Tracing Framework

Developer Guide

TABLE OF CONTENTS

1. Tracing Framework

The CIMOM infrastructure components will use the Tracing framework to provide trace messages that will be helpful in investigating a problem cause.

Tracing must be implemented on a per component basis, for example Repository, Configuration framework, Query processor etc.

1.1 Procedure for adding trace to a new Component.

1. Identify whether the component being added belongs to an existing list of trace components. Refer to the trace component list (TRACE_COMPONENT_LIST []) defined in <Pegasus/Common/TraceComponents.h> header file. If an appropriate component is found, note the component identifier (starts with a TRC_) from the list of TRACE_COMPONENT_IDs defined in <Pegasus/Common/TraceComponents.h> and skip step 2. If there is no appropriate component is available, move to step 2.

2. Add the new component to the <Pegasus/Common/TraceComponents.h> file. This step includes adding a component name and a component identifier.

    a. Add the <component name> to the TRACE_COMPONENT_LIST []. This name will be used to turn on tracing for the component.
    b. Add the component identifier to the TRACE_COMPONENT_ID list. The Component identifier must be formed by prefixing the component name with "TRC_"(E.g. TRC_<component name>). The developer should use the component identifier, to identify a component during any trace calls.

    (**NOTE**: It is important that both the TRACE_COMPONENT_ID and TRACE_COMPONENT_LIST [] are updated appropriately to include the new component.)

3. Add the trace calls to the component code. The TRACE_COMPONENT_ID should be used to identify a component in all trace calls.

1.2 Trace Interfaces

MACROS FOR TRACING

- PEG_FUNC_ENTER: Traces method entry
- PEG_FUNC_EXIT: Traces method exit

INLINE FUNCTIONS FOR TRACING

- Tracer::trace: Traces a given message.
- Tracer::trace: Traces a given message. Overloaded to include the file name and line number.
- The macros __LINE__ and __FILE__ should be used for passing the line number and file name arguments.

1.2.1 PEG_FUNC_ENTER

SYNOPSIS

PEG_ FUNC _ENTER (Uint32 traceComponent, char* methodName);

DESCRIPTION

PEG_FUNC_ENTER () logs a method entry message to the defined trace file.

traceComponent Refers to a trace Component defined in the tracer header file.

methodName The name of the method being entered.

EXAMPLE

```
PEG_FUNC_ENTER (
        TRC_QUERY_PROCESSOR,
        "WQLParser::getSelectClause");
```

1.2.5 PEG_FUNC_EXIT ()

SYNOPSIS

PEG_FUNC_EXIT (Uint32 traceComponent, char* methodName);

DESCRIPTION

PEG_FUNC_EXIT () logs a method exit message to the defined trace file.

traceComponent Refers to a trace Component defined in the tracer header file.

methodName The name of the method being exited.

EXAMPLE

```
PEG_FUNC_EXIT ( TRC_QUERY_PROCESSOR,
                "WQLParser::getSelectClause");
```

1.2.1 Tracer::trace

SYNOPSIS

```
Tracer::trace (
   Uint32 traceComponent,
   Uint32 traceLevel,
    char* formatString,
     …);
```

DESCRIPTION

Tracer::trace () logs a message to the defined trace file.

traceComponent	Refers to a trace Component defined in the tracer header file.
traceLevel	Sets the verbosity level of the trace message. NOTE: Level 1 is reserved for function entry/exit messages Level 2 Basic Flow trace messages, minimal data detail Level 3 Intra function logic flow, moderate data detail Level 4 High data detail
formatString	A printf style format string for the trace message.
…	Variable argument list matching the formatString.

