Focus on Module content

On this page

Useful links

Platform administration information

Unknown macro: 'link-to'

To better yield quality and quantity information through the CAST Engineering Dashboard, the definition of Modules is critical.

Why Module content matters

The size and split of Modules within Application impact the Application assessment results and the way people access and use assessment results.

Regarding Module size:
In CAST quality and quantity model, the Module is the smallest assessment entity: one **CAN NOT** get a quality or quantity assessment for an extract of a Module, one **CAN NOT** compare over time or over the Information System extracts of Modules, one **CAN NOT** run specific quality indicators on an extract of a Module...
Furthermore, when using optional Organization Tree, the Module also represents the smallest assignment entity (and, come to that, the largest entity as well): a Developer from the Organization Tree can be assigned Modules and can not be assigned extract of a Module. Then, the quality and quantity assessment of the deliveries of a given development entity will also be aligned on Module granularity level.
Therefore, Module composition is truly impacting the information restitution. Module should reflect the smallest assessment entity one would like to handle. E.g.: the part of an application of a given technology assigned to a given development entity; the part of an application of a given technology that handle a specific layer or a specific macroscopic functionality such as logging, security, presentation, persistence, business logic;...

Regarding Module split:
The way to split an Application into Modules impacts the Application assessment results. Indeed, creating a Module based on a small population highlights the behavior of this population. In a larger Module, some objects can more easily go unnoticed as they represent a smaller part of the group.

> *The analogy would be to assess the performance of a scholar establishment by assessing the population as a whole or as a group of classes. In the former case, poor-performing students will cause a minor variation of overall percentage; in the latter case, poor-performing students can cause a whole class to turn into a poor-performing class. Unraveling such situation has pro's and con's. In this analogy, is it a good thing or a bad thing to highlight the poor performance of a single class while the whole establishment is performing well?*

This is a matter of choice. Hence the ability to define Module the way one wants. The ability to create **a Module for each functional component** of an application is the best bet.

Methodology to create a Module based on filtered Analysis Results

Once the perimeter of the Module is decided, here is the methodology to actually configure the Module:

- Based on technology object organization, determine the apex of some "object cones", that is, the branches and leaves of the object browser you want to include in the Module
  - Use Discovery Portal/Enlighten to access the object's exact identification (short/full names, path)
  - Use Discovery Portal/Enlighten to get an idea of branches and leaves
- Define the way to target these "entry points" based on their type and/or their short/full name or path
  - Type alone: all objects of a given type

        - Names/paths alone: all objects whose name/path matches a given pattern
        - A combination of these
    - Once the "entry points" targeted, all sub-branches and leaves are automatically included into the Module
        - In case you need to exclude a sub-branch or a leave, you must define lower-grain "entry points"

Module "object cones"

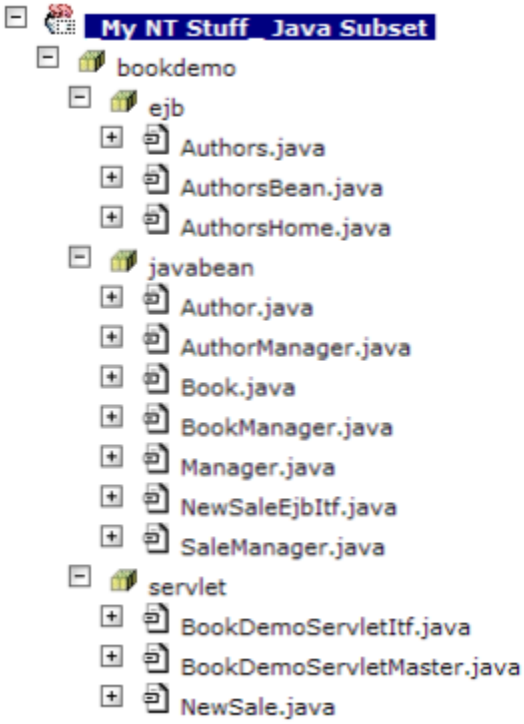
Module "object cone" concept is **critical** to master for accurate Module composition.
This concept lets you target objects by their [full] names and / or types to include **ALL THEIR INCLUDED OBJECTS** in the Module.
E.g.: target a Java Class to include the Class as well as all its Methods in a Module.

