CoCoA-5 - Bug #102

Packages: should exported functions be automatically "Protect"ed?

09 Mar 2012 16:17 - Anna Maria Bigatti

Status:	Closed	**Start date:**	09 Mar 2012
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Parser/Interpreter	**Estimated time:**	0.00 hour
Target version:	CoCoA-5.0.3	**Spent time:**	3.50 hours

Description

If Package $aaa exports a function Foo, then Foo cannot be exported by another Package $bbb.
And that's safe.

Similarly I thought that Foo would be automatically "Protect"ed and that I could not define @Foo! at TopLevel, but I can:

```
/**/ Deg := 1; // overwrites  Deg(...)  defined in $BackwardCompatible
/**/ deg := 1; // does not overwrite the builtin function
ERROR: Cannot set "deg" because it is a system variable
deg := 1;
^^^
```

I think exported functions should be automatically protected. ... or why not?

2013-01-22: here is another motivating example (from Robbiano)

```
/**/ Num:=1; // overwrites the Num fn exported from @BackwardCompatible@

/**/ DecimalStr(CpuTime());
ERROR: Expecting type FUNCTION, but found type INT (maybe you forgot "*"?)
WHERE: at line 157 (column 8) of float.cpkg5
  N := Num(X);
       ^^^
CONTEXT: function IntegerAndDecimal at line 157 of float.cpkg5
 CALLED BY: function DecimalStr at line 132 of float.cpkg5
  called at top-level
```

In contrast, a standard user defined fn (via Define) is not automatically protected, so that it may easily be redefined during development; when the user is sure it is right, he can explicitly Protect the name. We reject the alternative idea to allow Define to automatically Unprotect a name as that would defeat the purpose of protection.

Related issues:

Related to CoCoA-5 - Bug #94: Default reason for protected variables	**Closed**	**29 Feb 2012**
Related to CoCoA-5 - Feature #531: Package protected variables should know wh...	**Closed**	**09 Apr 2014**

History

#1 - 18 Apr 2012 16:52 - John Abbott

I agree that names exported from packages should be protected (probably with the reason being that it was exported from package XYZ). Making this change should not break any (good) existing code -- what happens if you load the same package twice? Presumably a package should be allowed to redefine the names protected by itself -- and this can be determined by looking at the reason associated with a protected name.

I can find any relevant emails, but I thought we had discussed the idea that variables set by Define would be protected (by "define"). Such a variable could be changed by a new Define if the reason was that it was set by an earlier Define command. This should offer a reasonable degree of safety while also allowing a user to develop new versions of his own functions (without having to unprotect explicitly).

What do you think?

#2 - 31 May 2012 18:10 - Anna Maria Bigatti

- Category set to Parser/Interpreter

#3 - 23 Jan 2013 17:42 - John Abbott

- Status changed from New to In Progress

- Assignee set to John Abbott

- Target version set to CoCoA-5.0.3

- % Done changed from 0 to 10

After considering typical use patterns, it makes most sense for **Export** to protect the exported names, and for **Define** not to protect (the automatic unprotection I hinted at above would largely defeat the purpose of protecting).

The only slight catch is that reloading a package must work reasonably. Anna reminds us that reloading a package should first **undefine all variables** it previously exported, and then make the new exports. This means that loading a package must be able to undefine the protected variables associated with it.

#4 - 24 Jan 2013 14:23 - John Abbott

- Status changed from In Progress to Feedback

- % Done changed from 10 to 90

JAA notes that *system protected* variables cannot be unprotected (*e.g.* the variable **QQ**).

Internally the interpreter marks names exported from packages as *package variables*.

As far as I can tell a variable can be in one of four states:

- **(A)** system protected variable
- **(B)** package variable
- **(C)** user protected variable
- **(D)** normal unprotected variable

The easiest solution for me was to make *package variables* behave like *system protected* variables. This also is the most sensible solution because fns implemented in packages are practically *built-in*, and built-in fns are *system protected*.

The relevant source code is in **Interpreter.C**, function **checkProtection** (around line 1300).

#5 - 18 Feb 2013 18:30 - John Abbott

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

After implementing the change about a month ago, no problems have cropped up.

Final decision: package exported names enjoy the same "protection" as system protected names (essentially built in fns, plus a handful of others).

NOTE: the commands **Protect** and **Unprotect** are not documented in CoCoA5!!