Start AD Usage functions from IDz editors

Currently, a developer has to copy the variable or section name and change the perspective (Application Discovery Browser) in order to then select all AD projects. Then the copied name must be pasted and the analysis "Use in program" started.

We believe that we can achieve a higher level of use of ADDI in IDz if the paths from the search query to the result can be done in a short and intuitive manner.
A developer is mainly in his source, which is opened in a language-specific editor (Cobol, PLI, ASM, JCL, Java, LPEX editor ...). They need important information during implementation, especially when it comes to incompatible changes to copies or subroutines. There is important information that calls this source point so that the developer can adapt it himself or inform the corresponding developer.

The result is prepared in detail in a tree table.
A faster navigation to the result would be given if the search can be started by selecting the name in the editor via the context menu.

Important functions for the everyday work would be:

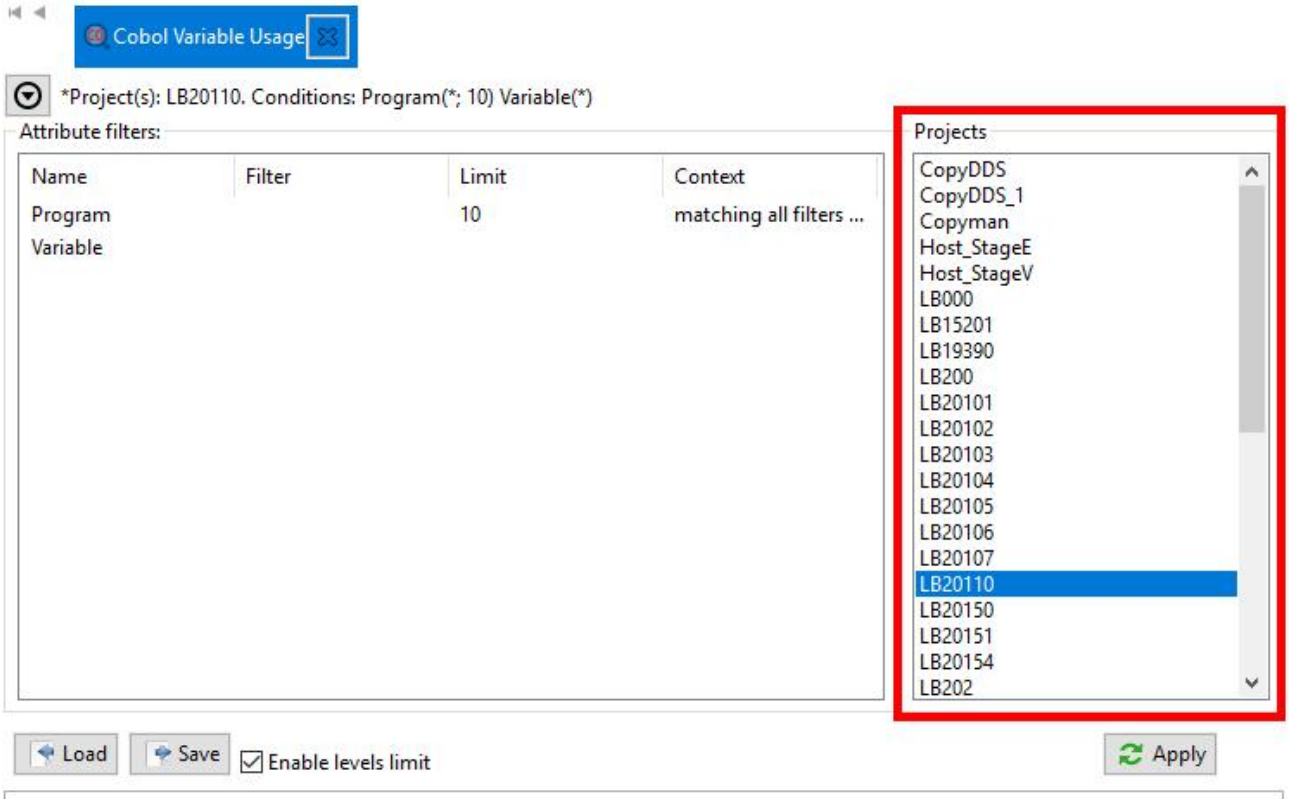
- Program Callgraph
- Variable Usage
- Include Usage
- SQL Table Usage
- SQL Table Field Usage
- Program Usage in Jobs

All functions should be executed according to the extension. In order for the developer to be able to continue working, the Usage View should automatically be displayed / opened

The selection of AD projects should:

a) should appear in a separate dialog if no further dialog input is required
   ⇨ Program Callgraph

b) should be integrated in the usage dialog so that there is no dialog sequence. Integration can take place via the wizard function or directly in the input area
   ⇨ Variable Usage with Program, Value and Statement Type
   ⇨ Include Usage with Statement Type and Program
   ⇨ SQL Table Usage with Statement Type
   ⇨ SQL Table Field Usage with SQL Table, Program, Statement
   ⇨ Pgm Usage in Jobs with Step, DD Name, Dataset, Program



The selection of the AD projects should be cached after changes by the user so that the selection does not have to be carried out again in subsequent analyzes

If no clear mapping between function and selection is possible, then the functions mentioned above should be visible in the context menu by default. Thus, a cross-language analysis is possible.

```
19
20 public class BS8906MapperAuszahlungsdatenStellvertreter extends Bs8906
21 {
22
23     private static final String BSNAME = "BS8906"
24
25     public BS8906MapperAuszahlungsdatenStellvertreter()
26     {
27         super();
28
29         setBSNAME(BSNAME);
30     }
31 */
1 public class TSErzeugeOnlinezugang extends AbstractKuOTransactionService
2 {
3
4     /**
5      * Kurzbeschreibung
6      * Transaktion  => BA250T
7      * Dispacher    => BA248U
8      * Servicemodul => BA497U
9      * Steuer       => 3002
0      */

    @Override
    protected String getTransaktionsCode()
    {
        return "BA250N1A";
    }
```

Result of the analyzes
A better representation option for the respective analysis result would be a tree table. The information from the call diagram analysis is presented in the tree table by presenting uses and references by opening the nodes. Different symbols indicate users and users.
The tree table would also be used in other analyzes to display a large amount of information from the properties view in the individual lines. As it is already done in the search function in the IDz (supplemented by the lines).