Once the apex of the "object cone" is targeted, **filters on [full] names and / or types are not applicable any more**.
That is, an "object cone" can contain objects with names that do not match the filtering criterion on [full] name or can contain objects of different types. As long as the "object cone" apex is a match, children objects are included.

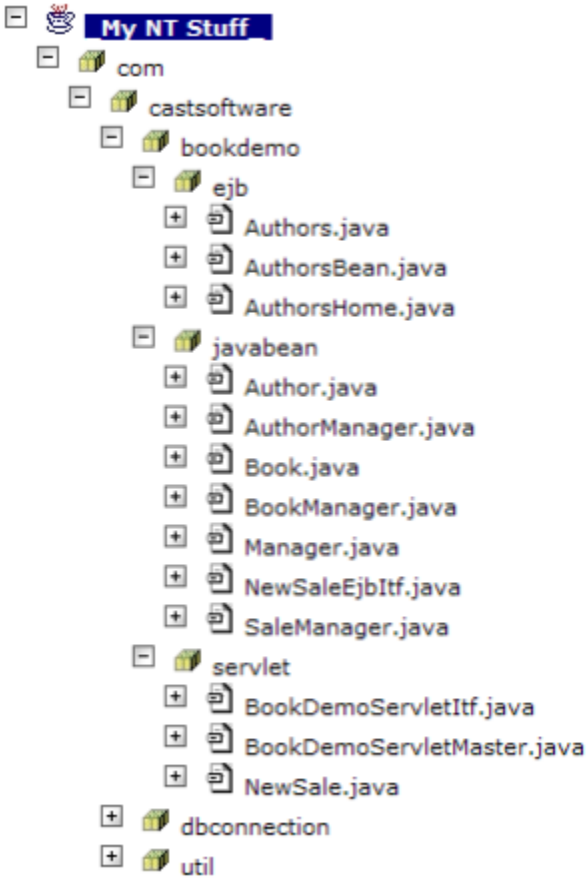
Example #1: create a Module based on a Java package

The following configuration:
Full Name Like com.castsoftware.bookdemo
Selected Types: Java Package

Leads to a Module composed of:



While the full analysis result is:



Explanation:
The filters on both the object type ('Java Package') and the object full name ('like com.castosftware.bookdemo') targeted the actual 'com.castsoftware.bookdemo' Java package; the Module was therefore build with this package and its content (that is, sub-packages, classes, methods and fields).

