Searching in DevOps Model RealTime Models

Author: Mattias Mohlin
Senior Software Architect
IBM

This document describes how to use the various search commands that are available in DevOps Model RealTime. The document was last updated for Model RealTime 12. All screen shots were captured on the Windows platform.

Introduction

An important capability of a modeling tool is the ability to let users quickly find elements of interest in the model. Model RealTime provides several commands for performing searches in a model, and depending on what you are looking for some of those commands may be more appropriate than others. There are often many ways to find the elements you are looking for, but some ways may be faster than others or provide more precise search results. It is therefore important to understand which of the search commands that is best to use for a particular situation.

This document describes the search commands that are provided by Model RealTime and gives some guidelines for when to use each of them.

Different Search Strategies

On a conceptual level there are three main strategies for searching a model:

- Index-based searching
- Memory-based searching
- View-based searching

Index-based search commands make use of a search index. This index contains various information about the elements of the models in the workspace, such as their names, types and locations. The index is automatically created and updated, and is by default stored in your workspace. There are some benefits with using a search index:

- Searching a big workspace can be done in a short time, provided that the index is always kept up-to-date.
- It is not required to load all models of the workspace into memory, since the index is separately stored as files (by default located in the workspace metadata).
- It is possible to put additional derived (i.e. computed) information about elements into the index, which then can be utilized when searching.
- It is possible to populate the index with information from models not contained in the workspace, for example external projects that may be imported into the workspace later.

However, index-based searching also has its limitations:

- The criteria you can use when searching are restricted to what information that is present in the index. Putting additional information in the index enables more kinds of search criteria, but also makes the index bigger and slower to create, update and use.
- Updating the index can take some time for a large workspace, in particular the first time when it has to be built from scratch. However, by automatically doing index updates as a background activity it becomes less noticeable for users.

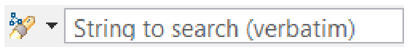
Memory-based search commands operate directly on the model that is loaded into memory. This allows them to use custom search criteria in order to find elements based on more advanced semantic rules than what is possible with index-based searching. The memory-based search commands require a relevant part of the model to be loaded and they do not consider any closed models.

Finally there are view-based search commands, which work by searching a particular view of the model, for example what is displayed in a diagram at a particular point in time. What is found by a view-based search command therefore depends not only on what is in the model, but also on preferences such as filtering preferences and other display options.

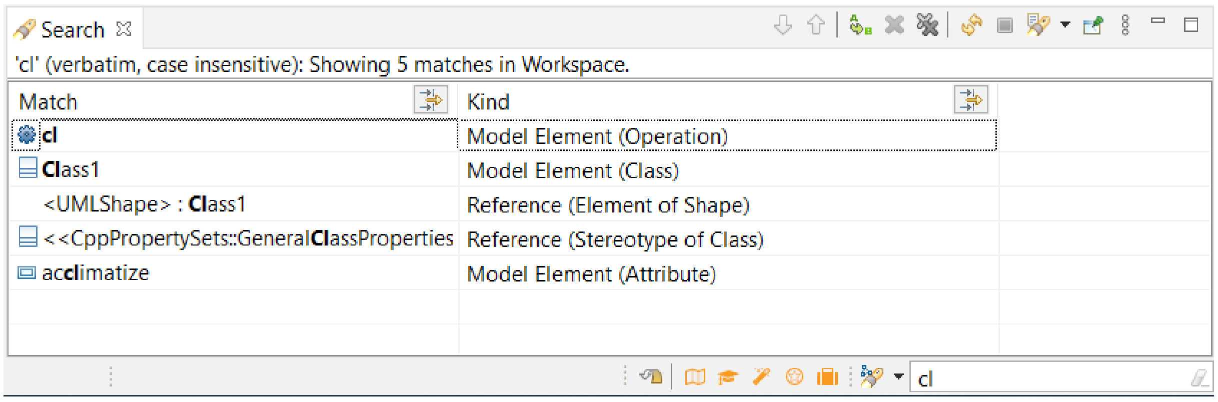
Most search commands in Model RealTime are index-based, but a few are memory-based (see Memory-based Search Commands) and one is view-based (see Incremental Textual Search in Diagrams).

Search Field

The easiest and quickest way to perform a search in Model RealTime is to use the search field that by default is located in the bottom right corner of the main window.


String to search (verbatim)

At any time you can just type a few characters in this field and press Enter to find matching elements in the model. Here is an example:



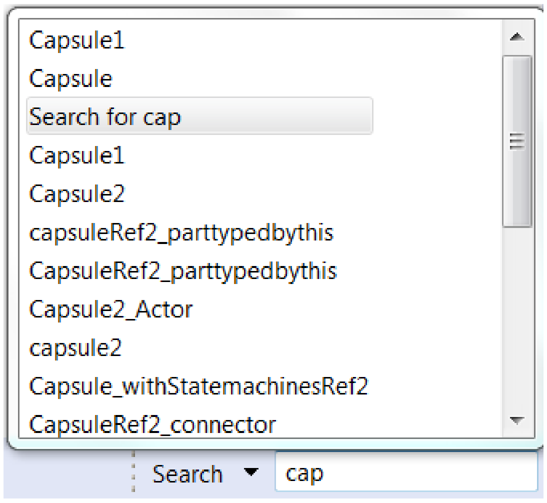
'cl' (verbatim, case insensitive): Showing 5 matches in Workspace.

Match	Kind
cl	Model Element (Operation)
Class1	Model Element (Class)
<UMLShape> : **Cl**ass1	Reference (Element of Shape)
<<CppPropertySets::General**Cl**assProperties	Reference (Stereotype of Class)
ac**cl**imatize	Model Element (Attribute)

Elements that match the search string are listed in the Search view, and the matching text is marked with boldface in the "Match" column.

Search String Proposals Popup

As you type characters in the search field, a pop-up will appear, suggesting names to search for.


Capsule1
Capsule
Search for cap
Capsule1
Capsule2
capsuleRef2_parttypedbythis
CapsuleRef2_parttypedbythis
Capsule2_Actor
capsule2
Capsule_withStatemachinesRef2
CapsuleRef2_connector

These names come from two sources:

- **Previously used search strings**
  These are the names that appear above the "Search for..." line. The order of the names is determined from when you most recently used them when searching. The most re-

cently used search string appears closest to the “Search for...” line. Therefore you can simply press the Arrow-Up key once if you want to repeat the latest search again.

- **Strings that are present in your model**
  The search index is used for getting these strings, and their order is determined by how frequently they occur in the model. The most common string is just below the “Search for...” line so you can get to it easily by pressing the Arrow-Down key once.

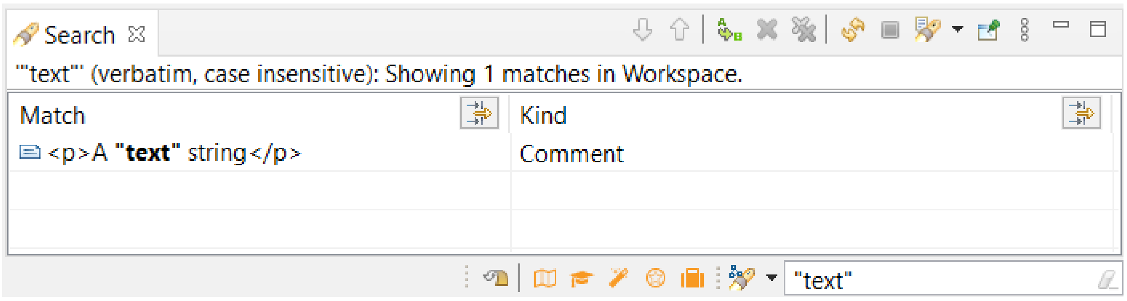
Note that only names that match what you already typed in the search field are displayed in the pop-up. If you didn’t type anything, and just want to reuse a previously used search string, just click in the search field and press Arrow-Up. Select the search string you want to use and press Enter to start the search. If the string you are looking for is not present in the pop-up you can close it by pressing Esc.

Verbatim versus Non-Verbatim Searching

By default the search string is interpreted verbatimly. This means that Model RealTime will try to find as exact matches as possible, where the searched for string occurs exactly as typed. In most cases this gives the search results you would expect. However, in some cases it can be useful to search in a non-verbatim way. For example, you may want to find places in the model where two or more words occur "close" to each other. By "close" we could mean different things; immediately adjacent to each other, on the same code line, in properties belonging to the same element etc. Non-verbatim searching can be enabled by turning off the preference *RealTime Development - Search - Use verbatim search*. Each word in the search string will then be searched for separately, and matches will be reported where the words occur "close" to each other. The closer the words occur, the better the search match is, and the higher it will be placed in the search result. The idea is that the most relevant search results should always appear on the top of the search result.

If you decide to turn off the *Use verbatim search* preference you can still search verbatimly by enclosing the search string in double quotes in the search field. With verbatim search enabled double quotes are interpreted literally. Here are a few examples:

Preference is set to verbatim search. The double quotes are included in the matched string.

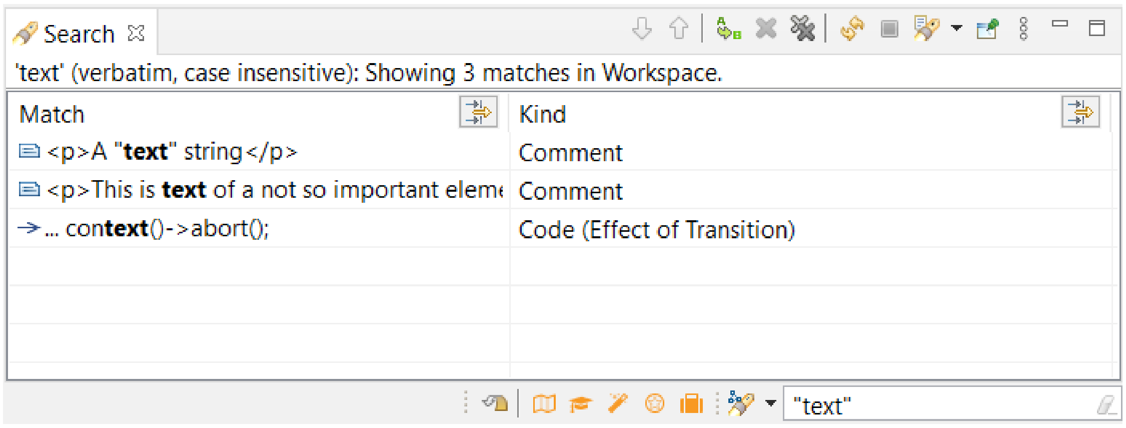


'"text"' (verbatim, case insensitive): Showing 1 matches in Workspace.

Match	Kind
<p>A **"text"** string</p>	Comment

"text"

Preference is set to non-verbatim search. The double quotes mean that the search will still be verbatim and they are not included in the matched string.

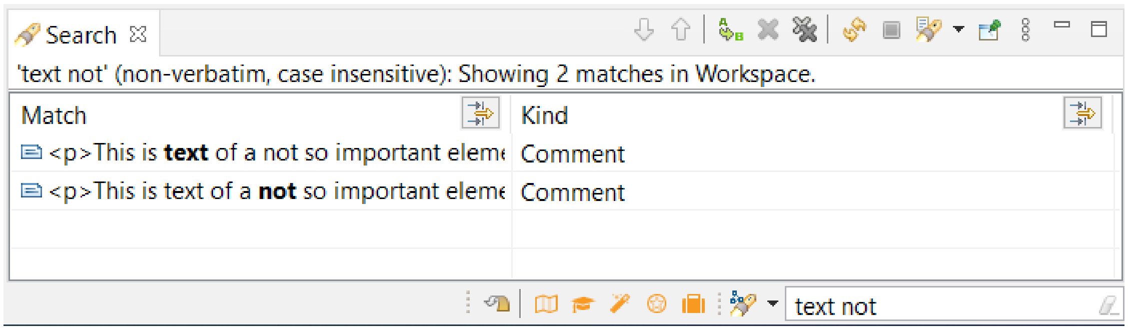

Search

'text' (verbatim, case insensitive): Showing 3 matches in Workspace.

Match	Kind
<p>A "**text**" string</p>	Comment
<p>This is **text** of a not so important elem	Comment
... con**text**()->abort();	Code (Effect of Transition)

"text"

Preference is set to non-verbatim search. Two words are searched for separately, without use of double quotes:


Search

'text not' (non-verbatim, case insensitive): Showing 2 matches in Workspace.

Match	Kind
<p>This is **text** of a not so important eleme	Comment
<p>This is text of a **not** so important eleme	Comment

text not

The Search view shows on the header line whether the search was performed verbatimly or not.