EXAMPLE

```
Tracer::trace (
        TRC_HTTP,
        Tracer::LEVEL3,
        "The Query String is: %s",
        queryString);
```

1.2.2 Tracer::trace() : Overloaded to include file name and line number

SYNOPSIS

```
Tracer::trace (
    char*   fileName,
    Uint32 lineNum,
    Uint32 traceComponent,
     Uint32 traceLevel,
     char* formatString,
     …);
```

DESCRIPTION

Tracer::trace () logs a message to the defined trace file.

fileName	filename of the trace originator. (Use __FILE__ macro)
lineNum	line number of the trace originator. (Use __LINE__ macro)
traceComponent	Refers to a trace Component defined in the tracer header file.
traceLevel	Sets the verbosity level of the trace message. NOTE: Level 1 is reserved for function entry/exit messages Level 2 Basic Flow trace messages, minimal data detail Level 3 Intra function logic flow, moderate data detail Level 4 High data detail
formatString	A printf style format string for the trace message.
…	Variable argument list matching the formatString.

EXAMPLE

```
Tracer::trace (
     __FILE__,
     __LINE__,
    TRC_HTTP,
    Tracer::LEVEL3,
    "The Query String is: %s",
    queryString);
```

1.3 Setting trace properties

Trace properties for components within the CIMOM are managed using the configuration framework.

For non-CIMOM components, the following Trace interfaces must be used to set the trace properties. All the trace properties, trace level, trace file and trace components must be set before using any of the trace calls.

1.3.1 Setting Trace Level

NAME

Tracer::setTraceLevel()

SYNOPSIS

Uint32 Tracer::setTraceLevel (Uint32 traceLevel);

DESCRIPTION

Tracer::setTraceLevel( ) Sets the trace level to the given trace level. Trace messages are written to the trace file only if they are at or above this level.

traceLevel Specifies the trace level. The valid values for the traceLevel are,

Tracer::LEVEL1
Tracer::LEVEL2
Tracer::LEVEL3
Tracer::LEVEL4

EXAMPLES

Tracer::setTraceLevel(Tracer::LEVEL4);

RETURN VALUE

0 if successful
1 if an invalid level is passed

1.3.2 Setting Trace File

NAME

Tracer::setTraceFile()

SYNOPSIS

Uint32 Tracer::setTraceFile (char* traceFile);

DESCRIPTION

Tracer::setTraceFile( ) Sets the trace file to the given trace file.

traceFile Specifies the trace file path.

EXAMPLES

Tracer::setTraceFile(“/tmp/mytrace”);

RETURN VALUE

0 if successful

1 if unable to open the file in append mode

1.3.3 Setting Trace Components

NAME

Tracer::setTraceComponents()

SYNOPSIS

Tracer:: setTraceComponents(String traceComponents);

DESCRIPTION

Tracer:: setTraceComponents ( ) Sets up the Components to be traced.

traceComponents	Comma separated list of trace Components. The Components are defined in the tracer header file.

EXAMPLES

Tracer::setTraceComponents(“Repository”);
Tracer::setTraceComponents(“Http,XML_Parser”);
Tracer::setTraceComponents(“ALL”);

Compiling out Trace code from Pegasus build:
The trace code can be optionally compiled out of the Pegasus build, by defining PEGASUS_REMOVE_TRACE (–D option) at compile time.

1.4 Guidelines for adding trace code to components

1. Identify the appropriate level for the trace call. Trace messages can be associated with the following levels:

Level 1	Function entry/exit.
Level 2	Basic logic flow trace messages, minimal data detail
Level 3	Intra function logic flow and moderate data detail
Level 4	High data detail

2. For all function entry/exit trace messages, use the following convention:
"<ClassName>::<MethodName>"

E.g. Tracer::PEG_FUNC_ENTER(<componentID>, "<ClassName>::<MethodName>");

3. Use METHOD_NAME constant for enter/exit method names.

4. Use the macros __FILE__ and __LINE__ to pass the information regarding origin of the trace message (File Name and Line Number).