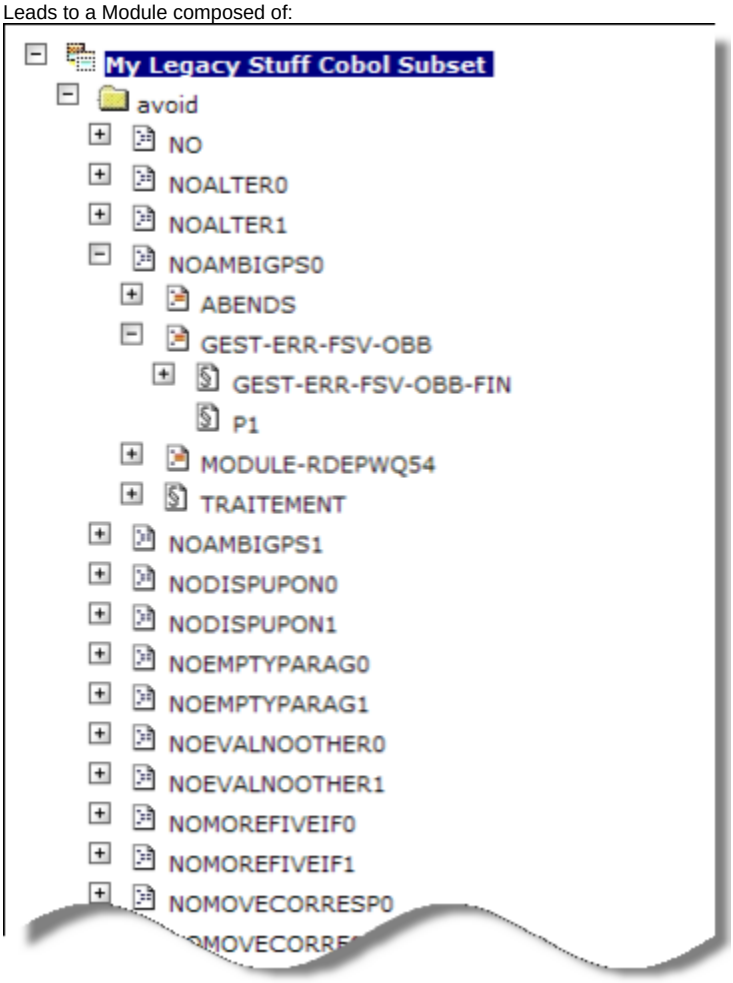
Use case:
Useful to handle functional area of an application based on the package organization.

> Note that when using the **Object Name** filter based on an **object's path** (as stored in the Analysis Service schema), some objects (notably Java Packages) are not saved with a path, therefore these objects will not be included in the module.

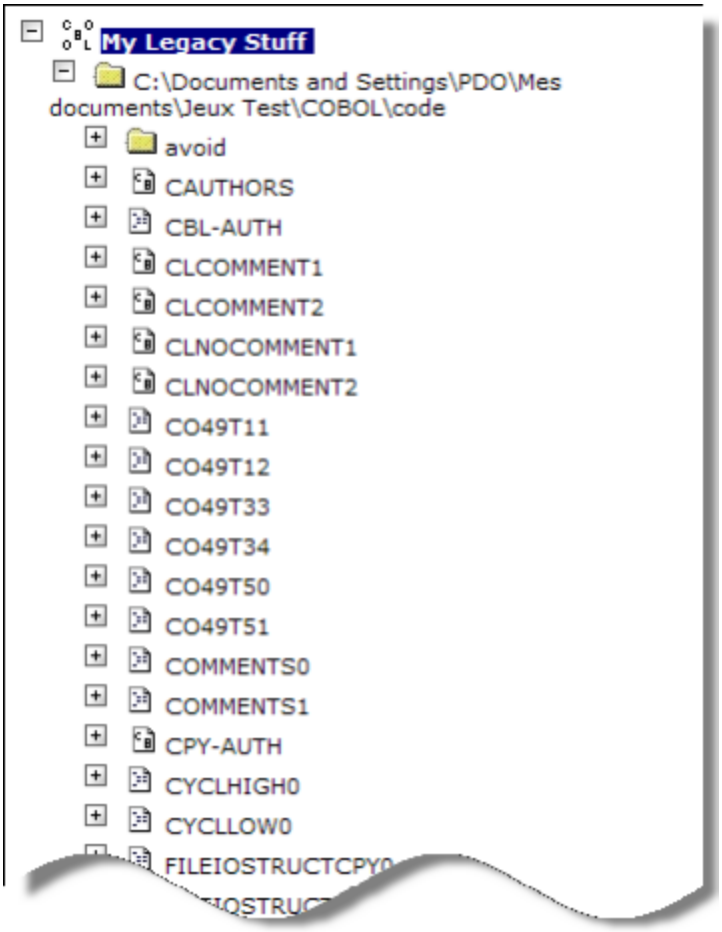
Example #2: create a COBOL Module based on a directory

The following configuration:
Full Name Like c:\Documents and Settings\PDO\Mes documents\Jeux Test\COBOL\code\avoid
Selected Types: COBOL Directory

