Modular Embedded DSL

Version 0.1, Dec 05, 2008

Markus Völter
voelter@acm.org
www.voelter.de

Abstract

This document provides a rough overview over the modular embedded DSL (i.e. C in MPS) that I have built so far. It explains the language structure and the most important design concepts.

Prerequisites

Note that this is not an MPS tutorial. I assume that you have a reasonable level of familiarity with MPS. Also, to be able to run the Windows examples you have to install the LCC compiler. I explained where to get it (easily) in the description of the Windows solution below. Finally, to run the examples on the Mindstorm you have to install the mindstorm drivers, nxtOsek and the GNUARM compiler. See the *mpscmindstorm* Google code page for details.

Language Overview

Existing Languages

MPS supports the definition and composition of language modules. Languages can you use or extend each other. So here is a list of languages (language modules) that are defined so far.

- **med.core:** The core language defines all the essential aspects of C. This includes the module system, types, expressions and statements.
- **med.unittest**: defines an extension of C to conveniently describe and execute unit tests.
- **med.tasks:** defines the abstraction of tasks that can be executed at various times during a program, for example in the background, upon start up or triggered by some kind of event
- **med.logging:** contains very simple statements for logging
- **med.quantities:** is an example language extension that provides physical quantities as data types in C
- **featuredepenencies:** supports the expression of (product line) feature dependencies in models and code
- **med.platform.win32:** generators that translate language elements that need specific translation for the Windows platform
- **med.platform.osek:** contains generators that require specific translation for the Osek platform

Modules and Resources

I will explain the details about each of these languages below, however, to be able to run the examples, it is important to understand the high-level structure of programs in MPS C.

In an environment where everything is a model it makes no sense to work with header files and textual includes. This is why the top-level abstraction of programs is a *Resource*. A resource contains any number of *Modules*. Modules can depend on, and consequently import, each other. If module A imports module B, then A can see all the exported elements in B. This mechanisms replaces the header file.

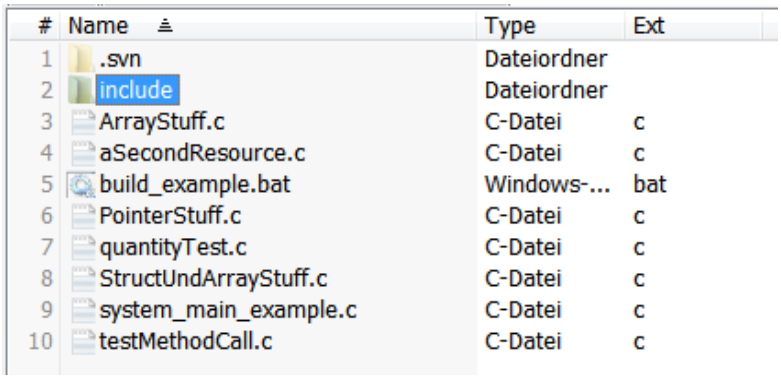
Within the module, there can be structs, procedures, tasks, quantity definitions, and all kinds of other things contributed by languages. To be able to contribute to modules, the language element has to implement the *IModuleContent* interface.

Solutions Overview

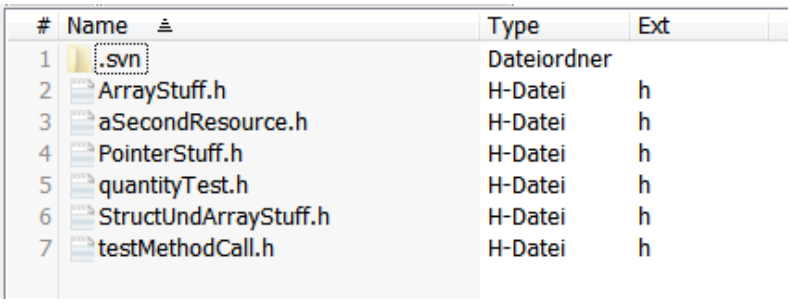
The project contains two example solutions (i.e. application project as opposed to languages).

modularEmbeddedDSL.sandbox

The *modularEmbeddedDSL.sandbox* is the main example that exercises all of language features currently supported. Is currently generated down to Windows. Generating this solution results in the following generated code:



