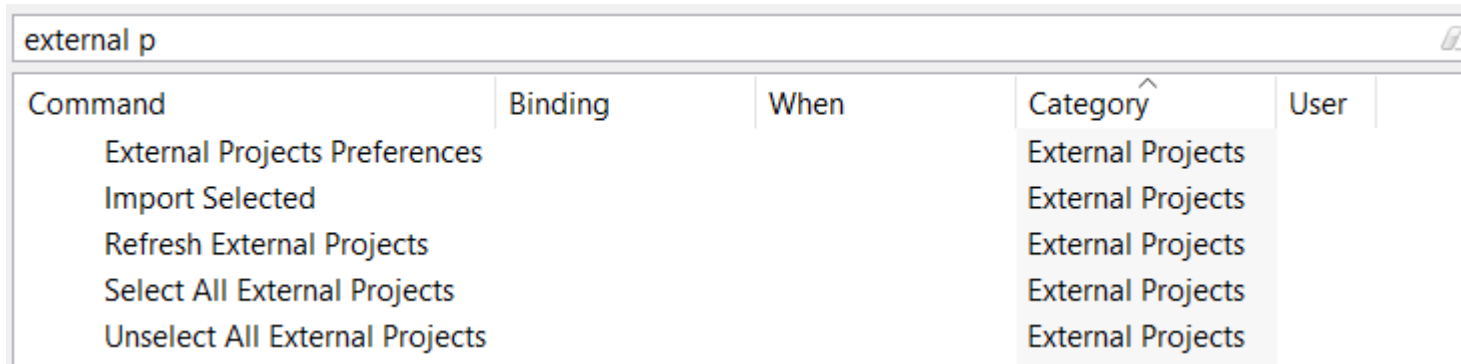
External Projects Import (1/2)

- ▶ A new view makes it easier to import external projects into the workspace
- ▶ Define where to look for external projects and then import found projects easily into the workspace
- ▶ Now possible to specify the location of external projects with a file on the same format as is supported by the model compiler



External Projects Import (2/2)

- ▶ Commands are available for working with the External Projects Import view using keybindings
 - Define your desired keybindings for the commands you want to use (there are no default keybindings for them)



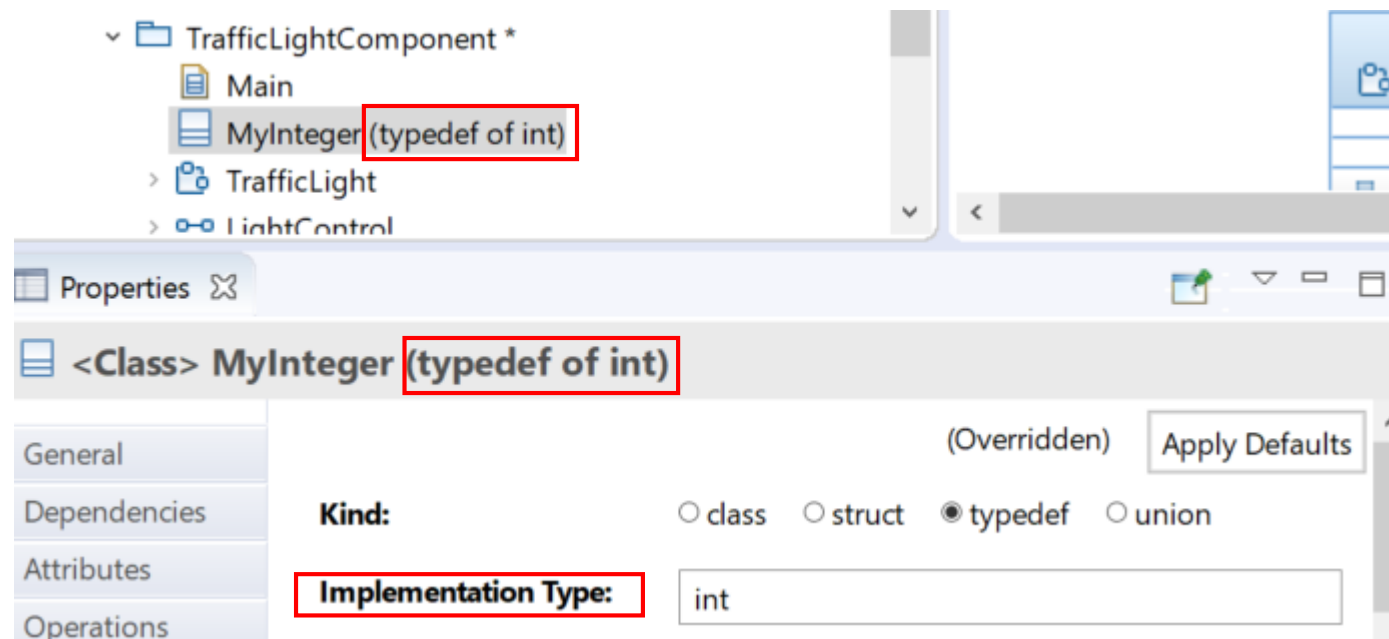
The screenshot shows a window titled "external p" containing a table with the following columns: Command, Binding, When, Category, and User. The Category column is currently selected, showing a dropdown menu with five entries, all labeled "External Projects".

Command	Binding	When	Category	User
External Projects Preferences			External Projects	
Import Selected			External Projects	
Refresh External Projects			External Projects	
Select All External Projects			External Projects	
Unselect All External Projects			External Projects	

- ▶ The External Projects Import view can now be filtered using project paths
- ▶ Often the view is automatically updated when external projects change (can for example be caused by switching branch in Git). If not, you can use the Refresh External Projects to update the list of external projects.

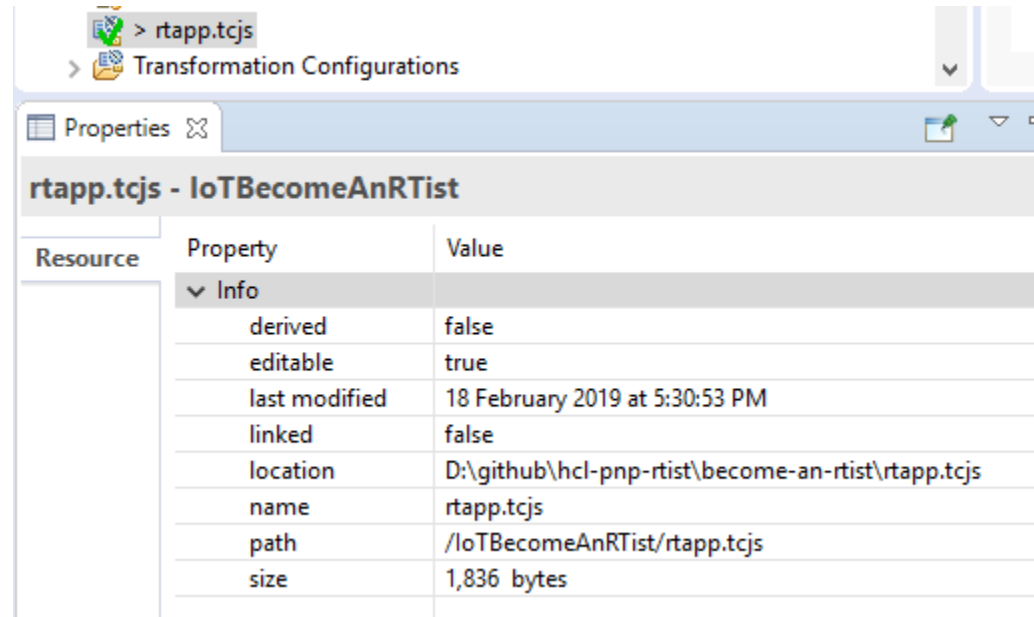
Typedef Types

- ▶ The typedef implementation type is now shown in the Project Explorer, Properties view etc.
- ▶ The “Implementation Type” property is now accessible without scrolling in the Properties view

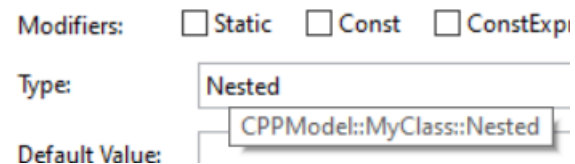


Properties View

- ▶ When a TC is selected information about the .tcjs file is now shown in the Properties view

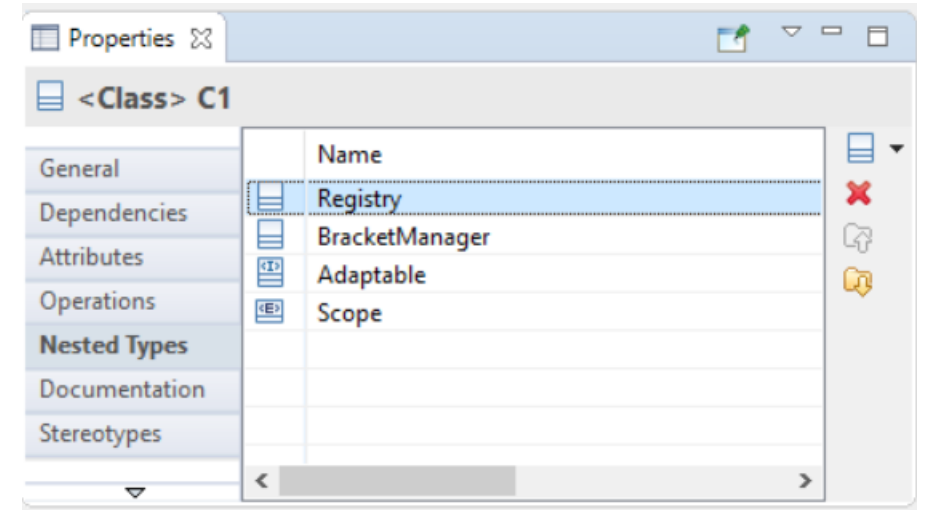


- ▶ The tooltip of a type reference now shows the fully qualified name of the type