Leads to a Module composed of:



While the full analysis result is:



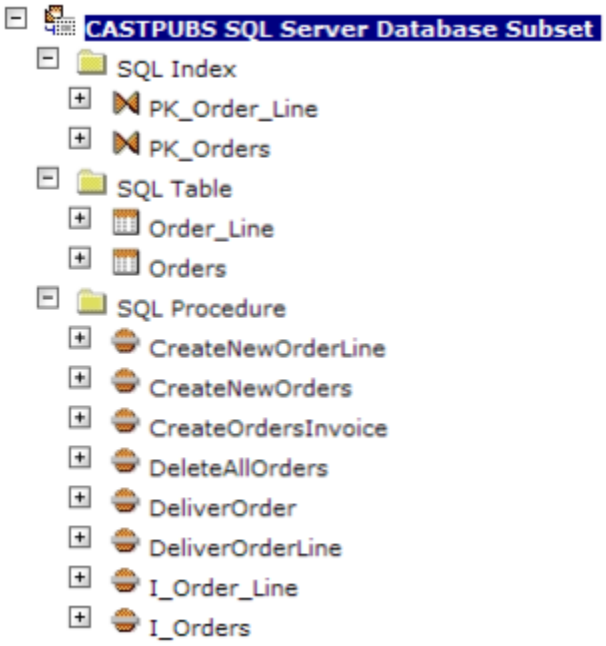
Explanation:
The filters on both the object type ('Cobol Directory') and the object path ('like C:\Documents and Settings\PDO\Mes documents\Jeux Test\COBOL\code\avoid\') targeted the actual Cobol Directory 'C:\Documents and Settings\PDO\Mes documents\Jeux Test\COBOL\code\avoid\' directory; the Module was therefore build with this directory and its content (programs, copybooks, sections, paragraphs...).

Use case:
Useful to handle generated code when such source code is stored in a separate directory.

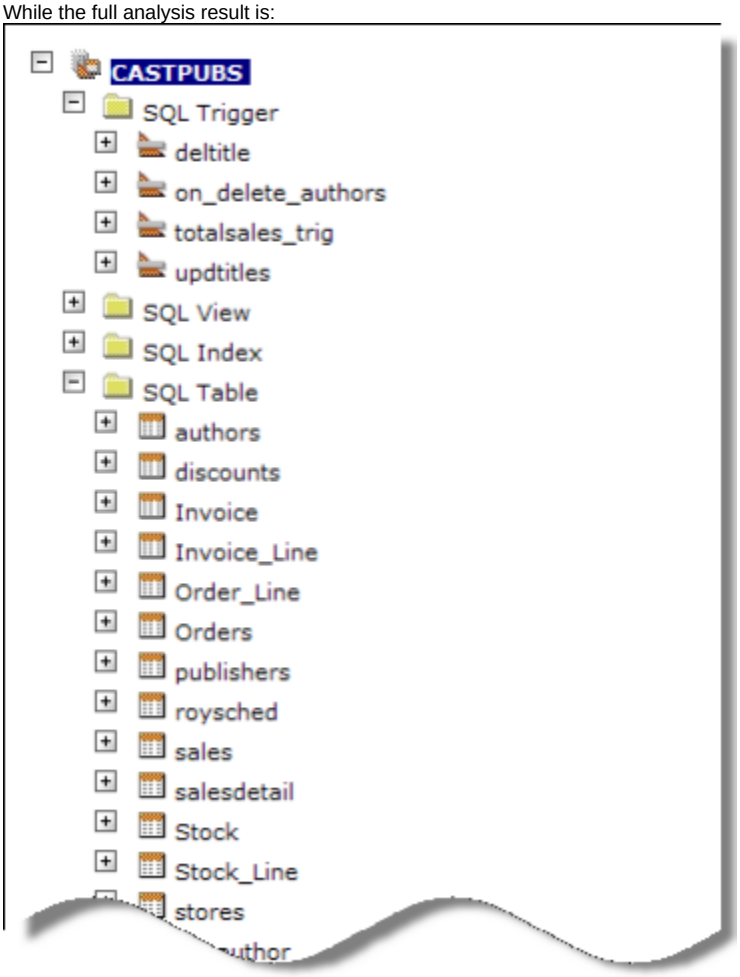
Example #3: create a DB Module based on object names

