

# **Rose Model Report**

*Generated August 26, 2004*

*2:46 PM*

TABLE OF CONTENTS

TABLE OF CONTENTS .....2

LOGICAL VIEW REPORT.....8

LOGICAL VIEW .....8

CBE .....8

*CBCommonBaseEvent* .....8

*CBExtendedDataElement*.....10

*CBContextDataElement* .....10

*CBComponentIdentification*.....10

*CBMsgDataElement*.....11

*CBDefaultEvent*.....11

*CBDefaultElement*.....12

*CBSituation* .....12

*CBStartSituation* .....12

*CBConnectSituation* .....12

*CBStopSituation* .....12

*CBReportSituation* .....13

*CBFeatureSituation*.....13

*CBConfigureSituation* .....13

*CBDependencySituation*.....13

*CBCreateSituation* .....14

*CBDestroySituation*.....14

*CBAvailableSituation*.....14

*CBRequestSituation*.....14

*CBOtherSituation*.....14

STATISTICAL.....15

*SDDescriptor* .....15

*SDMemberDescriptor*.....15

*SDGaugeRepresentation*.....15

*SDSampleDescriptor* .....16

*SDView* .....16

*SDSampleWindow*.....16

*SDSnapshotObservation*.....16

*SDDiscreteObservation* .....17

*SDContiguousObservation* .....17

*SDTextObservation*.....17

*SDRangeRepresentation*.....17

*SDCounterDescriptor* .....17

*SDRepresentation* .....18

*SDTextRepresentation* .....18

*SDDiscreteRepresentation*.....18

*SDContiguousRepresentation*.....18

TEST.....18

TESTPROFILE .....18

*TPFDeployment*.....18

*TPFLiteralAnyorNull*.....19

*TPFLiteralAny* .....19

*TPFInstanceValue* .....19

*TPFExecutionResult* .....19

*TPFLogAction* .....19

## LOGICAL VIEW REPORT

---

<i>TPFDefault</i> .....	20
<i>TPFDefaultApplication</i> .....	20
<i>TPFBehavior</i> .....	20
<i>TPFTestCase</i> .....	20
<i>TPFArbiter</i> .....	21
<i>TPFCodingRule</i> .....	21
<i>TPFSUT</i> .....	21
<i>TPFTestSuite</i> .....	21
<i>TPFTestObjective</i> .....	22
<i>TPFTestComponent</i> .....	22
<i>TPFTimezone</i> .....	22
<i>TPFValidationAction</i> .....	23
<i>TPFVerdict</i> .....	23
<i>TPFExecutionHistory</i> .....	23
<i>TPFExecutionEvent</i> .....	23
<i>TPFExecutionStatus</i> .....	24
<i>TPFExecutionType</i> .....	24
<i>TPFInvocationEvent</i> .....	24
<i>TPFVerdictEvent</i> .....	24
<i>TPFMessageEvent</i> .....	24
<i>TPFSeverity</i> .....	25
<i>TPFTypedEvent</i> .....	25
<i>TPFInvocationStatus</i> .....	25
<i>TPFVerdictReason</i> .....	25
<i>TPFInvocationReason</i> .....	25
<i>TPFTest</i> .....	26
<i>TPFLoopEvent</i> .....	26
<i>TPFTimedEvent</i> .....	26
<i>TPFWaitEvent</i> .....	26
BEHAVIOR.....	27
CONFIGURATION.....	27
<i>CFGLocation</i> .....	27
<i>CFGClass</i> .....	27
<i>CFGOperation</i> .....	27
<i>CFGArtifact</i> .....	28
<i>CFGInstance</i> .....	28
<i>CFGComparableProperty</i> .....	28
<i>CFGPropertyGroup</i> .....	28
<i>CFGCategory</i> .....	29
<i>hyadesOperatingSystemEnumeration</i> .....	29
<i>hyadesOperatingSystemCategory</i> .....	29
<i>hyadesProcessorTypeCategory</i> .....	30
<i>hyadesMemorySizeCategory</i> .....	30
<i>CFGCategorySelectionMode</i> .....	30
<i>hyadesProcessorSpeedCategory</i> .....	30
<i>hyadesProcessorTypeEnumeration</i> .....	31
<i>hyadesBrowserTypeCategory</i> .....	31
<i>hyadesBrowserVersionCategory</i> .....	31
<i>hyadesBrowserEnumeration</i> .....	32
<i>hyadesProcessorNumberCategory</i> .....	32
<i>hyadesDisplayColorDepthCategory</i> .....	32
<i>hyadesDisplayColorDepthEnumeration</i> .....	32
<i>hyadesDisplayHeightCategory</i> .....	33
<i>hyadesDisplayWidthCategory</i> .....	33
<i>hyadesDisplayHeightEnumeration</i> .....	33

## LOGICAL VIEW REPORT

---

<i>hyadesDisplayWidthEnumeration</i> .....	34
<i>hyadesDisplayNumberCategory</i> .....	34
<i>hyadesDatabaseCategory</i> .....	34
<i>hyadesDatabaseVersionCategory</i> .....	35
<i>hyadesDatabaseEnumeration</i> .....	35
<i>CFGMachineInstance</i> .....	35
<i>CFGMachineConstraint</i> .....	35
<i>CFGCategorySelectionAmount</i> .....	35
<i>CFGCategorySelectionMultiplicity</i> .....	36
<i>CFGConfigurableObject</i> .....	36
<i>hyadesHostnameCategory</i> .....	36
<i>hyadesWindowsDomainCategory</i> .....	36
<i>hyadesUsernameCategory</i> .....	36
<i>hyadesServicePackCategory</i> .....	37
<i>hyadesMajorVersionCategory</i> .....	37
<i>hyadesMinorVersionCategory</i> .....	37
<i>hyadesServicePackEnumeration</i> .....	37
<i>CFGArtifactLocationPair</i> .....	38
<i>CFGMachine</i> .....	38
<i>CFGPseudoEnumeration</i> .....	38
<i>hyadesPasswordCategory</i> .....	39
<i>hyadesClasspathCategory</i> .....	39
<i>hyadesRootDirectoryCategory</i> .....	39
DATAPOOL .....	39
<i>DPLDatapoolSpec</i> .....	39
<i>DPLEquivalenceClass</i> .....	40
<i>DPLRecord</i> .....	40
<i>DPLCell</i> .....	40
<i>DPLVariable</i> .....	40
<i>DPLRole</i> .....	40
<i>DPLDatapool</i> .....	40
COMMON .....	40
<i>CMNNamedElement</i> .....	41
<i>CMNMachine</i> .....	41
<i>CMNNodeType</i> .....	41
<i>CMNNodeInstance</i> .....	41
<i>CMNExtendedProperty</i> .....	41
<i>CMNDefaultProperty</i> .....	42
<i>CMNAnnotation</i> .....	42
SDB .....	42
<i>SDBRuntime</i> .....	42
<i>SDBSymptom</i> .....	43
<i>SDBMatchPattern</i> .....	43
<i>SDBSolution</i> .....	43
<i>SDBDirective</i> .....	44
HIERARCHY .....	44
<i>TRCCollectionMode</i> .....	44
<i>TRCProcessProxy</i> .....	45
<i>TRCOption</i> .....	45
<i>TRCAgent</i> .....	45
<i>TRCAgentProxy</i> .....	46
<i>TRCConfiguration</i> .....	47
<i>TRCEnvironmentVariable</i> .....	47
<i>TRCExecParameter</i> .....	47
<i>TRCFilter</i> .....	47

## LOGICAL VIEW REPORT

---

<i>TRCNode</i> .....	47
<i>TRCMonitor</i> .....	48
<i>AbstractDefaultEvent</i> .....	48
<i>AbstractTRCView</i> .....	48
<i>AbstractTRCDescription</i> .....	48
<i>AbstractTRCProcess</i> .....	49
<i>AbstractTRCCollectionBoundary</i> .....	49
<i>UnresolvedCorrelation</i> .....	49
<i>CorrelationSourceInfo</i> .....	49
<i>CorrelationContainer</i> .....	49
<i>CorrelationContainerProxy</i> .....	50
<i>CorrelationEntry</i> .....	50
<i>CorrelationEngine</i> .....	50
<i>BooleanArray</i> .....	50
<i>IntArray</i> .....	50
DATA .....	51
<i>TSTDataPool</i> .....	51
<i>TSTPicker</i> .....	51
<i>TSTEquivalenceClass</i> .....	51
EXTENSIONS .....	51
<i>Query</i> .....	51
<i>SimpleSearchQuery</i> .....	52
<i>InstanceQuery</i> .....	52
<i>OrderByElement</i> .....	52
<i>QueryResult</i> .....	52
<i>SimpleBinaryExpression</i> .....	52
<i>LeftOperand</i> .....	52
<i>RelationalOperators</i> .....	52
<i>OrderByOperators</i> .....	53
<i>ResultEntry</i> .....	53
<i>CorrelationQuery</i> .....	53
<i>TimeBasedCorrelationQuery</i> .....	53
FRAGMENTS .....	53
<i>BVRInteractionOccurrence</i> .....	53
<i>BVRPartDecomposition</i> .....	54
<i>BVRInteractionOperand</i> .....	54
<i>BVRInteractionConstraint</i> .....	54
<i>BVRInteractionOperator</i> .....	54
<i>BVRGate</i> .....	55
<i>BVRCombinedFragment</i> .....	55
<i>BVRInteraction</i> .....	55
INTERACTIONS .....	55
<i>BVRLifeline</i> .....	56
<i>BVRInteractionFragment</i> .....	56
<i>BVRMessage</i> .....	56
<i>BVRGeneralOrdering</i> .....	56
<i>BVREventOccurrence</i> .....	57
<i>BVRMessageEnd</i> .....	57
<i>BVRMessageSort</i> .....	57
<i>BVRExecutionOccurrence</i> .....	57
<i>BVRStateInvariant</i> .....	58
<i>BVRStop</i> .....	58
<i>BVRProperty</i> .....	58
ECORE .....	58
<i>EAttribute</i> .....	58

## LOGICAL VIEW REPORT

---

<i>EAnnotation</i> .....	58
<i>EClass</i> .....	59
<i>EClassifier</i> .....	59
<i>EDataType</i> .....	59
<i>EEnum</i> .....	59
<i>EEnumLiteral</i> .....	60
<i>EFactory</i> .....	60
<i>EModelElement</i> .....	60
<i>ENamedElement</i> .....	60
<i>EObject</i> .....	61
<i>EOperation</i> .....	61
<i>EPackage</i> .....	61
<i>EParameter</i> .....	61
<i>EReference</i> .....	62
<i>EStructuralFeature</i> .....	62
<i>ETypedElement</i> .....	62
<i>EBigDecimal</i> .....	62
<i>EBigInteger</i> .....	63
<i>EBoolean</i> .....	63
<i>EBooleanObject</i> .....	63
<i>EByte</i> .....	63
<i>EByteArray</i> .....	63
<i>EByteObject</i> .....	63
<i>EChar</i> .....	64
<i>ECharacterObject</i> .....	64
<i>EDate</i> .....	64
<i>EDiagnosticChain</i> .....	64
<i>EDouble</i> .....	64
<i>EDoubleObject</i> .....	64
<i>EEList</i> .....	65
<i>EEnumerator</i> .....	65
<i>EFeatureMap</i> .....	65
<i>EFeatureMapEntry</i> .....	65
<i>EFloat</i> .....	65
<i>EFloatObject</i> .....	66
<i>EInt</i> .....	66
<i>EIntegerObject</i> .....	66
<i>EJavaClass</i> .....	66
<i>EJavaObject</i> .....	66
<i>ELong</i> .....	66
<i>ELongObject</i> .....	67
<i>EMap</i> .....	67
<i>EResource</i> .....	67
<i>EResourceSet</i> .....	67
<i>EShort</i> .....	67
<i>EShortObject</i> .....	67
<i>EString</i> .....	68
<i>EStringToStringMapEntry</i> .....	68
<i>ETreeIterator</i> .....	68
TRACE.....	68
<i>TRCObject</i> .....	68
<i>TRCClass</i> .....	69
<i>TRCMethodInvocation</i> .....	70
<i>TRCProcess</i> .....	70
<i>TRCThread</i> .....	71