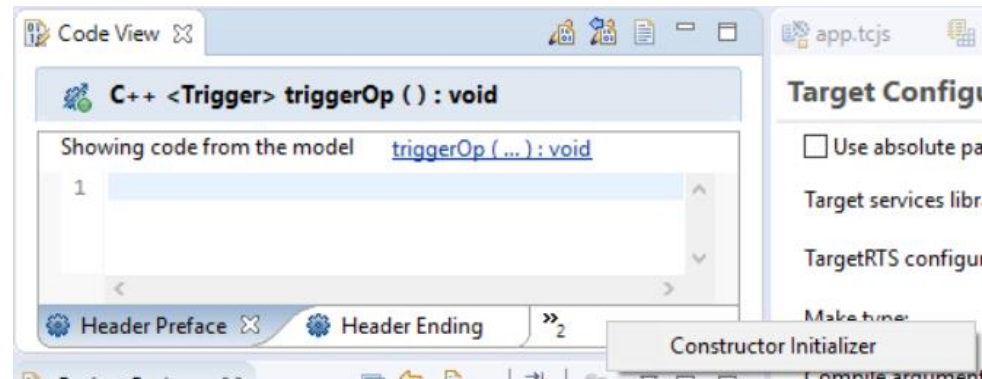
Properties View

- ▶ A new property page “Nested Types” for classes and capsules
- ▶ Makes it easier to control the order of nested types
 - This can sometimes be important to ensure that C++ classes are declared before they are referenced
- ▶ Also supports creating and deleting nested types (classes, interfaces and enumerations)



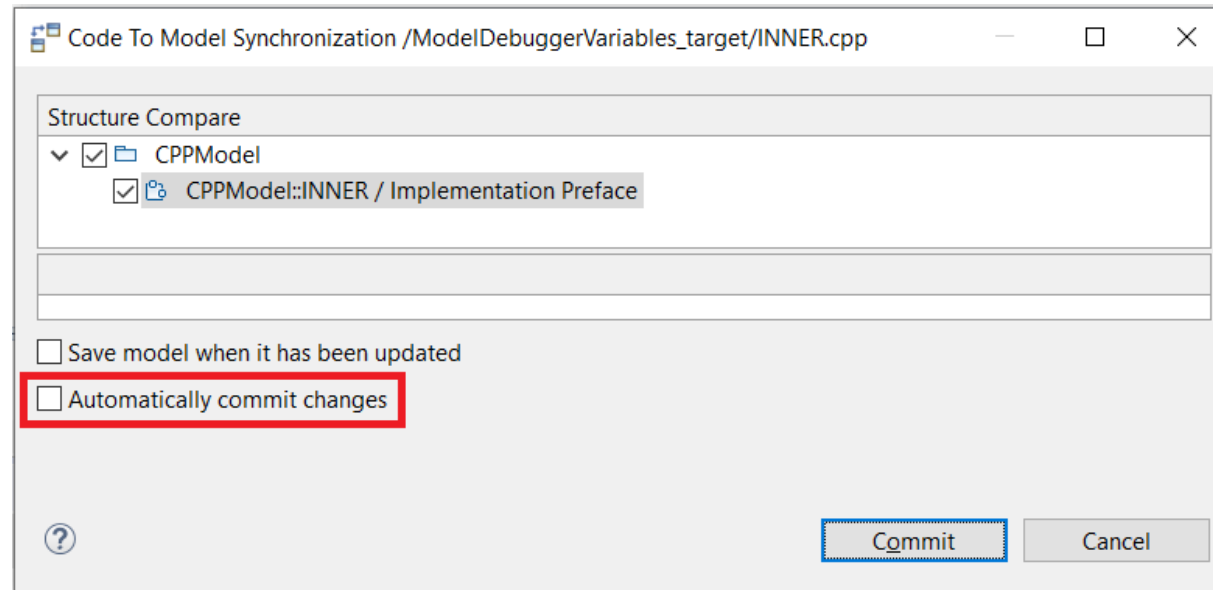
Code Editing

- ▶ Some irrelevant code snippets are now hidden from the Code view
 - Constructor initializer for non-constructor operations
 - Body for member interface operations and trigger operations
 - Implementation preface/ending for elements for which no code is generated in the .cpp file (e.g. non-static attributes)
- ▶ Hidden code snippets can be made visible by means of the context menu



Code to Model Synchronization

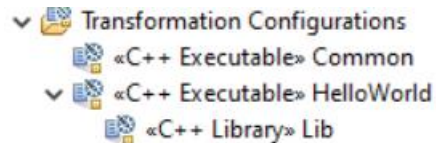
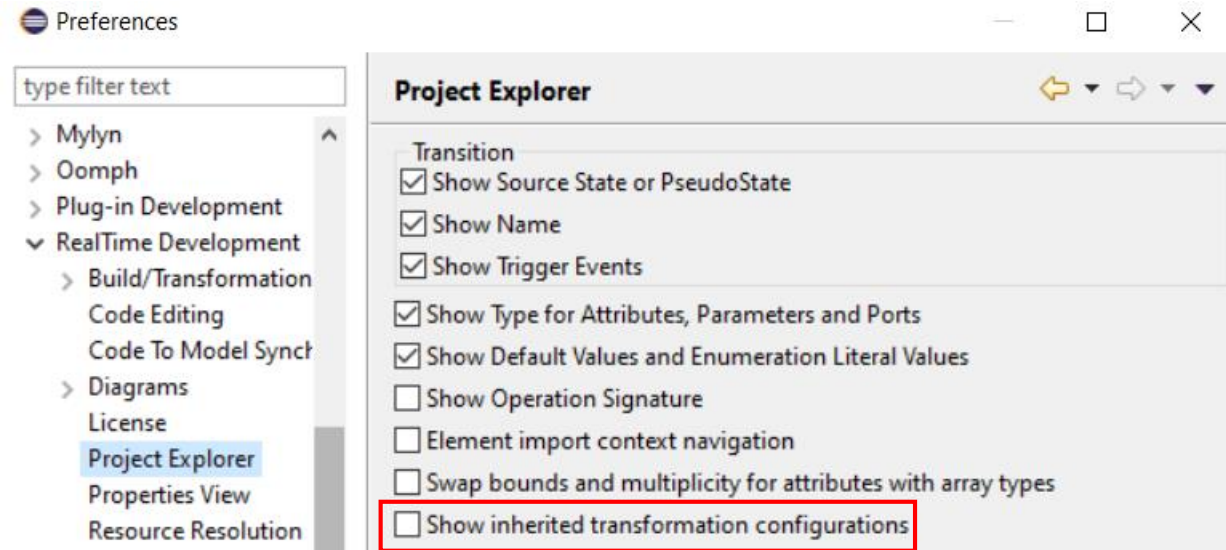
- ▶ The dialog that appears when synchronizing changes made in a generated file can now be automatically closed



- ▶ The new checkbox corresponds to a new preference *RealTime Development – Automatically commit changes from modified generated files*
 - When set, the dialog will be automatically closed after 2 seconds

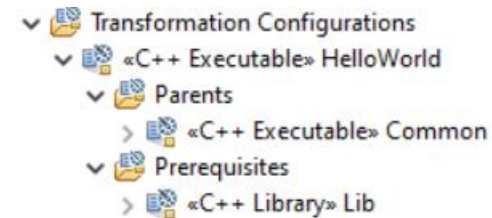
Hide or Show Inherited TCs in the Project Explorer

- ▶ A new preference allows you to hide inherited TCs in the Project Explorer
- ▶ By default inherited TCs are not shown. Makes the Project Explorer more compact.



with preference not set (default)

- Only prerequisite TCs are shown as nested TCs

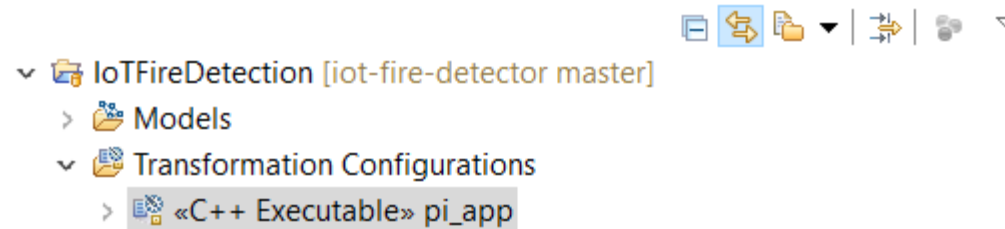


with preference set

- Both inherited and prerequisite TCs are shown as nested TCs (under virtual folders)

Transformation Configuration Editor (1/3)

- ▶ Now supports the “Link with Editor” button of the Project Explorer



- ▶ Improved the command “Navigate to Inherited Value” so that it works in more situations



Transformation Configuration Editor (2/3)