5. Be sure to trace at every point where your method may return (before returns or throws).

1.4 List of currently defined Trace components

Component	**Component Name**	**Component ID**
Channels	Channel	TRC_CHANNEL
XML Parser	XmlParser	TRC_XML_PARSER
XML Writer	XmlWriter	TRC_XML_WRITER
XML Reader	XmlReader	TRC_XML_READER
HTTP Protocol	Http	TRC_HTTP
CIM Types, Value, Objects	CimData	TRC_CIM_DATA
Provider Manager	ProvManager	TRC_PROV_MANAGER
Repository	Repository	TRC_REPOS
Request dispatching	Dispatcher	TRC_DISPATCHER
OS Abstractions	OsAbstraction	TRC_OS_ABS
Configuration	Config	TRC_CONFIG
Indication Delivery	IndDelivery	TRC_IND_DELIVERY

1.5 Sample code implementing trace calls

```
//------------------------------------------------------------------------
//
// testCimStartTag()
//
//     <!ELEMENT CIM (MESSAGE|DECLARATION)>
//     <!ATTRLIST CIM
//         CIMVERSION CDATA #REQUIRED
//         DTDVERSION CDATA #REQUIRED>
//
//------------------------------------------------------------------------

void XmlReader::testCimStartTag(XmlParser& parser)
{
    const char METHOD_NAME[] = “XMLReader::testCimStartTag”;
    XmlEntry entry;

    PEG_FUNC_ENTER(
        TRC_XML_PARSER,
        METHOD_NAME);

    Tracer::trace(
        __FILE__,
        __LINE__,
        TRC_XML_PARSER,
        Tracer::LEVEL2,
        "Checking for the start tag");

    XmlReader::expectStartTag(parser, entry, "CIM");

    const char* cimVersion;

    Tracer::trace(
        __FILE__,
        __LINE__,
        TRC_XML_PARSER,
        Tracer::LEVEL2,
        "Checking for the CIM version attribute");

    if (!entry.getAttributeValue("CIMVERSION", cimVersion))
    {
        Tracer::trace(
            __FILE__,
            __LINE__,
            TRC_XML_PARSER,
            Tracer::LEVEL2,
            "Missing CIM version attribute”);

        PEG_FUNC_EXIT(
            TRC_XML_PARSER,
            METHOD_NAME);

        throw XmlValidationError(
            parser.getLine(), "missing CIM.CIMVERSION attribute");
    }
```

```
Tracer::trace(
    __FILE__,
    __LINE__,
    TRC_XML_PARSER,
    Tracer::LEVEL2,
    "Comparing for the CIM version number");

if (strcmp(cimVersion, "2.0") != 0)
{
    Tracer::trace(
        __FILE__,
        __LINE__,
        TRC_XML_PARSER,
        Tracer::LEVEL3,
        "Expecting CIM version attribute 2.0, found : %s",
            cimVersion);

    PEG_FUNC_EXIT (
        TRC_XML_PARSER,
        METHOD_NAME);

    throw XmlValidationError(parser.getLine(),
        "CIM.CIMVERSION attribute must be \"2.0\"");
}

const char* dtdVersion;

Tracer::trace(
    __FILE__,
    __LINE__,
    TRC_XML_PARSER,
    Tracer::LEVEL2,
    "Checking for the DTD version attribute");

if (!entry.getAttributeValue("DTDVERSION", dtdVersion))
{
    Tracer::trace(
        __FILE__,
        __LINE__,
        TRC_XML_PARSER,
        Tracer::LEVEL2,
        "Missing DTD version attribute”);

    PEG_FUNC_EXIT (
        TRC_XML_PARSER,
        METHOD_NAME);

    throw XmlValidationError(
        parser.getLine(), "missing CIM.DTDVERSION attribute");
}

Tracer::trace(
    __FILE__,
    __LINE__,
    TRC_XML_PARSER,
```

```
        Tracer::LEVEL2,
        "Checking for the DTD version number");

    if (strcmp(dtdVersion, "2.0") != 0)
    {
        Tracer::trace(
            __FILE__,
            __LINE__,
            TRC_XML_PARSER,
            Tracer::LEVEL3,
            "Expecting DTD version attribute 2.0 found, : %s",
                dtdVersion);


        PEG_FUNC_EXIT (
            TRC_XML_PARSER,
            METHOD_NAME );

        throw XmlValidationError(parser.getLine(),
            "CIM.DTDVERSION attribute must be \"2.0\"");
    }

    PEG_FUNC_EXIT (
        TRC_XML_PARSER,
        METHOD_NAME);

}
```