Note that even with verbatim search, the search string is not always matched exactly as written. Use of wildcards and case insensitive searching are two reasons for that (see Search Patterns).

Search Patterns

Sometimes you may not remember the exact word (or words) you are looking for. In those cases it is useful to include wildcards in the search string. Wildcards cause the search string to match a pattern rather than an exact string. This is true both for verbatim and non-verbatim search.

The most common pattern is `*X*` which matches all strings that contain the substring X. Actually this is so common that * wildcards are always added implicitly before and after the search string you type. This explains the example mentioned previously where a search for `cl` also finds matches where "cl" occurs inside the text (for example `acclimatize` or `Class1`).

The downside of implicitly adding these wildcards before and after the typed search string is that if you happen to know the exact spelling of the word you search for, you will in most cases get additional matches in the search result where the search string appears as a substring. However, the search matches will be ordered in the Search view so that the “best” matches appear on top. And a match where the search string matches exactly is considered a better match than one where it only partly matches the search string.

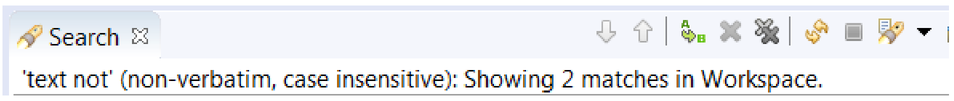
The * wildcard matches zero to many arbitrary characters in the name. You can place the * wildcard at any place in the search string to create different search patterns. And you can of course use more than one * wildcard. Note that even if you add your own wildcards the implicit * before and after the search string will still be added. However, the part of the string that matches those implicit wildcards will not be marked in boldface. Here are some examples (for clarity underline is used instead of boldface to show which part of the string that matches the search string):

`*Data`	Matches all strings that contain zero to many arbitrary characters followed by `Data`, for example `TestData`, `ProxyData` or just `Data`. Due to the implicit trailing * wildcard it also matches `DataStore` and `MyDataStore`.
`Test*Capsule`	Matches all strings that contain `Test` followed by some arbitrary characters and then followed by `Capsule`, for example `TestModelCapsule` or `TestCapsule`. Due to the implicit * wildcards it also matches `MyTest-Capsule` and `TestForCapsules`.
`T*M*C`	Matches all strings that contain the letters `T`, `M` and `C` with zero or many letters in between, for example `TestModelCapsule`.

Another wildcard you can use is `?` which matches exactly one arbitrary character. It is much less commonly used than `*`, but when your model elements follow certain naming conventions it may be handy. Some examples:

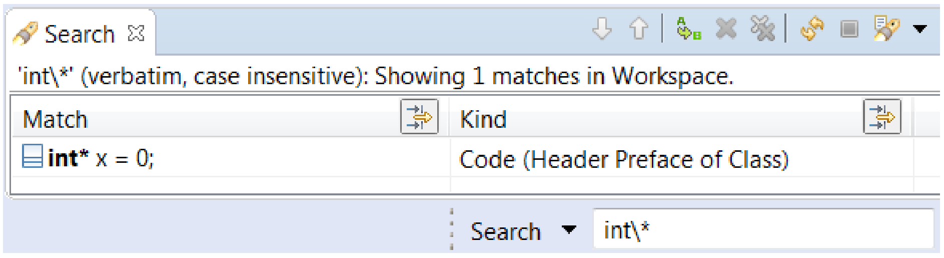
`Test?MyClass`	Matches strings that contain `Test` and `MyClass` and have exactly one character in between, for example `Test1MyClass` or `Test_MyClass`.
`???MyClass`	Matches all strings that contain `MyClass` and that have 3 letters before it, for example `XYZMyClass` or `abcMyClass`. Due to the implicit * wildcards it will also match strings with more than 3 letters before "MyClass", for example `the123MyClasses`.

Another form of search pattern comes from the interpretation of the search string as either case-sensitive or case-insensitive. If you only use lower-case letters in the search string, the search will be case-insensitive, while if it contains at least one upper-case letter, it will be case-sensitive. This information is printed at the top of the Search view when you have performed the search, so you can know how the search string was interpreted. For example:



'text not' (non-verbatim, case insensitive): Showing 2 matches in Workspace.

If you want a * or ? in your search string to be interpreted literally, and not as a wildcard, you must escape it using a backslash. For example:

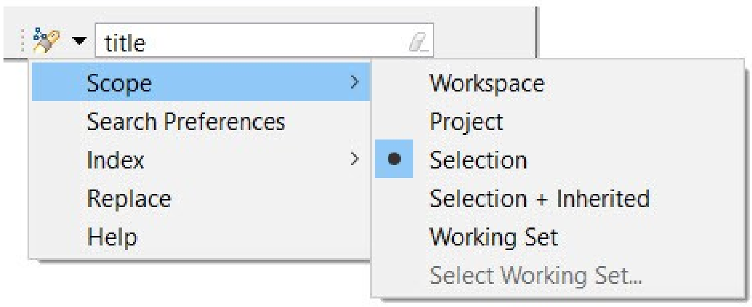

'int*' (verbatim, case insensitive): Showing 1 matches in Workspace.

Match	Kind
int* x = 0;	Code (Header Preface of Class)

Note that the backslash character is only interpreted as an escape character if it is immediately followed by * or ?. This means that if you for example want to search for a Windows-style path in a TC file, you can just type the path name as expected, for example "C:\temp\folder". If you want to avoid that a backslash acts as an escape character in front of * or ? you can escape it. For example, to search for the string '*' literally, you need to type \\\* in the search field.

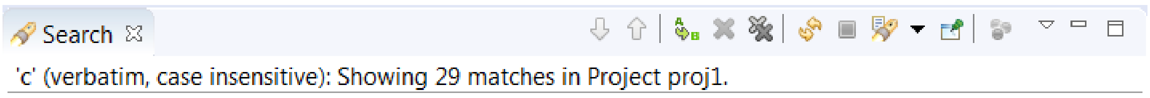
Scope

By default the search takes place in all projects in the workspace (closed projects are excluded). You can change the scope of the search by using the menu that appears when clicking the black arrow to the left of the Search field.



The scopes "Project" and "Selection" require an element to be selected in the Project Explorer. Set the scope to "Project" to only search in the project to which the selected element belongs, and set the scope to "Selection" to only search within the selected element. The scope "Selection + Inherited" works the same as "Selection" but will in addition search in elements from which the selected element inherits (see Searching in Inheritance Hierarchy).

The search scope that is used in the search is printed at the top of the Search view:



'c' (verbatim, case insensitive): Showing 29 matches in Project proj1.

If the scope is set to "Project" or "Selection" and there is no element selected, then the search will be performed in the workspace scope.

The scope "Workspace and External Projects" is only available if you have configured Model RealTime to support external projects. It allows search to find matches in external projects that are not currently present in the workspace. See Index management and external projects for more information about how to enable the support for external projects and how to specify where to find them.

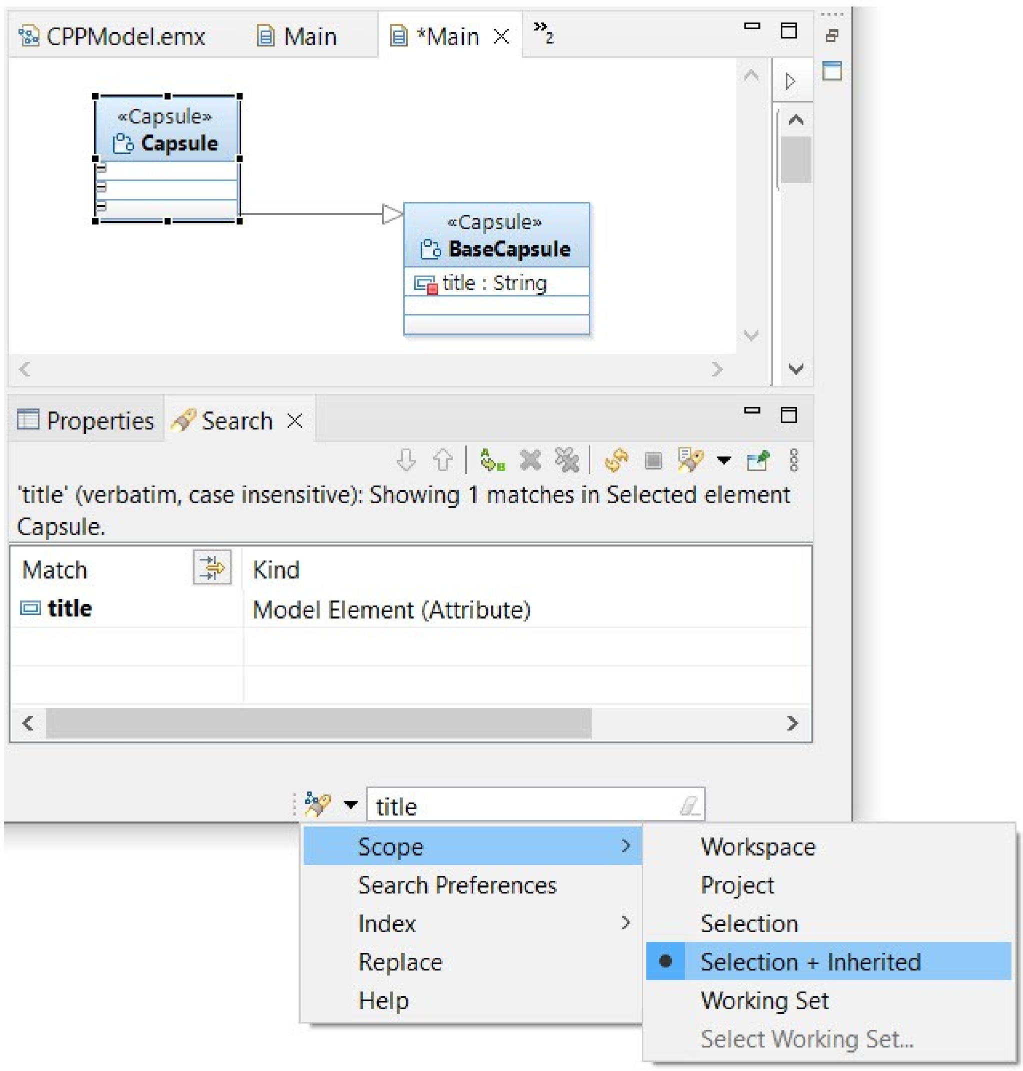
Select the scope “Working Set” if you want to search in several projects contained in a working set. Then use the command “Select Working Set” to specify the working set.

Searching in Inheritance Hierarchy

Model RealTime lets you set the search scope to “Selection + Inherited” to perform quick searches for selected inherited elements, such as attributes of capsules.

Consider a scenario where you have a capsule that inherits from another base capsule. Let’s say, you want to search for an attribute named ‘title’ in the inherited capsule. For this, you can simply select the child capsule, set "Selection + Inherited" as the scope, and type the attribute name ‘title’ in the search field.

As you can see from the image below, Model RealTime searches and matches the ‘title’ attribute in the selected and inherited capsules and displays the result in the Search view.



'title' (verbatim, case insensitive): Showing 1 matches in Selected element Capsule.

Match	Kind
title	Model Element (Attribute)

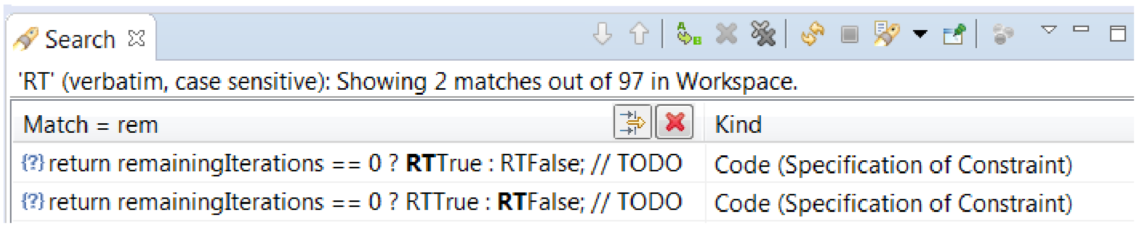
title

Alternatively, you can also use the Search dialog to search selected inherited elements (see Find/Replace).

When searching an inheritance hierarchy the search result will be sorted so that elements locally defined in the selected element will come before inherited elements. Also, matches in direct base elements come before matches in indirect base elements. This means that if you for example search for a redefined operation in a class the found operations will be reported according to where they are present in the inheritance hierarchy. If an operation calls the operation it redefines you can therefore easily find that redefined operation as the next operation in the search result.

