Error handling

All error/exception handling is done in Logtalk by using the ISO defined `catch/3` and `throw/1` predicates [ISO 95]. Some Prolog compilers do not implement these predicates or, if they do, the implementation is not compatible with the standard. Furthermore, the nature of these predicates does not allow their definition by the user. For these reasons, we should check our Prolog compiler before trying to add error handling code to your Logtalk applications.

Errors thrown by Logtalk defined built-in predicates have the following format:

```
error(Error, Call)
```

For example:

```
error(type_error(object_identifier, 33), current_object(33))
```

Errors thrown while processing a message have the following format:

```
error(Error, Message, Sender)
```

For example:

```
error(permission_error(modify, private_predicate, p(_)), foo::abolish(p/1), user)
```

Compiler warnings and errors

The Logtalk preprocessor/compiler uses the `read_term/3` ISO Prolog defined built-in predicate to read and process a Logtalk source file. One consequence of this is that invalid Prolog terms or syntax errors may abort the compilation process with limited information given to the user (due to the inherent limitations of the `read_term/3` predicate).

If all the (Prolog) terms in a source file are valid, then there is a set of errors or potential errors, described below, that the preprocessor will try to detect and report, depending on the used compiler flags (see the Installing and running Logtalk section of this manual for details).

Unknown entities

The Logtalk preprocessor/compiler will warn us of any referenced entity that is not currently loaded. The warning may reveal a misspell entity name or just an entity that it will be loaded next.

Singleton variables

Singleton variables in a clause are often misspell variables and, as such, one of the most common errors when programming in Prolog. If your Prolog compiler complies with the Prolog ISO standard or at least supports the ISO predicate `read_term/3` called with the option `singletons(S)`, then the Logtalk preprocessor/compiler will warn us of any singleton it finds while compiling a Logtalk entity.

Redefinition of Prolog built-in predicates

The Logtalk preprocessor/compiler will warn us of any redefinition of a Prolog built-in predicate inside an object or category. Sometimes the redefinition is intended. In other cases, the user may not be aware that the subjacent Prolog compiler may already provide the predicate as a built-in or we may want to ensure code portability among several Prolog compilers with different sets of built-in predicates.

Redefinition of Logtalk built-in predicates

Similar to the redefinition of Prolog built-in predicates, the Logtalk compiler will warn us if we try to redefine a Logtalk built-in. The redefinition will probably be an error in almost all (if not all) cases.

Redefinition of Logtalk built-in methods

An error will be thrown if we attempt to redefine a Logtalk built-in method inside an entity. The default behavior is to report the error and abort the compilation of the offending entity.

Misspell calls of local predicates

A warning will be reported if Logtalk finds (in the body of a predicate definition) a call to a local predicate that is not defined, built-in (either in Prolog or in Logtalk) or declared dynamic. In most cases these calls are simple misspell errors.

Portability warnings

A warning will be reported if a predicate clause contains a call to a non-ISO defined built-in predicate.

Other warnings and errors

The Logtalk preprocessor/compiler will throw an error if it finds a predicate clause or a directive that cannot be parsed. The default behavior is to report the error and abort the compilation of the offending entity.

Runtime errors

This session briefly describes runtime errors that result from misuse of Logtalk built-in predicates, built-in methods or from message sending. For a complete and detailed description of runtime errors please consult the Reference Manual.

Logtalk built-in predicates

All Logtalk built-in predicates checks the type and mode of the calling arguments, throwing an exception in case of misuse.

Logtalk built-in methods

Every Logtalk built-in method checks the type and mode of the calling arguments, throwing an exception in case of misuse.

Message sending

The message sending mechanisms always check if the receiver of a message is a defined object and if the message corresponds to a declared predicate within the scope of the sender.

Last updated on: October 26, 2005