Advanced - Mixing a Universal Analyzer job and a Universal Importer job

Preamble

Consider the following:

- The **Universal Analyzer** analyzes unsupported (by a standard CAST Analyzer) source code files according to the "instructions" defined in a specific Language Pack. The results are stored in the CAST Analysis Service and can be exploited by relevant CAST Products (CAST Engineering Dashboard, CAST Enlighten etc.)
- The **Universal Importer** simply injects information about a language's objects, links and properties stored in a specific format in XML based files (.UAX) into the CAST Analysis Service. These objects, links and properties can be exploited by relevant CAST Products (CAST Engineering Dashboard, CAST Enlighten etc.). As with the Universal Analyzer, a specific Language Pack for the language that is being injected must already be registered in the CAST Analysis Service.

Mixing jobs

Now take a situation where you have files (XML based files (.UAX) resulting from an external analyzer or other system or processor) that contain objects, properties and links **in addition to** references to source code files or actual source code for some of the objects. This situation may be evident in an ERP where an existing repository may already contain a hierarchy of objects and their properties as well as the source code.

In previous versions of CAST, both the Universal Analyzer (by analyzing the source code or references to the source code in the .UAX files) and the Universal Importer (by injecting the objects, links and properties defined in the .UAX files) were used together to "import" the data into the CAST Analysis Service for further exploitation.

Now, however, this is no longer the case and you can simply use the Universal Importer on its own - it will handle both the analysis of the source code (via the Universal Analyzer) or references to the source code in the .UAX files and the injection of objects, links and properties defined in the .UAX files.

If you need to do this, these tips will help:

- You must register a **Language Pack** for the target language in the CAST Analysis Service, prior to using the Universal Importer. See the Creating or modifying a custom extension using the CAST Universal Analyzer framework for more information about creating a Language Pack.
- You should then use the **Universal Importer** to analyze the source code or references to the source code in the .UAX files and then inject the existing objects, links and properties contained in the .UAX files
- You should ensure that:
  - You include the attribute: **<CAST_CodeExpandable sourcePath="..." />** in the .UAX file to define the location (**relative** or **absoute** - see below for more information) of the of the source code
  - You include the attribute: **<CAST_UA_CodeExpandable technologies="..." />** in the .UAX file to define the technology of the source code that will be analyzed.

If you still use both the Universal Analyzer and the Universal Importer in "combined mode" (rather than using just the Universal Importer as explained above) you may find that the CAST Engineering Dashboard will report an incorrect number of **Total Checks.** The number of Total Checks for a specific measure will be higher than the actual number of objects that really exist. This is because when you use the Universal Analyzer and the Universal Importer in "combined mode", objects are attached to two sets of projects at the same time (Universal Analyzer and the Universal Importer) and are therefore counted more than once. CAST has provided a workaround to resolved this issue. More information can be found in Advanced - Resolving erroneous number of Total Checks reported by the CAST Engineering Dashboard.

Example

Take the following example for analyzing and injecting an object called "PageEvent.1" into the CAST Analysis Service using the Universal Importer:

```
<instance id="PageEvent.1" instanceOf="PSoft_PageEvent">
  <identification name="ACL_COMPONENT2.Activate" fullName="ACL_COMPONENT2.Activate"/>
  <CAST_CodeExpandable sourcePath="PageEvent_ACL_COMPONENT2.Activate.src.PSoft_PageEvent.ACL_COMPONENT2..
Activate"/>
  <CAST_UA_CodeExpandable technologies="PeopleSoft"/>
</instance>
```

In this situation, the file that will be analyzed by the Universal Importer and that contains the object **ACL_COMPONENT2.Activate** must be in the following location - this is using a **Relative** path (CAST recommends using relative paths to avoid potential problems):

```
PageEvent_ACL_COMPONENT2.Activate.src.
```

Below is a copy of the file containing the source code of the object **ACL_COMPONENT2.Activate**:

```
CAST_EXPORTED_PEOPLECODE_PAGE_BEGIN
NAME=ACL_COMPONENT2.Activate
FULLNAME=ACL_COMPONENT2.Activate
BEGIN_PEOPLECODE
Local Grid &grid;
grid = GetGrid(Panel.ACL_COMPONENT2, 'ACLCOMPONENT_V2');
&column = &grid.GetColumn('VIEW_CREF');
&column.Label = MsgGetText(48, 274, 'Message Not Found');
END_PEOPLECODE
CAST_EXPORTED_PEOPLECODE_END
```

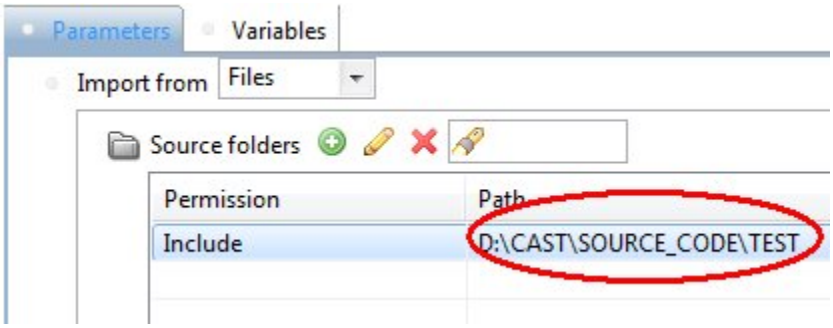
Absolute/Relative path of source code

To define the location of the source code files in your .UAX files, you can use either a **relative path** or an **absolute path**:

Relative	<CAST_CodeExpandable sourcePath="PageEvent_ACL_COMPONENT2.Activate.src.PSoft_PageEvent.ACL_COMPONENT2..Activate"/>
Absolute	<CAST_CodeExpandable sourcePath="P:\PageEvent\PageEvent_ACL_COMPONENT2.Activate.src.PSoft_PageEvent.ACL_COMPONENT2..Activate"/>

CAST strongly recommends that you use relative paths to avoid any potential problems linked to the location of the source code

When using **Relative** paths in your .UAX files,ensure that the path you enter in the Universal Importer tool is in fact the root path:



This path will then be used to determine the location of source code files specified in the .UAX files. In the above code examples, you would need to ensure the **Root Path** as **P:\PageEvent**