Filtering

If a search results in too many items in the Search view you can either repeat the search with a more specific search string, or you can filter the search result to only show some of the matches. Each of the columns in the Search view can be filtered separately. Click the Edit Filter button ( ) in the header of the column you want to filter. Then type a filter string in the text box that appears. For example:

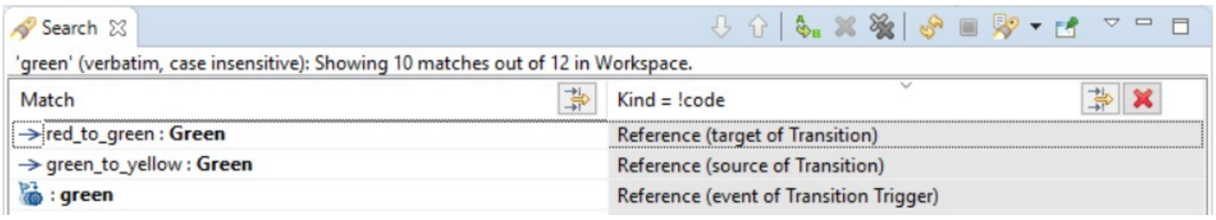


'RT' (verbatim, case sensitive): Showing 2 matches out of 97 in Workspace.

Match = rem	Kind
return remainingIterations == 0 ? **RT**True : RTFalse; // TODO	Code (Specification of Constraint)
return remainingIterations == 0 ? RTTrue : **RT**False; // TODO	Code (Specification of Constraint)

The Search view is filtered by hiding all matches where the text in the filtered column does not match the specified filter text. The filter string is interpreted in the same way as the search string when it comes to case-sensitivity, and you can use the same wildcard characters (see Search Patterns). If you set a filter for multiple columns, they will be combined so that only search matches that "pass through" all the filters will remain visible.

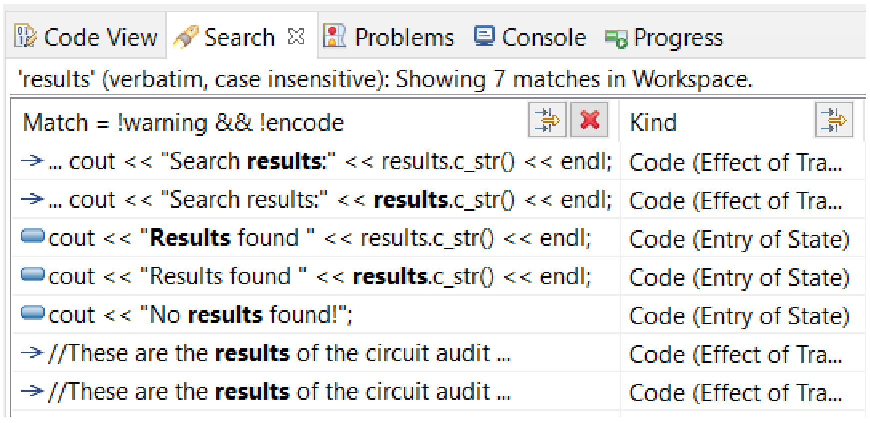
Sometimes it's useful to do the opposite, that is to filter the Search view so that it instead shows all matches where the text in the filtered column does not match the specified filter text, and hides all other matches. You can accomplish this by writing an exclamation mark (i.e. the logical NOT operator) in front of the filter string. This means that the filter is negated. Here is an example of showing all matches that are not code matches:



'green' (verbatim, case insensitive): Showing 10 matches out of 12 in Workspace.

Match	Kind = !code
red_to_green : **Green**	Reference (target of Transition)
green_to_yellow : **Green**	Reference (source of Transition)
: **green**	Reference (event of Transition Trigger)

You can also use && (the logical AND operator) for combining multiple filters for the same column. And it can be combined together with ! if you want to filter out multiple different kinds of matches. Here is an example:



'results' (verbatim, case insensitive): Showing 7 matches in Workspace.

Match = !warning && !encode	Kind
→ ... cout << "Search **results**:" << results.c_str() << endl;	Code (Effect of Tra...
→ ... cout << "Search results:" << **results**.c_str() << endl;	Code (Effect of Tra...
cout << "**Results** found " << results.c_str() << endl;	Code (Entry of State)
cout << "Results found " << **results**.c_str() << endl;	Code (Entry of State)
cout << "No **results** found!";	Code (Entry of State)
→ //These are the **results** of the circuit audit ...	Code (Effect of Tra...
→ //These are the **results** of the circuit audit ...	Code (Effect of Tra...

If you need to use the characters ! or && in the filter string, without interpreting them as logical operators, you can enclose the whole filter string in double quotes. The filter will then be interpreted verbatimly.

Remove an applied filter by pressing the Clean Current Filter button (✖), or simply delete all text from the filter box.

You can also apply a filter by simply starting to type in the Search view. Use the Tab / Shift+Tab keys to move the filter box between columns.

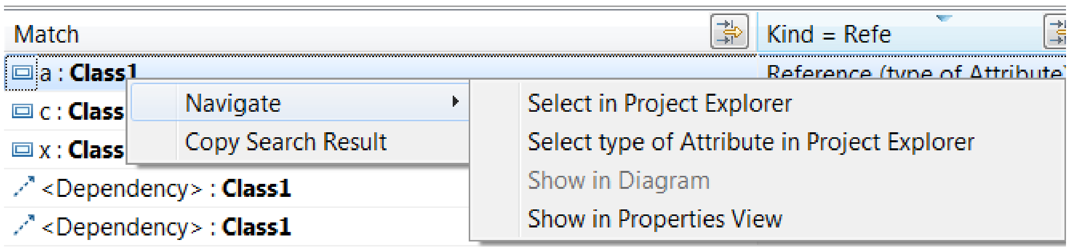
Any column in the Search view can be filtered, but often it's most useful to filter on the Kind column. Such filtering allows you to only show certain kinds of matches, such as code matches, or reference matches.

Navigation

Once you have found the element you were looking for you can navigate to it by double-clicking the match in the Search view. This performs a default navigation which will take you to the most appropriate view or editor in Model RealTime depending on the kind of match. For example, double-clicking a code match will open the Code editor, while double-clicking a match in a transformation configuration will open the TC editor. In most cases the search string will be highlighted in the opened view or editor.

Sometimes you may want to show the matching element in another place. Or you may want to navigate to another element that is related to the matching element. In this case you can right-click on the match in the Search view and find all available navigation commands in the "Navigate" sub context menu.

For example, assume you have found a reference to an attribute "a" that is typed by a class "Class1". The default navigation command, which will run if you double-click on this match, will select the attribute in the Project Explorer. However, if you instead would like to show it in the Properties view, you can right-click on the match and perform *Navigate - Show in Properties View:*



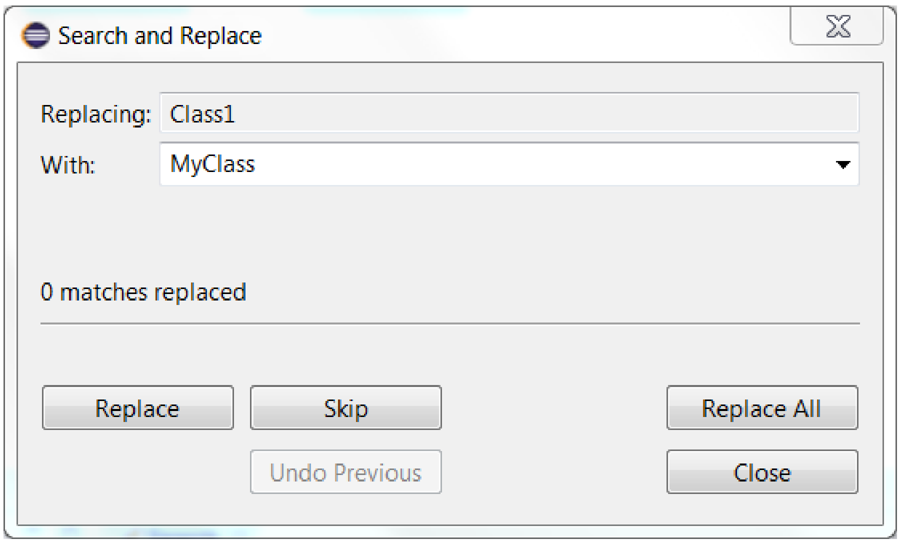
You also find a command there for navigating to the type of the attribute (i.e. Class1).

Sometimes it's not necessary to perform an explicit navigation from a match. The Code view and the Properties view are updated to show code and properties for a match that is selected in the Search view. This is convenient if you have many matches to go through, and quickly want to look at code or properties for each of them. Note that showing code for a selected match requires its model element to be loaded. There is a preference *RealTime Development - Search - Load model when try to display match in the Code view* which controls if the model should be automatically loaded when a match is selected in the Search view. Having this preference enabled is convenient but comes with a small performance cost when selecting matches in the Code view (since loading the required model elements takes some time).

When Search finds something in an external project, that project must first be imported into the workspace before navigation can take place. This happens automatically when you double-click such a match in the Search view. There is a preference *RealTime Development – Load models on external projects import* which can be set to enforce automatic loading of models of imported external projects. It is recommended to have this preference enabled if you search in external projects, as it will make navigation commands work from the Search view almost as if the project was already present in the workspace. Read more about searching in external projects here.

Replace

Sometimes the purpose of searching for a string is to replace it with another string. You can accomplish this by pressing the Replace button ( ) after the search has completed, or you can use the command Replace in the Search button menu. This will open the "Search and Replace" dialog which allows you to go through all search result items and replace the search string with another string.



As you press the Replace or Skip buttons the item that is selected in the Search view will be replaced or skipped. Press “Undo Previous” to undo the most recent replacement and go back to the previously selected item in the Search view. If you know that you want to replace all items you may press “Replace All”.

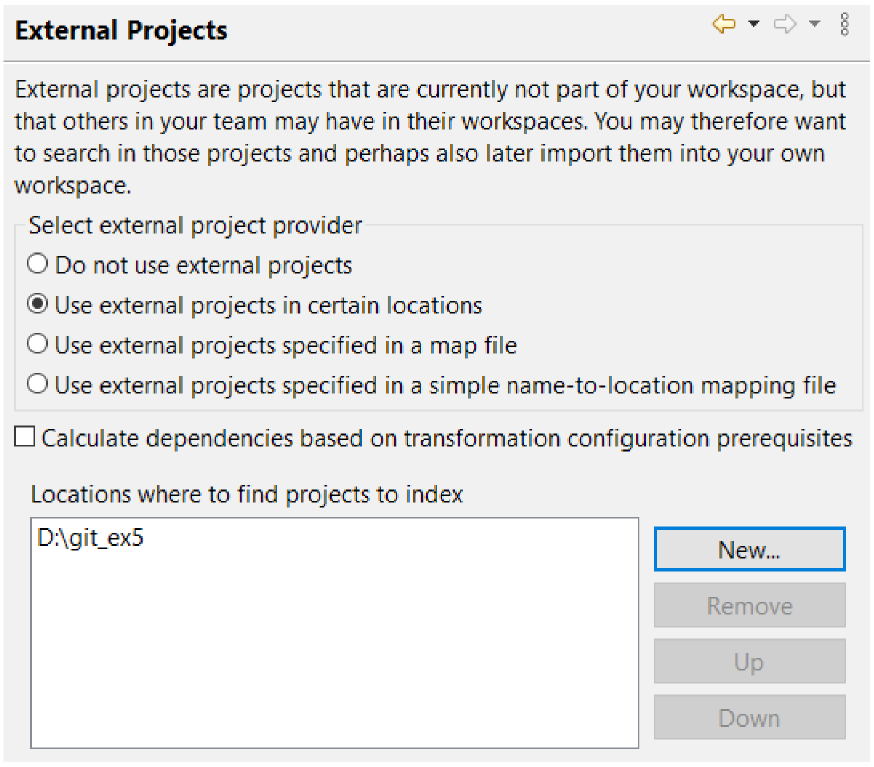
By default all items in the Search view will be iterated by the “Search and Replace” dialog. If you only want to replace some of the found matches you can select them and then press the Replace button.

Note that some kinds of matches cannot be replaced. For example, a reference match cannot be replaced as this could cause the model to become inconsistent. If a match cannot be replaced you will see the message “This match is not a subject for replace” in the “Search and Replace” dialog. Also, note that Replace is not meaningful for non-verbatim searching of multiple words.

Index management and external projects

The search index is automatically created and updated in the background as you work in Model RealTime. The index consists of one folder per project and is by default placed in your workspace metadata (under `com.ibm.xtools.umldt.rt.indexer`). However, the preferences under *RealTime Development - Search - Index - Index location* allow you to place it somewhere else. If you set the preference *Keep index in project* the index folder will be placed next to the project file (the index folder is in this case called `.index2`). You can for example use this option if you want to commit the index to an SCM system (could be useful for projects that are used frequently but are modified rarely). You can also choose to place the index in a custom folder.