The following configuration:
Full Name Like %Order%

Leads to a Module composed of:



While the full analysis result is:



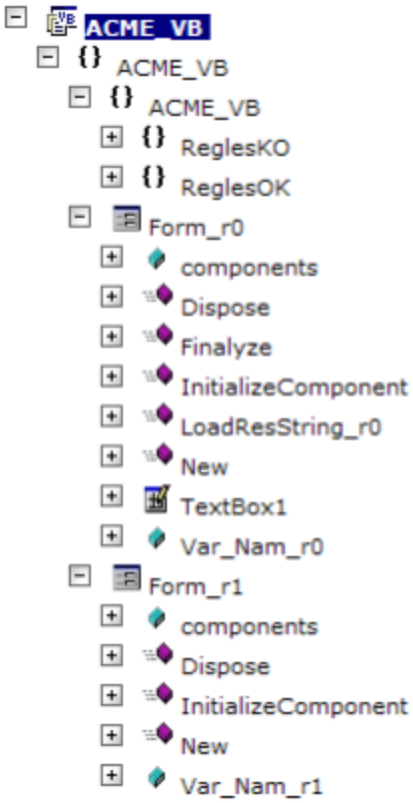
Explanation:
The filters on the full name ('like %Order%') targeted the actual SQL objects whose name contains 'Order'; the Module was therefore build with these SQL objects and their content (N/A with these types of objects).

Use case:
Useful to handle developments on top of packaged applications where the name alone can help discriminate objects.

Example #4: create a .NET Module based on .NET projects

The following configuration:
Full Name Like [ACME_VB]

Leads to a Module composed of:



While the full analysis result is composed of ACME_VB, ACME_CS, and IBM-T42.CASTPUBS used by My .NET Stuff.

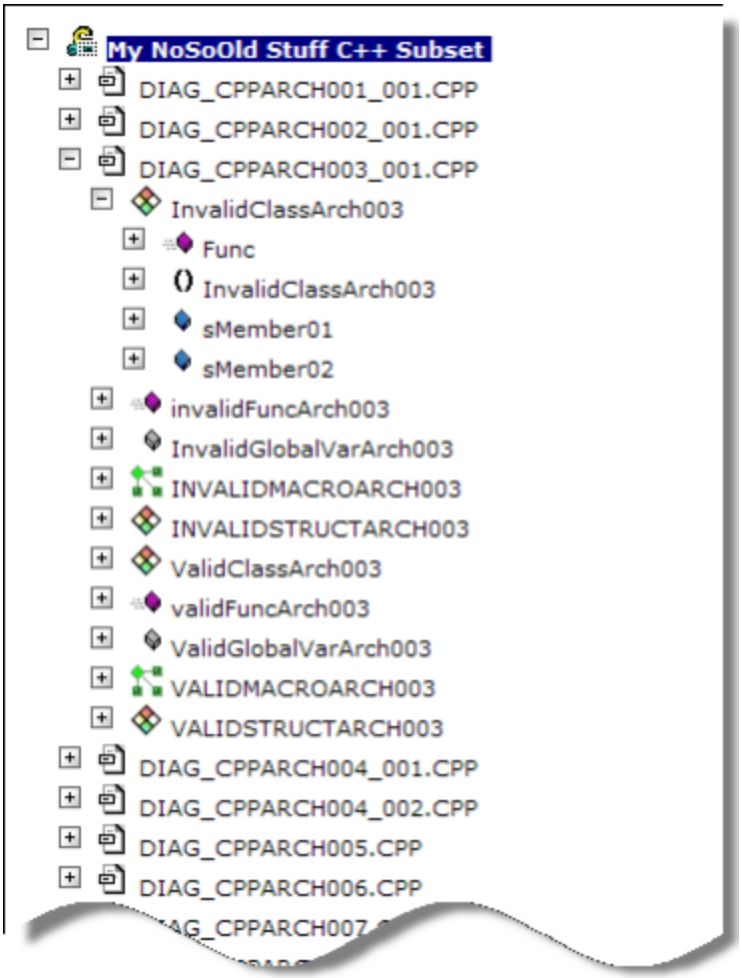
Explanation:
The filters on the full name ('like [ACME_VB]') targeted the actual .NET project name '[ACME_VB]'; the Module was therefore build with this project and its content (namespaces, forms, ...).