- ▶ If a TC contains anything but single and simple assignments of TC properties, only the Code tab can be shown
 - Other tabs are hidden since form-based editing is not possible in this case
- ▶ An annotation marker is shown in the ruler of the Code tab to pinpoint these JavaScript statements
 - Remove these statements to get back all tabs in the TC editor

```
9 tc.createTargetProject = true;
10x) tc.createTargetProject = false;
11 tc.targetProject = '/Course_target';
12 tc.compilationMakeCommand = 'mingw32-make';
13x) if (tc.createTargetProject)
14     tc.compilationMakeType = MakeType.MS_nmake;
15 tc.compileArguments = '${DEBUG_TAG}';
16x) JavaScript statement beyond simple assignment
17 tc.targetConfiguration = 'WinT.x64-VisualC++-15.0';
18 tc.targetServicesLibrary = 'D:\\git\\rsarte-target-rt\\r
19 tc.targetPlatform = 'platform/resource/Course/CDDMod
20 <
```

Code

Transformation Configuration Editor (3/3)

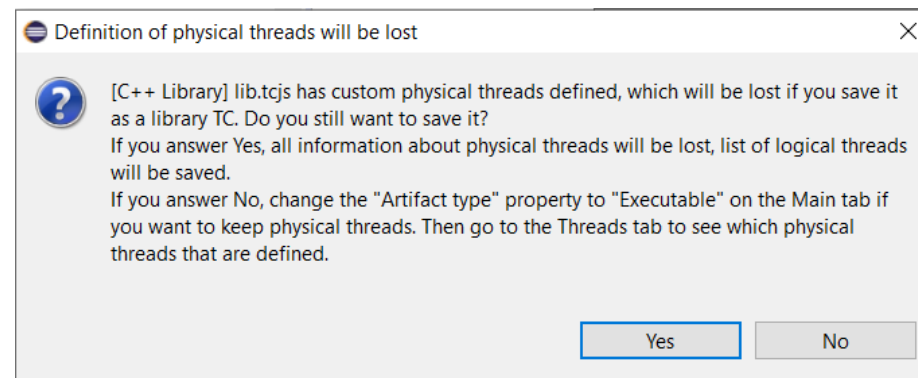
- ▶ Relative resource URIs can now be used
 - Use of such URIs makes it possible to copy the project that contains the TC (references bind to contents in the copied project, and not in the original)
 - However, there are some limitations for relative resource URIs. TC reference search cannot find them and navigation in the TC editor does not always work
- ▶ The representation for logical threads in library TCs was simplified
 - Now only the logical thread names are stored in the TC file
 - You will be warned if the conversion of an executable TC to a library TC will lead to thread information loss

References

Inherited transformation configurations:

```
../CppExecutableDefaults.tcjs
```

```
tc.threads = [  
    'Main',  
    'CapsuleThread',  
    'ExtThread'  
];
```



Access to Top-Level TC Properties from Prerequisite TCs

- ▶ A prerequisite TC can now access properties from the built TC (a.k.a. the “top-level TC”)
- ▶ This is useful when creating libraries to be built in many different contexts

For example:

```
let tc = TCF.define(TCF.CPP_TRANSFORM);
```

```
let topTC = TCF.getTopTC().eval;
```

```
tc.targetConfiguration = topTC.targetConfiguration || 'WinT.x64-VisualC++-15.0';
```

*eval evaluates all TC properties to simple values,
taking default values and inheritance into account*

*if built as a prerequisite, use
the target configuration from
the top-level TC*

*otherwise, fallback to a default
target configuration*

Enable Support for File Artifacts

- ▶ The TC property “Enable Support for file artifacts” has been removed
- ▶ File artifacts will be translated to C++ if they are present among the source elements (i.e. they are now handled in the same way as all other kinds of model elements)

Support for In-Class Member Initializers (C++ 11)

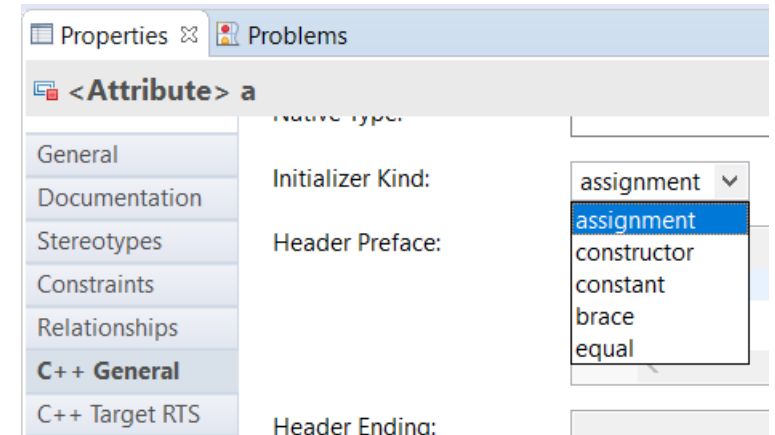
- ▶ Two new kinds of initializers are now supported for an attribute. They will both cause attribute initialization to be generated in the class definition itself, in the header file (supported since C++ 11).

- **Brace** – Use brace-style initialization of the corresponding C++ variable

```
int a { 4 } ;
```

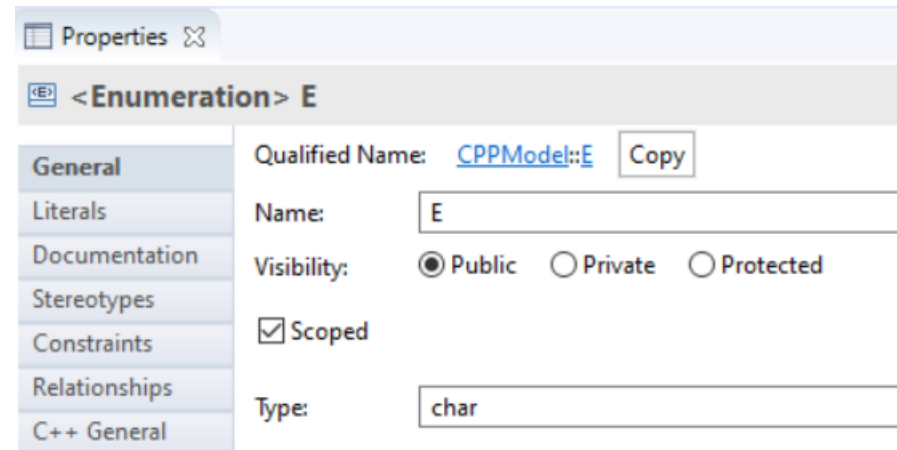
- **Equal** – Initialize the corresponding C++ variable with an “equal assignment”

```
int a = 4;
```



Support for Scoped Enums (C++ 11)

- ▶ New properties for Enumerations: Scoped and Type
 - Set **Scoped** to represent a scoped enum, and set **Type** to use another underlying type for the enum than the default (int)



- ▶ Note: The TargetRTS will not make a difference between scoped and unscoped enums. Their values are shown as integers at run-time (e.g. in traces and in the Variables view)

Support for Rvalue References (C++ 11) and Better Handling of Native Types




- ▶ Previously, native C++ type references were parsed and the parsed representation was then used when generating the code. There were several issues with this approach:
 1. Limitations in what kinds of types that could be supported (e.g. rvalue references (&&) were not supported)
 2. Compiler specific extensions were not supported (e.g. `__stdcall` on Windows)
 3. Informal content such as a comment in the type reference was lost
 4. Mistakes made in the native type could often go unnoticed since the parser accepted even some syntax that is illegal in C++
- ▶ Now the general rule is to print native C++ types exactly as specified in the model
 - Thereby the model compiler now supports arbitrary complex native types (including rvalue references, compiler specific extensions and comments).
 - For backwards compatibility parsing is still done to handle a few important special cases (such as automatic generation of qualifiers for references to nested types) and detection of potential problems in type references will be reported with warnings.
 - **However, if your model contains faulty native type references, which were not previously detected, they might now lead to compiler errors. You should correct your model to fix such problems.**


```
public:  
    virtual ~MyClass( void );  
    MyClass( const MyClass & rtg_arg );  
    MyClass & operator=( const MyClass & rtg_arg );  
    MyClass & operator=( MyClass && rtg_arg ) = default;  
    MyClass( void );  
    MyClass( MyClass && /* ELENA V */ other );  
    unsigned int __stdcall ServerThread( unsigned int elenaParam01 );  
};
```

Operations without Return Parameters

- ▶ It's no longer necessary to explicitly set the return type to 'void' for an operation that does not return a result
 - An operation without a return parameter now translates to a void function without causing a warning
- ▶ Inconsistencies related to the return parameter are now detected (e.g. multiple return parameters, return parameter without specified type etc)

Return Type: [Set return type ...](#)