The search index can also be configured to include contents for projects that are currently not present in the workspace. Such external projects can sometimes be useful to search in, since you may want to import external projects into the workspace if searching finds something interesting in them. You can set the search scope to “Workspace and External Projects” to include external projects in the search. But you also must specify where the external projects are located. This is done in *Preferences – Version Control (Team) – External Projects*:


External Projects

External projects are projects that are currently not part of your workspace, but that others in your team may have in their workspaces. You may therefore want to search in those projects and perhaps also later import them into your own workspace.

Select external project provider

- ○ Do not use external projects
- ● Use external projects in certain locations
- ○ Use external projects specified in a map file
- ○ Use external projects specified in a simple name-to-location mapping file

☐ Calculate dependencies based on transformation configuration prerequisites

Locations where to find projects to index

D:\git_ex5

External projects can be specified either as a list of locations in the file systems (i.e. folders), or a map file can be used for specifying the external projects. The map file is a text file where each external project is specified on a separate line using the syntax

```
name:path[::dependents]
```

where `name` is the name of the project and `path` is its physical location. If the path contains a colon (for example a full path on Windows), you may instead use semicolon as separator. The optional `dependents` is a regular expression which may be used to specify that the external project is a dependent project for other projects mentioned in the map file. This expresses that whenever any of those projects are imported into the workspace, that external project should also be imported.

For flexibility it's possible to use environment variables both within the map file itself as well as in the path specified in the preference. Refer to an environment variable either with the syntax `$(var)` or `${var}`.

Here is an example of a map file:

```
base:$(PRJ_ROOT)/c1/base::model|allModels
base:$(PRJ_ROOT)/c2/base::model1
model:$(PRJ_ROOT)/model
model1:$(PRJ_ROOT)/model1
# You may use comments
allModels:$(PRJ_ROOT)/allModels
```

There is also a simpler form of map file (called "simple name-to-location mapping file" in the preference page). This kind of map file is what also the model compiler supports with its `--`

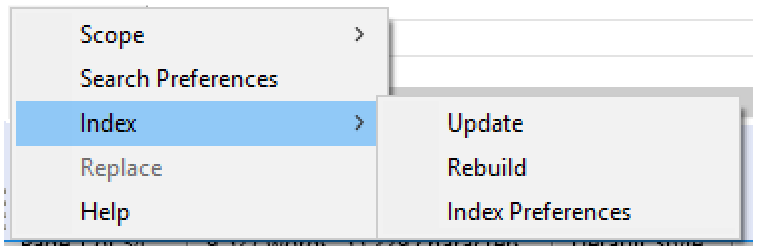
`root` option and it's a text file where each line specifies a project followed by the path to its project folder. The path may either be absolute, or relative to the location of the map file. For example:

```
MyProject=D:\projs\MyProject
MyLocalProj=./MyLocalProj
```

The preference page also provides a checkbox which can be set in order to analyze transformation configurations that are present in the external projects. Prerequisites of these transformation configurations will be used for including the projects containing those prerequisites.

If Search finds a match in an external project, that project has to be imported into the workspace before you can navigate to it. This happens automatically when you perform the navigation, but of course such a navigation will take slightly longer than usual since both the external project, and all its dependent projects, first need to be imported into the workspace.

If you change any of the above mentioned preferences related to the search index you should perform an update of the index for the change to take effect. You can do this from the menu of the Search button:

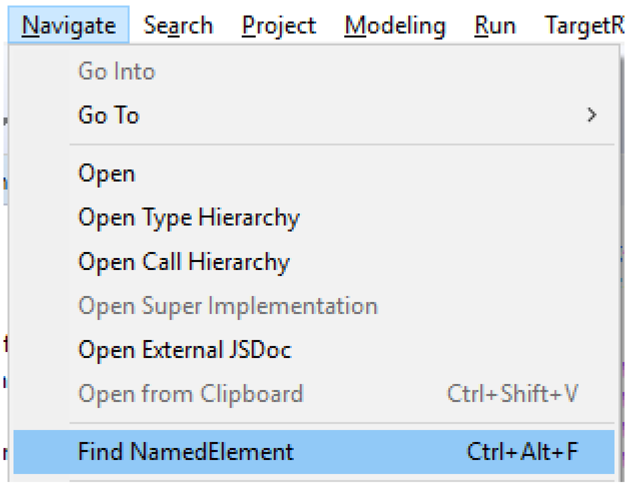


Choose *Index - Update* to update the index, and *Index - Rebuild* to completely rebuild the index from scratch.

Find NamedElement

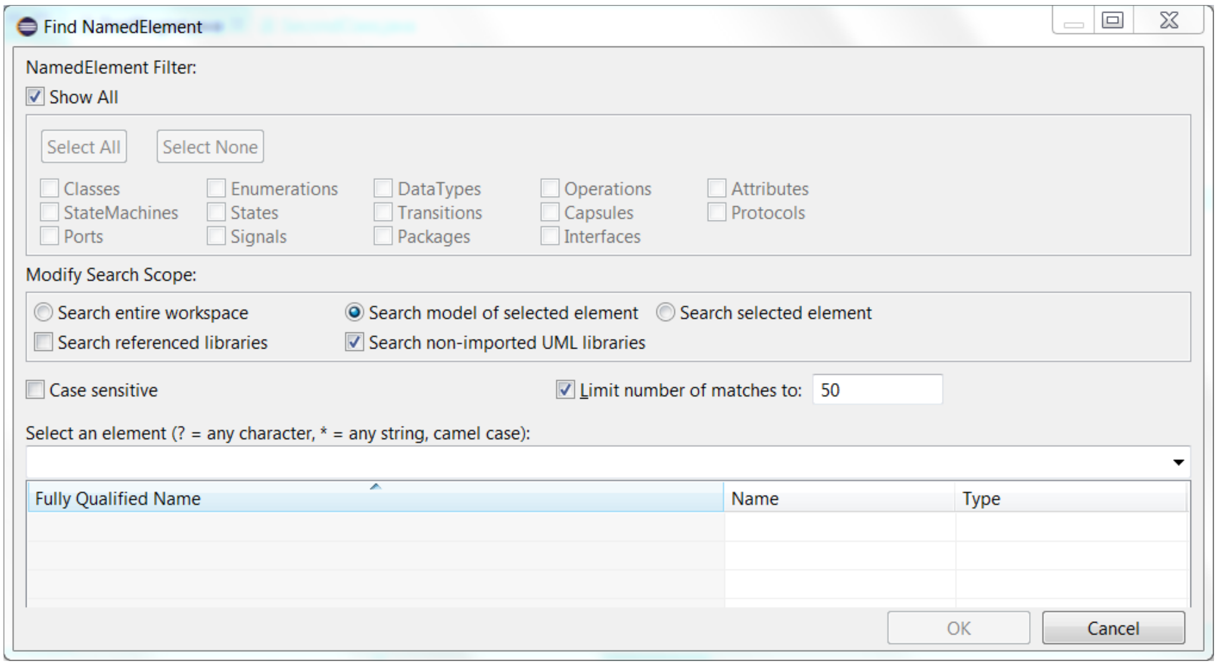
A NamedElement is UML terminology for a model element that may be given a name. Most elements in a UML model may have a name. For some kinds of elements a name is mandatory, while for most elements the name is optional and may be empty.

Model RealTime provides a command called “Find NamedElement” for finding elements with a certain name. It is an index-based search command and is available in the global “Navigate” menu. Its default keybinding is Ctrl+Alt+F:



The main reason for when you may want to use this command, instead of simply typing the name to search for in the search field, is when you want to do the searching incrementally as you type. This can for example be useful if you are not quite sure about the name of the element you are looking for, and want to try searching for different names.

Searching for NamedElements are done from the Find NamedElement dialog:


Find NamedElement

NamedElement Filter:

[✓] Show All

Select All | Select None

☐ Classes	☐ Enumerations	☐ DataTypes	☐ Operations	☐ Attributes
☐ StateMachines	☐ States	☐ Transitions	☐ Capsules	☐ Protocols
☐ Ports	☐ Signals	☐ Packages	☐ Interfaces	

Modify Search Scope:

○ Search entire workspace [● Search model of selected element] ○ Search selected element

☐ Search referenced libraries [✓] Search non-imported UML libraries

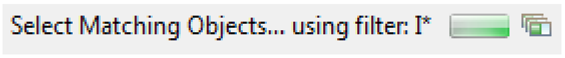
☐ Case sensitive [✓] Limit number of matches to: 50

Select an element (? = any character, * = any string, camel case):

Fully Qualified Name	Name	Type

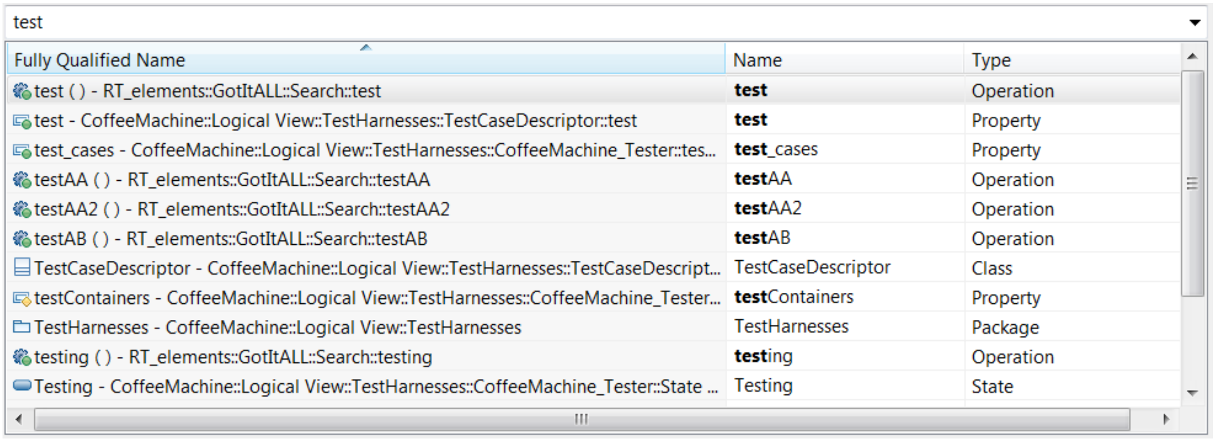
OK | Cancel

For now let's ignore the various settings which are provided by this dialog, and jump directly to the text field in the middle of the dialog. Here you can type the name of the element you are looking for. As soon as you stop typing for a moment, a search will start in the background. A message is shown in the status bar while the search runs:


Select Matching Objects... using filter: I*

When the search finishes all elements in the model which have a name that matches the search string are listed in the table at the bottom of the dialog. Names are not required to match exactly. For example if you search for the string “Test” you will not only find the elements that have the name “Test” but also all elements with names that start with “Test”. This often helps in case you don't remember all of the name to search for.

Here is an example:


test

Fully Qualified Name	Name	Type
test ( ) - RT_elements::GotItALL::Search::test	**test**	Operation
test - CoffeeMachine::Logical View::TestHarnesses::TestCaseDescriptor::test	**test**	Property
test_cases - CoffeeMachine::Logical View::TestHarnesses::CoffeeMachine_Tester::tes...	**test**_cases	Property
testAA ( ) - RT_elements::GotItALL::Search::testAA	**test**AA	Operation
testAA2 ( ) - RT_elements::GotItALL::Search::testAA2	**test**AA2	Operation
testAB ( ) - RT_elements::GotItALL::Search::testAB	**test**AB	Operation
TestCaseDescriptor - CoffeeMachine::Logical View::TestHarnesses::TestCaseDescript...	TestCaseDescriptor	Class
testContainers - CoffeeMachine::Logical View::TestHarnesses::CoffeeMachine_Tester...	**test**Containers	Property
TestHarnesses - CoffeeMachine::Logical View::TestHarnesses	TestHarnesses	Package
testing ( ) - RT_elements::GotItALL::Search::testing	**test**ing	Operation
Testing - CoffeeMachine::Logical View::TestHarnesses::CoffeeMachine_Tester::State ...	Testing	State

Resulting elements are by default sorted alphabetically on their name. This means that those elements that match the search string exactly are on the top of the list. You can click on the table columns to sort the list differently, either according to the fully qualified name of the element or according to the type of element. This can often help you to find the element you are looking for if you don't remember all of its name, but know for example its type or approximately where in the model it is located.

