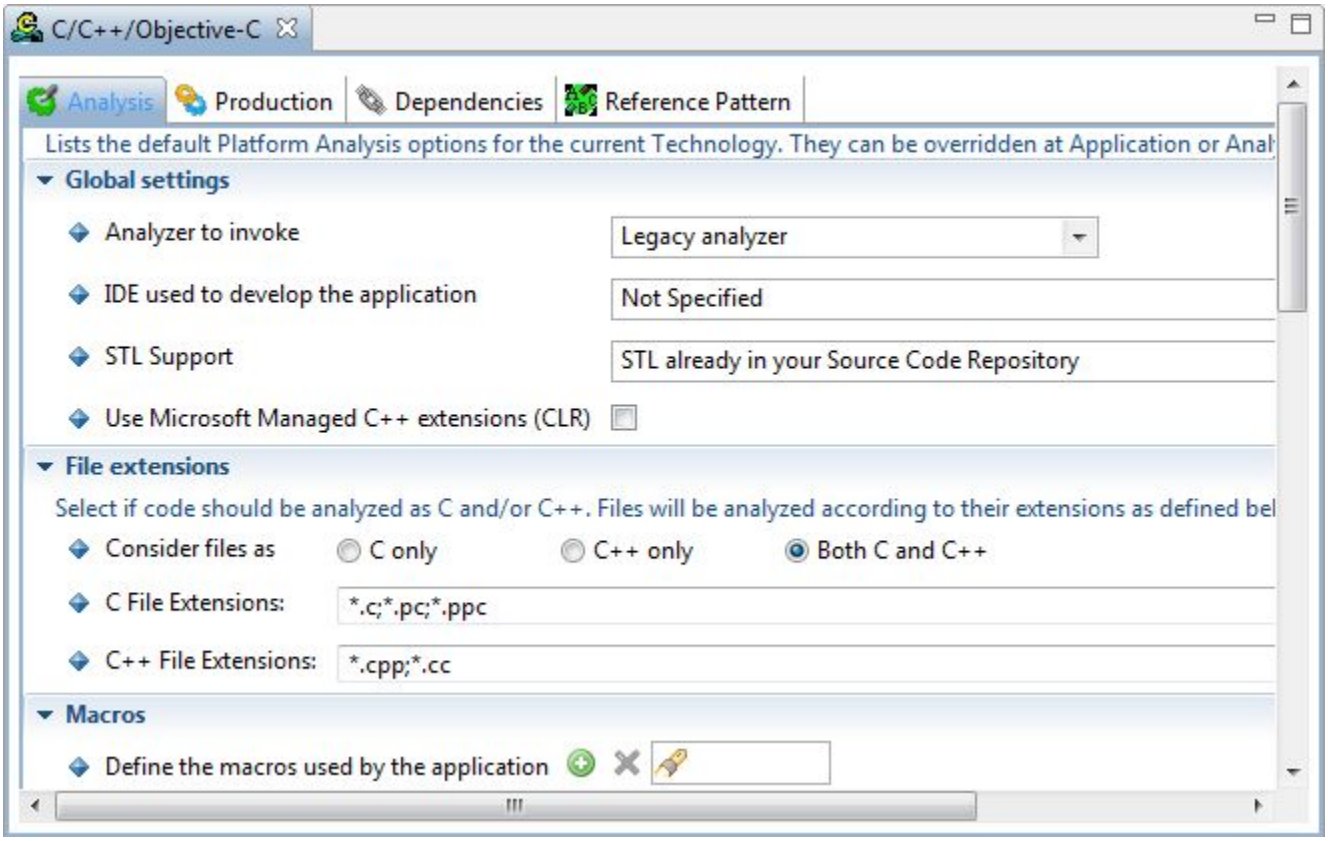
CMS - C/C++/Objective-C Technology options

C/C++/Objective-C Technology options



The **C/C++/Objective-C Technology editor** contains various tabs. Each is explained below.

Notes

- The options in this Technology editor are the default options that will be used to populate the same fields in **Applications** and **Analysis Units**. If you need to define specific options for a specific Application or Analysis Unit, then please use the Application editor (Analysis tab) or the Analysis Unit editor.
- If you make a change to a specific option at Application or Analysis Unit level, and then subsequently change the same option via this editor, this setting will NOT be mirrored back to the Application or Analysis Unit - this is because specific settings at Application and Analysis Unit level have precedence if they have been changed from the default setting available in this editor.
- The C++ Reference Guide contains miscellaneous technical information about the C/C++ analysis process.