Use case:
Useful to handle projects within a VB/VB.NET/.NET solution.

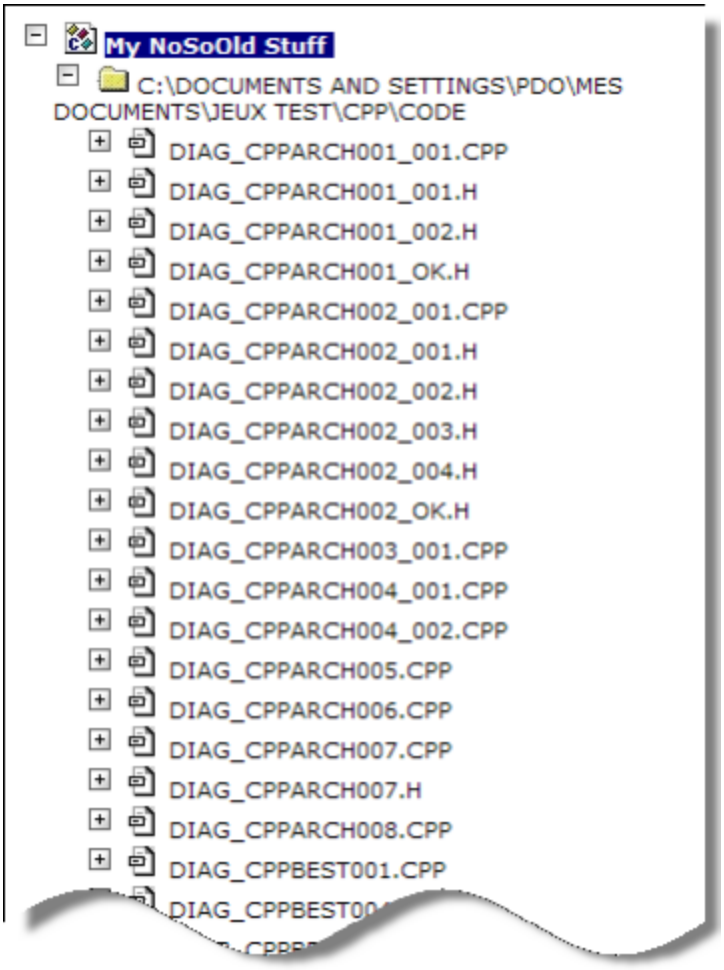
Example #5: create a Module based on .cpp files

The following configuration:
Full Name Like %.cpp

Leads to a Module composed of:



While the full analysis result is composed of:



Explanation:
The filters on both the full name ('like %.cpp') targeted the actual C++ implementation files and left aside the header files; the Module was therefore build with these files and their content (classes, methods,...).

Use case:
Useful to avoid assessment pollution of batches of header files. Note the directory selection can also be used, depending on the source code organization.

Example #6: create a Module based on SAP Tables

The following configuration:
Full Name Like SAP_TABLE/%

Leads to a Module composed of all SAP Tables from the SAP analysis while the full analysis also contains programs, etc.

Explanation:
The filters on the full name ('like SAP_TABLE/%') targeted all objects whose full name starts with 'SAP_TABLE/' which is the naming convention within the AIP of analyzed SAP tables.