For the above example, assume you are looking for a state which you think is called something that starts with “Test”. By sorting the result list on the “Type” column and scrolling down until you see the elements that are states, you quickly find out that the state you are looking for is the “Testing” state.

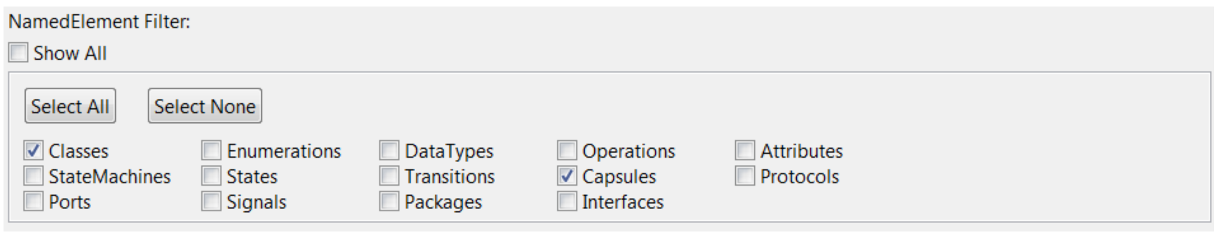
Once you have found the element you are looking for you can select it in the list and press the OK button (or you can just double-click the element). This closes the Find NamedElement dialog and the element you selected will be highlighted in the Project Explorer.

Note that by default only 50 matching elements will be listed in the table. This limit ensures that searches are reasonably quick because the most time consuming part of a search is actually to present the search result in the list. Also, it is rarely useful to get more than 50 matches since such a long list is quite tedious to go through. It is then usually better to type a more precise search string or use some of the search settings that are described later in this chapter in order to reduce the number of found elements. It is possible to not limit the number of found elements by unchecking the “Limit number of matches to” checkbox, but this is not recommended since it may cause Model RealTime to become unresponsive for a long time if a very large number of elements are found by a search.

The rest of this chapter will dig into the various settings that are provided by the Find NamedElement dialog, and also some more advanced ways of finding the element you are looking for.

NamedElement Filter

The upper part of the Find NamedElement dialog lets you specify a filter for the search. By default “Show All” is marked which means that no filter is used and all kinds of elements will be searched for. If you uncheck “Show All” you can mark specific kinds of elements using the checkboxes below. For example, in order to only search for Capsules and Classes with a certain name, mark checkboxes as shown below:


NamedElement Filter:

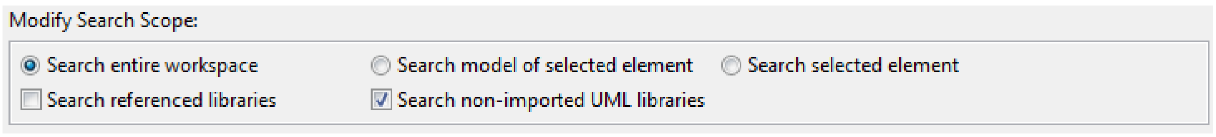
☐ Show All

✓ Classes	☐ Enumerations	☐ DataTypes	☐ Operations	☐ Attributes
☐ StateMachines	☐ States	☐ Transitions	✓ Capsules	☐ Protocols
☐ Ports	☐ Signals	☐ Packages	☐ Interfaces	

Note that there are many more kinds of elements in a model than shown above. The checkboxes only let you filter on those kinds of elements that are most commonly used in Model RealTime models. Therefore, even if you press the “Select All” button to mark all checkboxes you may still find fewer elements than if no filtering is used.

Search Scope

The ”Modify Search Scope” group in the middle of the dialog lets you set the scope of the search:


Modify Search Scope:

● Search entire workspace ○ Search model of selected element ○ Search selected element

☐ Search referenced libraries ☑ Search non-imported UML libraries

The broadest scope is to search in the entire workspace, and you can use this if you don't have any clue about where the element you are looking for is located. Searching in the entire workspace can take quite some time if you have a big workspace. Note that if you launched the Find NamedElement dialog without any element selected in the Project Explorer, then your only choice is to search the entire workspace, since the other two choices require a selected element.

If you know that the element you are looking for is located in a particular model, then you may select in the Project Explorer any element in that model before you launch the Find NamedElement dialog. Then you can mark the option ”Search model of selected element” in order to reduce the scope of the search to only look in that model. By model we here mean a logical model, i.e. an .emx file and zero to many .efx files.

The search will be even faster if you know that the element you are looking for is located in a particular model element. You can then select that element in the Project Explorer before you launch the Find NamedElement dialog, and then mark the option ”Search selected element” in order to reduce the scope of the search to only look at elements that are contained within that selected element. For example, assume you are looking for a class that you know should be located in a certain package. Before you run the Find NamedElement command select that package in the Project Explorer and then mark ”Search selected element” before you perform your search.

There are two additional settings that relate to the scope of the search.

- **Search referenced libraries**
  If this checkbox is marked the search scope will be extended to also contain all packages that are imported by the package to which the selected element belongs. This choice is hence only available if you selected an element in the Project Explorer before launching the Find NamedElement dialog.
- **Search non-imported UML libraries**
  If this checkbox is marked the search scope will be extended to include all packages that are imported by packages other than the package to which the selected element belongs. This setting is useful when you search in the entire workspace, and you want to search for an element that is located in a package that is imported by any of the packages that exist in the workspace, but not by the package of the element that was selected in the Project Explorer (if any).

Name Patterns

Sometimes you may not remember the exact name of the element you are looking for. In those cases it is useful to search for elements whose names match a certain pattern. You can use the same wildcard characters as described in Search Patterns.

Another kind of pattern is to search by means of abbreviations. This is useful if your model contains elements that have long names with "camel case", i.e. each part of a name starts with an uppercase letter. In those cases you can abbreviate the search string by just specifying those uppercase letters. For example, the search string "TMC" matches all of the following names: "TestModelCapsule", "TestMockingCode" and "TinyModelController". Of course, an element with the name "TMC" would also match. If you get too many matches when using an abbreviation consisting of all uppercase letters you can add some lowercase letters to make the search more precise. For example, if you would use the search string "TMCo" you would get matches for "TestMockingCode" and "TinyModelController" but not for "TestModelCapsule". And if you would search for "TeMC" you would get matches for "TestModelCapsule" and "TestMockingCode" but not for "TinyModelController". Note that the search string must start with an uppercase letter, and it must not contain anything but letters (for example no numbers, wildcards or other special characters) in order to be recognized as an abbreviation.

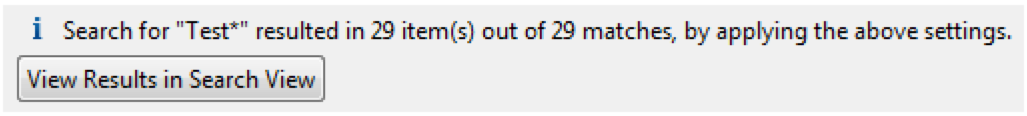
The Find NamedElement dialog has a checkbox "Case sensitive" which can be marked in order to perform case sensitive matches of names. You can use this checkbox to reduce the number of elements in the search result if you know that the case you used for the search string is the correct one. It is recommended to have "Case sensitive" enabled when you search by means of abbreviations, to make sure that you only find matches where the search string letters appear in upper-case.

Showing Search Result in Search View

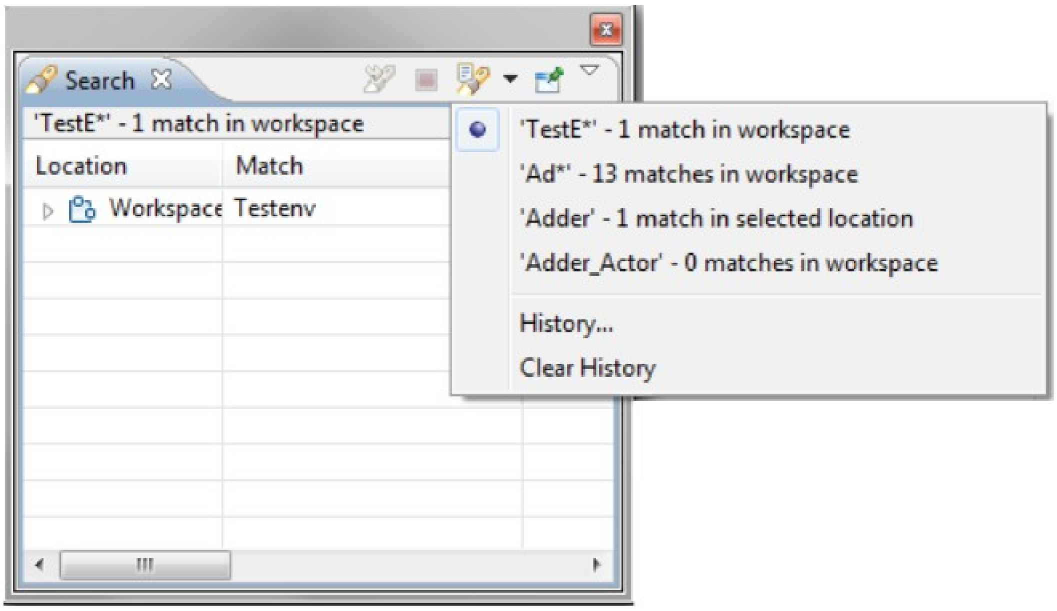
The Find NamedElement dialog is modal which usually is not a problem since you are supposed to find the element you are looking for by inspecting the search result list, and then close the dialog and navigate to that element in the Project Explorer. However, sometimes you may want to use the Find NamedElement dialog for slightly different purposes:

- From the limited information that is presented in the search result list you may not know which of the found elements that is the one you are looking for. For example, assume you are looking for an operation called "ComputeResult" which has an implementation that contains a certain comment. You can find all operations called "ComputeResult" but you cannot look into their implementation without first closing the Find NamedElement dialog.
- Your goal may not be to find only one particular element, but a group of elements that have a certain name. For example, assume you need to find all states called "HandleRequest" in order to check if they handle a particular event. You can find all the states with this name using the Find NamedElement dialog, but then you want to close the dialog and continue to work with this list using other Model RealTime views and editors.

In scenarios like these it is useful to be able to propagate the search result from the Find NamedElement dialog to the Search view. This can be done by pressing the button "View Results in Search View" which is available in the bottom left corner of the dialog when a search has been performed that resulted in a non-empty search result list.


i Search for "Test*" resulted in 29 item(s) out of 29 matches, by applying the above settings.

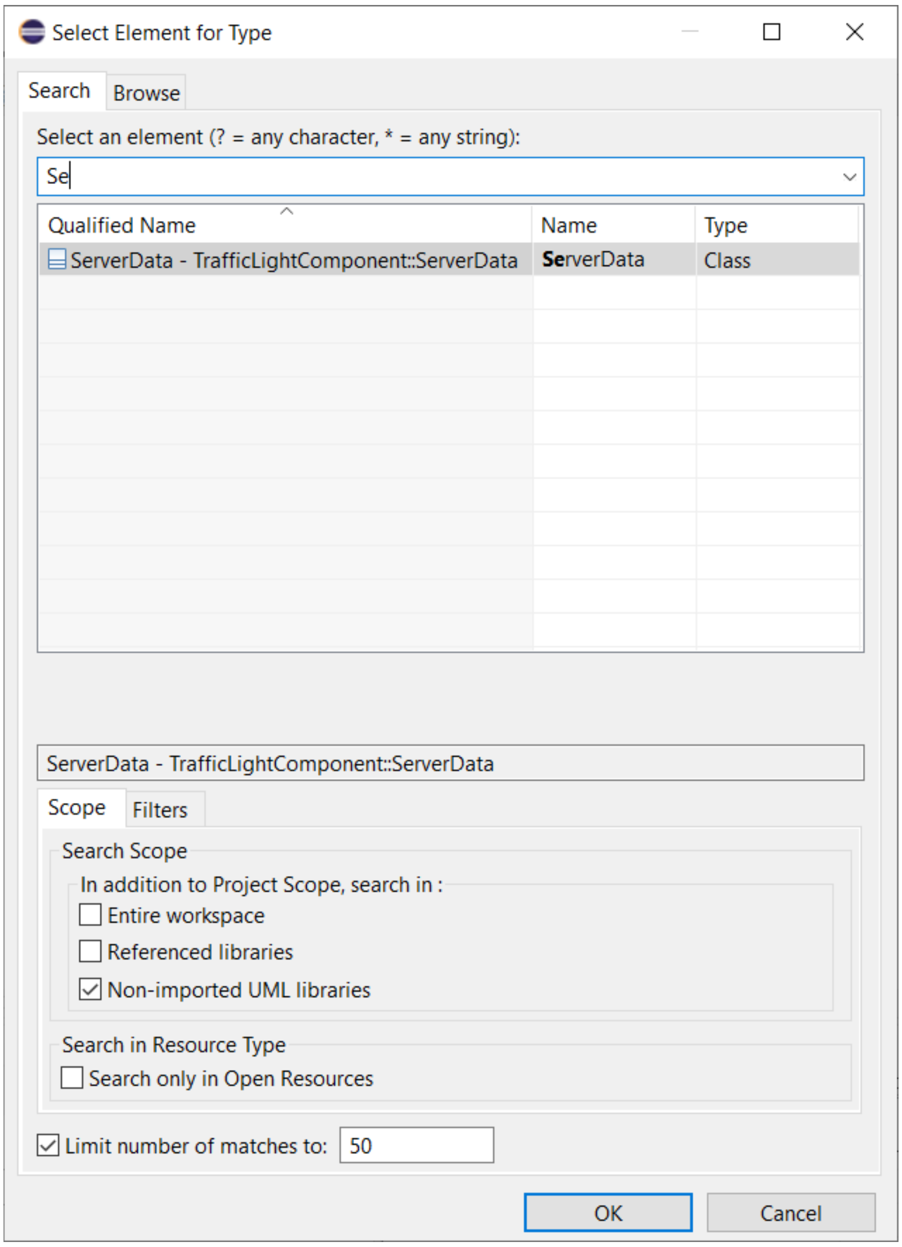
The Search view maintains a history so you can perform multiple searches in the Find NamedElement dialog and after each search press the button to populate the Search view with the results. Then when you have closed the dialog you can use the Search view's “Show Previous Searches” button in order to access the saved search results.


