xxxCastMetrics.xml - defining new metrics for a UA language

CAST AIP provides the possibility to extend the standard set of metrics provided by default. This extension of the "out of the box" configuration can be defined using an XML file called **XXXCastMetrics.xml**. For each technology on which you want to define custom metrics, one corresponding file must be created.

Custom metrics can be defined for:

- standard **CAST supported technologies**
- **custom technologies** configured via the **CAST Universal Analyzer**

Custom metrics are grep based metrics that use regular expressions to generate the required data. They are not based on the technology's grammar - if, indeed there is any.

Please note that:

- The legacy CastMetrics.xml file that is stored in the CAST installation folder is no longer used during an analysis/snapshot generation. If you have customized this file in the past, its content must be moved to one or several XXXCastMetrics.xml files. It is highly recommended to have one file per custom language or one file for custom metrics for a standard language.
- You do not need to use the **Load MetaModel from disk** option in CAST Server Manager - as long as the XXXCastMetrics.xml file is present when the analysis is run/snapshot is generated, then it will be taken into account.

File naming conventions

The XXX in the name of the XML file signifies that you are free to choose how you name the file provided that you retain **CastMetrics.xml** as the end part of the filename. CAST suggests using the **name of the technology as a prefix** to CastMetrics.xml but this is not obligatory. In addition you are not limited to three characters. You could, for example, use a file called CASTXSLCastMetrics.xml.

Ensuring the XXXCastMetrics.xml file is taken into account

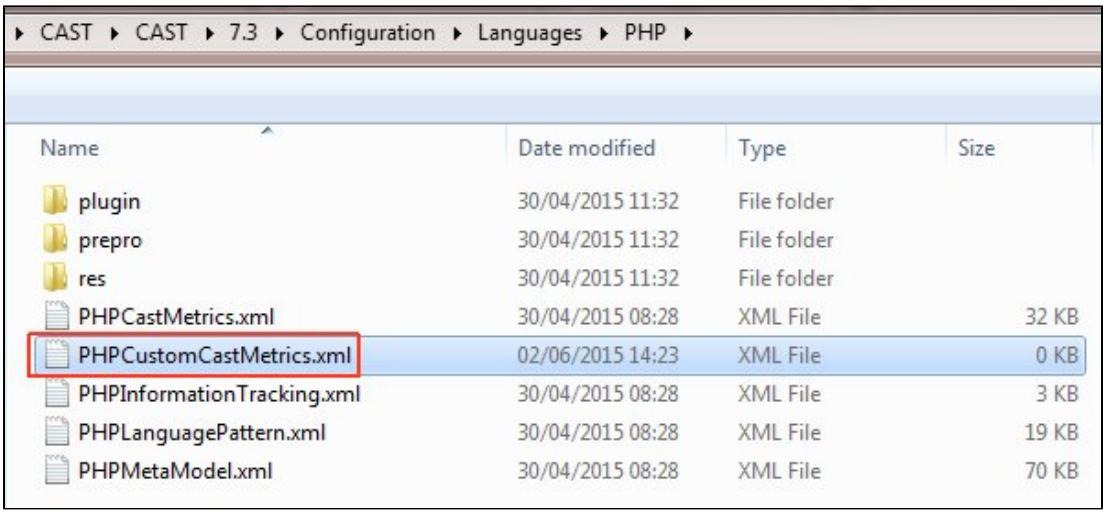
Step 1 - File location

The XXXCastMetrics.xml file must be manually copied to the following location so that it is taken into account during the analysis:

```
%PROGRAMDATA%\CAST\CAST\<version>\Configuration\Languages\<technology folder name>\XXXCastMetrics.xml
```

When extending metrics for technologies analyzed by a custom UA language pack

- When extending metrics for technologies that are analyzed by a custom UA language pack, the **<technology folder name>** folder will already exist. It is the same folder as is used to store the XXXMetaModel.xml file and the XXXLanguagePattern.xml files used in the language pack.
- In addition, the **XXXCastMetrics.xml** file **may already exist** in this folder - if this is the case, CAST recommends creating an **additional** XXXCastMetrics.xml file to configure your bespoke metrics to avoid modifying the XXXCastMetrics.xml file configured for the language pack. Take the example of this PHP Language Pack - this already has a **PHPCastMetrics.xml** file and an additional **PHPCustomCastMetrics.xml** has been created for bespoke metrics:

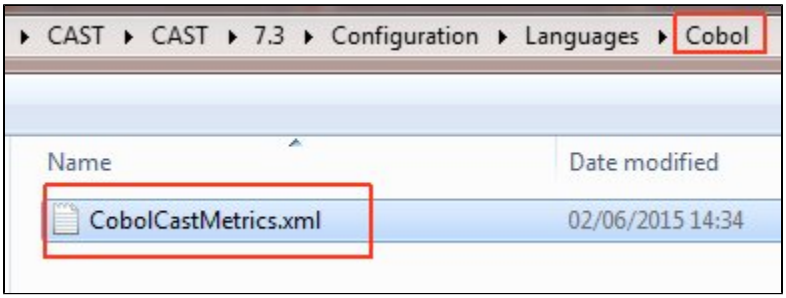


When extending metrics for technologies analyzed by a standard CAST AIP analyzer

- When extending metrics for technologies such as Cobol, VB or other technologies supported by a standard CAST AIP analyzer, you will need to create the **<technology folder name>** folder yourself. You can find out what name to use by using the same folder name as is used in AIP - navigate to the following location to see a list of core AIP technology names:

```
<CAST_AIP_installation_folder>\Configuration\Languages\
```

- Take the example of extending the Cobol technology - the **CobolCastMetrics.xml** file should located in a folder called **Cobol**:



Step 2 - Install the <all users dir> extension with CAST Server Manager

You now need to run **CAST Server Manager** > **Manage Extensions** to install the **<all users dir>** extension. This will ensure that the custom **XXXCastMetrics.xml** is installed correctly. This is explained in more detail here:

- With new CAST schemas
- With existing CAST schemas

> This step must be completed when creating custom metrics for both custom UA language packs AND core AIP technologies.

XML file encoding

The XXXCASTMetrics.xml file used in the Language Package must use the **UTF-8** encoding as follows:

- The **XML declaration** (the first line of the file) must include the UTF-8 encoding attribute:

```
<?xml version="1.0" encoding="utf-8" ?>
```

- The XML file must be **saved** in **UTF-8** format

Structure

General Structure of the file XXXCastMetrics.xml

An example XXXCastMetrics.xml file is shown below:

```
<METRIC_LIST>
   <METRIC Name="<Name of the metric>" TYPE="<INF SUB TYP>">
      <LANGUAGE NAME="AAA-LANGUAGE">
         <!-- description of the metric -->
      </LANGUAGE>
   </METRIC>
   <METRIC Name="<Name of the another metric>" TYPE="<INF SUB TYP>">
      <LANGUAGE NAME="AAA-LANGUAGE">
         <!-- Description of the metric -->
      </LANGUAGE>
   </METRIC>
</METRIC_LIST>
```

Each <METRIC> tag defines the name and the type of the metric to be created. The type is the ID of the metric, also called InfSubTyp in the CAST Analysis Service.

The name and the ID of the metric must be unique for all the metrics in the CAST Analysis Service (please refer to the Configuration\Metrics\Configuration\MetricsName.xml located in your CAST installation folder for more information about IDs)

Each metric must be declared for each technology we want to make it available for. In the example below the metric "**Use of printf in a do/while**" has the ID 3000000 and is available for C++.

```
   <METRIC Name="Use of printf in a do/while" TYPE=3000000>
      <LANGUAGE Name="C-LANGUAGE">
         <!-- Description of the metric for  C/C++ -->
      </LANGUAGE>
</METRIC>
```

Note that the ID of a User Metric must be higher than 2000000.

Language Name

Each metric must contain a <LANGUAGE NAME> tag. For example:

```
<LANGUAGE NAME="C-LANGUAGE">
```

Please use one of the following:

- ASP-LANGUAGE
- ASPX-LANGUAGE
- ABAP-LANGUAGE
- ANSISQL-LANGUAGE
- ASETSQL-LANGUAGE
- BO-LANGUAGE
- C-LANGUAGE
- COBOL-LANGUAGE
- C-SHARP-LANGUAGE
- DB2-LANGUAGE
- DB2ZOS-LANGUAGE
- PROPFORMS-LANGUAGE
- FORMS-LANGUAGE
- HTML-LANGUAGE
- IMS-LANGUAGE
- JSP-LANGUAGE
- JAVA-LANGUAGE
- JCL-LANGUAGE
- JAVA-SCRIPT-LANGUAGE
- MSTSQL-LANGUAGE
- PB-LANGUAGE
- VB-LANGUAGE
- VB-SCRIPT-LANGUAGE
- VB-DOTNET-LANGUAGE

- UNIVERSAL-LANGUAGE

Configuring the Search Mode

In order to optimize the search of your code, there are several tags that can be activated:

- For the search in the source code use the tag : <SEARCH_IN_CODE>
- For the search in comments use the tag <SEARCH_IN_COMMENT>
- For the search in strings use the tag <SEARCH_IN_STRING>
- For the search in embedded SQL use the tag <SEARCH_IN_EMBEDDEDSQL>
- For a case sensitive search use the tag <SEARCH_CASE_SENSITIVE>
- If we need to match the whole word only, then use the tag <MATCH_WHOLE_WORD_ONLY>

Search section and Regular expressions

There are two methods that can be used to search your code:

- Simple search
- Nested search

The tag <INIT> provide the possibility of the initialise the value of the metric for which the pattern matched. For example, if the metric contains:

```
<INIT>3</INIT>
```

this means that the value 3 will be added to every value of the metric pushed to the CAST Analysis Service. There is no initialization of the metrics for the objects that don't match.

Simple Search

The simple search is based on one or several regular expressions (RegExp). The search has succeeded if one or several regular expressions have matched.

Nested Search

The type of search is recommended for the following patterns:

```
BEGIN
  CONTENT
END
```

For example:

```
...
   <EMBEDDED>
      <BEGIN>
         <REGEXP> RegExp for the begining of the Pattern </REGEXP>
      </BEGIN>
      <VALUE>
         <REGEXP> RegExp for the content of the pattern</REGEXP>
      </VALUE>
      <END>
         <REGEXP> RegExp for the END of the Pattern </REGEXP>
      </END>
   </EMBEDDED>
...
```

Extra options

<THRESHOLD>

The option <THRESHOLD> provides the possibility to define a threshold on the number of matches for a given object. If this value is lower that the threshold then it is ignored. If this is not the case the value "1" is set for this metric in the CAST Analysis Service.

For example, consider the following metric:

```
...
   <EMBEDDED>
      <THRESHOLD>5</THRESHOLD>
      <BEGIN>
         <REGEXP>do</REGEXP>
      </BEGIN>
      <VALUE>
         <REGEXP>printf</REGEXP>
      </VALUE>
      <END>
         <REGEXP>while</REGEXP>
      </END>
   </EMBEDDED>
...
```

on the following C++ source code: **TestThreshold.cpp**

```
void testThreshold()
{
  printf("Out of the loop");
  do
  {
     printf("Hello World!");
     printf("Hello World!");
     printf("Hello World!");
     printf("Hello World!");
  }
  while(true);
}
```

The metric will match four times (one match for every **printf** present in the do/while). However, as the threshold is set to 5, no value will be saved in the CAST Analysis Service. If the threshold had been set lower at 4, the value 1 would have been saved.

<ADD>

The tag <ADD> provides the possibility to force the metric to count the nested level of the matched pattern. The value defined here will be added to the metric value for every nested BEGIN/END statement.

Example1: Consider the following metric:

```
...
   <EMBEDDED>
      <ADD>2</ADD>
      <BEGIN>
         <REGEXP>do</REGEXP>
      </BEGIN>
      <VALUE>
         <REGEXP>printf</REGEXP>
      </VALUE>
      <END>
         <REGEXP>while</REGEXP>
      </END>
   </EMBEDDED>
...
```

For the following C++ source code**: TestAdd.cpp**

```
void testAdd()
{
  printf("Out of the loop");
  do
  {
    do
    {
      do
      {
        printf("Hello World!");
      }
      while(true);
    }
    while(true);
  }
  while(true);
}
```

For the example above, the metric will find one match for BEGIN/CONTENT/END and six matched for the three do/while nested loops. As such, the value 7 will be saved in the CAST Analysis Service.

Example 2: Consider metric A, defined with (simplified syntax):

```
INIT=5
ADD=2
BEGIN=AAA
VALUE=ZZZ
END=BBB
```

And metric B with:

```
INIT=5
ADD=0
BEGIN=AAA
VALUE=ZZZ
END=BBB
```

Then for the following code:

```
AAA
   ZZZ
BBB
```

- metric A will be valued 5 + 2 + 1 = 8 (i.e. INIT + ADD + Number of MATCHES)
- metric B will be valued 5 + 1 = 6 (i.e. INIT + Number of MATCHES)

While for the following code:

```
AAA
   AAA
      ZZZ
   BBB
BBB
```

- metric A will have 5 + 2 + 2 + 1 = 10 (i.e. INIT + 2 x ADD + Number of MATCHES)
- metric B will have 5 + 1 = 6 (i.e. INIT + 2 x ADD + Number of MATCHES)