## LOGICAL VIEW REPORT

---

<i>TRCMethod</i> .....	71
<i>TRCPrimitiveType</i> .....	72
<i>EObjectID</i> .....	72
<i>EMethodID</i> .....	72
<i>TRCPackage</i> .....	73
<i>TRCCollectionBoundary</i> .....	73
<i>TRCClassLoader</i> .....	74
<i>EClassID</i> .....	74
<i>TRCSignatureNotation</i> .....	74
<i>TRCSourceInfo</i> .....	74
<i>TRCMethodProperties</i> .....	74
<i>TRCHeapObject</i> .....	75
<i>TRCFullTraceObject</i> .....	75
<i>TRCTraceObject</i> .....	75
<i>TRCFullHeapObject</i> .....	75
<i>TRCObjectReference</i> .....	76
<i>TRCHeapDump</i> .....	76
<i>TRCAggregatedMethodInvocation</i> .....	76
<i>TRCFullMethodInvocation</i> .....	76
<i>TRCHeapRoot</i> .....	77
<i>TRCGCRootType</i> .....	77
<i>TRCArrayClass</i> .....	77
<i>TRCAggregatedObjectReference</i> .....	77
<i>TRCThreadEvent</i> .....	78
<i>TRCThreadSleepingEvent</i> .....	78
<i>TRCThreadWaitingForObjectEvent</i> .....	78
<i>TRCThreadWaitingForLockEvent</i> .....	78
<i>TRCThreadRunningEvent</i> .....	79
<i>TRCThreadDeadEvent</i> .....	79
<i>TRCMethodWithLLData</i> .....	79
<i>TRCLLData</i> .....	79
<i>TRCSourceInfoWithLLData</i> .....	79
<i>TRCLineCoverageData</i> .....	80
<i>LLUnitData</i> .....	80
<i>TRCObjectValue</i> .....	80
<i>TRCInputOutputEntry</i> .....	80
<i>TRCInputOutputContainer</i> .....	80
<b>PACKAGE STRUCTURE</b> .....	<b>81</b>

## LOGICAL VIEW REPORT

---

### Logical View

Copyright (c) {date} IBM Corporation and others.

All rights reserved. This program and the accompanying materials are made available under the terms of the Common Public License v1.0 which accompanies this distribution, and is available at <http://www.eclipse.org/legal/cpl-v10.html>

Contributors:

- IBM Corporation - initial implementation
- Rational Software - initial implementation
- Scapa Technologies - initial implementation

### cbe

The Common Base Event model is used to describe a basic model for storing "messages" or logs from a product. The Common Base Event is the core structure for holding such messages which are often referred to as events from the products that log them.

### CBECommonBaseEvent

This class provides the the basic level of common properties for all problem artifacts.

*Derived from CBEDefaultEvent*

#### Public Properties:

---

**localInstanceId** : String

Source supplied event identifier. There is no guarantee that this value is globally unique and stays constant for the life of the event.

**globalInstanceId** : String

A globally unique identifier for this specific Event instance.

**creationTime** : double

The time the artifact was created as defined by CIM datetime data type. If not defined then the event receiver must timestamp it.

Following is excerpt regarding format is from the standard:

yyyymmddhhmmss.mmmmmmsutc (Must always be 25 characters - zero-padded if necessary)

yyyy is a 4 digit year

mm is the month

## LOGICAL VIEW REPORT

---

dd is the day  
hh is the hour  
mm is the minutes  
ss is the second  
mmmmmm is the number of microseconds  
s is a "+" or "-" indicating the sign of UTC (Universal Coordinate Time - same as Greenwich Mean Time) or simply a ":" indicating a time interval (in case of using : yymm are interpreted as days)  
utc is the offset from UTC in minutes and ignored for the time interval

For example 19980525145013.0000000-300 is Monday, May 25, 1998 at 2:50:13 PM EST.

**timeZone : short**

The time zone part from the original XML fragment (eg. -240), see creationTime.

**severity : short**

Events can be classified with different severity levels. This enables administrators to focus on the most severe problems currently occurring in the enterprise. The event adapter can optionally assign the value for this attribute. If the event adapter does not assign a severity level, the default severity level in the event class definition is assigned by the event server. This attribute is extensible; you can add a new severity level by appending to the severities list defined in the event class definition files. The predefined severity levels, in order of increasing severity, are as follows: Information

- " Unknown
- " Harmless error that has no effect on the resource
- " Warning should be used when it's appropriate to let the user decide if action is needed.
- " Minor should be used to indicate action is needed, but the situation is not serious at this time.
- " Critical should be used to indicate action is needed NOW and the scope is broad (perhaps an imminent outage to a critical resource will result).
- " Fatal should be used to indicate an error occurred, but it's too late to take remedial action.

Enumeration values start at 0 and go to 60. Default is unknown (10)

Default value is Unknown.

**priority : short**

Defines the importance of the event

- " Unknown
- " Ignore
- " Low
- " Medium
- " High
- " Critical

Default value is Unknown.

**msg : String**

The content of this optional string is determined by the event creator. A typical usage would be the fully resolved and human readable message associated with this event.

**repeatCount : short**

This optional property is the number of occurrences of a given message artifact for a specific time interval. The Time interval is indicated by the ElapsedTime property described below.

**elapsedTime : long**

This optional property is the time interval or the elapsed time for the number occurrences of message artifact provided by the MessageCount property. This property is expressed in microseconds.

**sequenceNumber : long**

Optional sequence numbers are used to provide order at a greater granularity than the time stamps.

version : String = commonbaseevent1\_0  
otherData : String

### CBEExtendedDataElement

The ExtendedDataElement class provides a name-value pair for a Common Base Event. For example you may choose to have a name-value pair to provide return error code, extended problem situation description, or provide any additional data for the consumer of the event.

Token names may be referenced by correlation rules, however the ContextDataElement's prime purpose is to hold data used for correlation. The support of this class relieves management tools of the need to be able to tokenize application specific artifacts. The token definitions are assumed to be specific to a resource. The data format is assumed to be in big endian.

*Derived from CBEDefaultElement*

#### Public Properties:

---

type : String  
hexValue : String

### CBEContextDataElement

The Context Data Element provides data that is used to establish the environment of an event. This data is intended to be used when post processing the model in order to associate or "correlate" multiple events together.

#### Public Properties:

---

contextId : String  
type : String  
name : String  
contextValue : String

### CBEComponentIdentification

A component is a modular unit of a running system that has been defined to be a source of events, or a reporter of events. Typically these are large grained parts of a system, or services that are provided to a system. However the notion of a component is up to the system itself to define.

### Public Properties:

---

**location** : String  
**locationType** : String  
**application** : String  
**executionEnvironment** : String  
**component** : String  
**subComponent** : String  
**componentIdType** : String  
**instanceId** : String  
**processId** : String

This property identifies the processID of the "running" component or subcomponent that generated the artifact. This property is optional.

**threadId** : String

This property identifies the ThreadID of the component or subcomponent indicated by the Process ID that generated the artifact. A running process may spawn one or more thread to carry its function and/or incoming requests, the ThreadID will change accordingly. This property is optional.

**componentType** : String

## CBEMsgDataElement

When an event is used to represent a message, a message data element captures the several parts of the message information. For example "RDB2742E: an Error was detected in foo indicating that bar had problems" which has a msg id as well as substituted "foo" and "bar" variables.

### Public Properties:

---

**msgId** : String

ZThis is where you would find the RDB1234E string that identifies the message.

**msgIdType** : String

the id type is used to indicate the structure of the message id. for example IBM341 indicates a 3 charactercomponent id (RDB), a 4 character number (1234), and a 1 character severity level (E) making up the example RDB1234E.

**msgCatalogId** : String

A

**msgCatalogTokens** : String

**msgCatalog** : String

**msgLocale** : String

**msgCatalogType** : String

## CBEDefaultEvent

A default event is the parent of any class that you wish to pass to an analysis engine. It will be marked isAnalyzed by the engine as appropriate.

The default event is also the most minimal event that can be logged. As an instance, all values logged are held in a structure of default elements.

*Derived from AbstractDefaultEvent*

### Public Properties:

---

**analyzed** : boolean = false

this member is used to indicate if the event has been analyzed by an engine.

extensionName : String

### CBEDefaultElement

This is the most primitive holder of data. A simple name value pair.

#### Public Properties:

---

name : String

The Name property names the token. DefaultElement typically define token names that can be used to identify data specific to an event. For example, the elements of an xml fragment that go beyond the default event or the CBE.

values : String

This is a sequence of values associated with the name.

### CBESituation

#### Public Properties:

---

categoryName : String

reasoningScope : String

### CBEStartSituation

*Derived from CBESituation*

#### Public Properties:

---

successDisposition : String

situationQualifier : String

### CBEConnectSituation

*Derived from CBESituation*

#### Public Properties:

---

successDisposition : String

situationDisposition : String

### CBEStopSituation

*Derived from CBESituation*

Public Properties:

---

successDisposition : String  
situationQualifier : String

## CBEReportSituation

*Derived from CBESituation*

Public Properties:

---

reportCategory : String

## CBEFeatureSituation

*Derived from CBESituation*

Public Properties:

---

featureDisposition : String

## CBEConfigureSituation

*Derived from CBESituation*

Public Properties:

---

successDisposition : String

## CBEDependencySituation

*Derived from CBESituation*

Public Properties:

---

dependencyDisposition : String

### CBECreateSituation

*Derived from CBESituation*

Public Properties:

---

successDisposition : String

### CBEDestroySituation

*Derived from CBESituation*

Public Properties:

---

successDisposition : String

### CBEAavailableSituation

*Derived from CBESituation*

Public Properties:

---

operationDisposition : String  
processingDisposition : String  
availabilityDisposition : String

### CBERequestSituation

*Derived from CBESituation*

Public Properties:

---

successDisposition : String  
situationQualifier : String

### CBEOtherSituation

*Derived from CBESituation*

### Public Properties:

**anyData** : String

---

## statistical

Internal package, please do not extend or use.

This package is intended to describe structured and dynamically typed statistical data.