Search
'TestE*' - 1 match in workspace
Location
Match
Workspace
Testenv
'TestE*' - 1 match in workspace
'Ad*' - 13 matches in workspace
'Adder' - 1 match in selected location
'Adder_Actor' - 0 matches in workspace
History...
Clear History

Searching in the Select Element Dialog

The Select Element dialog appears in many situations when you are supposed to locate a particular element in the model. For example, the dialog appears if you press the “Select type” button in the Properties view in order to set the type of an attribute. There are two tabs in the Select Element dialog. The “Browse” tab can be used if you know exactly where in the model the element is located. Otherwise you can use the “Search” tab in order to find the element by searching for it.

Here is an example of what the Select Element dialog may look like when you have pressed the “Select type” button in the Properties view in order to set the type of an attribute:



Select an element (? = any character, * = any string):

Se

Qualified Name	Name	Type
ServerData - TrafficLightComponent::ServerData	ServerData	Class

ServerData - TrafficLightComponent::ServerData

Search Scope

In addition to Project Scope, search in :

- [ ] Entire workspace
- [ ] Referenced libraries
- [✓] Non-imported UML libraries

Search in Resource Type

- [ ] Search only in Open Resources

[✓] Limit number of matches to: 50

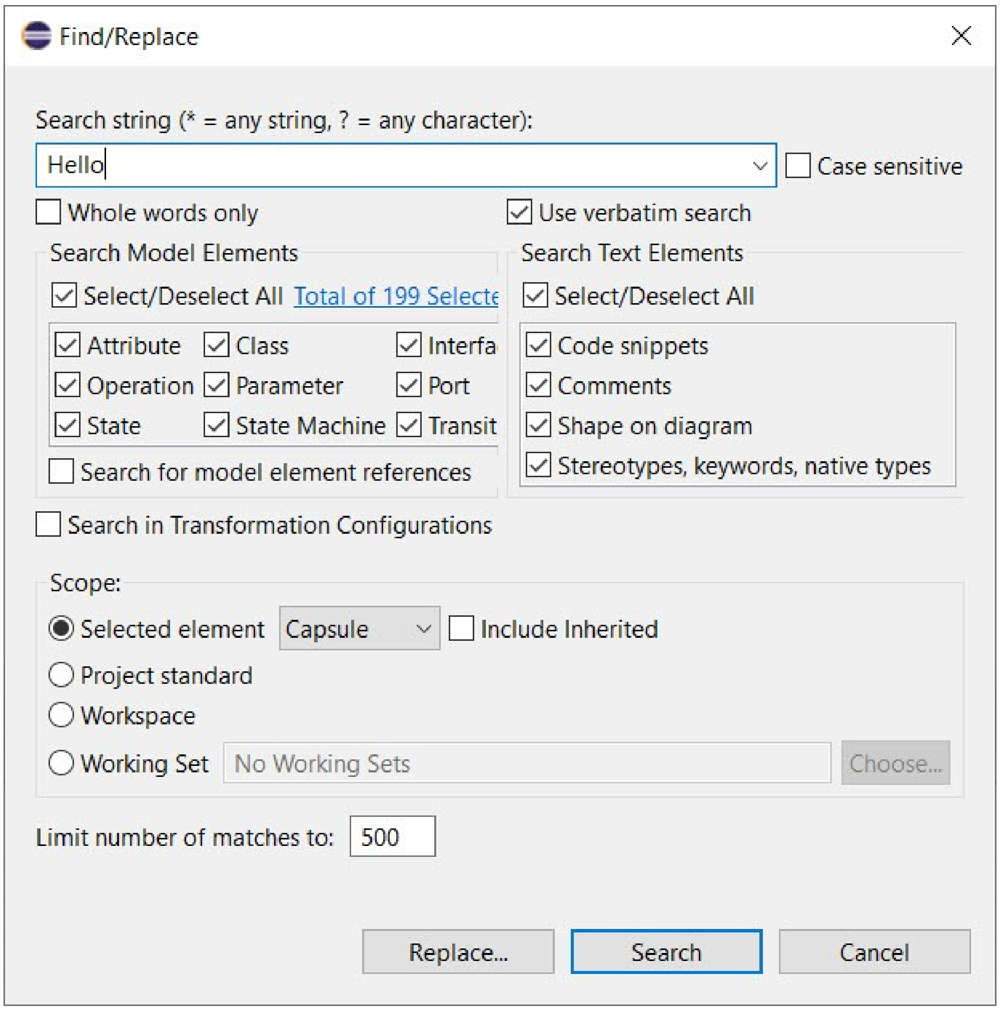
Just like the Find NamedElement dialog the Search tab in the Select Element dialog makes an index-based search for the name (or name pattern) of the element. Many of the search settings provided by the Search tab are also the same as in the Find NamedElement dialog. But there are also some differences, and the following are worth noticing:

- It is not possible to restrict the search scope to a particular element in the model. If you unmark all checkboxes in the Search Scope group the search takes place in project scope (i.e. in the models of the project to which the element belongs from where the dialog was launched).
- There is an additional checkbox "Search only in Open Resources" which allows you to avoid searching in closed models.
- The Filters tab allows you to specify what kinds of model elements you want to find. Note that the Select Element dialog applies an implicit filter based on the context from where it was launched. For example, if you launch the dialog in order to set the type of an attribute, the Filters tab will only show those element types that may be used for that purpose. If there is only one allowed element type, then the Filters tab will not appear at all.

Note also that contrary to Find NamedElement, which is an Model RealTime specific search command, the Select Element dialog provides a general search capability which works the same regardless of the kind of model on which it is used. This means that you may often find element types in the Filters tab which are not applicable for Model RealTime models, and Model RealTime specific element types (like capsules) are also missing there. Because of this a search may give matching elements of a kind that does not make sense in an Model RealTime model.

Find/Replace

The Find/Replace dialog comes in two variants in Model RealTime. You can either use it as a stand-alone dialog by means of the Find/Replace command (default keybinding Ctrl + F) that is available in the Edit menu and in the context menu of an element in the Project Explorer.


Find/Replace

Search string (* = any string, ? = any character): Hello

[ ] Case sensitive

[ ] Whole words only

[✓] Use verbatim search

Search Model Elements

[✓] Select/Deselect All Total of 199 Selecte

[✓] Attribute [✓] Class [✓] Interfa

[✓] Operation [✓] Parameter [✓] Port

[✓] State [✓] State Machine [✓] Transit

[ ] Search for model element references

Search Text Elements

[✓] Select/Deselect All

[✓] Code snippets

[✓] Comments

[✓] Shape on diagram

[✓] Stereotypes, keywords, native types

[ ] Search in Transformation Configurations

Scope:

● Selected element: Capsule [ ] Include Inherited

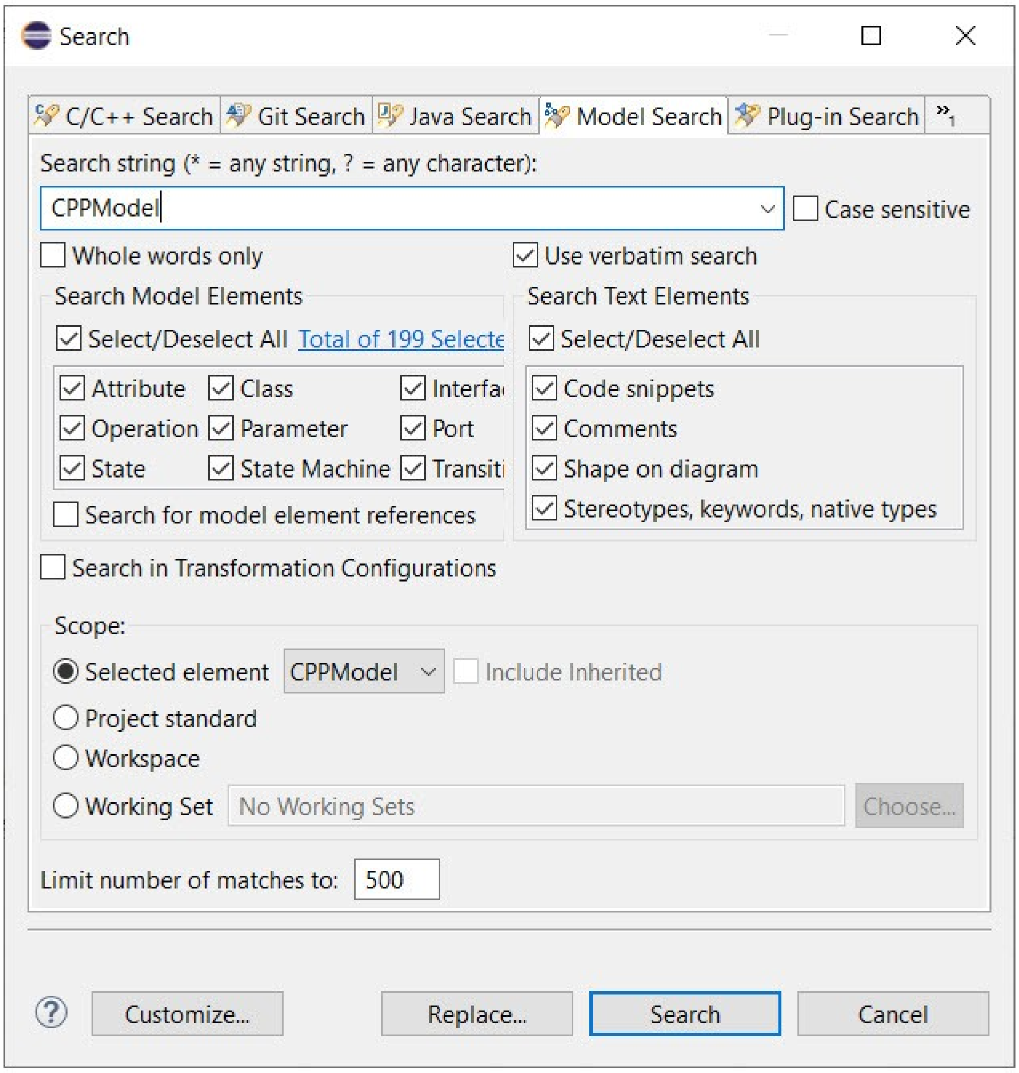
○ Project standard

○ Workspace

○ Working Set: No Working Sets

Limit number of matches to: 500

Another way is to use the Model Search command that is available in the Search menu (default keybinding Ctrl + Shift + M). It opens the general Eclipse Search dialog which contains many different search features, and Model Search is one of them.


Search

C/C++ Search | Git Search | Java Search | Model Search | Plug-in Search

Search string (* = any string, ? = any character):

CPPModel

- [ ] Case sensitive
- [ ] Whole words only
- [✓] Use verbatim search

Search Model Elements

- [✓] Select/Deselect All Total of 199 Selecte
- [✓] Attribute
- [✓] Class
- [✓] Interfa
- [✓] Operation
- [✓] Parameter
- [✓] Port
- [✓] State
- [✓] State Machine
- [✓] Transiti
- [ ] Search for model element references

Search Text Elements

- [✓] Select/Deselect All
- [✓] Code snippets
- [✓] Comments
- [✓] Shape on diagram
- [✓] Stereotypes, keywords, native types

- [ ] Search in Transformation Configurations

Scope:

- [●] Selected element CPPModel
- [ ] Include Inherited
- ○ Project standard
- ○ Workspace
- ○ Working Set No Working Sets Choose...