Analysis tab

Global settings

Analyzer to invoke	Use this option to choose between the analysis engine that will be used to run the analysis:	**Legacy analyzer** This option invokes the legacy C/C++ analyzer as provided in previous releases of CAST AIP. By default this option **will always be active** at Technology/Application/Analysis Unit level.
		Clang analyzer This option is only relevant for those that are using the **CAST AIP C Family Extension** in order to analyze **Objective C** source code. This option must be active in order for the Objective C source code to be analyzed by correctly. If this option is **invoked** and: • the **CAST AIP C Family Extension** is **not** installed, then an error will be displayed when the analysis is run: `The C-Family analyzer is not installed. Please use the Extension Downloader to get it (com.castsoftware.cfamily)` • the **CAST AIP C Family Extension** is installed but your source code does **not contain Objective C** (i.e. it is a traditional Visual C/C++ application) then results will not be consistent with the output of the "legacy analyzer". In this situation where only traditional Visual C/C++ source code is being analayzed, CAST recommends that the **Legacy analyzer** option is always used.

IDE used to develop the Application/Analysis Unit	Use this option to choose the development environment of your source code files. The **CAST Management Studio** then applies a default **Environment Profile** (these can be viewed here: <install_folder>\EnvProf\CPP) that matches the selected **Development Environment**. If you are analysing Objective C with the C Family Extension, please ensure that you select the **Xcode** option from the drop down list. Note that if the CAST Delivery Manager Tool detects the source code as a variant of Visual C++, then the appropriate option will be set at Analysis Unit level.	
STL Support	This option allows you to choose how you want CAST to handle **STL header files** that are referenced in your source code. Not all other Environment Profiles provided by CAST contain STL header files (for copyright reasons) and this can lead to an incomplete analysis, and thus to incomplete results. As a direct result of this, CAST has introduced this option that allows you to tell the CAST Management Studio whether the STL header files are available or not:	
	CAST emulation	If STL header files **are not available** in your source code, select this option. This will then force the CAST Management Studio to activate a basic (i.e. non exhaustive) emulation of the most common STL headers (i.e. <string>, <vector>, <algorithm> etc.), thus avoiding an incomplete analysis or ambiguous/erroneous analysis results in most circumstances. This option is provided via a pre-defined Environment Profile, and you can list the headers that are handled by viewing the contents of the following folder: `<install_folder>\EnvProf\CPP\STLforUnix`
	STL already in your Source Code Repository	If STL header files **are available** in your source code, select this option. The STL header files will be analyzed along with the rest of your source code as normal. This is the default option.
Use Microsoft Managed C++ extensions (CLR)	Select this option if your Source Code contains Microsoft Managed C++ extensions targeted at the Common Language Runtime (CLR). A corresponding Environment Profile will then be applied. See: `<install_folder>\EnvProf\CPP\Microsoft Managed C++` Note that this option is automatically ticked if the project is configured to execute code under the Common Language Runtime virtual machine.	

File Extensions

Consider Files As	**C only**	If this option is selected, C++ specifics are ignored (templates, namespaces, classes, inheritance and polymorphism) and are not taken into account when the source files are analyzed. This option allows optimized handling of C files.
	C++ only	If this option is selected, C++ specifics (templates, namespaces, classes, inheritance and polymorphism) are taken into account when the source files are analyzed.
	Both C and C++	If this option is selected only one distinct job is generated and executed: one **C++ technology** job using the **Consider Files As > C++** option. However the results stored in the CAST Analysis Service will contain both **C and C++ object types**. In other words, both file types are analyzed as **C++ files** and **Line Of Code Count** (LoC) for C files will be included in the C++ technology rather than the C technology.
C File Extensions	Use this option to specify the file extensions that will be considered as **C** files for analysis purposes.	
C++ File Extensions	Use this option to specify the file extensions that will be considered as **C++** files for analysis purposes.	

Impacts of choosing one parsing option over the other.

Selecting one of these options will cause the source code to be handled differently by the analyzer:

- A function or a method contains any number of declarations and at least one definition. The grouping of these blocks in one unique object is done differently by the analyzer according to whether the C++ or C option has been selected:

  - With option C, declaration or definition blocks with the same name are grouped together in the same function or in the same method without taking into account any parameters.