Use case:
This example is valuable with a refined full name to build modules containing some of the tables, according to a naming convention.

Example #7: create a Module based on a .Net namespace

The following configuration:
Full Name Like n1
Selected Types: .Net namespace

Explanation:
The filters on both the object type ('.Net namespace') and the object full name ('like n1') targeted the actual 'n1' .Net namespace; the Module was therefore build with this namespace and its content (that is, sub-namespaces, classes, methods and fields).

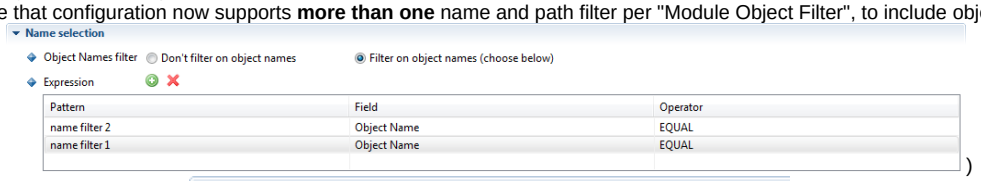
In order to distribute source files between modules, the Module "build by namespace" requires that you create filters to distribute them between Modules. As they can contain objects from multiple namespaces, they are not "belonging to" any namespace. For instance, add a filter on "full name like %xyz%.cs"

Use case:
Useful to handle functional area of an application based on the package organization.

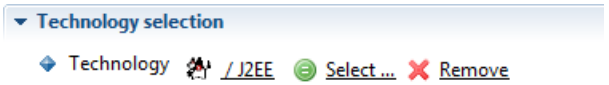
More...

In addition to the capabilities to build Modules by targeting the apex of "object cones" by their type and full name, the CAST Management Studio also support:

- filtering on object names and path
  (please note that configuration now supports **more than one** name and path filter per "Module Object Filter", to include objects **matching one of them (OR)**)

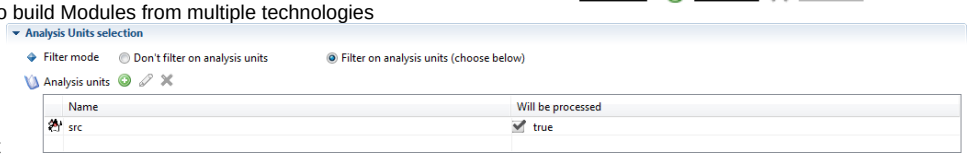


- filtering on any given technology

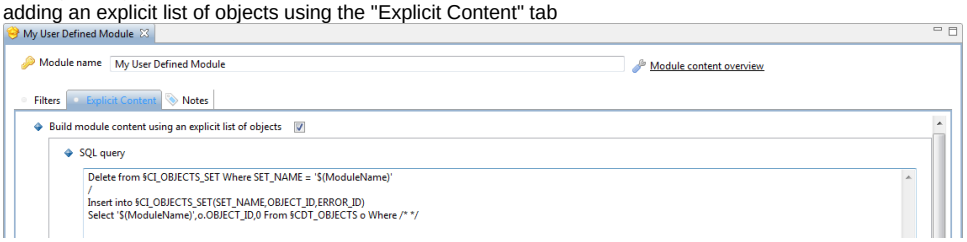


(please leave it blank if you want to build Modules from multiple technologies)

- filtering on any given Analysis Unit



- adding an explicit list of objects using the "Explicit Content" tab



In case of doubts on the results of the filtering options, "Module Content Overview" tool is available in the Module editor within CAST Management Studio User Interface. Please note that you have to have up-to-date analysis results in the Analysis Service so that the feature delivers the best results.

About database Subsets

Background information

Database Subsets of a Module are sets of server-side objects that are directly - or indirectly - called by the client-side objects from the concerned Module.

They are helpful when one wants to assess the quality and measure the quantity of the server-side code actually used by the server-side code (frequently, database contain the server-side code for multiple Applications or some leftover server-side code that shouldn't be part of the assessment and measurement results).