▼  doItJustDoIt ()
  return : int
  return2 : bool

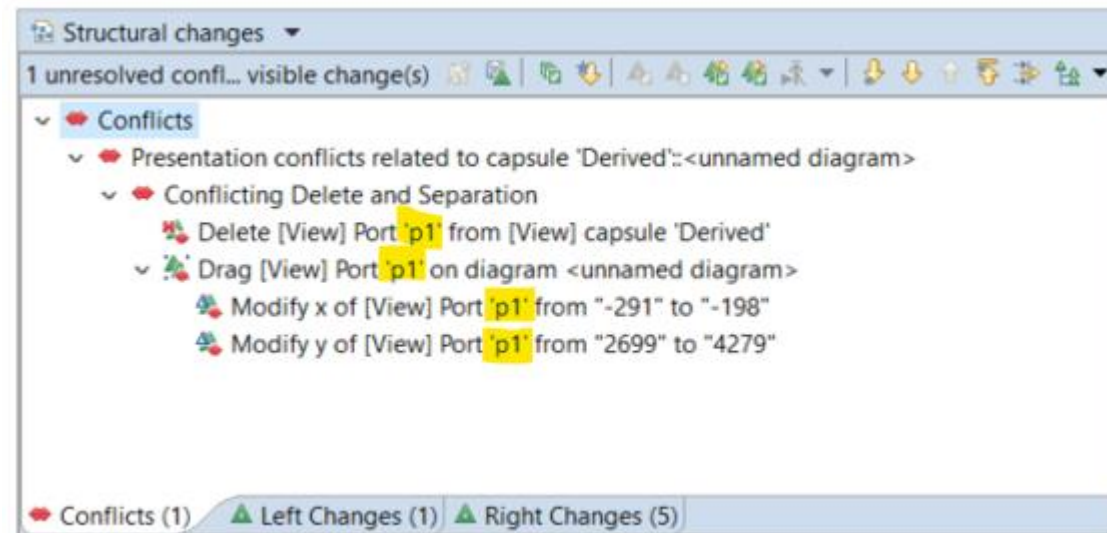
 <Operation> doItJustDoIt ()

General	Qualified Name: CPPModel::TOP::doItJustDoIt Copy
Parameters	Name: <input type="text" value="doItJustDoIt"/>
Exceptions	Visibility: <input checked="" type="radio"/> Public <input type="radio"/> Private <input type="radio"/> Protected
Documentation	Modifiers: <input type="checkbox"/> Static <input type="checkbox"/> Pure Virtual <input type="checkbox"/> Const <input type="checkbox"/> Virtual
Stereotypes	Return Type: <input type="text" value="int"/> Set return type ...
Constraints	Signature: <input type="text" value="doItJustDoIt () : bool"/>
Relationships	
C++ General	

```
10:39:30 : INFO : Transforming model...
10:39:30 : ERROR : CPPModel::TOP::doItJustDoIt : Several return types are specified : [int, bool]
10:39:30 : ERROR : Transformation completed with 1 error(s)
10:39:30 : ERROR : Skipping build step because of transformation errors
10:39:30 : INFO : Done. Elapsed time 0.373 s
```

Compare / Merge

- ▶ Changes and conflicts with a reference to a port now display the port name
 - Now works even if the port is defined in a file that is not part of the compare / merge session



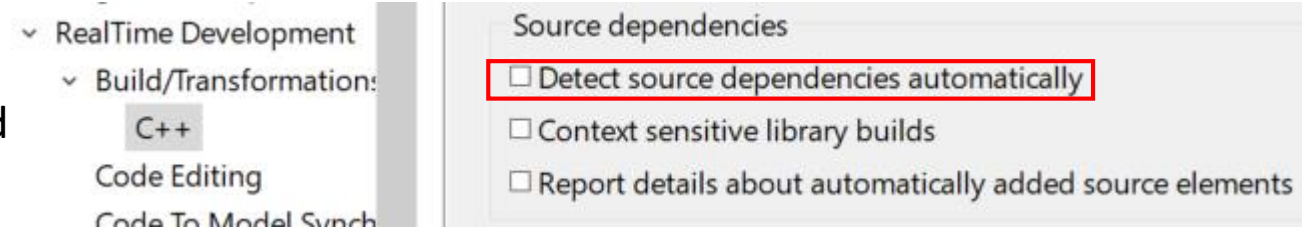
Model Compiler (1/5)

- ▶ Support for automatic detection of source dependencies

- The Sources list of the built TC can be dynamically extended to include all elements that are referenced (directly or indirectly) from built elements

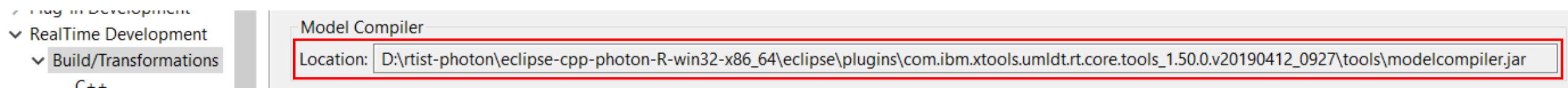
- A new message is printed for all automatically added source elements (so they can be added in the Code tab of the TC editor)

```
11:20:49 : INFO : Adding missing sources on-the-fly to "DetectSourceDependenciesAutomatically/HelloWorld.tcjs" :  
'platform:/resource/DetectSourceDependenciesAutomatically/HelloWorld.emx#_s1xKwPoYEemJs6K8xfurHQ' /* HelloWorld::HelloWorld */,  
'platform:/resource/DetectSourceDependenciesAutomatically/HelloWorld.emx#_utwnIPoYEemJs6K8xfurHQ' /* HelloWorld::MyClass */
```



- ▶ The location of the model compiler is now shown in the Preference page

- Makes it easier to run it from the command line



Model Compiler (2/5)

- ▶ Running the model compiler with no command-line arguments will now show its help
- ▶ Make is now always run as part of building (it will detect whether something needs to be built or not)
- ▶ There are new preferences for the build server (i.e. the model compiler running in server mode)
 - It can now be disabled, so it isn't automatically launched at start-up (useful in case no build features are used)
 - A new button for opening the build server log (helps troubleshooting)
 - A new preference for automatically restarting the build server when it has been idle for a certain time

Build Server

Launch build server at start-up

VM arguments: -Xverify:none -Xmx4024m

Port range (syntax is lower:upper), restart is required : 61004:61020

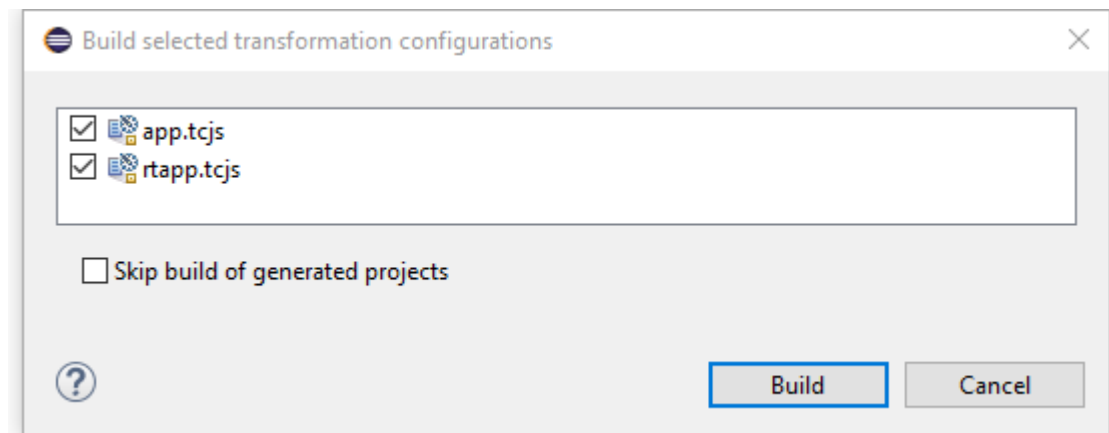
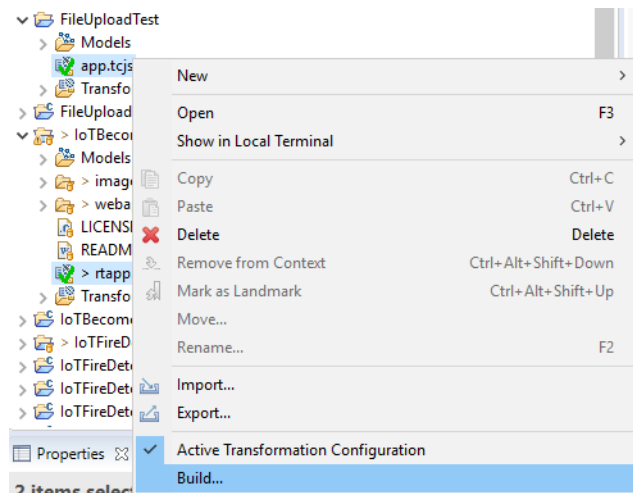
Status: server is running at port 61004

Restart Server Log

Restart build server on idle time (in mins) 30

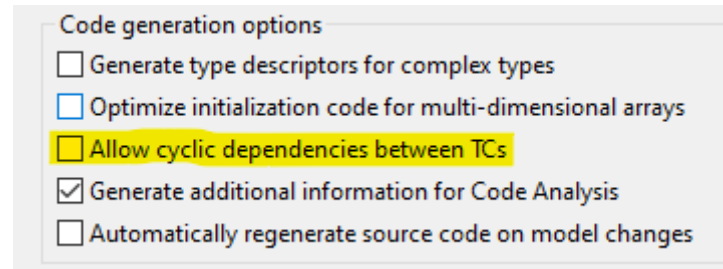
Model Compiler (3/5)

- ▶ It's no longer necessary to specify a license when launching the model compiler from command-line
 - The license of the RTist installation will be used
- ▶ Multiple TCs can now be built in one go
 - Select all TCs to build in the UI and perform the Build context menu command
 - The selected TCs will be built one by one consecutively



Model Compiler (4/5)