  - With option C++, declaration or definition blocks with the same name and the same parameters are grouped together in the same function or in the same method.

  For example:

  ```
  void foo();
  void foo(int a)
  {
  }
  ```

  Using the option C++ here would create two functions called "foo" in the Analysis Service, whereas using the C option would only create one "foo" function.
- The analyzer detects calling links differently depending on whether the C or C++ option is selected:

  - With option C, the analyzer believes that the function or the method is called as soon as an object with the same name is called.

  - With option C++, the analyzer believes that the function or the method is called if an object with the same name and comparable parameters is also called.

  For example (and using the previous example):

  ```
  void main ()
  {
  foo();
  }
  ```

  Because using the C++ option with the previous example would create two functions "foo" in the Analysis Service - one with a parameter, the other without - the analyzer will only create a Call type link to the function WITHOUT the parameter. Using the C option, the analyzer will create a link to the one function even if it has parameters.

Notes

- A file written in class C can be analyzed with the option **C++ only** with no risk. The only consequences are that the job may take slightly longer to run and some groupings of functions/methods with the same name but different parameters will not be carried out.
- A file written in C++ must NOT be analyzed with the option **C File**.
- When C/C++ Analysis Units are auto created in the CAST Delivery Manager Tool from Visual Studio project files (.vcproj files) and these project files have the Visual Studio option **Compile As** NOT set in **bold** (i.e. this reflects a default Visual Studio value which is not written in the .vcproj file), the CAST Management Studio will not be able to detect the **Compile As** value. As a result the **Compile As** value cannot be reflected in the Analysis Unit under the option **Consider File As**. In this instance, the source file extension will be used to determine the Analysis Unit **Consider File As** value.

  For example, you may have a project which has the value **Compile As** set to **Compile as C++ Code** in the Visual Studio environment (NOT set in bold). In this instance, when the Analysis Units are auto created, the CAST Management Studio cannot detect the **Compile As** value and will set the Analysis Unit **Consider File As** option to **C only** (using the file extension).
- If you have an application where **.c files** must be analyzed as **C++ files**, you should set the .c extension as a C++ extension - i.e. add it to the **C++ File Extensions** field.

Includes - only visible at **Application** and **Analysis Unit** level

The options **Force Include File** and **Define the Include Paths** enable you to define **additional paths** for files that have been specified in the #include part of your project files. This is to ensure the correct resolution of links to files located in other directories.

The analyzer will take into account the two "include" styles:

- **#include "fileA.h"**
- **#include <fileB.h>**

In the first case (where an #include uses " "), the analyzer will automatically search the path of the file that contains the #include, THEN the additional paths you have added and in the order specified.

Where < > is used in an #include command, the analyzer will search the additional paths first then, THEN the path of the the file that contains the #include.

Force Include File	This option enables you to specify macros in file format: • Use the **Browse** button to browse for the file you want to include. This file will contain the macros that you want the analyzer to take into account and must have the same structure as any C/C++ file (in other words, this file can contain "#define" orders to define macros, "#include" orders to include files, but also declare functions). It is possible to enter a simple filename (for example test.h) without an absolute path. In this case, the analyzer will search for the file as follows: • in the associated Source Repository • then in any defined **Include Path** (defined in **this section** or in an **Environment Profile** defined via the **IDE used to develop the application** option above or via a **Custom Environment Profile** defined in the **Custom Environment Profiles** section below). Please see the C++ Reference Guide for more information about how this option is handled in CAST Enlighten (links and book marks).

Define the Include Paths used by the Analysis Unit

This option enables you to add your include paths:

Adding a path

1. Click the  button. A new path called "Folder" will be displayed in the list.
2. To define the path (using the Browse option) and whether it is **recursive** or not, select the "Folder" item and use the relevant fields directly below the list.
3. The **Recursive** option indicates whether any sub-folders and the files they contain that are located in the chosen path will be included for resolution purposes (TRUE or FALSE). By default all paths that you add to the list will be flagged as FALSE - i.e. their sub-folders and any paths they contain will not be included. Choose whether the path is recursive as appropriate.

