Glossary

ancestor
: A class or parent that contributes (via inheritance) to the definition of an object. For class-based hierarchies, the ancestors of an object are its class(es) and all the superclasses of its class(es). For prototype-based hierarchies, the ancestors of an object are its parent(s) and the ancestors of its parent(s).

category
: A set of predicates directives and clauses that can be imported by any object.

class
: An object that defines the common predicates of a set of objects (its instances).

  abstract class
  : A class that cannot be instantiated. Usually used to store common predicates that are inherited by other classes.

  metaclass
  : The class of a class, when we see it as an object. Metaclass instances are themselves classes. In a reflexive system any metaclass is also an object.

  subclass
  : A class that is a specialization, direct or indirectly, of another class.

  superclass
  : A class from which another class is a specialization (directly or indirectly via another class).

directive
: A Prolog term that affects the interpretation of Prolog code. Directives are represented using the `:-/1` prefix operator.

  entity directive
  : A directive that affects how Logtalk entities (objects, protocols, or categories) are used or compiled.

  predicate directive
  : A directive that affects how predicates are called or compiled.

  source file directive
  : A directive that affects how a source file is compiled.

encapsulation
: The hiding of an object implementation. This promotes software reuse by isolating users from implementation details.

entity
: Generic name for Logtalk compilation units: objects, categories, and protocols.

event
: The sending of a message to an object. An event can be expressed as an ordered tuple: `(Event, Object, Message, Sender)`. Logtalk distinguish between the sending of a message - `before` event - and the return of control to the sender - `after` event.

grammar rule
: An alternative notation for predicates used to parse or generate sentences on some language. This notation hides the arguments used to pass the lists of tokens being processed, thus simplifying the representation of grammars. Grammar rules are represented using the infix operator `-->/2` instead of the operator used on predicate clauses (`:-/2`).

  grammar rule non-terminal
  : A syntactic category of words of phrases. A non-terminal is identified by its name and number of arguments using the notation `<name>//<nargs>`.

  grammar rule terminal
  : A word or a basic symbol of a language.

identity
: Property of an entity that distinguish it from every other entity. In Logtalk an entity identity can be an atom or a compound term. All Logtalk entities (objects, protocols, and categories) share the same name space.

inheritance
: An object inherits predicate directives and clauses from other objects that it extends or specializes. If an object extends other object then we have a prototype-based inheritance. If an object specializes or instantiates another object we have a class-based inheritance.

  private inheritance
  : All public and protected predicates are inherited as private predicates.

  protected inheritance
  : All public predicates are inherited as protected. No change for protected or private predicates.

  public inheritance
  : All inherited predicates maintain the declared scope.

instance
: The same as object. This term is used when we want to emphasize that an object characteristics are defined by another object (its class).

instantiation
: The process of creating a new class instance. In Logtalk does not necessarily implies dynamic creation of an object at runtime; an instance may also be defined as a static object in a source file.

library
: A directory containing source files. The directory name is used as the library name.

message
: A request for a service, sent to an object. In more logical terms, a message can be seen as a request for proof construction using an object's predicates.

metainterpreter
: A program capable of running other programs written in the same language.

method
: Set of predicate clauses used to answer a message sent to an object. Logtalk uses dynamic binding to find which method to run to answer a message.

monitor
: Any object that is notified when a spied event occurs. The spied events can be set by the monitor itself or by any other object.

object
: An entity characterized by an identity and a set of predicate directives and clauses. In Logtalk objects can be either static or dynamic, like any other Prolog code.

  parametric object
  : An object whose name is a compound term containing free variables that can be used to parameterize the object predicates.

  parametric object proxy
  : A compond term with the same functor and usually the same number of arguments as the identifier of a parametric object.

parent
: An object that is extended by another object.

predicate
: Predicates describe what is true about the application domain. A predicate is identified by its name and number of arguments using the notation `<name>/<nargs>`.

  local predicate
  : A predicate that is defined in an object (or in a category) but that is not listed in a scope directive. These predicates behave like private predicates but are invisible to the reflection methods.

  meta-predicate
  : A predicate where one of its arguments will be called as a goal. For instance, `findall/3` and `call/1` are Prolog built-ins meta-predicates.

  private predicate
  : A predicate that can only be called from the object that contains the scope directive.

  protected predicate
  : A predicate that can only be called from the object containing the scope directive or from an object that inherits the predicate.

  public predicate
  : A predicate that can be called from any object.

  visible predicate
  : A predicate that is declared for an object, a built-in method, or a Prolog or Logtalk built-in predicate.

profiler
: A program that collects data about other program performance.

protocol
: A set of predicates directives that can be implemented by an object or a category (or extended by another protocol).

prototype
: A self-describing object that may extend or be extended by other objects.

self
: The original object that received the message under processing.

sender
: An object that sends a message to other object.

specialization
: A class is specialized by constructing a new class that inherit its predicates and possibly add new ones.

this
: The object that contains the predicate clause under execution.