- ▶ Environment variables can now be used in the "Build folder" of an external library TC
- ▶ Build variant scripts can now update TC prerequisites during the build. Also, there are now error messages printed if a build variant script tries to modify any TC properties that are not allowed to be modified (e.g. Type or Parents). In general, the handling of errors that may arise when interpreting a built variant script has been improved.
- ▶ The model compiler now detects if names of generated files clash (i.e. accidental overwriting of another file with the same name)
 - Can for example happen if you have model elements with the same name in different packages
- ▶ The model compiler can now build TCs with cyclic prerequisites (i.e. cyclic library dependencies)
 - A new preference must be set



Model Compiler (5/5)

- ▶ Fully qualified state names can now be generated
 - Controlled by a new TC property
 - Useful if state names are not unique within a hierarchical state machine

```
static const char * const rtg_state_names[] =  
{  
    "<machine>"  
    , "Waiting"  
    , "Waiting::Engine1Stalled"  
    , "Waiting::Engine2Stalled"  
    , "Engine1Stalled"  
};
```

- ▶ When running the model compiler from the command-line RSA_RT_HOME is now automatically set
 - You can override it using an environment variable or a Java system variable
- ▶ When building several TCs from the command-line the model compiler now stops if one TC fails to build
 - A new command-line option --keepBuilding can be used to always build all TCs regardless if errors occur

Code Generation

Top capsule: HelloWorld::HelloWorld

Compile data classes individually

Generate code qualifiers

Generate fully qualified state names

Optimize handling of frequent triggers

Cleaning External Libraries

- ▶ External Library TCs have a new property for specifying a Clean Command
 - Use for example "make clean", invoke a "clean script" or something else
- ▶ When it has been specified it's possible to use the Clean command also on external library TCs
- ▶ This is currently an experimental feature
 - Enable it in *Preferences – Experimental Features*

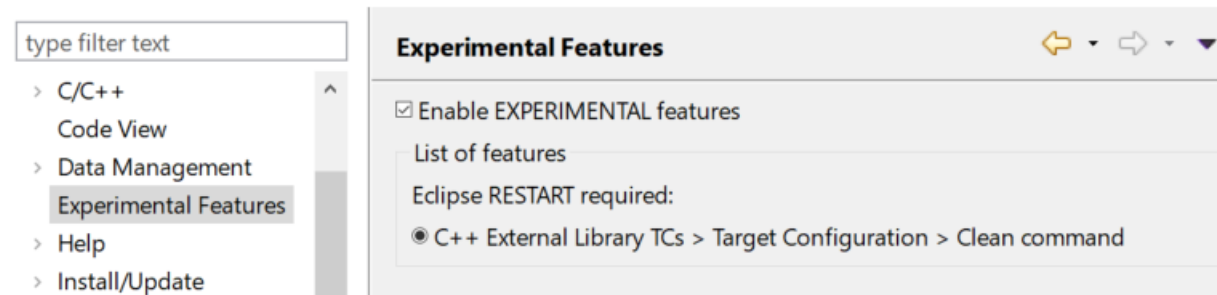
Target Configuration

Generate make file for external CDT project

Build command:

Build folder:

EXPERIMENTAL Clean command:



Code Analysis

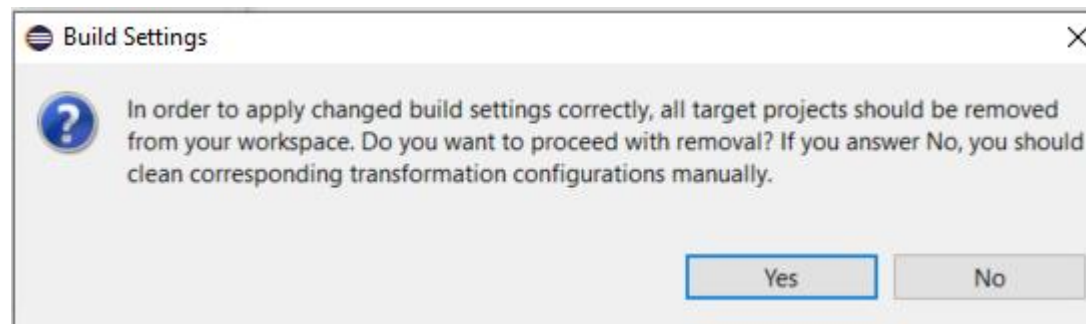
- ▶ Generated CDT projects are now set-up to use CDT GCC Built-in Compiler settings
 - Makes code analysis work better

```
14:19:58 : INFO : Generating XML file with CDT projects info...
```

```
14:19:58 : INFO : CDT GCC Built-in Compiler Settings provider will be applied to generated target projects for Code Analysis support
```

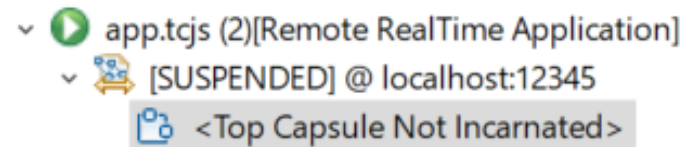
```
14:19:59 : INFO : Generating XML file with CDT projects info... Done
```

- Disable this in Preferences if not using a GNU compiler (*RealTime Development – Build/Transformations – C++ - Generate additional information for Code Analysis*)
- Note: When this preference is changed all target projects should be removed. A dialog appears to do this automatically.

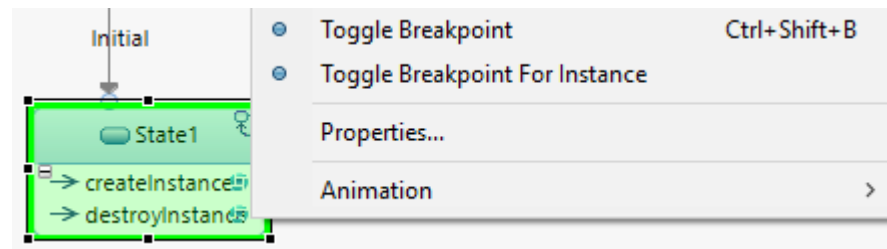


Model Debugger (1/5)

- ▶ The Debug view now shows more clearly when the top capsule has not yet been incarnated
 - Also, ports for the top capsule are not shown until it has been incarnated



- ▶ “Type-wide” breakpoints are now supported
 - Breakpoints set before starting a model debug session applies to all instances of a capsule (“type-wide”)
 - In an instance diagram you can now choose if a breakpoint should be “type-wide” or only apply on that particular instance



Model Debugger (2/5)

- ▶ Breakpoints are now more clearly described in the Breakpoints view
 - Distinction between “type-wide” and instance breakpoints



- ▶ In-line editing of data for an event in the Events view is now supported
 - Only use the browse (...) button when entering data spanning over multiple lines

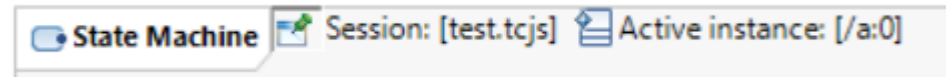
yellow	void	
createInstance	int	1
setTimer	int	1

- ▶ The number of instances in a capsule part are now shown in the Debug view
 - A special icon is used to denote empty capsule parts

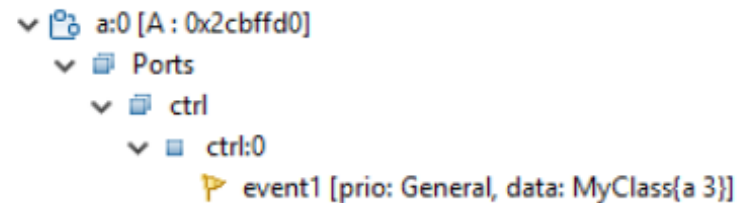


Model Debugger (3/5)

- ▶ Instance diagram labels are more compact now
 - Shows the launch configuration name (by default named as the TC) and the instance path



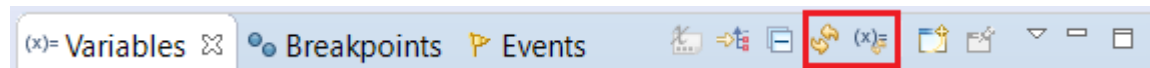
- ▶ Event queues are now shown in the Debug view. Under each port instance you can see the events waiting to be dispatched to that particular port instance.
 - The event priority and data (if any) is shown



- ▶ The Variables view now shows also private inherited attributes of a capsule instance

Model Debugger (4/5)

- ▶ The Variables view can now show values of attributes also when the application is running
 - It's no longer necessary to suspend the application to view attribute values
 - The view is updated each time a new capsule instance is selected in the Debug view
 - It's also possible to force an update by pressing a new Refresh button in the tool bar
 - A new toggle button Toggle Monitor Variables lets the view automatically update whenever the value of an attribute changes at run-time