### SDDescriptor

This is the base class for all the descriptors on the model. This class provides the ability to structure the descriptor data in a tree (parent/child relationship).

This class, and its subclasses, does not hold any data fields proper, but rather contain all the metadata that describe the counters, statistics, etc. that can be collected from an agent instance.

*Derived from AbstractTRCDescription*

### Public Properties:

**id** : String

**name** : String

**description** : String

---

### SDMemberDescriptor

This is the abstract class that all descriptor types need to realize. Specializations of this class represent the different types of data that are collected.

*Derived from SDDescriptor*

### SDGaugeRepresentation

Observations of this type has a certain range and a maximum and minimum threshold. These threshold values can be set by the UI and stored here. It is then possible to observe the model as it is built and provide notification actions and the like.

*Derived from SDRangeRepresentation*

### Public Properties:

**maxThreshold** : int

**minThreshold** : int

---

### SDSampleDescriptor

This is a descriptor type for statistical data. Statistical data is data that is not necessarily updated all the time but rather is updated in the agent only periodically at some frequency.

*Derived from* *SDMemberDescriptor*

#### Public Properties:

---

**updateFrequency** : double

### SDView

The view is a named collection of windows. The view allows you to navigate the windows so that data can be propagated through the windows if that behaviour is desired. For example data may be collected at 1 minute intervals and used to populate a window which contains the most recent 1 hour of data. There may be a second window that contains a set of data points that summarize each of the previous 23 hours plus the hour represented in the first window. A third window may reflect the data for a given week, and so on.

A view associates these windows of related data and provides a mechanism to manage the cascading of data across these windows over the data.

*Derived from* *AbstractTRCView*

#### Public Properties:

---

**name** : String

### SDSampleWindow

This is a window of data. Windows contain a fixed number of snapshots that typically represent some period of time.

This class can be specialized to describe processing of the observations for a particular use-case.

### SDSnapshotObservation

All observations are specializations of this class. The individual observation values reside in the specialization instances.

#### Public Properties:

---

**validityMask** : byte

This is a bitmask for the indices of the observation instance array fields. It is possible for a counter to become active/inactive at any point during a window. This mask allows a means of marking whether a particular field contains a valid value.

### SDDiscreteObservation

The collection of discrete values associated with a counter during a sample window.

*Derived from SDSnapshotObservation*

Public Properties:

---

value : int

### SDContiguousObservation

The collection of contiguous values associated with a counter during a sample window.

*Derived from SDSnapshotObservation*

Public Properties:

---

value : double

### SDTextObservation

The collection of text values associated with a counter during a sample window.

*Derived from SDSnapshotObservation*

Public Properties:

---

textValue : String

### SDRangeRepresentation

Observations of this type have a value within a certain range

*Derived from SDDiscreteRepresentation*

Public Properties:

---

min : int

max : int

### SDCounterDescriptor

This class is a descriptor for a counter that is constantly updated. Observations of this type are considered to be up to the moment values.

*Derived from SDMemberDescriptor*

### **SDRepresentation**

This is the root of all the metadata types that describe how to render a observation. The intention of the representation class is to provide a description of observations so that a generic viewer can be provided for the observations with the same representation specialization

### **SDTextRepresentation**

Represent observations of this type as text values.

*Derived from SDRepresentation*

### **SDDiscreteRepresentation**

Observations of this type have discrete values (integers).

*Derived from SDRepresentation*

### **SDContiguousRepresentation**

Observations of this type have contiguous values (float).

*Derived from SDRepresentation*

### **test**

### **testprofile**

### **TPFDeployment**

A deployment is the allocation of the Instances into Locations.

A Test Suite might be associated to a number of deployment specifying with configurations should be used to run the Test Suite.

*Derived from CFGConfigurableObject*

### **TPFLiteralAnyOrNull**

LiteralAnyOrNull is a literal representing any value out of a set of possible values, or the lack of a value.

### **TPFLiteralAny**

LiteralAny is a literal representing any value out of a set of possible values

### **TPFInstanceValue**

An Instance Value is a specification of a named attribute value. The value can take several forms including literal values, expressions, existing InstanceValues, or constraints.

*Derived from CMNNamedElement*

### **TPFExecutionResult**

The ExecutionResult (renamed Trace concept from the Test Profile) represents a snapshot of the execution of a TestCase or TestSuite. It contains deployment information, the ordered set of log actions performed during the test case, and the final verdict.

*Derived from CMNNamedElement*

#### **Public Properties:**

---

testVersion : String  
verdict : TPFVerdict  
type : String

### **TPFLogAction**

A LogAction is an EventOccurrence that specifies that an entity should be logged to the execution trace for further analysis. The target of a log action is a logging mechanism in the run-time system.

*Derived from BVREventOccurrence*

### TPFDefault

A default is a behavior that is activated when an unexpected event occurs on a test component.

*Derived from TPFBehavior*

### TPFDefaultApplication

A default application is the a reference to a default behavior, owned by an element deifning its scope. This allows the default to be activated should an unexpected event occur on a component, based on its scope.

### TPFBehavior

Behavior represents the dynamic behavior of a TestSuite, TestCase, TestComponent used to test an SUT, or Arbiter. In the context of a TestCase, it represents the composite behavior of the different TestComponents realizing the test, and/or the behavior of the TestCase in the case where the TestSuite classifier interacts directly with the SUT.

In the Hyades meta-model Behaviors are modeled using Interactions (i.e. Sequence Diagrams) enabling the easy modeling of composite behaviors.

*Derived from CMNNamedElement*

#### Public Properties:

---

**isReentrant** : Boolean

**resource** : String

**location** : String

### TPFTestCase

A test case is a specification of one case to test the system, including what to test with which input, result, and under which conditions.

A test case belongs to a test suite. It has a behavior specifying how a set of cooperating test components interacting with a system under test realize a test objective.

It may invoke other test cases.

A test case uses an arbiter to evaluate the outcome of its test behavior.

*Derived from TPFTest*

### TPFArbiter

An Arbiter represents an implementation of the IArbiter interface. The purpose of an IArbiter implementation is to determine the final verdict for a test case. This determination is done according to a particular arbitration strategy, which is provided in the implementation of the arbiter interface.

*Derived from CFGClass*

### TPFCodingRule

Coding rules are strings referencing coding rules such as those defined for ASN.1, CORBA or XML. Coding rules are basically applied to InstanceValue to denote the concrete encoding and decoding for these values during test execution.

#### Public Properties:

---

**coding** : String

### TPFSUT

The SUT is the system under test. The SUT provides only a set of operations via publicly available interface(s) to enable blackbox testing.

*Derived from CFGClass*

#### Public Properties:

---

**location** : String  
**resource** : String

### TPFTestSuite

A Test Suite acts as a grouping mechanism for a set of test cases.

It can be related to a TestObjective, defining the motivation of the TestSuite.

The behavior of a test suite is generally used for control the execution flow between its test cases or to invokes external test suites or test cases.

*Derived from TPFTest*

*Derived from CFGClass*

### Public Properties:

**persistenceId** : String

---

## TPFTestObjective

A Test Objective is a dependency used to specify the objectives of a Test Case or Test Suite. A Test Case or Test Suite can have any number of objectives. A Test Objective being a dependency to the objective itself, it is associated to only one Test Case or Test Suite.

*Derived from CMNNamedElement*

### Public Properties:

**type** : String

**reference** : String

---

## TPFTestComponent

A test component is commonly an active class used to specify test cases as interactions between a number of test components with the SUT. The behavior of a test component can be used to specify low level test behavior, or it can be automatically derived from all the test cases behaviors in which the component takes part.

*Derived from CFGClass*

### Public Properties:

**type** : String

---

## TPFTimezone

Timezones serve as a grouping mechanisms for test components within a test system. Each test component belongs at most to one timezone. Test components in the same timezone have the same perception of time, i.e. test components of the same timezone are considered to be time synchronized. The time zone of a test component can be accessed both in the model and in run-time.

Comparing time-critical events within the same timezone is allowed. Comparing time-critical events of different timezones is a matter of semantic variation point and should be decided by the tool vendor. By default, comparison between events in two different timezones is illegal.

### Public Properties:

**name** : String

**description** : String

---

### TPFValidationAction

A ValidationAction is an EventOccurrence that performs some kind of verification.

When a validation action is specified, it indicates that at run-time, the expression within the action will be evaluated. If the expression evaluates to true, the verdict pass is sent to the arbiter using the setVerdict operation. If the expression is false, the verdict fail is sent to the arbiter. Instead of an expression, valid verdicts for the arbiter implementation may also be used.

*Derived from BVREventOccurrence*

### TPFVerdict

The verdict is a predefined enumeration datatype which contains at least the values fail, inconclusive, pass, error indicating how this test case execution has performed.

- A pass verdict indicates that the test case is successful and that the SUT has behaved according to what should be expected.
- A fail verdict on the other hand shows that the SUT is not behaving according to the specification.
- An inconclusive verdict means that the test execution cannot determine whether the SUT performs well or not.
- An error verdict tells that the test system itself and not the SUT fails.

#### Public Properties:

---

**inconclusive** : String  
**pass** : String  
**fail** : String  
**error** : String

### TPFExecutionHistory

### TPFExecutionEvent

*Derived from CMNNamedElement*

Public Properties:

---

ownerId : String  
timestamp : long  
text : String  
eventType : String

### TPFExecutionStatus

### TPFExecutionType

Public Properties:

---

start : String  
stop : String

### TPFInvocationEvent

*Derived from TPFExecutionEvent*

Public Properties:

---

status : TPFInvocationStatus  
reason : TPFInvocationReason

### TPFVerdictEvent

*Derived from TPFExecutionEvent*

Public Properties:

---

verdict : TPFVerdict  
reason : TPFVerdictReason

### TPFMessageEvent

*Derived from TPFExecutionEvent*

Public Properties:

severity : TPFSeverity

---

### TPFSeverity

Public Properties:

info : String

error : String

warning : String

---

### TPFTypedEvent

*Derived from TPFExecutionEvent*

Public Properties:

type : TPFExecutionType

---

### TPFInvocationStatus

Public Properties:

unattempted : String

successful : String

unsuccessful : String

---

### TPFVerdictReason

Public Properties:

unknown : String

none : String

seeDescription : String

abort : String

didNotComplete : String

---

### TPFInvocationReason

Public Properties:

---

unknown : String  
none : String  
seeDescription : String  
preconditionNotMet : String  
noMatchingConfiguration : String  
noBehavior : String  
didNotStart : String

## TPFTest

A test is an abstract concept for TestSuite and Test Case. It enables the association of both concepts to a TestObjective, a TestExecutionResult and a Behavior.

As such a Test has no semantic associated with it.

*Derived from CMNNamedElement*

Public Properties:

---

type : String

## TPFLoopEvent

*Derived from TPFFExecutionEvent*

## TPFTimedEvent

A TPFTimedEvent is an event that is specifically timed. It contains both a start and end timestamp, and can have its duration calculated and displayed.

*Derived from TPFFExecutionEvent*

Public Properties:

---

endTimestamp : long

## TPFWaitEvent

A TPFWaitEvent is a specialization of TPFTimedEvent. The distinction between the two types is that a TPFWaitEvent is an event that logs a deliberate wait or sleep, while a TPFTimedEvent may log the duration of any arbitrary sequence.