#	Name	Type	Ext
1	.svn	Dateiordner	
2	include	Dateiordner	
3	ArrayStuff.c	C-Datei	c
4	aSecondResource.c	C-Datei	c
5	build_example.bat	Windows-...	bat
6	PointerStuff.c	C-Datei	c
7	quantityTest.c	C-Datei	c
8	StructUndArrayStuff.c	C-Datei	c
9	system_main_example.c	C-Datei	c
10	testMethodCall.c	C-Datei	c



#	Name	Type	Ext
1	.svn	Dateiordner	
2	ArrayStuff.h	H-Datei	h
3	aSecondResource.h	H-Datei	h
4	PointerStuff.h	H-Datei	h
5	quantityTest.h	H-Datei	h
6	StructUndArrayStuff.h	H-Datei	h
7	testMethodCall.h	H-Datei	h

Among other things it generates a *build_example.bat* file which builds the system. This file uses the LCC 32 C compiler for Windows. It also depends on a bunch of custom made batch files which it calls. You can get the LCC compiler including all the necessary batch files from: *http://mydrive.ch*, username *messelcast*, password *fgd1928*, folder *mpscmindstorms*.

To run it, make sure that the *c-build* and *c-build\lcc\bin* are in the system path. You also have to set the *%lccinclude%* environment variable to point into *c-build\lcc\include.* Finally you'll have to adapt the absolute path in *link.bat* and the one in *libpath.txt* to your environment. It is a hack to supply the library path to the linker - I have no idea how to make this more elegant (except, of course, using one single *LCC_HOME* variable to make all of the others relative... something yet to be done).

You can then run *build_example*. It should generate an *example.exe* system which you can run. The output should say something about successful tests:

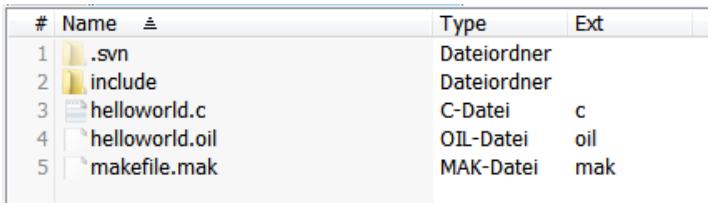
```
running test: testAdding
running test: addTwoPhysicalSpeeds
running test: addingPhysicalANdInternal
running test: testPointers
running test: testReferences
running test: createArrays
```

```
running test: staticstuff
```

MindstormHelloWorld

This project contains the hello world for Lego Mindstorms. Here is what you need to do to run it.

First you need to generate the files in the solution. Here are the files this generates:



#	Name	Type	Ext
1	.svn	Dateiordner	
2	include	Dateiordner	
3	helloworld.c	C-Datei	c
4	helloworld.oil	OIL-Datei	oil
5	makefile.mak	MAK-Datei	mak

To be able to execute the makefile it is useful to put the whole project into your Osek samples folder because then you can reuse all the infrastructure that comes with Osek. My example is in *nxtOsek/samples_c/mvtest*.

You can then open a Cygwin console in this directory and call *make -f makefile.mak all*. If this succeeds, this will give you all kinds of files, one of them is called *rxeflash*. Running this should flash the application onto your mindstorm or you can run it.

Language Descriptions

This section contains a description of the most important language elements in the various languages. Note that this cannot be complete it is intended as a way to get a first understanding of how it works. You will have to dig down into the language implementation in the system.

med.core - the core language

Language Concepts

The core language contains all the essential ingredients for C. Within the language definition you can see various subfolders (they are called virtual folders in the MPS). The best one to start is the *modules* folder since it contains all the elements for the high-level structure of MPS C programs.

Resource	**The equivalent to a file. Can contain any number of modules**
Module	Basically a namespace. Can depend on (and

	use) other modules.
ImplementationModule	Contains the actual program content (procedures, structs, etc.) ExternalModules are used to interface to existing header files and libraries.
IModuleContent	interface implemented by everything that wants to be a direct content of modules
IExportableModuleContent	module contents that have an export flag. Necessary for the import between modules to work

Generation