 **Notes**

- Please note that CAST only recommends selecting the **Recursive** option if you know exactly what you want to achieve. In the vast majority of situations, you should instead define each include path exactly as it is defined in the C++ project itself.

  For example, if your C++ project uses the "comp/include" include path, and your sources reference the file "stat.h" through:

  #include <sys/stat.h> // comp/include/sys/stat.h intended

  just add the path "comp/include" here, like in your C++ project, without selecting the **Recursive** option.

  Recursive is only useful if you really want to list all subdirectories of a directory, and spare some typing. This is generally not the case.

Removing a path

To remove a path that you do not want to include in the analysis:

1. Select the path that you want to remove from the list by left-clicking with the mouse.
2. Click the **Remove** button  to clear the selected item(s). The **Delete** key and the right-click short cut menu can also be used. You can remove multiple items by selecting the items in the list and then clicking the **Remove** button.

Editing an existing path

If you want to edit an existing path:

- Select the item in the list and use the fields directly below the list to make your changes.

Up and Down buttons

If you have two or more paths, then you can use the **Up**  and **Down**  arrow buttons to **re-order** the paths. This is **important** as the order they appear in the list is the order in which the paths are dealt with during the analysis process. To move a path down one level, select the path, then click **Down**; to move it up one level, click **Up**.

 **Notes**

- Please note that you can also use the CAST environment variable **$(CastCommonDir)** when adding an include path. This variable corresponds to the CAST installation folder "**%CommonProgramFiles%\CAST\CAST**". When you are using a non-English version of Windows, this variable will be localized.
- Note that it is also possible to define **persistent Include Paths** via a custom Environment Profile (see **Custom Environment Profiles** section below).
- Note that include paths configured in the current **Includes section** will be processed BEFORE any definitions made in the Environment Profiles associated to the **IDE used to develop the Analysis Unit** option, and to any **Custom Environment Profiles** (see **Custom Environment Profiles** section) you have added.

Defining include paths for Microsoft compilers

Usually, if the Microsoft C++ compiler is installed on the analysis machine, you just need to select the default "Microsoft VCxxxx" Environment Profile (via the **IDE used to develop the Analysis Unit** option above). In some cases, you may also need to provide your include list yourself. The following rules should be followed:

1. Do not add more directories then required
2. Do not select the "recursive" option for these paths
3. Do not select a path such as **C:\Program Files\Microsoft Visual Studio 10.0\VC\crt\src** (this typically contains files that are reserved for Microsoft internal use and debugger)

In any of the three cases above, you will get an error message during the analysis such as:

```
#error directive encountered 'Use of C runtime library internal header file... (file 'C:
\PROGRAM FILES\MICROSOFT[...]\VC\CRT\SRC\CRTDEFS.H")'
```

Include paths for Microsoft compilers should resemble the following:

For Microsoft VC 9.0 (2008) and higher:

- C:\Program Files\Microsoft Visual Studio 10.0\VC\include
- C:\Program Files\Microsoft Visual Studio 10.0\VC\atlmfc\include
- C:\Program Files\Microsoft SDKs\Windows\vxx.xx\include

(the third one is for <windows.h> etc., the exact value for xx.xx depends on your exact installation)

For older compilers:

- C:\Program Files\Microsoft Visual Studio 8\VC\include
- C:\Program Files\Microsoft Visual Studio 8\VC\atlmfc\include
- C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDKinclude

For even older MS Dev 6 (VC98):

- C:\Program Files\Microsoft Visual Studio\VC98\Include
- C:\Program Files\Microsoft Visual Studio\VC98\MFC\Include
- C:\Program Files\Microsoft Visual Studio\VC98\atl\Include

Macros

Define the macros used by the application/analysis unit

Use this option to define any macros that are present in your source code and that need to be taken into account during the analysis.

Adding a macro

1. Click the  button. A new macro called "MY_MACRO" will be displayed in the list.
2. Define the macro's name and value in the fields that are displayed below the macro list.

Removing a macro

To remove a macro that you do not want to include in the analysis:

1. Select the macro that you want to remove from the list by left-clicking with the mouse.
2. Click the **Remove** button  to clear the selected item(s). The **Delete** key and the right-click short cut menu can also be used. You can remove multiple items by selecting the items in the list and then clicking the **Remove** button.