*Derived from TPFTimedEvent*

## behavior

## configuration

### CFGLocation

A CFGLocation is the representation of a physical or logical entity into which a CFGInstance can be deployed.

Examples of Location include Processors, Application Servers, Containers, JVM, Processes, Threads.

*Derived from CFGConfigurableObject*

### CFGClass

A CFGDeployable is an element that describes behavioral (operation/behavior), and structural features (Instance) and may be instantiable. Mapped to a programming language like java, it comes in several specific forms, including class, interface, etc.

It owns CFGInstances which define its internal structure (class attributes).

A CFGDeployable has a behaviorResource and behaviorLocation which describe where its behavior is stored: Resource URL and Location within the resource .

*Derived from CMNNamedElement*

### CFGOperation

An operation is a service that can be requested from a deployable.

*Derived from CMNNamedElement*

### CFGArtifact

This class is a grouping of CFGInstances that share a common CFGDeploymentSpec and CFGLocation. In other words, CFGInstances within the same CFGArtifact will be deployed to the same location, and within the same {process / thread}.

*Derived from CFGConfigurableObject*

### CFGInstance

CFGInstance represents an instance variable of a CFGClass. The objects represented by these instance variables are grouped in one or more artifacts. The maxCount attribute of CFGInstance represents the maximum number of instances of the object represented by this instance variable which will be created during a test execution.

*Derived from CMNNamedElement*

**Public Properties:**

---

**maxCount** : int  
**initialValue** : String

### CFGComparableProperty

*Derived from BVRProperty*

**Public Properties:**

---

**type** : CFGCategory  
**operator** : String

### CFGPropertyGroup

*Derived from CMNNamedElement*

**Public Properties:**

---

**propertyGroupID** : String

## CFGCategory

*Derived from CMNNamedElement*

### Public Properties:

---

name : String  
displayName : String  
selectionMode : CFGCategorySelectionMode  
selectionAmount : int  
selectionMultiplicity : CFGCategorySelectionMultiplicity

## hyadesOperatingSystemEnumeration

*Derived from CFGPsudoEnumeration*

### Public Properties:

---

category : CFGCategory = hyadesOSCategory  
OS\_1 : String = "OS.Microsoft.Windows.95"  
OS\_2 : String = "OS.Microsoft.Windows.98"  
OS\_3 : String = "OS.Microsoft.Windows.98SE"  
OS\_4 : String = "OS.Microsoft.Windows.NT"  
OS\_5 : String = "OS.Microsoft.Windows.2000.Professional"  
OS\_6 : String = "OS.Microsoft.Windows.2000.Server"  
OS\_7 : String = "OS.Microsoft.Windows.2000.Advanced Server"  
OS\_8 : String = "OS.Microsoft.Windows.XP.Home"  
OS\_9 : String = "OS.Microsoft.Windows.XP.Media Center"  
OS\_10 : String = "OS.Microsoft.Windows.XP.Professional"  
OS\_11 : String = "OS.Microsoft.Windows.XP.Tablet PC"  
OS\_12 : String = "OS.Microsoft.Windows.XP.64-Bit"  
OS\_13 : String = "OS.Microsoft.Windows.Server 2003"  
OS\_14 : String = "OS.Microsoft.Windows.Small Business Server 2003"  
OS\_50 : String = "OS.Apple.OS 7xx"  
OS\_51 : String = "OS.Apple.OS 8xx"  
OS\_52 : String = "OS.Apple.OS 9xx"  
OS\_53 : String = "OS.Apple.OS X"  
OS\_54 : String = "OS.Apple.OS X.Panther"  
OS\_55 : String = "OS.Apple.OS X.Server"  
OS\_60 : String = "OS.IBM.AIX"  
OS\_61 : String = "OS.IBM.OS/400"  
OS\_62 : String = "OS.IBM.Linux"  
OS\_63 : String = "OS.IBM.zOS"  
OS\_64 : String = "OS.IBM.OS/390"  
OS\_65 : String = "OS.IBM.OS/2"  
OS\_70 : String = "OS.Sun.Solaris"  
OS\_71 : String = "OS.Sun.Linux"  
OS\_80 : String = "OS.HP.HP-UX"  
OS\_81 : String = "OS.HP.Linux"  
OS\_82 : String = "OS.HP.MPE/iX"  
OS\_90 : String = "OS.RedHat.Linux"  
OS\_100 : String = "OS.SuSE.Linux"

## hyadesOperatingSystemCategory

*Derived from CFGCategory*

Public Properties:

---

name : String = "OS"  
displayName : String = "Operating System"  
selectionMode : CFGCategorySelectionMode = selectValue

## hyadesProcessorTypeCategory

*Derived from CFGCategory*

Public Properties:

---

name : String = "PROC\_TYPE"  
displayName : String = "Processor Type"  
selectionMode : CFGCategorySelectionMode = selectValue

## hyadesMemorySizeCategory

*Derived from CFGCategory*

Public Properties:

---

name : String = "MEM\_SZ"  
displayName : String = "Memory Size (MB)"  
selectionMode : CFGCategorySelectionMode = enterValue

## CFGCategorySelectionMode

## hyadesProcessorSpeedCategory

*Derived from CFGCategory*

Public Properties:

---

name : String = "PROC\_SPD"  
displayName : String = "Processor Speed"  
selectionMode : CFGCategorySelectionMode = enterValue

## hyadesProcessorTypeEnumeration

*Derived from CFGPsudoEnumeration*

### Public Properties:

---

category : CFGCategory = hyadesProcessorTypeCategory  
PR\_1 : String = "PR.Intel.8086"  
PR\_2 : String = "PR.Intel.8088"  
PR\_3 : String = "PR.Intel.80286"  
PR\_4 : String = "PR.Intel.80386"  
PR\_5 : String = "PR.Intel.80486"  
PR\_6 : String = "PR.Intel.Pentium"  
PR\_7 : String = "PR.Intel.Pentium Pro"  
PR\_8 : String = "PR.Intel.Pentium MMX"  
PR\_9 : String = "PR.Intel.Pentium II"  
PR\_10 : String = "PR.Intel.Pentium II Xeon"  
PR\_11 : String = "PR.Intel.Pentium III"  
PR\_12 : String = "PR.Intel.Pentium III Xeon"  
PR\_13 : String = "PR.Intel.Pentium III E"  
PR\_14 : String = "PR.Intel.Pentium 4"  
PR\_15 : String = "PR.Intel.Pentium 4 Xeon"  
PR\_16 : String = "PR.Intel.Pentium 4 Mobile"  
PR\_17 : String = "PR.Intel.Pentium M"  
PR\_18 : String = "PR.Intel.Pentium Celeron"  
PR\_19 : String = "PR.Intel.Itanium"  
PR\_50 : String = "PR.Motorola.68000"  
PR\_51 : String = "PR.Motorola.68020"  
PR\_52 : String = "PR.Motorola.68030"  
PR\_53 : String = "PR.Motorola.68040"  
PR\_54 : String = "PR.Motorola.68050"  
PR\_55 : String = "PR.Motorola.PowerPC 601"  
PR\_56 : String = "PR.Motorola.PowerPC 603"  
PR\_57 : String = "PR.Motorola.PowerPC 605"  
PR\_58 : String = "PR.Motorola.PowerPC G3"  
PR\_59 : String = "PR.Motorola.PowerPC G4"  
PR\_60 : String = "PR.Motorola.PowerPC G5"

## hyadesBrowserTypeCategory

*Derived from CFGCategory*

### Public Properties:

---

name : String = "WBR\_TYPE"  
displayName : String = "Web Browser Type"  
selectionMode : CFGCategorySelectionMode = selectValue

## hyadesBrowserVersionCategory

*Derived from CFGCategory*

Public Properties:

---

name : String = "WBR\_VER"  
displayName : String = "Web Browser Version"  
selectionMode : CFGCategorySelectionMode = enterValue

## hyadesBrowserEnumeration

*Derived from CFGPsudoEnumeration*

Public Properties:

---

category : CFGCategory = hyadesBrowserTypeCategory  
WB\_1 : String = "WB.Microsoft.Internet Explorer"  
WB\_2 : String = "WB.Netscape.Netscape"  
WB\_3 : String = "WB.Mozilla.Mozilla"  
WB\_4 : String = "WB.Apple.Safari"  
WB\_5 : String = "WB.Opera.Opera"  
WB\_6 : String = "WB.NCSA.Mosaic"  
WB\_7 : String = "WB.ISC.Lynx"

## hyadesProcessorNumberCategory

*Derived from CFGCategory*

Public Properties:

---

name : String = "PROC\_NUM"  
displayName : String = "Number of Processors"  
selectionMode : CFGCategorySelectionMode = enterValue

## hyadesDisplayColorDepthCategory

*Derived from CFGCategory*

Public Properties:

---

name : String = "DISP\_DEPTH"  
displayName : String = "Display Color Depth (Bits)"  
selectionMode : CFGCategorySelectionMode = selectAndEnterValue

## hyadesDisplayColorDepthEnumeration

*Derived from CFGPseudoEnumeration*

Public Properties:

---

category : CFGCategory = hyadesDisplayColorDepthCategory  
DCD\_1 : String = "1"  
DCD\_2 : String = "4"  
DCD\_3 : String = "8"  
DCD\_4 : String = "16"  
DCD\_5 : String = "24"  
DCD\_6 : String = "32"  
DCD\_7 : String = "64"  
DCD\_8 : String = "128"

## hyadesDisplayHeightCategory

*Derived from CFGCategory*

Public Properties:

---

name : String = "DISP\_HEIGHT"  
displayName : String = "Display Height (Pixels)"  
selectionMode : CFGCategorySelectionMode = selectAndEnterValue

## hyadesDisplayWidthCategory

*Derived from CFGCategory*

Public Properties:

---

name : String = "DISP\_WIDTH"  
displayName : String = "Display Width (Pixels)"  
selectionMode : CFGCategorySelectionMode = selectAndEnterValue

## hyadesDisplayHeightEnumeration

*Derived from CFGPseudoEnumeration*

Public Properties:

---

category : CFGCategory = hyadesDisplayHeightCategory  
DH\_1 : String = "480"  
DH\_2 : String = "600"  
DH\_3 : String = "768"  
DH\_4 : String = "1024"  
DH\_5 : String = "1050"  
DH\_6 : String = "1200"  
DH\_7 : String = "1440"  
DH\_8 : String = "1536"

## hyadesDisplayWidthEnumeration

*Derived from CFGPseudoEnumeration*

Public Properties:

---

category : CFGCategory = hyadesDisplayWidthCategory  
DW\_1 : String = "640"  
DW\_2 : String = "800"  
DW\_3 : String = "1024"  
DW\_4 : String = "1280"  
DW\_5 : String = "1400"  
DW\_6 : String = "1600"  
DW\_7 : String = "1920"  
DW\_8 : String = "2048"

## hyadesDisplayNumberCategory

*Derived from CFGCategory*

Public Properties:

---

name : String = "DISP\_NUM"  
displayName : String = "Number of Displays"  
selectionMode : CFGCategorySelectionMode = enterValue