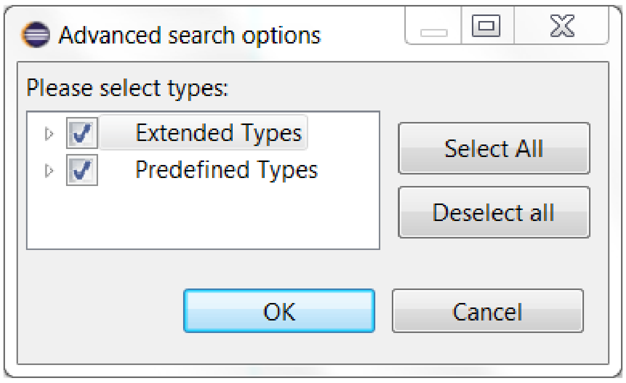
Limit number of matches to: 500

In both these cases the dialog works in the same way, and allows you to perform a textual search and also to replace found matches with another string. The main difference compared to searching from the Search Field is that the Find/Replace and Search dialogs allow you to specify certain filters and conditions before searching. For example, you can specify that you only want to look for certain kinds of model elements such as classes. Of course, you can still apply additional filters in the Search view after the search has been performed (see Filtering).

If you are looking for an element with a certain name, you could also consider using the Find NamedElement dialog instead, especially if you are not completely sure about its name and need to try searching for different names. However, one advantage compared to the Find NamedElement dialog is that the Find/Replace and Search dialogs allow you to filter on many more kinds of model elements (actually on all available kinds of model elements). To do this, press the hyperlink that shows the total number of selected element types:

- [✓] Select/Deselect All Total of 199 Selected

The Model RealTime specific element types, such as Capsule and Protocol, are located under “Extended Types”.



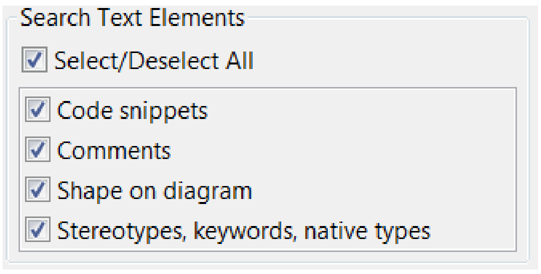
If you use the Find/Replace or Search dialog for finding elements with a certain name you should deselect the checkbox in "Search Text Elements". Otherwise a free-text search (see Free Text Searching) will also take place which may lead to additional search results and also slower search performance.

The checkbox "Use verbatim search" can be checked if you want to search verbatimly even if the corresponding workspace preference is turned off. Hence, marking this checkbox has the same effect as enclosing the search string in double quotes in the search field. See Verbatim versus Non-Verbatim Searching for more information about the difference of these two ways of searching.

Free Text Searching

The Find/Replace and Search dialogs can be used for free-text searching of strings within a model. However, it is usually more convenient to instead use the Search Field for this.

If you anyway want to use Find/Replace or Search dialog for free-text searching, mark the kinds of text elements you want to include in the search:



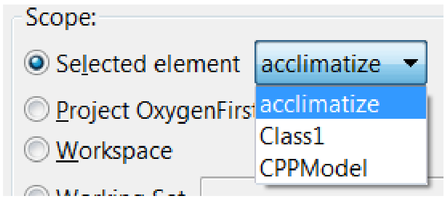
- **Code snippets**
  Mark this if you want to search for code snippets in the model that contain the search string. For example, if you want to find all code references to a certain attribute, you may search in code snippets for the name of that attribute.
- **Comments**
  Mark this if you want to search for model comments that contain the search string.
- **Shape on diagram**
  Mark this if you want to search for shapes on diagrams that have a text label that contains the search string. You can for example use this checkbox if you want to search in the text of Note symbols.

- **Stereotypes, keywords, native types**
  Mark this if you want to search in applied stereotypes, keywords on model elements and native C/C++ types (which includes all text you can write in the Type field for an element).

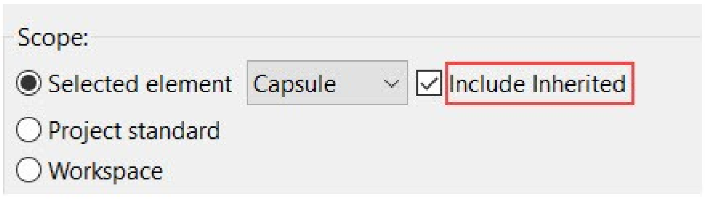
Search Scope

The Find/Replace and Search dialogs support the same search scopes as the search field (see Scope).

If an element was selected when opening the Find/Replace dialog you can set the selected element as the search scope. You can then also use the drop-down menu to extend the search scope to any element that contains the selected element:



The scope section also provides an "Include Inherited" checkbox to search in the inheritance hierarchy of the selected element.



Note that this checkbox is only enabled if the selected element can inherit (that is, it is either a class, capsule, interface or protocol).

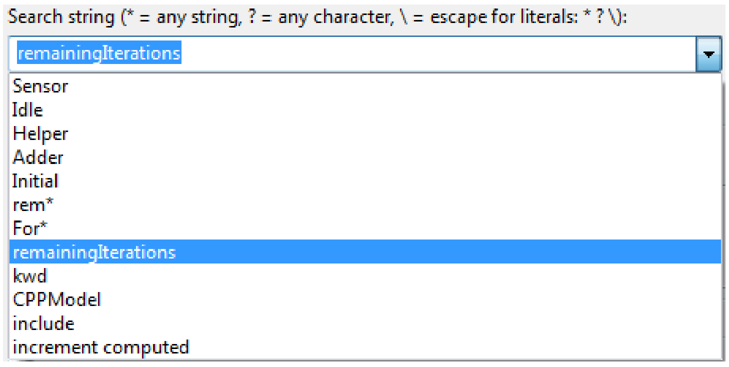
Search Options

The Find/Replace and Search dialogs support the same search patterns with wildcards as described in Search Patterns. It also allows you to limit the number of matches, which is recommended to avoid getting too many search result items (with poor performance as a consequence). Case-sensitive searching is supported by marking the checkbox “Case sensitive”.

Mark the checkbox "Search for model element references" if you want to include references to model elements in the search result. For example, searching for a class will then also report matches for all attributes that are typed by that class. If you only are interested in finding the references, a better approach is described in Searching for References to an Element.

Mark the checkbox "Search in Transformation Configurations" if you want to search also inside transformation configuration files.

The dialog remembers search strings that have been used in the past. Initially, when the dialog is launched, the search string is by default set to the name of the element that is selected in the Project Explorer or in a diagram. If you instead want to search for something that you have searched for in the past, you can use the dropdown menu in the combo box to pick a previously used search string.

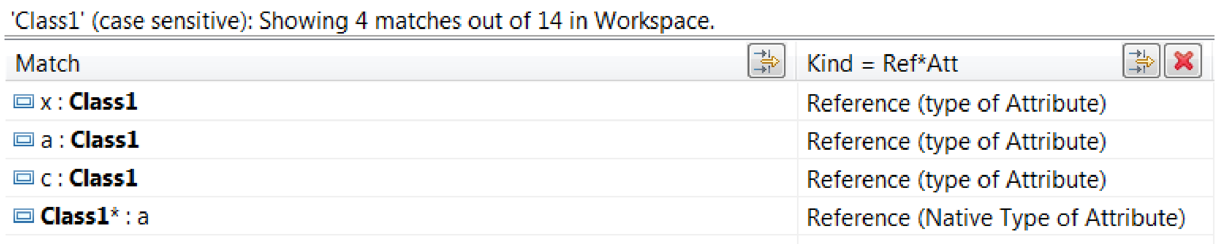

Search string (* = any string, ? = any character, \ = escape for literals: * ? \):

remainingIterations

Sensor
Idle
Helper
Adder
Initial
rem*
For*
remainingIterations
kwd
CPPModel
include
increment computed

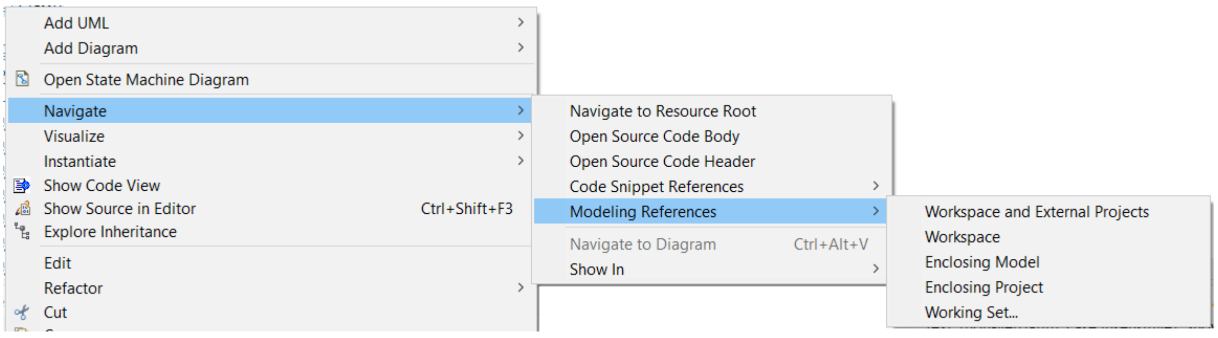
Searching for References to an Element

If you want to find all elements in a model which refer to a particular element it's often just enough to type the name of that element in the Search Field and then filter the Kind column to only show Reference matches. You can even filter more specifically, to only consider references to some kinds of elements.
As an example, assume you want to find all attributes which are typed by a certain class "Class1". Type this name in the search field and run a search. Then filter the Kind column with the pattern "Ref*Att":


'Class1' (case sensitive): Showing 4 matches out of 14 in Workspace.

Match	Kind = Ref*Att
x : **Class1**	Reference (type of Attribute)
a : **Class1**	Reference (type of Attribute)
c : **Class1**	Reference (type of Attribute)
Class1* : a	Reference (Native Type of Attribute)

However, if the referenced element does not have a name, this approach will not work. In that case you can instead use the Modeling References command which is available in the context menu of an element that is selected in the Project Explorer:



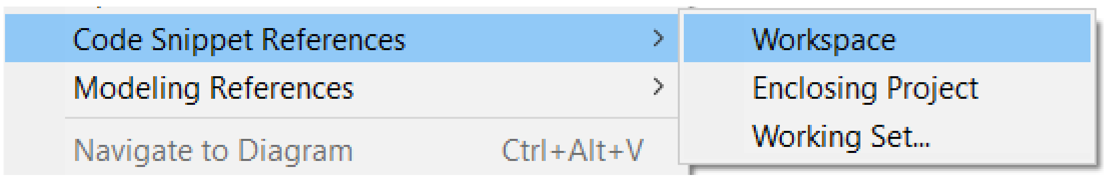
Modeling References is an index-based search command, and it finds all references to the selected element within the model. However, it does not find references that are located inside code snippets. To search for references inside code snippets you should use the Search Field instead. Alternatively you can use the Find/Replace dialog (see Free Text Searching). Yet another possibility is to use the command Code Snippet References which also is available in the context menu of the Project Explorer. Note that Code Snippet References makes use of CDT and hence requires the presence of generated C++ code.

The search scope for the Modeling References command is set when you choose any of the items in the sub menu shown above. The following search scopes are available:

- **Workspace and External Projects**
  Search for references to the selected element in all models of the workspace, and also in external projects.
- **Workspace**
  Search for references to the selected element in all models of the workspace.

- **Enclosing Project**
  Search for references to the selected element in all models of the project to which it belongs.
- **Enclosing Model**
  Search for references to the selected element in the model to which it belongs. Model here means "logical model", i.e. an .emx file and zero to many .efx files.
- **Working Set**
  Search for references to the selected element in models specified by a working set.

The Code Snippet References command has a similar sub menu, but supports fewer search scopes:

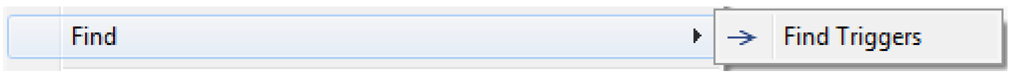


References that are found by the Modeling References and Code Snippet References commands are presented in the Search view. From there you can double-click to navigate to the element that contains the reference. Depending on the kind of element you may also find other navigation possibilities in the context menu, for example "Navigate to Diagram" for elements that are shown in a diagram. There are a few cases where the element that contains the reference may not have any presentation in any view in the Model RealTime user interface. This happens for example for internal elements created by Model RealTime itself. Those references cannot be navigated to.