Editing a macro	If you want to edit an existing macro: • Select the item and edit the name and value as necessary in the fields that are displayed below the macro list • You can also double click the macro to right click it and select **Edit** from the shortcut menu to edit the name and values in a separate editor window.

Notes

- Note that it is also possible to define **persistent Macros** via a custom Environment Profile (see **Custom Environment Profiles** - only in the Application editor (Analysis tab) or the Analysis Unit editor)
- Note that macro definitions configured here will be processed AFTER any definitions made in the Environment Profiles associated to the **IDE used to develop the Application/Analysis Unit** option (see **IDE** section above), and to any **custom Environment Profiles** (see **Custom Environment Profiles** - only in the Application editor (Analysis tab) or the Analysis Unit editor) you have added.
- If you include a file (see below) containing a list of macros, this file is included in each main file that is analyzed. In other words, and for illustrative purposes only, it would be as if each file selected for analysis started with "#include "autoinclude.h" " (where autoinclude.h is the name of the file selected in the **Includes** section below). These macros are processed AFTER the macro definitions made in this section.
- Most, if not all applications developed with Microsoft Visual C++ rely on a set of (standard or not) header files provided with the compiler (stdlib.h, windows.h...). These headers themselves heavily rely on various macros that are internally defined by the compiler. These macros are predefined via Environment Profiles that are applied automatically to the source code following the selections made in the **IDE used to develop the Application/Analysis Unit** option (see **IDE** section above), as such there is no need to explicitly add them.

Precompiled Headers (PCH)

Precompiled header	Enter the name of a header file, that the majority of the source code files in the project include first, and that self includes many other library files. Typically "stdafx.h" on most Microsoft projects.
PCH generator	A source file that includes the Precompiled header file (as referenced above), and that should be analyzed first. Typically "stdafx.cpp" on most Microsoft projects.
Disable use of PCH	Selecting this option will cause Precompiled headers to be ignored. This is generally not recommended, as PCHs improve analysis speed.

Parsing

>> allowed to close nested template list	Many compilers allow to close two nested template lists with >> (instead of ">(blank)>"). There is no reason to deactivate this option, unless "x>>y" is used in any template expression.
Accept Trigraphs	Select this option only if your code contains trigraphs. The analyzer will then interpret normalized character sequences as a single character - for example **??<** will be interpreted as **{**. Note also that this option will also force the analyzer to accept digraphs.
Standard scope of "for" loops	Variables declared inside "for" loops are not visible outside the loop. Activating this option could generate messages about unknown variables on some pre-ISO compilers, like Microsoft VC6.
Standard processing of "typename"	According to the standard, "T::X", not preceded by "typename" (where T is a template parameter) should always be interpreted as an expression. Activating this option is recommended on most compilers, except Microsoft compilers.
Detect unknown compilation directives	When this option is activated, the analyzer tries to detect compilation directives that have not been defined as macros (e.g. WINAPI and the like). Since this option is not 100% reliable, it is recommended to disable it once the analysis has been properly configured.

Custom Environment Profiles - only visible at **Application** and **Analysis Unit** level

This option enables you to select C/C++ based Environment Profiles that you have created yourself (as oppose to the Environment Profiles that are pre-defined by CAST and applied when the **IDE used to develop the Application/Analysis Unit** option is set)

Environment Profiles are a set of predefined configuration settings that can be included in an analysis. An Environment Profile can be particularly useful where you have several applications that rely on the same specific settings. By creating an Environment Profile that defines these settings, you then simply include the Environment Profile in the analysis. When the analysis is run, the settings in the Environment Profile are taken into account.