## hyadesDatabaseCategory

*Derived from CFGCategory*

Public Properties:

---

name : String = "DB"  
displayName : String = "Database"  
selectionMode : CFGCategorySelectionMode = selectValue

## hyadesDatabaseVersionCategory

*Derived from CFGCategory*

### Public Properties:

---

name : String = "DB\_VER"  
displayName : String = "Database Version"  
selectionMode : CFGCategorySelectionMode = enterValue

## hyadesDatabaseEnumeration

*Derived from CFGPseudoEnumeration*

### Public Properties:

---

category : CFGCategory = hyadesDatabaseCategory  
DB\_1 : String = "DB.IBM.DB2"  
DB\_2 : String = "DB.Oracle.Oracle"  
DB\_3 : String = "DB.Microsoft.Access"  
DB\_4 : String = "DB.Sybase.SQLAnywhere"

## CFGMachineInstance

*Derived from CMNNodeInstance*

## CFGMachineConstraint

*Derived from CMNNodeType*

## CFGCategorySelectionAmount

## CFGCategorySelectionMultiplicity

## CFGConfigurableObject

*Derived from CMNNamedElement*

## hyadesHostnameCategory

*Derived from CFGCategory*

**Public Properties:**

---

**name** : String = "HNAME"  
**displayName** : String = "Hostname"  
**selectionMode** : CFGCategorySelectionMode = enterValue

## hyadesWindowsDomainCategory

*Derived from CFGCategory*

**Public Properties:**

---

**name** : String = "WIN\_DOM"  
**displayName** : String = "Windows Domain"  
**selectionMode** : CFGCategorySelectionMode = enterValue

## hyadesUsernameCategory

*Derived from CFGCategory*

**Public Properties:**

---

**name** : String = "USR\_NAME"  
**displayName** : String = "Username"  
**selectionMode** : CFGCategorySelectionMode = enterValue

## hyadesServicePackCategory

*Derived from CFGCategory*

### Public Properties:

---

**name** : String = "SVC\_PAC"  
**displayName** : String = "Service Pack"  
**selectionMode** : CFGCategorySelectionMode = enterAndSelect

## hyadesMajorVersionCategory

*Derived from CFGCategory*

### Public Properties:

---

**name** : String = "MAJ\_VER"  
**displayName** : String = "Major Version Number"  
**selectionMode** : CFGCategorySelectionMode = enterValue

## hyadesMinorVersionCategory

*Derived from CFGCategory*

### Public Properties:

---

**name** : String = "MIN\_VER"  
**displayName** : String = "Minor Version Number"  
**selectionMode** : CFGCategorySelectionMode = enterValue

## hyadesServicePackEnumeration

*Derived from CFGPseudoEnumeration*

Public Properties:

---

category : CFGCategory = hyadesServicePackCategory  
SP\_1 : String = "1"  
SP\_2 : String = "2"  
SP\_3 : String = "3"  
SP\_4 : String = "4"  
SP\_5 : String = "5"  
SP\_6 : String = "6"  
SP\_7 : String = "7"  
SP\_8 : String = "8"  
SP\_9 : String = "9"  
SP\_100 : String = "1a"  
SP\_101 : String = "1b"  
SP\_200 : String = "2a"  
SP\_201 : String = "2b"  
SP\_300 : String = "3a"  
SP\_301 : String = "3b"  
SP\_400 : String = "4a"  
SP\_401 : String = "4b"  
SP\_500 : String = "5a"  
SP\_501 : String = "5b"  
SP\_600 : String = "6a"  
SP\_601 : String = "6b"  
SP\_700 : String = "7a"  
SP\_701 : String = "7b"  
SP\_800 : String = "8a"  
SP\_801 : String = "8b"  
SP\_900 : String = "9a"  
SP\_901 : String = "9b"

### CFGArtifactLocationPair

### CFGMachine

*Derived from CFGLocation*

Public Properties:

---

hostName : String

### CFGPseudoEnumeration

Public Properties:

---

values : String  
category : CFGCategory

## hyadesPasswordCategory

*Derived from CFGCategory*

### Public Properties:

---

**name** : String = "PASSWD"  
**displayName** : String = "Password"  
**selectionMode** : CFGCategorySelectionMode = password

## hyadesClasspathCategory

*Derived from CFGCategory*

### Public Properties:

---

**name** : String = "CLSPATH"  
**displayName** : String = "Classpath"  
**selectionMode** : CFGCategorySelectionMode = enterValue

## hyadesRootDirectoryCategory

*Derived from CFGCategory*

### Public Properties:

---

**name** : String = "ROOTDIR"  
**displayName** : String = "Root Directory"  
**selectionMode** : CFGCategorySelectionMode = enterValue

## datapool

## DPLDatapoolSpec

*Derived from CMNNamedElement*

**DPLEquivalenceClass**

*Derived from CMNNamedElement*

**DPLRecord**

**DPLCell**

**DPLVariable**

*Derived from CMNNamedElement*

**DPLRole**

**DPLDatapool**

*Derived from CFGClass*  
*Derived from CMNNamedElement*

**common**

Internal package, please do not extend or use.

The common package provides the base classes that are similar to the UML concepts for defining classes and behaviour. This package is not intended to provide a general meta model for these concepts. It will be replaced if a UML model were to appear in Eclipse that can be used.

### CMNNamedElement

**Public Properties:**

---

**id** : String  
**description** : String  
**name** : String

### CMNMachine

*Derived from CMNNodeType*

### CMNNodeType

*Derived from CFGLocation*

**Public Properties:**

---

**hostname** : String

### CMNNodeInstance

*Derived from CFGLocation*

**Public Properties:**

---

**hostname** : String

### CMNExtendedProperty

This class stores the name, type and value of an extended property. The type field specifies an XSD datatype, and the value specifies an XML instance of an object of the specified type.

*Derived from CMNDefaultProperty*

**Public Properties:**

---

**name** : String

**type** : String

The type field specifies an XSD type from xmlns="http://www.w3.org/2001/XMLSchema"

**value** : String

### **CMNDefaultProperty**

CMNDefaultProperty is a property class with no methods and no attributes. Its role is as a tag interface within the model, to allow for future use of SDOs or Dynamic EMF classes to be used as properties.

Any Hyades participating vendor should discuss the use of this class with the Test Model team before utilizing it. This class should only be used to store properties that are not of general interest to the test model. Any properties that are of general interest should be discussed with the test model team, and upon agreement, added to the model directly.

### **CMNAnnotation**

A CMNAnnotation represents a model annotation which is stored as a named file attachment in the same EMF resource as the model element that has been annotated. Each annotation has a name and a URI.

*Derived from CMNNamedElement*

**Public Properties:**

---

**URI** : String

**type** : String

### **sdb**

Internal package, please do not extend or use.

The symptom data base is used to contain symptoms to be used by an analysis engine whne analyzing common base events

### **SDBRuntime**

A runtime instance typically represents a product or a system, such as a Relational Database or a web server.

### Public Properties:

---

**id : String**

Each runtime is uniquely identified by this value.

**name : String**

The name hold a short human readable title for the runtime. For example "MyRDB product V6"

**symptomUrl : String**

The symptom data base can have a URL associated with it to identify where it was downloaded from. This is a typical model to support refreshing of the DB.

**localExternalFileLocation : String**

A symptom DB for a given product can be provided as a local file, and this property captures the location as a convenience for refreshing.

**description : String**

A description of the runtime represented. This is human readable information and is determined by the runtime provider.

### SDBSymptom

A symptom is a unique set of match criteria that can be used to identify a problem. The uniqueness of the symptom is up to the instance provider.

### Public Properties:

---

**description : String**

A human readable description of the problem is captured in this property. An example may be "system hangs after loading file X"

**id : String**

The id of a symptom uniquely identifies a symptom within a runtime. It is actually the only thing that must be unique within the set of symptoms for a given runtime.

### SDBMatchPattern

Match patterns are used in sequence to determine if an event being analyzed meets the criteria of a symptom.

### Public Properties:

---

**name : String**

The name property holds the name of the field in an event that should be compared to the value as a test for this particular match pattern.

**value : String**

Value holds the literal string being used in the comparison.

### SDBSolution

A solution is an action that can be taken to address the symptom. A solution can take the form of a message to a user or automated actions, and that is all defined by the directives.

Solutions are owned by and unique to a runtime.

### Public Properties:

#### **description : String**

A human readable brief description an answer to the problem. For example a symptom that captures a performance throughput problem may have a solution of "adjust the size fo the thread pool". This solution may be a fix for several different symptoms, and it may be detailed with several directives.

#### **id : String**

A solution is unque within a runtime environment.

## SDBDirective

Directives are the actual steps or actions that are recommended for a given solution.

Although not required to be mutually exclusive the description and directive string are often not both used at the same time.

### Public Properties:

#### **id : String**

A directive has a unique identifier relative to the runtime that owns it.

#### **description : String**

In the case that the action or directive take the form of a textual instruction for a user, that text would be held in this property. An example might be "power on the server".

#### **directiveString : String**

A directive string is intended to hold a meaningful string for the runtime to use in an automated or at least runtime unique reaction ot the symptom.

For example the directive may contain a class name that the runtime could load and execute in order to collect more data or automate an adjustment.

## hierarchy

Internal package, please do not extend or use.

The "hierarchy" package is used to capture the base topology used to provide a hierarchical view that is used to organize and interact with the various agent contents.

## TRCCollectionMode

Public Properties:

---

HEAP\_STATISTICS\_ONLY : = 0  
HEAP\_FULL :  
HEAP\_AND\_EXECUTION\_FULL :  
HEAP\_AND\_EXECUTION\_STATISTICS\_ONLY :  
EXECUTION\_STATISTICS\_ONLY :  
EXECUTION\_NO\_INSTANCES :  
EXECUTION\_FULL :  
HEAP\_FULL\_AND\_EXECUTION\_STATISTICS\_ONLY :

### TRCProcessProxy

The proxy for a real OS process. Has properties that represent the execution context. The reason this is referred to as a proxy is that if an agent is actually collecting an execution trace, it will contain the relevant details at runtime of the process that is being monitored.

Public Properties:

---

**name** : String  
**runtimeId** : String  
**pid** : int  
**classpath** : String  
The process classpath  
**parameters** : String  
The process command-line parameters  
**launchMode** : int  
The launching mode for the current process,  
1 if the trace was attached to the process, 0 if the  
process was launched by the trace process.  
**location** : String  
The initial current directory for the process  
**vmArguments** : String  
**active** : boolean

### TRCOption

This is a name/value pair, used to define options for the profiling agent. Eg.:  
key="ALLOCATION\_INFORMATION" optionValue="all"

Public Properties:

---

**key** : String  
A name used to identify an option  
**value** : String  
The value of the option

### TRCAgent

An agent is analogous to a "trace file". An agent is typed to hold a particular type of trace, and the TRCAgent object owns by value the instances of the trace data. Because agents are typed, there can be zero or more agents associated with a given TRCMonitor, TRCNode or TRCProcessProxy. However, a given agent instance can only be associated

with one of those objects.

This agent is the model entity for a data provider. Providers provide metadata as a collection of descriptors and the data itself as observations.

