Documenting Logtalk programs

Logtalk automatically generates a documentation file for each compiled entity (object, protocol, or category) in XML format. Contents of the XML file include the entity name, type, and compilation mode (static or dynamic), the entity relations with other entities, and a description of any declared predicates (name, compilation mode, scope, ...).

The XML documentation files can be enriched with arbitrary user-defined information, either about an entity or about its predicates, by using the two directives described below.

Documenting directives

Logtalk supports two documentation directives for providing arbitrary user-defined information about an entity or a predicate. These two directives complement other Logtalk directives that also provide important documentation information like `uses/1`, `calls/1`, or `mode/2`.

Entity directives

Arbitrary user-defined entity information can be represented using the `info/1` directive:

```
:- info([
    Key1 is Value1,
    Key2 is Value2,
    ...]).
```

In this pattern, keys should be atoms and values should be ground terms. The following keys are pre-defined and may be processed specially by Logtalk:

`comment`
: Comment describing entity purpose (an atom).

`author`
: Entity author(s) (an atom or a compound term `{entity}` where `entity` is the name of a XML entity defined in the `custom.ent` file).

`version`
: Version number (a number).

`date`
: Date of last modification (formatted as Year/Month/Day).

`parameters`
: Parameter names and descriptions for parametric entities (a list of key-values where both keys and values are atoms).

`parnames`
: Parameter names for parametric entities (a list of atoms; a simpler version of the previous key, used when parameter descriptions are deemed unnecessary).

`copyright`
: Copyright notice for the entity source code (an atom or a compound term `{entity}` where `entity` is the name of a XML entity defined in the `custom.ent` file).

`license`
: License terms for the entity source code; usually, just the license name (an atom or a compound term `{entity}` where `entity` is the name of a XML entity defined in the `custom.ent` file).

`remarks`
: List of general remarks about the entity using the format *Topic* `-` *Text*. Both the topic and the text must be atoms.

For example:

```
:- info([
    version is 2.1,
    author is 'Paulo Moura',
    date is 2000/4/20,
    comment is 'Building representation.',
    diagram is 'UML Class Diagram #312']).
```

Use only the keywords that make sense for your application and remember that you are free to invent your own keywords.

Predicate directives

Arbitrary user-defined predicate information can be represented using the `info/2` directive:

```
:- info(Functor/Arity, [
    Key1 is Value1,
    Key2 is Value2,
    ...]).
```

Keys should be atoms and values should be ground terms. The following keys are pre-defined and may be processed specially by Logtalk:

`comment`
: Comment describing predicate purpose (an atom).

`arguments`
: Names and descriptions of predicate arguments for pretty print output (a list of key-values where both keys and values are atoms).

`argnames`
: Names of predicate arguments for pretty print output (a list of atoms; a simpler version of the previous key, used when argument descriptions are deemed unnecessary).

`allocation`
: Objects where we should define the predicate. Some possible values are `container`, `descendants`, `instances`, `classes`, `subclasses`, and `any`.

`redefinition`
: Describes if the predicate can be redefined and, if so, in what way. Some possible values are `never`, `free`, `specialize`, `call_super_first`, `call_super_last`.

`exceptions`
: List of possible exceptions throw by the predicate using the format *Description* `-` *Exception term*. The description must be an atom. The exception term must be a non-variable term.

`examples`
: List of typical predicate call examples using the format *Description* `-` *Predicate call* `-` *Variable bindings*. The description must be an atom. The predicate call term must be a non-variable term. The variable bindings term uses the format `{`*Variable*`=`*Term*, ...`}`. When there are no variable bindings, the success or failure of the predicate call should be represented by the terms `{yes}` or `{no}`, respectively.

For example:

```
:- info(color/1, [
    comment is 'Table of defined colors.',
    argnames is ['Color'],
    constraint is 'Only a maximum of four visible colors allowed.']).
```

Use only the keywords that make sense for your application and remember that you are free to invent your own keywords.

Processing and viewing documenting files

The XML documenting files are (by default) automatically generated when you compile a Logtalk entity. For example, assuming the default filename extensions, compiling a `trace` object and a `sort(_)` parametric object contained in a source file will result in `trace_0.xml` and `sort_1.xml` XML files.

Each XML file contains references to two other files, a XML specification file and a XSL style-sheet file. The XML specification file can be either a DTD file (`logtalk.dtd`) or a XML Scheme file (`logtalk.xsd`). The XSL style-sheet file is responsible for converting the XML files to some desired format such as HTML or PDF. The default names for the XML specification file and the XSL style-sheet file are defined in the configuration files. The `xml` sub-directory in the Logtalk installation directory contains the XML specification files described above, along with several sample XSL style-sheet files and sample scripts for converting XML documenting files to several formats. Please read the `NOTES` file included in the directory for details. You may use the supplied sample files as a starting point for generating the documentation of your Logtalk applications.

The Logtalk DTD file, `logtalk.dtd`, contains a reference to a user-customizable file, `custom.ent`, which declares XML entities for source code author names, license terms, and copyright string. After editing the `custom.ent` file to reflect your personal data, you may use the XML entities on `info/1` documenting directives. For example, assuming that the XML entities are named *author*, *license*, and *copyright* we may write:

```
:- info([
    version is 1.1,
    author is {author},
    license is {license},
    copyright is {copyright}).
```

The entity references are replaced by the value of the corresponding XML entity when the XML documenting files are processed (**not** when they are generated; this notation is just a shortcut to take advantage of XML entities).

There is a set of compilers options, used with the Logtalk `logtalk_load/2` or the `logtalk_compile/2` built-in predicates, that can be used to control the generation of the XML documentation files. Please see the Running Logtalk section of this manual for details.

Last updated on: October 26, 2005