They are also helpful when one wants to assess the quality of an entire functional component, including both client-side and server-side code.

If you want to include the whole server-code and to create a dedicated Module for the server-side code, you can decide to not generate and deal with Database Subsets as described below.

In AIP 7.x (x1)

Last minute information

In release 7.x (x1), the database Subsets can be generated using the "Database subset" option in the "Module" tab of the "Application" editor, however, they are not included in the Modules that contain the client code.

To overcome this, please use the "Explicit content" capability to enrich target Modules with the appropriate database Subsets.

To include the content of the database Subsets related to client code from a Module, please use the following query in the "Explicit content" tab of the "Module" editor:

SQL Query for Module explicit content configuration

```
Delete from §CI_OBJECTS_SET Where SET_NAME = '$(ModuleName)'
/
Insert into §CI_OBJECTS_SET(SET_NAME,OBJECT_ID,ERROR_ID)
Select distinct  '$(ModuleName)',op.IdObj,0
From §Keys k , §SETROOT sr , §AnaProSet aps, §AnaProSet apss , §ObjPro op
Where k.KeyNam = '$(ModuleName)'
And k.ObjTyp = 20000
And k.IdKey = sr.IdSet
And sr.IdRoot = aps.IdPro
And aps.IdJob = apss.IdJob
And Not Exists (Select Distinct 1 From §SETROOT srs Where srs.IdRoot = apss.IdPro)
And op.IdPro = apss.IdPro
/
```

Do not forget to check the injection status of "Explicit content":

SQL Query for Module explicit content injection status

```
Select cos.SET_NAME, cos.OBJECT_ID, k.KeyNam, cos.ERROR_ID
  From CI_OBJECTS_SET cos, Keys k
  Where cos.OBJECT_ID = k.IdKey
  Order By ERROR_ID Desc
```

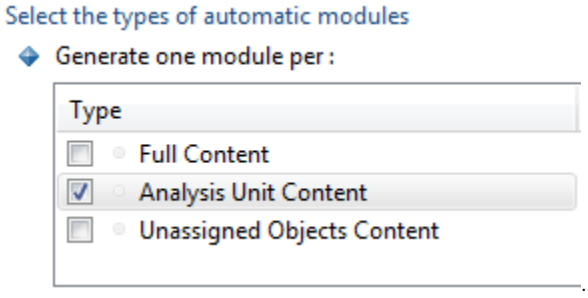
Except for database Subset objects themselves which should show with an ERROR_ID of 5003, all other Objects should have an ERROR_ID equal to 0.

Methodology to automatically create some specific Modules

Although strongly recommended to create your own Modules that will add value to the assessment results, there are ways to create some specific Modules to ease your task:

- create a Module with the content of the whole Application: default strategy to enable you to generate a Snaphost on an Application as soon as the Application is assigned some Source Repository, yet does not add value to the assessment results
- create a Module with the content of each Analysis Unit: optional strategy to enable fast configuration with technologies that lead to multiple Analysis Units with quite-meaningful names (e.g.: one Analysis Unit per .NET C# project)
- create a Module with unassigned objects: optional strategy to be sure that, snapshot after snapshot, there are no major addition of source code that would go "unnoticed", that is, that would not be part of existing Modules; as soon as this specific Module appears, this means there is a need for Module configuration update to deal with the new source code.

To do so, you simply need to check the appropriate option in the Module tab of the Application editor (Advanced audience required):



Note that these are on/off options that can lead to cumulating the "My Application full content" Module, with one Module per Analysis Unit, with the user-defined Module, hence creating a situation with **shared objects**.

Some situation may require this configuration, while other may not. E.g.: have both the "My Application full content" Module, with one Module per Analysis Unit to have the grades of each Analysis Units and of the whole Application content (without the "My Application full content" Module, you would get the Application grade as the weighted average of Analysis Units' grades).