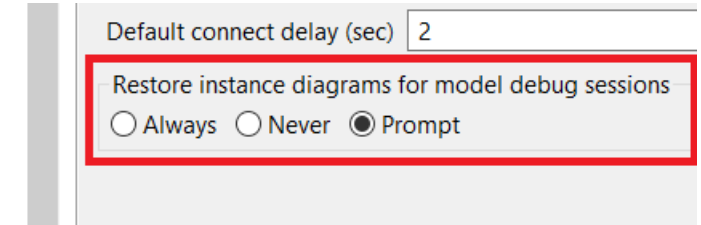
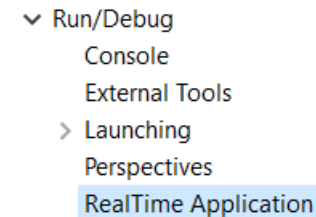
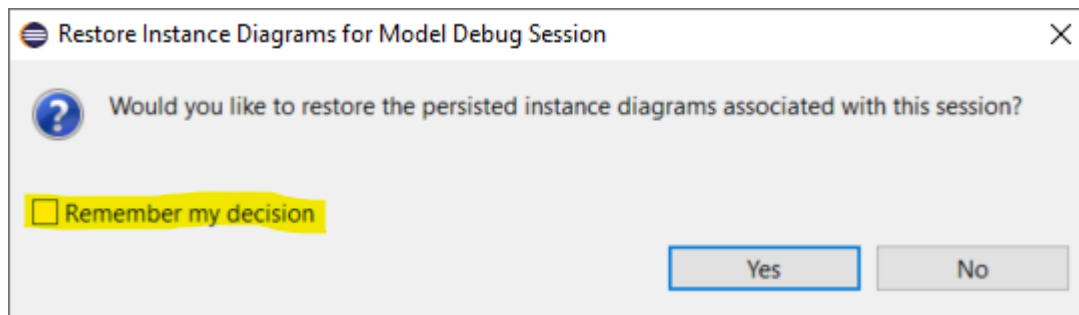


- ▶ Capsule incarnation initialization data can now be traced
 - Traced by means of a new “initialize” event

4	Capsule Incarnated			new INNER()	iINNER:0	36565 ms
5	Event Received on Port	initialize	int 0	Initialize@<system>	iINNER:0	36565 ms

Model Debugger (5/5)

- ▶ The file format for traces is now independent on the locale of the machine and on the workspace
 - Makes it easier to share trace files between users
- ▶ Now it's possible to avoid being prompted for restoring instance diagrams when launching a model debug session
 - New checkbox in dialog, and new corresponding preference



CORBA IDL Support

- ▶ The CORBA IDL Transform from 10.2 is now back again
 - It was temporarily removed as part of a refactoring of the transformation framework
 - Install it as a separate component “RTist CORBA Support”
- ▶ New documentation is now available for this feature
 - **RTist User’s Guide – Articles – CORBA IDL Support**



Rose RT Importer

- ▶ The Rose RT Importer now supports interface realizations
 - Created for Rose RT classes that inherit from an «interface» class
- ▶ The Rose RT Import now handles variables better
 - `$name` is now imported as `$(TCONFIG_NAME)` (name of current TC) instead of being expanded
 - The default output directory `$/@/$name` in Rose RT is now imported to the default output directory in RTist (previously `$name` was expanded in that case)
- ▶ The make file generator was updated accordingly
 - Better support for `$(TCONFIG_NAME)`
 - Definition of `CONNEXIS_HOME` to support migrated models that use Connexis

External Port Data

- ▶ The TargetRTS implementation of external ports has been extended to make it easier for code running in an external thread to pass data to an RTist application thread.

- Data can now be passed when raising an external event

```
// This function may be used only on threads other than the one on
// which the owner capsule executes. Data can optionally be included in the request.
// After calling this the owner capsule is automatically disabled from receiving another 'raise'
// request until 'enable' is called again.
int raise(const void * data = 0, const RTObject_class * type = 0);
```

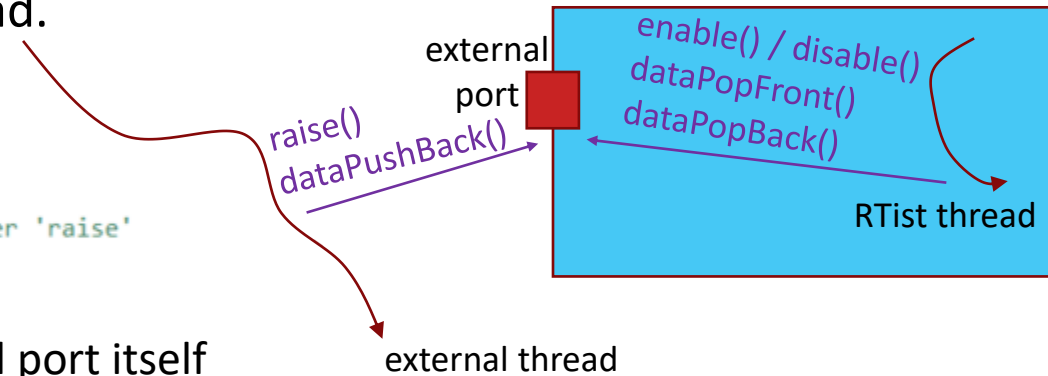
- A thread-safe shared data area (a list) also exists on the external port itself

```
// Put a data object at the end of the externalData list. The data object should be non-null and dynamically allocated by
// code in the external thread, which calls this function.
void dataPushBack(void*);

// Pops the front data object off the externalData list. Returns the number of remaining data objects in the list
// (the data will be null in case of error, for example when attempting to pop from an empty list).
// This function is usually called by the owner capsule thread, and it should delete the popped data object when done with it.
unsigned int dataPopFront(void**);

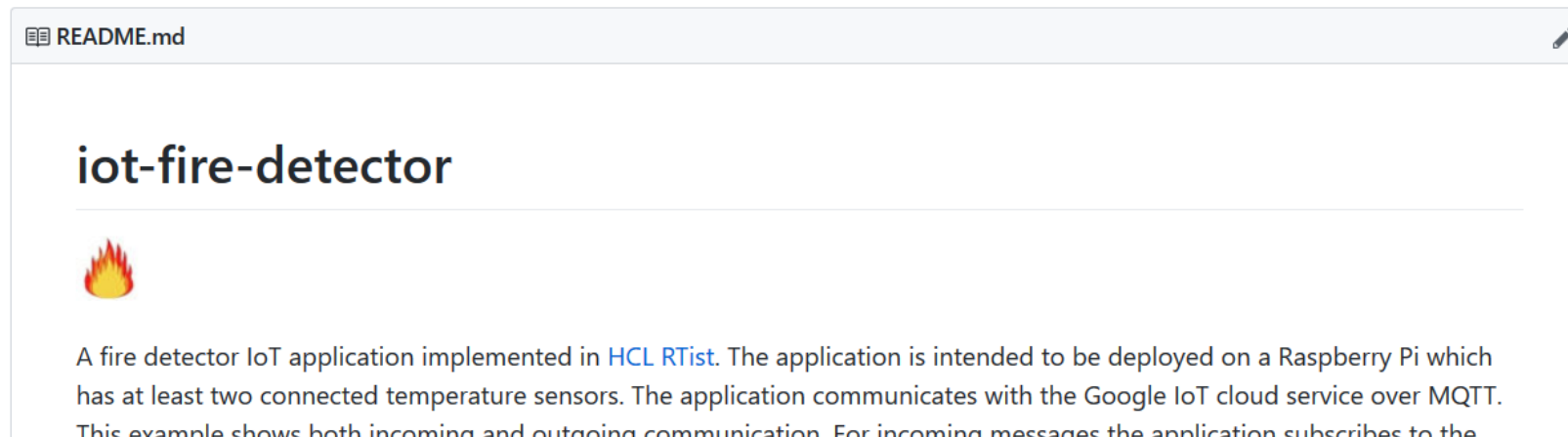
// Pops the back data object off the externalData list. Returns the number of remaining data objects in the list
// (the data will be null in case of error, for example when attempting to pop from an empty list).
// This function is usually called by the owner capsule thread, and it should delete the popped data object when done with it.
unsigned int dataPopBack(void**);

// Attempts to find a specific data object in the externalData list. If it is found it will be deleted.
void dataDelete(void*);
```



GitHub Repositories

- ▶ A number of RTist sample applications have been created on GitHub
 - Focus is to show various ways how to send messages to or from a generated C++ application
- ▶ Also used for providing libraries of reusable functionality
- ▶ Repositories are located in the <https://github.com/hcl-pnp-rtist> organization and are MIT licensed
- ▶ Pull requests are welcome



Connexis

- ▶ The Connexis library ("DCS"), for building distributed real-time applications, is now supported by the model compiler
- ▶ Use of the Connexis profile is now optional (but can make it easier to use the Connexis library)
 - Edit Connexis stereotype properties to automatically add or remove usage of Connexis library elements
- ▶ The Connexis documentation has been reworked
- ▶ A new Connexis sample is included in the installation

<Capsule> «Connexis Feature» EnglishServer

Keywords:

Applied Stereotypes:

Stereotype	Profile	Required	Marking Model
Connexis Feature	Connexis Profile	False	HelloWorldOverflowToBackupService
GeneralCapsuleProperties	CapPropertySet	False	HelloWorldOverflowToBackupService

Apply Stereotypes... Unapply Stereotypes

Stereotype Properties:

Property	Value
▼ Connexis Capsule	
CDM Transport	False
CRM Transport	False
Locator Functionality	False
RTD InitStatus Port	True
RTD Metrics Port	False
Target Agent Functionality	False

New Example

Select a wizard

This HelloWorld model implements a simple distributed model using Connexis. The model demonstrates the use of the Connexis locator service and one simple way of providing overflow handling by directing all overflow traffic to a backup server in a distributed environment using the built in Connexis Locator Ranking service.