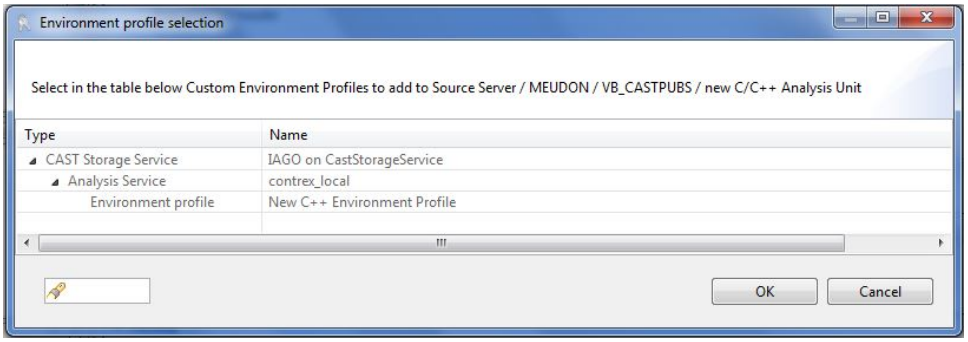
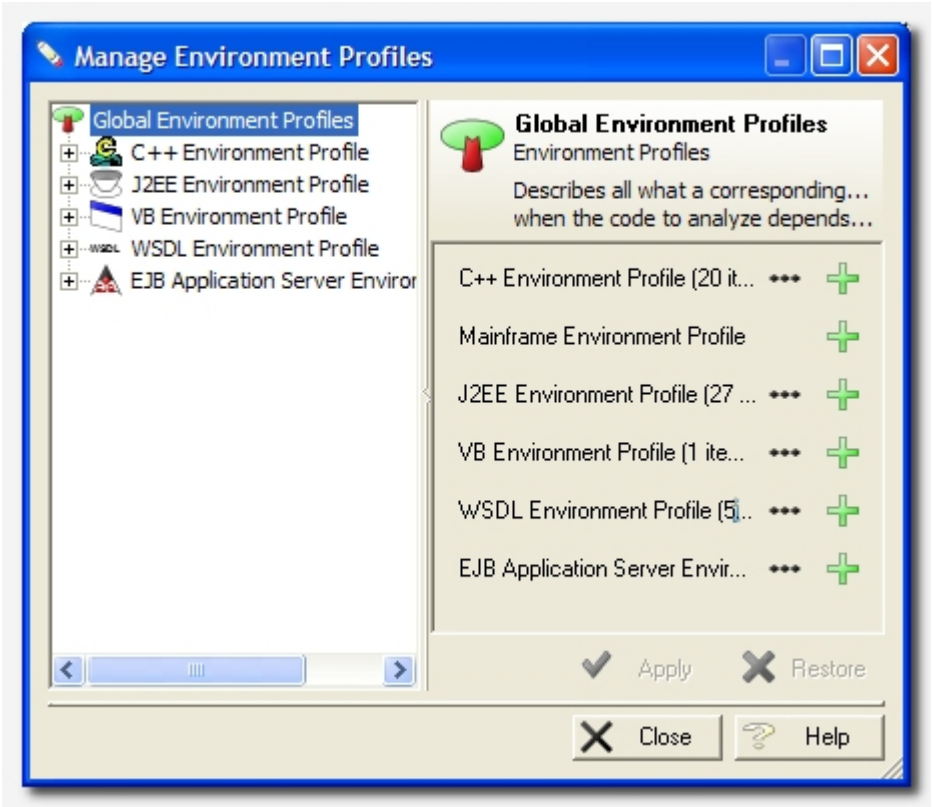
For **C/C++ technologies**, you can define:

- Include Paths
- Macros

Add a custom Environment Profile

To add a custom Environment Profile:

- click the  button
- You will be prompted to choose between adding an existing custom Environment Profile (**Add** option) or creating a new custom Environment Profile (**New** option):

Add	A dialog box will be displayed listing all existing custom Environment Profiles that have been created:  Select the Environment Profile you require and click **OK**. The profile will then appear in the list.
Manage Environment Profiles	The **Environment Profile Manager** will be displayed, enabling you to create your custom Environment Profile:  See Using the Environment Profile Manager for more information about how to create **new custom Environment Profiles**.

Remove a custom Environment Profile

To remove a profile that you do not want to include in analysis:

1. Select the item that you want to remove from the list by left-clicking with the mouse.
2. Click the **Remove** button to clear the selected item(s). The **Delete** key and the right-click short cut menu can also be used

You can remove multiple items by selecting the items in the list and then clicking the **Remove** button.

Production tab

The **Production tab** is only visible at **Technology** and **Application** level.

Process Settings

Number of Instances	This option is only relevant when the "**Legacy analyzer**" option is active. It allows you to **limit the number of objects held in memory** before they are committed to disk during the save process of an analysis. Please contact CAST Support before modifying this option.

Dependencies tab

Please see the Dependencies tab for more information about this.

Reference Pattern tab

Please see the Reference Pattern tab for more information about this.

See Also

C++ Reference Guide