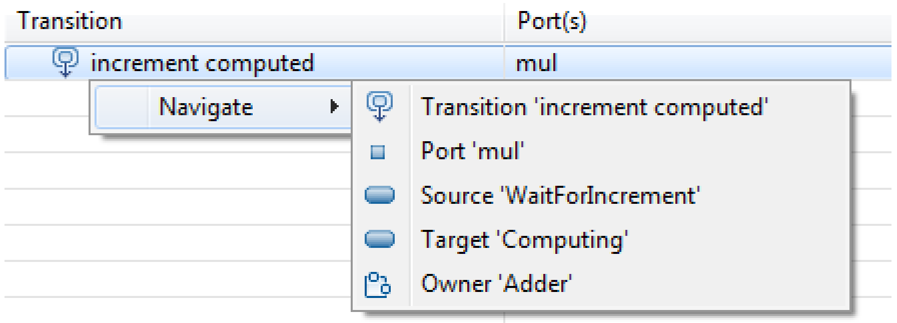
Memory-based Search Commands

All search commands described so far are index-based because they search using an index. Model RealTime also provides a few search commands that do not make use of an index. Instead they operate directly on the model that has been loaded into memory. These search commands are therefore referred to as memory-based search commands, since they work by traversing the model that has been loaded into memory and collecting result elements according to certain rules that are specific for each such search command. Contrary to the index-based search commands, memory-based search commands are therefore special purpose search commands. They can find elements from the model based on a specific criterion that may be hard or impossible to accomplish by means of the more general index-based search commands.

A memory-based search command always starts searching from a context element. This is the element that is selected in the Project Explorer or in a diagram. The memory-based search commands that are available for the selected element appear in the context menu, usually in a Find sub menu. For example:



One way to think about memory-based search commands is like running queries on the model. Starting from the context element a memory-based search command collects all elements that are returned by its query. These resulting elements are presented in the Search view. From there you can navigate by double-clicking on the result elements. You can also navigate to other elements that are related to a result element in one way or another. Those additional navigation commands are present in the context menu for an element in the Search view. For example:


Transition	Port(s)
increment computed	mul

Here the memory-based search command has returned a transition in the model. In addition to the transition itself you can navigate to the port referenced by one of its triggers, the source and target states as well as the owner capsule to which the transition belongs.

Keep in mind that the scope of a memory-based search command is the set of models that are currently open (i.e. loaded into memory). Closed models are not searched. If you want to search in all models of the workspace you should therefore make sure that all those models are loaded before running the memory-based search command. The easiest way to do this is by using the command *Modeling – Load UML Models*.

The rest of this chapter describes the memory-based search commands that are available in Model RealTime.

Find Triggers

This command finds transition triggers. It is available for the following kinds of context elements:

- **Protocol event**
  Finds the transition triggers which trigger on the protocol event.
- **Behavior port**
  Finds the transition triggers which trigger when an event arrives at the port.
- **Trigger operation**
  Finds the transition triggers which trigger when the trigger operation is called.
- **Call event**
  Finds the transition triggers which refer to the call event.

For example, assume you want to know where a particular protocol event is handled in the model. To find out you can run the Find Triggers command from the context menu of the protocol event.

The following information is provided in the Search view for each found transition trigger:

- The transition to which the trigger belongs.
- The port(s) that are referenced by the trigger, i.e. the port(s) through which an event must arrive in order to enable that particular transition trigger to trigger its transition.
- The guard condition of the trigger, i.e. the condition that must be fulfilled in order to enable that particular transition trigger.
- The guard condition of the transition to which the trigger belongs. This condition must be fulfilled in order to trigger the transition.
- The source and target elements of the transition in the state machine, for example a state or a pseudo state.
- The class or capsule that owns the state machine to which the transition of the trigger belongs.

You can navigate to each of these related elements, except the guard conditions, from the Navigate context menu of each found transition trigger.

Find Triggering Events

This command finds out which events that may trigger a transition in a certain context. It is available for the following kinds of context elements:

- **State**
  Finds the events that may trigger a transition when a certain state is active in a state machine.
- **State machine**
  Finds all events that a state machine may handle.
- **Capsule or class with a state machine**
  Finds all events that the state machine of the capsule or class may handle.

For example, assume you want to know what events that may trigger a transition when a certain state is active. This information is not so easy to obtain just by looking at the state machine diagram, because it's not enough to just consider the transitions that leave that particular state. If the state is enclosed in a hierarchy of composite states, then any transition that leaves one of those enclosing states also need to be considered. And if the selected state itself is a composite state it may contain sub states with outgoing transitions that also may trigger when that state is active. Finally, any state that is redefined may have additional outgoing transitions inherited from a base state machine. The Find Triggering Events command makes it easy to collect the complete list of events that may trigger a transition when a particular state is active.

The command returns a list of transition triggers, and for each such trigger the following information is provided in the Search view:
- The event of the trigger, i.e. the event that must arrive in order to trigger a transition.
- The transition to which the trigger belongs, i.e. the transition which may be triggered when the event arrives.
- The port(s) that are referenced by the trigger, i.e. the port(s) through which an event must arrive in order to enable that particular transition trigger.
- The guard condition of the trigger, i.e. the condition that must be fulfilled in order to enable that particular transition trigger.
- The guard condition of the transition to which the trigger belongs. This condition must be fulfilled in order to trigger the transition.
- The source and target elements of the transition in the state machine, for example a state or a pseudo state.

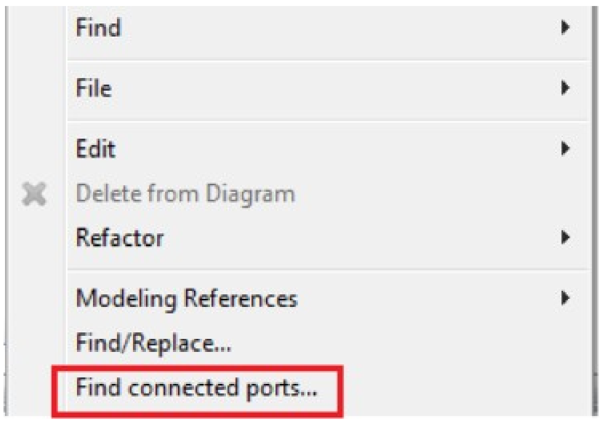
You can navigate to each of these related elements, except the guard conditions, from the Navigate context menu of each found transition trigger.

For each transition trigger that is found an order id is also presented in the Search view. The order id consists of two numbers separated by a dot. The first number tells you where in the composite state hierarchy that particular transition trigger is located. If it is located on a transition that leaves the context state itself, it gets the number 0. If it is located on a transition that leaves an enclosing state it gets a positive number (1 means one level up in the state hierarchy, 2 means two levels up, and so on). If the trigger is located on a transition that leaves a state that is a sub state of the context state it gets a negative number (-1 means one level down in the state hierarchy, -2 means two levels down, and so on). The second number in the order id is just increased by one for each found transition trigger at that level, to ensure that they all get a unique id.

It's often useful to sort the search result by clicking on the column headers in the Search view. For example, assume you want to find out if a capsule is prepared for handling a certain event. By running the Find Triggering Events command on the capsule, and then sort on the name of the event, you can easily find out if a particular event may be received by the capsule, and which transitions that then may trigger.

Find Connected Ports

This command finds out which other ports a selected port may communicate with at run-time, either by means of the static connector structure (for wired ports) or by means of dynamically established connections (for unwired ports). Note that this command for historical reasons is not present in the Find context menu, but directly on the top level in the context menu of a port:



When you run this memory-based search command you will first get a question whether the command should traverse relay ports. If you answer No, then the command will find the ports which may be involved in direct communication with the selected port. Some of those ports may be relay ports. However, usually it's more interesting to answer Yes, since the search then proceeds through relay ports to find out the final behavior port to which a message would be delivered at run-time.

For an unwired port the command cannot use the connector structure to find out the connected ports. Instead it uses the registration kind property to find out how the ports will be connected at runtime. However, ports that have their registration kind set to Application will not be considered by the search command, since those ports are connected programmatically and the service name is not statically known.

The following information is provided in the Search view for each found connected port:

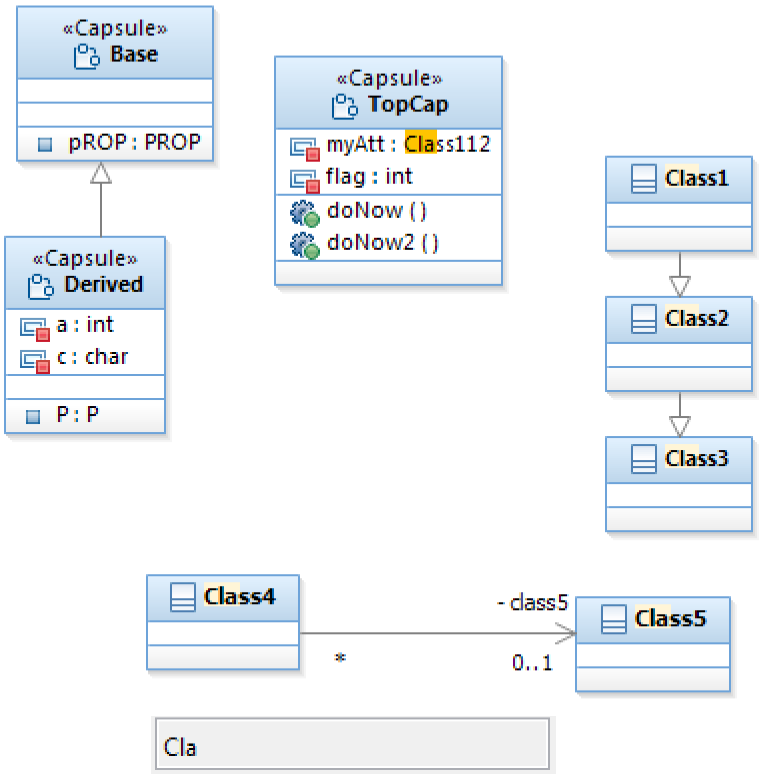
- The name of the connected port
- The capsule part and its type, on which the connected port is defined
- The protocol of the port
- The capsule where the port can be seen in its composite structure diagram

Incremental Textual Search in Diagrams

Model RealTime provides a command for searching for text strings inside diagrams. The command is incremental which means that you can type the text string to search for character-by-character, and after each keystroke Model RealTime will highlight matching text strings in the diagram. The command is very similar to the incremental search that Eclipse provides for C++ source code editors. This way of searching is useful if you want to know if a certain piece of text is shown in a diagram. It is especially valuable on large and cluttered diagrams where there are many text labels present.

To perform an incremental textual search open the diagram you want to search in and invoke the Incremental Find command by means of its keybinding, which by default is Ctrl+J. A text area in the status bar will then appear, showing that you are now running an incremental find in the diagram:

Then start to type a few characters that match the text you are looking for. As you type Model RealTime will highlight matching texts in the diagram. The first matching text that is found is highlighted in a strong yellow color. Other matching texts in the diagram are highlighted using a bleak yellow color. Here is an example of what a class diagram may look like when searching for the text string "Cla":



The text that is being searched for is shown in the text area in the status bar.

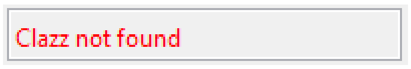
By pressing Ctrl+J again you will advance the search so that the next matching text gets highlighted with strong yellow color. Thereby you can go through all texts that match a particular search string by repeatedly pressing Ctrl+J.

The diagram is searched in the direction from the top-left corner down to the bottom-right corner. This order corresponds to a left-to-right reading order. To search the diagram in the reverse order you can instead invoke the command Reverse Incremental Find which has the default keybinding Ctrl+Shift+J.

An incremental diagram text search can wrap, which means that when you have moved to the last matching text label and then press Ctrl+J once more, then the search starts over from the top-left corner again so that the first match gets highlighted. A reverse search will also wrap if pressing Ctrl+Shift+J when the first match is highlighted. Then the last match becomes highlighted.

The diagram scrolls automatically to make the text that is highlighted visible on the screen. Hence you don't have to zoom out before starting to search in case you have a big diagram that doesn't fit fully on the screen.

If there are no text labels in the diagram that match the characters you have typed, a message will be printed in the status bar text area. For example:



```
Clazz not found
```

If this happens you can backspace and type some other characters to change the search string.

The incremental diagram text search command terminates when you have stopped typing and click somewhere else, for example in the background of the diagram.