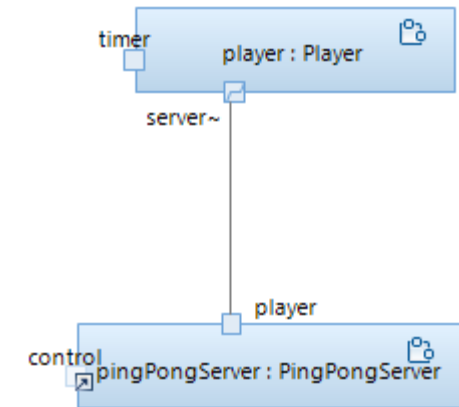
Wizards:

type filter text

- > RMP (RTist Modeling Platform) Plug-ins
- ▼ UML Capsule Development
 - Connexis_HelloWorldOverflowToBackupService

JSON API for External Communication with RTist Applications

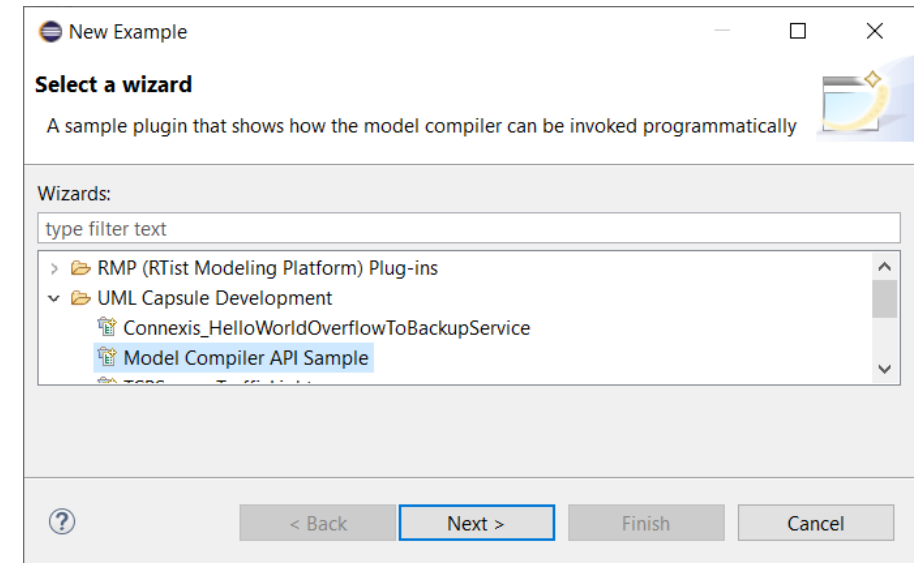
- ▶ [lib-tcp-server](#) is a library that allows an RTist application to communicate with other applications over TCP
 - Provides a JSON-based API for sending and invoking events on certain ports
 - Outgoing events on those ports can be routed to an external application
 - The external application can also reply on an outgoing invoke
 - Available on GitHub as an open-source library, and also in the RTist installation
- ▶ The API can be used for developing distributed applications
 - A light-weight alternative to using Connexis
 - [pingpong-distributed](#) is a sample distributed application built this way
- ▶ The API can also be used as a way to test applications
 - Black-box testing of the system interface, or white-box testing where some parts of the application are mocked out



```
{ "command": "sendEvent", "port" : "trafficLight", "event" : "test_int", "data" : "int 15" }
```

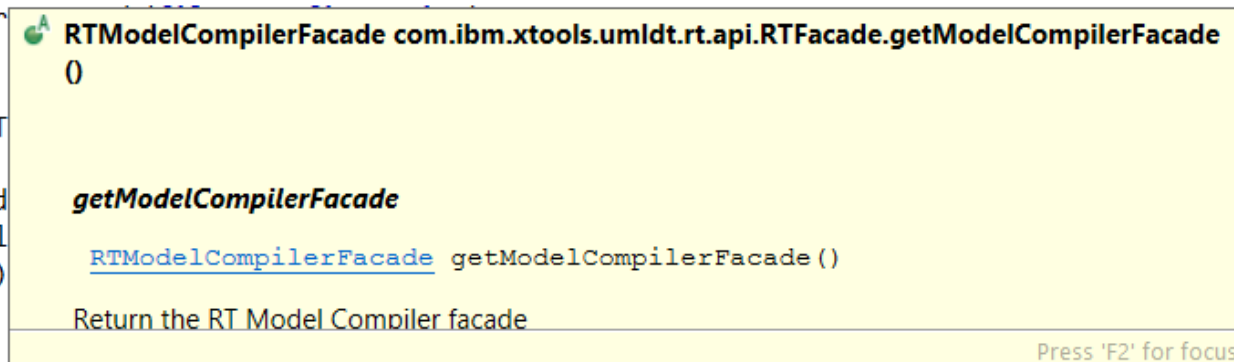
Model Compiler API

- ▶ The RTist "Tool API" was extended to support programmatic invocation of the model compiler
 - Possible to get notified when the build is finished, and access build messages
 - An example Eclipse plugin is available
- ▶ JavaDoc is available for the new API
 - All documentation related to Java APIs is now located in a new chapter "RTist Java APIs" in the Help



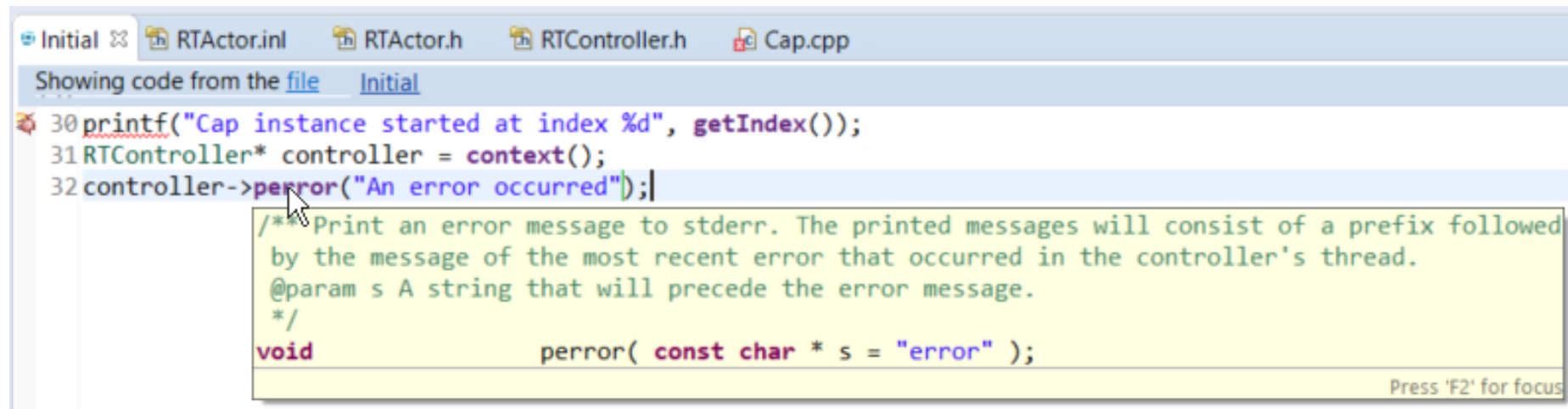
```
IBuilder builder = rtFacade.getModelCompilerFacade().getAsyncBuilder();
IBuildTask task = builder.createBuildTask();
final CountdownLatch wait = new CountdownLatch(1);

task.addListener(new IBuildTaskListener() {
    @Override
    public void stateChanged(IBuildTask task, IBuildState newState) {
        if (newState.isFinal()) {
            wait.countDown();
        }
    }
});
```



TargetRTS Documentation

- ▶ The most commonly used classes and functions in the TargetRTS are now documented using Doxygen comments
- ▶ Generated Doxygen documentation is included in the Help
- ▶ This documentation is also useful when editing or viewing code in the Code editor



The screenshot shows a code editor with several tabs: Initial, RTActor.inl, RTActor.h, RTController.h, and Cap.cpp. The main window displays code from the 'Initial' file. The code includes a printf statement, a controller initialization, and a call to perror. A tooltip is visible over the perror function call, providing documentation for the function. The tooltip text is as follows:

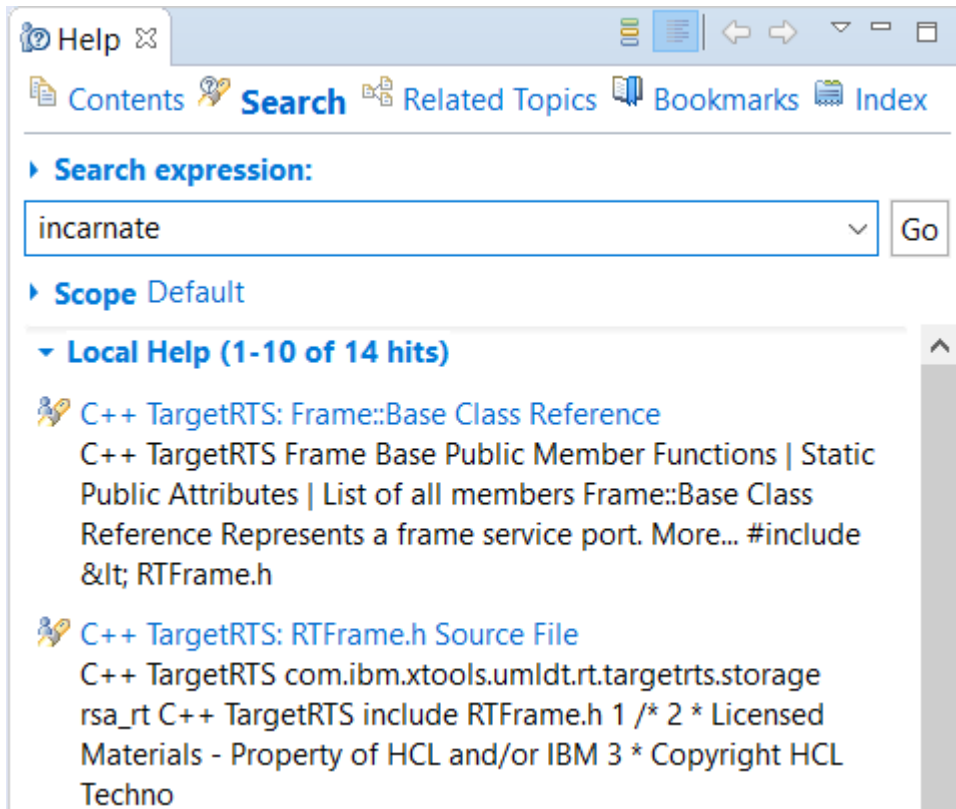
```
/* Print an error message to stderr. The printed messages will consist of a prefix followed by the message of the most recent error that occurred in the controller's thread.
@param s A string that will precede the error message.
*/
void perror( const char * s = "error" );
```