The agent is also the granularity of persistence.

### Public Properties:

---

**name : String**

A descriptive agent name, like "Java Profiler Agent"

**type : String = \_unknown**

The type of the agent, eg. "Profiler"

**runtimeId : String**

A GUID based ID to identify an monitoring agent instance

**startTime : double**

Marks the starting point of the collecting session

**stopTime : double**

Marks the end point of the collecting session

**collectionMode : TRCCollectionMode = 0**

Defines different collection modes (some include filtering of data on agent side some on loader side and some are combined). This is used by loaders and UI to decide the model building algorithms and the presentation mode. This is has the same value as

TRCAgentProxy.collectionMode attribute.

**version : String**

The version of an agent (for example "1.0.0")

## TRCAgentProxy

A TRCAgent proxy, to enable support for loading and unloading the agent resource. This class shadows most of the properties of the TRCAgent: please check the documentation on TRCAgent for duplicated fields.

### Public Properties:

---

**name : String**

**type : String = \_unknown**

**runtimeId : String**

**startTime : double**

**stopTime : double**

**collectionMode : TRCCollectionMode = 0**

Defines different collection modes (some include filtering of data on agent side some on loader side and some are combined). This is used by loaders and UI to decide the model building algorithms and the presentation mode.

**active : boolean = false**

**attached : boolean = false**

**collectionData : boolean = false**

**monitored : boolean = false**

**profileFile : String**

### Public Methods:

---

**boolean isToProfileFile();**

### TRCConfiguration

A configuration is used to hold a set of options and filters. This would normally be used to capture a set of options and filters that are active for a given agent.

#### Public Properties:

---

**name** : String  
**active** : boolean

### TRCEnvironmentVariable

A name-value pair to capture an environment variable. For example classpath

#### Public Properties:

---

**name** : String  
**value** : String

### TRCExecParameter

#### Public Properties:

---

**key** : String  
**value** : String

### TRCFilter

A filter describes a pattern that can be used during data collection to control the data being collected.

#### Public Properties:

---

**type** : String  
**pattern** : String  
**mode** : String  
**active** : Boolean  
**operation** : String = ""

### TRCNode

A Node is a machine, or at least a machine execution partition. It owns process proxies for processes with agents attached and data collected.

There is one server/service installed for each Node. An analogy is a service running on NT that hooks to the process being monitored.

### Public Properties:

---

**runtimeId** : String

**timezone** : double

**port** : int = 10002

**deltaTime** : double

Used to synchronize entry, exit events between different nodes, on a distributed application. It is in microseconds.

**name** : String

The machine/host name

**description** : String

**ipAddress** : String

The IP address of the node

### TRCMonitor

A monitor is in effect a complete set of traces that may have been collected from one or more agents. It is the logical root for a persisted model instance that may reflect a complete distributed application or simply a set of data collection points.

### Public Properties:

---

**name** : String

A user provider name

**startTime** : double

The overall start time (the earliest start time of all the agents in this monitor)

**stopTime** : double

The overall stop time (the latest stop time of all the agents in this monitor)

### AbstractDefaultEvent

This class allows for a clean plugin dependency packaging. Hyades needs to allow different plugins for each model sub-package in order to avoid all of them having to always be present and possibly loaded.

### AbstractTRCView

This class allows for a clean plugin dependency packaging. Hyades needs to allow different plugins for each model sub-package in order to avoid all of them having to always be present and possibly loaded.

### AbstractTRCDescription

This class allows for a clean plugin dependency packaging. Hyades needs to allow different plugins for each model sub-package in order to avoid all of them having to always be present and possibly loaded.

### AbstractTRCProcess

This class allows for a clean plugin dependency packaging. Hyades needs to allow different plugins for each model sub-package in order to avoid all of them having to always be present and possibly loaded.

### AbstractTRCCollectionBoundary

This class allows for a clean plugin dependency packaging. Hyades needs to allow different plugins for each model sub-package in order to avoid all of them having to always be present and possibly loaded.

#### Public Properties:

---

**name** : String

**startTime** : double

The creation time of this object, relative to hierarchy.TRCAgent.startTime

**collectionMode** : TRCCollectionMode = 0

Defines different collection modes (some include filtering of data on agent side some on loader side and some are combined). This is used by loaders and UI to decide the model building algorithms and the presentation mode.

### UnresolvedCorrelation

Class used to persist the unresolved correlations. When a correlation is resolved the object should be removed from the list.

#### Public Properties:

---

**contextId** : String

String based look-up key (eg. in Trace submodel case this is a concatenation of runtimeIds, ticket, sequence and threadId). The schema of this ID can be arbitrary chosen and has to be known by both the producer and the consumer of this class.

### CorrelationSourceInfo

Describes the known information from the received event.

### CorrelationContainer

This is a root in a correlation instance resource. For example the correlation instance is created when a correlation operation/action is used or by the loaders when an

AssociationEngine event is received.

### CorrelationContainerProxy

A UI proxy class, to avoid loading the whole correlation instance resource.

#### Public Properties:

---

**name** : String

**creationTime** : long

### CorrelationEntry

Represents a 1-many relation between a source event and it's associated events (an event can be a CBE, method invocation etc)

### CorrelationEngine

The id should uniquely identify the engine.

#### Public Properties:

---

**type** : String

Engines are typed with this attribute. Examples of types are "Correlation based on URI", "root"

**name** : String

Each engine has a name. This is not ensured to be unique. The name is intended to be human readable.

**id** : String

The id of an engine is something that uniquely identifies the engine from all other engines installed.

The uniqueness is managed outside the model and is the responsibility of the engine provider.

### BooleanArray

#### Public Properties:

---

**boolean[]** :

### IntArray

Public Properties:

int[] :

## data

### TSTDataPool

A DataPool is a grouping mechanism aimed at holding a set of Equivalence Classes themselves grouping a set of Instance Values.

A DataPool is referenced by TestSuite making use of it.

### TSTPicker

A Picker is a class realizing the association between a message and the corresponding Instance Values.

The Picker role is to decide at runtime which values need to be picked for a given message.

### TSTEquivalenceClass

An Equivalence class is a grouping mechanism for a set of Instance Values related. The relationship between those Instance Values is generally that their outcome is the same from the standpoint of the TestCase where they are used.

## extensions

Resource extensions

## Query

## SimpleSearchQuery

*Derived from Query*

## InstanceQuery

A simple query, in XMI case could have an optimized implementation

*Derived from Query*

## OrderByElement

## QueryResult

Mixed list of output elements instances.

## SimpleBinaryExpression

## LeftOperand

## RelationalOperators

## OrderByOperators

## ResultEntry

## CorrelationQuery

*Derived from SimpleSearchQuery*

## TimeBasedCorrelationQuery

*Derived from CorrelationQuery*

## fragments

### **BVRInteractionOccurrence**

An InteractionOccurrence refers to an Interaction. The InteractionOccurrence is a shorthand for copying the contents of the referred Interaction where the InteractionOccurrence is.

It is common to want to share portions of an interaction between several other interactions. An InteractionOccurrence allows multiple interactions to reference an interaction that represents a common portion of their specification.

*Derived from BVRInteractionFragment*

### **BVRPartDecomposition**

PartDecomposition is a description of the internal interactions of one Lifeline relative to an Interaction.

A Lifeline represents a DeployableInstance. A DeployableInstance is compositional and therefore its constituents interactions might be described using a PartDeocomposition.

*Derived from BVRInteractionOccurrence*

### **BVRInteractionOperand**

An InteractionOperand is contained in a CombinedFragment. An InteractionOperand represent one operand of the expression given by the enclosing CombinedFragment.

An InteractionOperand is an InteractionFragment with an optional guard expression. An InteractionOperand may be guarded by a InteractionConstraint. Only InteractionOperands with a guard that evaluates to true at this point in the interaction will be considered for the enclosing CombinedFragment.

*Derived from BVRInteractionFragment*

### **BVRInteractionConstraint**

An InteractionConstraint is a boolean expression that guards an operand in a CombinedFragment.

**Public Properties:**

---

**description** : String  
**constraint** : String

### **BVRInteractionOperator**

The InteractionOperator is an enumerated type holding the different kinds of oeprators to be used in a Combined Fragment.

- alt designates that the CombinedFragment represents a choice of behavior.
- opt designates that the CombinedFragment represents a choice of behavior where either the (sole) operand happens or nothing happens.
- par designates that the CombinedFragment represents a parallel merge between the behaviors of the operands. The eventoccurrences of the different operands can be interleaved in any way as long as the ordering imposed by each operand as such is preserved.
- loop designates that the CombinedFragment represents a loop. The loop operand will be repeated a number of times.

- neg designates that the CombinedFragment represents traces that are defined to be invalid.

### Public Properties:

---

alt :  
opt :  
par :  
loop :  
neg :

## BVRGate

A Gate is a connection point for relating a Message outside an InteractionFragment with a Message inside the InteractionFragment.

This allows the definition of Interaction Fragments as reusable blocks., that can be further linked using Gates.

*Derived from BVRMessageEnd*

## BVRCombinedFragment

A combined fragment defines an expression of interaction fragments. A combined fragment is defined by an interaction operator and corresponding interaction operands. Through the use of CombinedFragments the user will be able to describe a number of traces in a compact and concise manner.

*Derived from BVRInteractionFragment*

### Public Properties:

---

interactionOperator : BVRInteractionOperator

## BVRInteraction

## interactions

### BVRLifeline

A lifeline represents an individual participant in the Interaction. Participants represents instances of Deployable.

*Derived from CMNNamedElement*

### BVRInteractionFragment

InteractionFragment is an abstract notion of the most general interaction unit. An interaction fragment is a piece of an interaction. Each interaction fragment is conceptually like an interaction by itself.

The containers of an InteractionFragment are:

- Behavior
- CombinedFragment
- InteractionOperand

*Derived from CMNNamedElement*

### BVRMessage

A Message defines a particular communication between Lifelines of an Interaction.

A Message defines one specific kind of communication in an Interaction. A communication can be e.g. raising a signal, invoking an operation, creating or destroying an instance. The Message specifies not only the kind of communication given by the dispatching ExecutionOccurrence, but also the sender and the receiver.

A Message associates normally two EventOccurrences - one sending EventOccurrence and one receiving Event-Occurrence.

#### Public Properties:

**messageSort** : BVRMessageSort

### BVRGeneralOrdering

A GeneralOrdering represents a binary relation between two Eventoccurrences, to describe that one Eventoccurrence must occur before the other.

This allows for instance the synchronization between two test components.

### **BVREventOccurrence**

EventOccurrences represents moments in time to which Actions are associated. An EventOccurrence is the basic semantic unit of Interactions. The sequences of Eventoccurrences are the meanings of Interactions. Messages are sent through either asynchronous signal sending or operation calls. Likewise they are recieved by Receptions or actions of consumption.

*Derived from BVRInteractionFragment*

### **BVRMessageEnd**

A MessageEnd is an abstract concept that represents what can occur at the end of a Message.

*Derived from BVREventOccurrence*

### **BVRMessageSort**

The MessageSort captures the different kind of messages that can be used in an Interaction.