Press 'F2' for focus

- Note that the Code view does not support showing these tooltips

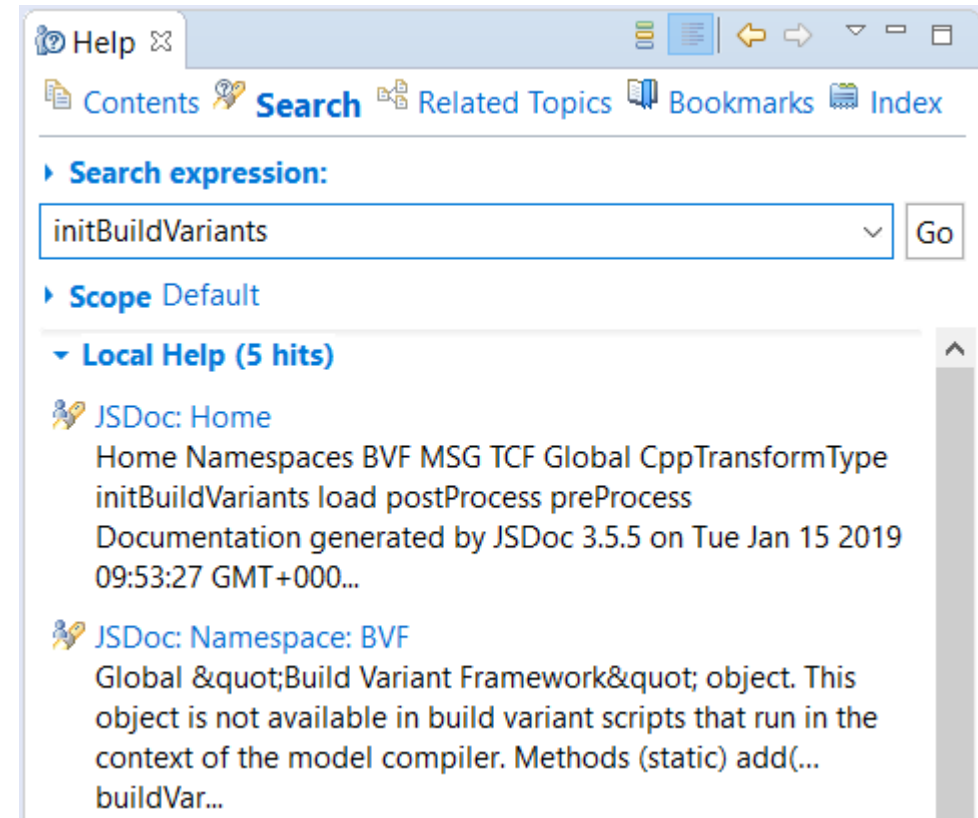
Searchable API Documentation

- ▶ Both the C++ TargetRTS APIs and JavaScript APIs are now searchable from the Help



The screenshot shows the Help search interface. The search expression is "incarnate". The results are categorized under "Local Help (1-10 of 14 hits)".

- C++ TargetRTS: Frame::Base Class Reference**
C++ TargetRTS Frame Base Public Member Functions | Static Public Attributes | List of all members Frame::Base Class Reference Represents a frame service port. More... #include < RTFrame.h
- C++ TargetRTS: RTFrame.h Source File**
C++ TargetRTS com.ibm.xtools.umldt.rt.targetrts.storage rsa_rt C++ TargetRTS include RTFrame.h 1 /* 2 * Licensed Materials - Property of HCL and/or IBM 3 * Copyright HCL Techno

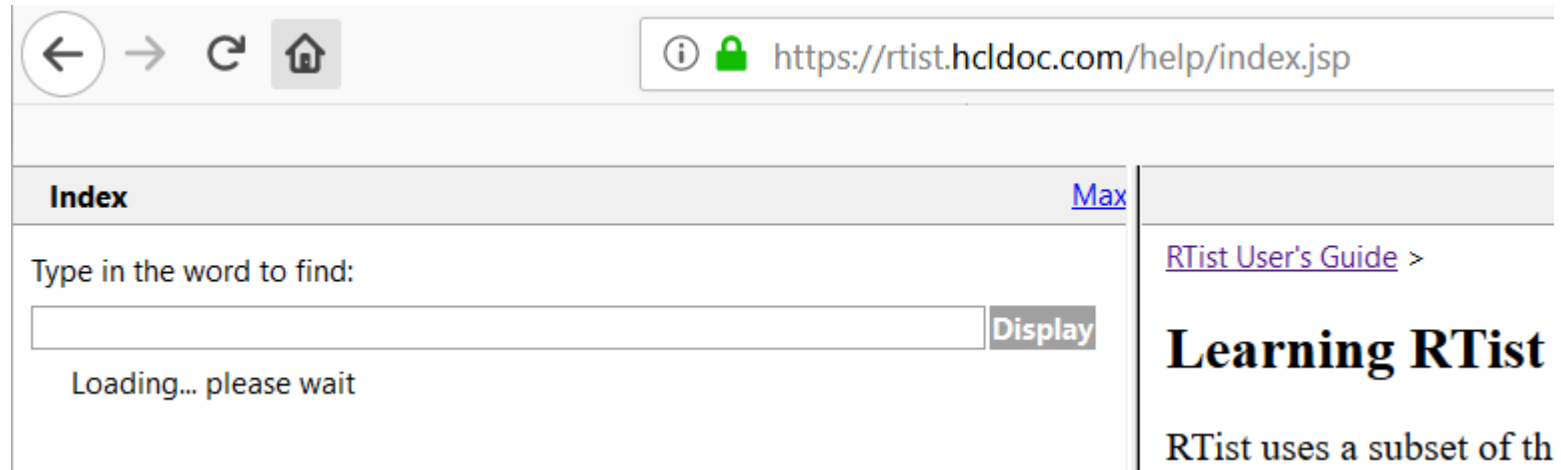


The screenshot shows the Help search interface. The search expression is "initBuildVariants". The results are categorized under "Local Help (5 hits)".

- JSDoc: Home**
Home Namespaces BVF MSG TCF Global CppTransformType initBuildVariants load postProcess preProcess Documentation generated by JSDoc 3.5.5 on Tue Jan 15 2019 09:53:27 GMT+000...
- JSDoc: Namespace: BVF**
Global "Build Variant Framework" object. This object is not available in build variant scripts that run in the context of the model compiler. Methods (static) add(... buildVar...

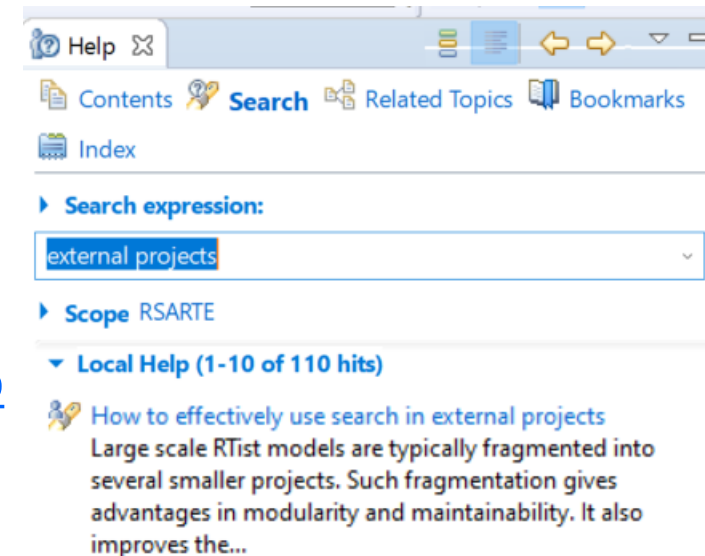
Online Documentation

- ▶ The RTist Eclipse Help documentation is now also available on: <https://rtist.hcldoc.com/help/index.jsp>
- ▶ This documentation is now tagged to enable search engines to index the contents



IBM developerWorks Documentation

- ▶ The IBM developerWorks Connections platform will be sunset on March 31
- ▶ All contents from there have been migrated to the RTist Eclipse Help
 - The new location is in the User's Guide
- ▶ Some of the contents is also included inside the RTist IDE
 - For example, all Newsletters are now there
 - As a benefit, information in the newsletters can now be found by **Help – Search**
- ▶ Web documentation is published to <https://rtist.hcldoc.com/help/index.jsp> as soon as its changed
 - Not necessary to wait until a new release is made
- ▶ All references to developerWorks within RTist have now been replaced for the new location



HCL

*Relationship*TM
BEYOND THE CONTRACT

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



WATCH THE FILM