**Public Properties:**

---

synchCall :  
synchSignal :  
asynchCall :  
asynchSignal :

### **BVRExecutionOccurrence**

An ExecutionOccurrence is an instantiation of a unit of behavior within the Lifeline. An example of such a behavior would be the invocation of a pre-existing behavior of the DeployableInstance to which the Lifeline belongs.

Since the ExecutionOccurrence will have some duration, it is represented by two Eventoccurrences, the start EventOccurrence and the finish Event-Occurrence.

*Derived from BVRInteractionFragment*

### BVRStateInvariant

A StateInvariant is a constraint on the state of a Lifeline. In this case we mean by state also the values of eventual attributes of the DeployableInstance the Lifeline refers to.

*Derived from BVRInteractionFragment*

### BVRStop

A Stop is an EventOccurrence that defines the termination of the instance specified by the Lifeline on which the Stop occurs

*Derived from BVREventOccurrence*

### BVRProperty

*Derived from CMNNamedElement*

Public Properties:

---

value : String

### ecore

### EAttribute

*Derived from EStructuralFeature*

Public Properties:

---

iD : boolean

### EAnnotation

*Derived from EModelElement*

Public Properties:

source : String  
details : EStringToStringMapEntry

---

## EClass

*Derived from EClassifier*

Public Properties:

abstract : boolean  
interface : boolean

---

Public Methods:

boolean isSuperTypeOf(EClass someClass);  
EStructuralFeature getEStructuralFeature(int featureID);  
EStructuralFeature getEStructuralFeature(String featureName);

---

## EClassifier

*Derived from ENamedElement*

Public Properties:

instanceClassName : String  
instanceClass : EJavaClass  
defaultValue : EJavaObject

---

Public Methods:

boolean isInstance(EJavaObject object);  
int getClassifierID();

---

## EDataType

*Derived from EClassifier*

Public Properties:

serializable : boolean = true

---

## EEnum

*Derived from EDataType*

Public Methods:

---

EEnumLiteral getEEnumLiteral(String name);  
EEnumLiteral getEEnumLiteral(int value);

## EEnumLiteral

*Derived from ENamedElement*

Public Properties:

---

value : int  
instance : EEnumerator

## EFactory

*Derived from EModelElement*

Public Methods:

---

EObject create(EClass eClass);  
EObject createFromString(EDataType eDataType, String literalValue);  
String convertToString(EDataType eDataType, EJavaObject instanceValue);

## EModelElement

*Derived from EObject*

Public Methods:

---

EAnnotation getEAnnotation(String source);

## ENamedElement

*Derived from EModelElement*

Public Properties:

---

name : String

## EObject

### Public Methods:

---

```
EClass eClass();
boolean eIsProxy();
EResource eResource();
EObject eContainer();
EStructuralFeature eContainingFeature();
EReference eContainmentFeature();
EEList eContents();
ETreeIterator eAllContents();
EEList eCrossReferences();
EJavaObject eGet(EStructuralFeature feature);
EJavaObject eGet(EStructuralFeature feature, boolean resolve);
eSet(EStructuralFeature feature, EJavaObject newValue);
boolean eIsSet(EStructuralFeature feature);
eUnset(EStructuralFeature feature);
```

## EOperation

*Derived from ETypedElement*

## EPackage

*Derived from ENamedElement*

### Public Properties:

---

```
nsURI : String
nsPrefix : String
```

### Public Methods:

---

```
EClassifier getEClassifier(String name);
```

## EParameter

*Derived from ETypedElement*

## EReference

*Derived from EStructuralFeature*

### Public Properties:

---

containment : boolean  
container : boolean  
resolveProxies : boolean = true

## EStructuralFeature

*Derived from ETypedElement*

### Public Properties:

---

changeable : boolean = true  
volatile : boolean  
transient : boolean  
defaultValueLiteral : String  
defaultValue : EJavaObject  
unsettable : boolean  
derived : boolean

### Public Methods:

---

int getFeatureID();  
EJavaClass getContainerClass();

## ETypedElement

*Derived from ENamedElement*

### Public Properties:

---

ordered : boolean = true  
unique : boolean = true  
lowerBound : int  
upperBound : int = 1  
many : boolean  
required : boolean

## EBigDecimal

### Public Properties:

---

java.math.BigDecimal :

## EBigInteger

Public Properties:  
java.math.BigInteger :

---

## EBoolean

Public Properties:  
boolean :

---

## EBooleanObject

Public Properties:  
java.lang.Boolean :

---

## EByte

Public Properties:  
byte :

---

## EByteArray

Public Properties:  
byte[] :

---

## EByteObject

Public Properties:

[java.lang.Byte](#) :

---

## EChar

Public Properties:

[char](#) :

---

## ECharacterObject

Public Properties:

[java.lang.Character](#) :

---

## EDate

Public Properties:

[java.util.Date](#) :

---

## EDiagnosticChain

Public Properties:

[org.eclipse.emf.common.util.DiagnosticChain](#) :

---

## EDouble

Public Properties:

[double](#) :

---

## EDoubleObject

Public Properties:

[java.lang.Double](#) :

## EEList

Public Properties:

[org.eclipse.emf.common.util.EList](#) :

## EEnumerator

Public Properties:

[org.eclipse.emf.common.util.Enumerator](#) :

## EFeatureMap

Public Properties:

[org.eclipse.emf.ecore.util.FeatureMap](#) :

## EFeatureMapEntry

Public Properties:

[org.eclipse.emf.ecore.util.FeatureMap\\$Entry](#) :

## EFloat

Public Properties:

[float](#) :

## EFloatObject

Public Properties:

---

java.lang.Float :

## EInt

Public Properties:

---

int :

## EIntegerObject

Public Properties:

---

java.lang.Integer :

## EJavaClass

Public Properties:

---

java.lang.Class :

## EJavaObject

Public Properties:

---

java.lang.Object :

## ELong

Public Properties:

---

long :

## ELongObject

Public Properties:

---

[java.lang.Long](#) :

## EMap

Public Properties:

---

[java.util.Map](#) :

## EResource

Public Properties:

---

[org.eclipse.emf.ecore.resource.Resource](#) :

## EResourceSet

Public Properties:

---

[org.eclipse.emf.ecore.resource.ResourceSet](#) :

## EShort

Public Properties:

---

[short](#) :

## EShortObject

Public Properties:

java.lang.Short :

---

## EString

Public Properties:

java.lang.String :

---

## EStringToStringMapEntry

Public Properties:

key : String  
value : String

---

## ETreeIterator

Public Properties:

org.eclipse.emf.common.util.TreeIterator :

---

## trace

The purpose of the trace model is to capture execution behavior of an application. This includes call stack tracing as well as object/memory observation. All over multi process/tier environments.

## TRCObject

Represents an instance of a type (defined by TRCClass)

Public Properties:

id : EObjectID

This is a static ID for the object. It must be unique in the trace.

size : int

The size of the array (number of bytes required to store this array, not the elements in it), computed based on the element type, see TRCArrayType

## TRCClass

This represents a complex type (a Java class for example)

- calls, baseTime, and cumulativeTime: these are aggregate measures of all the method invocations whose receiver object is of the given class (or package, or process). For cumulativeTime, time spent in any method invocations that call each other is not counted more than once for that class (or package, or process).

- inheritedCalls, inheritedBaseTime, and inheritedCumulativeTime: these are aggregate measures of all the method invocations whose receiver object is of the given class (or package, or process), but limited to those whose method is not implemented in the class of the receiver object (i.e. it is implemented in a class further up in the class hierarchy).

### Public Properties:

---

**id : EClassID**

An unique ID for this class during a monitoring session. By persisting this we can have sessions with disconnected segments, e.g. attach, start monitoring, pause monitoring, shutdown client, restart client, re-attach to same session and continue monitoring.

**name : String**

The class name, without the package qualified part of the name. In the case of synthetic names they will be captured in the form that is meaningful to the synthesizer.

Classes where the name is unknown will have a class name of \_unknown.

**size : int**

The size of one instance of this class. If the class is an array, this field specifies the size of the type of the elements in the array and the size of array it is specified in the size field of TRCArrayObject. [Note: the representation of array types is undergoing design changes.]

**loadTime : double**

When the class was loaded, relative to hierarchy.TRCAgent.startTime

**unloadTime : double**

When the class was unloaded, relative to hierarchy.TRCAgent.startTime

**interface : boolean**

True if the class is an interface

**lineNo : int**

The line number in the source info where the declaration of this class starts.

**baseTime : double**

Time spent in methods of the owning element

**cumulativeTime : double**

Time spent in methods of this owning element + all methods called from these methods (but owned by other elements)

**calls : int**

number of calls to the owning element

**inheritedCalls : int**

Number of calls to methods defined in superclasses of classes from the owning element

**inheritedBaseTime : double**

Time spent in methods defined in superclasses of this classes from the owning element

**inheritedCumulativeTime : double**

Time spent in methods of classes from the owning element + all methods called from the methods defined in superclasses of classes from the owning element

**totalSize : int**

The total size of memory allocated

**totalInstances : int**

Number of instances created

**collectedSize : int**

The total size of memory garbage collected

**collectedInstances : int**  
Number of instances garbage collected

### TRCMethodInvocation

A method invocation is the entry and exit of the actual method implementation.

The base class does not track any attributes, for minimum overhead.

### TRCProcess

TRCProcess represents a process. In this case it is one that has been monitored. It is the root container of a profiling trace.

- calls, baseTime, and cumulativeTime: these are aggregate measures of all the method invocations whose receiver object is of the given class (or package, or process). For cumulativeTime, time spent in any method invocations that call each other is not counted more than once for that class (or package, or process).

- inheritedCalls, inheritedBaseTime, and inheritedCumulativeTime: these are aggregate measures of all the method invocations whose receiver object is of the given class (or package, or process), but limited to those whose method is not implemented in the class of the receiver object (i.e. it is implemented in a class further up in the class hierarchy).

*Derived from AbstractTRCProcess*

#### Public Properties:

---

**pid : int**

Process ID as defined natively on the execution platform

**name : String**

The name of the executable. The name is totally dependant on what the execution environment is able to provide, and what is collected by the agent. On some operating systems in an attach scenario for example the process name is simply not available, however in a scenario where the process is launched by the workbench, the tools can apply a robust name that is very meaningful to the end user.

**id : String**

A GUID based ID to identify a process instance

**startTime : double**

The time when the process was started, relative to hierarchy.TRCAgent.startTime

**stopTime : double**

The time when the process was stopped, relative to hierarchy.TRCAgent.startTime

**lastEventTime : double**

The time of the last event received on this process, relative to hierarchy.TRCAgent.startTime

**initFinishedTime : double**

The time when the JVM initialization ended

**baseTime : double**

Time spent in methods of the owning element

**cumulativeTime : double**

Time spent in methods of this owning element + all methods called from these methods (but owned by other elements)

**calls : int**  
number of calls to the owning element

**inheritedCalls : int**  
Number of calls to methods defined in superclasses of classes from the owning element

**inheritedBaseTime : double**  
Time spent in methods defined in superclasses of this classes from the owning element

**inheritedCumulativeTime : double**  
Time spent in methods of classes from the owning element + all methods called from the methods defined in superclasses of classes from the owning element

**totalSize : int**  
The total size of memory allocated

**totalInstances : int**  
Number of instances created

**collectedSize : int**  
The total size of memory garbage collected

**collectedInstances : int**  
Number of instances garbage collected

**referencePointerSize : short = 4**  
Represents the number of bytes used for a reference pointer.

**threadStates : String**  
The name of the executable. The name is totally dependant on what the execution environment is able to provide, and what is collected by the agent. On some operating systems in an attach scenario for example the process name is simply not available, however in a scenario where the process is launched by the workbench, the tools can apply a robust name that is very meaningful to the end user.

### TRCThread

A monitored thread, scoped by its owning process.

#### Public Properties:

---

**id : int**  
A unique thread ID assigned by the agent. Even if the monitored environment can re-use thread IDs, this must be unique.

**name : String**  
The name of the thread, to the extent it is available from the monitored environment.

**groupName : String**  
The thread's group name. This is Java specific

**startTime : double**  
The thread start time, relative to hierarchy.TRCAgent.startTime

**stopTime : double**  
The thread stop time, relative to hierarchy.TRCAgent.startTime

**maxStackDepth : int**  
The maximum stack depth reached in this thread during a specific monitoring session. A value of zero indicates the depth has not been calculated, or there has been no observed activity on this thread.

### TRCMethod

A method definition

### Public Properties:

---

**id : EMethodID**

See TRCClass.id

**name : String**

The name of the method. This can have language specific names like <init> and <clinit> in Java, which can be translated in more descriptive names (init will become the same as the name of the class).

In order to get a fully qualified name you can navigate to the class and package.

**signature : String**

The full signature of the method using the notation described in TRCMethod.notation. (eg. "(Ljava/lang/String;)Z" for "boolean methodName(String s)" in JNI notation).

**modifier : int**

This will be set with a combination of bitmasks defined by the values from TRCMethodProperties

**notation : TRCSignatureNotation**

The notation format (use to select the decoding mechanism of the signature, eg. "JNI" notation)

**lineNo : int**

The line number in the SourceInfo where this method declaration starts. This is only as precise as the data collection can provide. Depending on the compiler, it may be a starting brace, the start of the actual signature declaration, or something else.

**baseTime : double**

Time spent in methods of the owning element

**cumulativeTime : double**

Time spent in methods of this owning element + all methods called from these methods (but owned by other elements)

**calls : int**

number of calls to the owning element

## TRCPrimitiveType

### Public Properties:

---

**JAVA\_REFERENCE : = 1**

**JAVA\_BOOLEAN :**

**JAVA\_BYTE :**

**JAVA\_CHAR :**

**JAVA\_SHORT :**

**JAVA\_INT :**

**JAVA\_LONG :**

**JAVA\_FLOAT :**

**JAVA\_DOUBLE :**

## EObjectID

Custom EMF EDataType to support conversion from a special object ID format to long.

## EMethodID

Custom EMF EDataType to support conversion from a special method ID format to int.

## TRCPackage

Logical representation of a Java package

- `calls`, `baseTime`, and `cumulativeTime`: these are aggregate measures of all the method invocations whose receiver object is of the given class (or package, or process). For `cumulativeTime`, time spent in any method invocations that call each other is not counted more than once for that class (or package, or process).

- `inheritedCalls`, `inheritedBaseTime`, and `inheritedCumulativeTime`: these are aggregate measures of all the method invocations whose receiver object is of the given class (or package, or process), but limited to those whose method is not implemented in the class of the receiver object (i.e. it is implemented in a class further up in the class hierarchy).

### Public Properties:

---

**name : String = Default**

The fully qualified name of the package. The dot separated notation is used. This may become more complex once further language-neutrality is added to the model.

**baseTime : double**

Time spent in methods of the owning element

**cumulativeTime : double**

Time spent in methods of this owning element + all methods called from these methods (but owned by other elements)

**calls : int**

number of calls to the owning element

**inheritedCalls : int**

Number of calls to methods defined in superclasses of classes from the owning element

**inheritedBaseTime : double**

Time spent in methods defined in superclasses of this classes from the owning element

**inheritedCumulativeTime : double**

Time spent in methods of classes from the owning element + all methods called from the methods defined in superclasses of classes from the owning element

**totalSize : int**

The total size of memory allocated

**totalInstances : int**

Number of instances created

**collectedSize : int**

The total size of memory garbage collected

**collectedInstances : int**

Number of instances garbage collected

## TRCCollectionBoundary

A collection boundary (segment boundary) of a trace. This class marks the beginning of a segment. It can have a method invocation associated with it (the first method invocation that was received in that slice). The end of a segment is marked by the end of the trace or by the next segment.

*Derived from AbstractTRCCollectionBoundary*

### TRCClassLoader

Defines a class loader

### EClassID

Custom EMF EDataType to support conversion from a special class ID format to int.

### TRCSignatureNotation

#### Public Properties:

---

JNI : = 0

### TRCSourceInfo

This is an anchor point to capture the information about a specific piece of source relative to a specific file.

#### Public Properties:

---

**location : String**

This contains a URL kind of location (which can represent a query too) for the source file where the method/class is declared. Sometimes it can contain partial info (for example only the file name without the path).

**language : String**

This field specifies the programming language used in this source file. For classes with methods written in different languages you need to create a SourceInfo object per method in order to specify the language.

### TRCMethodProperties

Used as a bitmask for the TRCMethod.modifier field.

Public Properties:

---

JAVA\_NATIVE : = 1  
JAVA\_SYNCHRONIZED : = 2  
JAVA\_PUBLIC : = 4  
JAVA\_PRIVATE : = 8  
JAVA\_PROTECTED : = 16  
JAVA\_DEFAULT\_VISIBILITY : = 32  
JAVA\_STATIC : = 64  
JAVA\_CONSTRUCTOR : = 128  
JAVA\_ABSTRACT : = 256  
JAVA\_FINALIZER : = 512

## TRCHeapObject

*Derived from TRCObject*

## TRCFullTraceObject

*Derived from TRCTraceObject*

Public Properties:

---

**createTime** : double  
When the object was created (allocated), relative to hierarchy.TRCAgent.startTime  
**collectTime** : double  
When the object was collected by GC, relative to hierarchy.TRCAgent.startTime  
**baseTime** : double  
Time spent in methods of the owning element  
**cumulativeTime** : double  
Time spent in methods of this owning element + all methods called from these methods (but owned by other elements)  
**calls** : int  
number of calls to the owning element

## TRCTraceObject

*Derived from TRCObject*

## TRCFullHeapObject

*Derived from TRCHeapObject*

Public Properties:

heapDumpFirstSeen : short

### TRCObjectReference

### TRCHeapDump

Public Properties:

id : short  
entryTime : double  
exitTime : double  
name : String

### TRCAggregatedMethodInvocation

The TRCAggregatedMethodInvocation represents a collapsed view of several MethodInvocation along identical call chains, and maintains a count of how often the chain was repeated.

*Derived from TRCMethodInvocation*

Public Properties:

count : int

### TRCFullMethodInvocation

TRCFullMethodInvocation carries attributes about a method invocation that are used when the agent was tracking performance information about each function call.

*Derived from TRCMethodInvocation*

Public Properties:

stackDepth : short  
The true stack depth for this invocation, even counting stack frames which don't appear in the TRCMethodInvocation.invokes chain.  
entryTime : double  
Method entry time, relative to hierarchy.TRCAgent.startTime

**exitTime** : double

Method exit time, relative to hierarchy.TRCAgent.startTime

**ticket** : long

A ticket is a sequential incremental counter used to correlate invocations that can not be sorted by time alone. This allows for events to arrive out of sequence or for invocations to filtered out and skipped.

**overhead** : double = 0.0

Overhead introduced by tracing the application. Tracking this time on a per-invocation basis is important because the cost of tracing is not uniform.

**callerLineNo** : int

The source line number of the calling statement.

### TRCHeapRoot

Public Properties:

---

**type** : TRCGCRootType

**frame** : int = -1

-1 == unknown

### TRCGCRootType

Public Properties:

---

UNKNOWN : = -1

JNI\_GLOBAL :

JNI\_LOCAL :

JAVA\_FRAME :

NATIVE\_STACK :

STICKY\_CLASS :

THREAD\_BLOCK :

MONITOR\_USED :

### TRCArrayClass

*Derived from TRCClass*

Public Properties:

---

**arrayType** : TRCPrimitiveType

Specifies if the class is an array and the array type (see TRCArrayType for valid values of this field). [Note: the representation of array types is undergoing design changes.]

### TRCAggregatedObjectReference

This will be used in HEAP\_STATISTICS\* collection modes. This reference object will

be linked at class object level and it will look like a class to class reference.

*Derived from TRCObjectReference*

**Public Properties:**

---

**ownerSize : int**

the total size of all owners

**targetSize : int**

the total size of all owners

**count : int**

the number of references between owner objects to target objects collapsed in this reference at class level

## TRCThreadEvent

**Public Properties:**

---

**time : double**

## TRCThreadSleepingEvent

*Derived from TRCThreadEvent*

**Public Properties:**

---

**sleepingTime : long**

## TRCThreadWaitingForObjectEvent

*Derived from TRCThreadEvent*

**Public Properties:**

---

**timeout : long**

## TRCThreadWaitingForLockEvent

*Derived from TRCThreadEvent*

## TRCThreadRunningEvent

*Derived from TRCThreadEvent*

## TRCThreadDeadEvent

*Derived from TRCThreadEvent*

## TRCMethodWithLLData

*Derived from TRCMethod*

## TRCLLData

### Public Properties:

---

**summaryUnits** : int  
**summaryNonzero** : int  
**typeId** : String  
**headings** : String

### Public Methods:

---

**int getUnitCount();**  
Returns the number of LLUnits for which detailed information is available. If this returns zero, only the summary information is available.  
**LLUnitData getUnitData(int unitNumber, LLUnitData instance);**  
Returns an LLUnitData object with detailed information about the indicated unit\_number. If instance is not null, then that instance is re-used (overwriting its fields with new data) and returned. Reusing the object instance saves construction time and heap traffic. See below for restrictions on re-using an LLUnitData instance. Unit numbers range from zero to GetUnitCount()-1.

## TRCSourceInfoWithLLData

*Derived from TRCSourceInfo*

## TRCLineCoverageData

*Derived from TRCLLData*

## LLUnitData

### Public Properties:

---

`org.eclipse.hyades.models.trace.util.LLUnitData :`

## TRCObjectValue

This represent the toString() value of an object. The collection agent can send a different string representation of the object if the output from toString() method is not appropriate.

Later we can extend this class with specialized value classes.

## TRCInputOutputEntry

## TRCInputOutputContainer

The container of all parameters/return values and also of the Map entries (key=TRCMethodInvocation, value=list of TRCObjectValue).

The instances of this class will be root objects in resources with trciovxmi extension.

## PACKAGE STRUCTURE

---

### Logical View

- hierarchy
  - extensions
- sdb
- cbe
- common
  - behavior
    - interactions
    - fragments
  - testprofile
  - configuration
  - datapool
- trace
- statistical
- test
- data
- ecore