Building RCP Apps with MPS

Markus Voelter

independent/itemis

Abstract.

1 Introduction

The term RCP is borrowed from the Eclipse ecosystem and stands for Rich Client Application. It refers to a stripped down version of the IDE that only contains those artifacts that are necessary for a given business purpose. In the MPS world it refers to a stripped down version of MPS that only contains those artifacts that are necessary to use a small domain-specific set of languages. Various aspects of MPS can be removed or customized to deliver a custom user experience for a given group of users. This article describes how. In particular, the following aspects of MPS can be customized:

- various icons, slogans, splash screens and images
- the help URL
- the set of languages available to the users
- the set of plugins available in MPS

2 Process Overview

To build a custom RCP version of MPS, you have to create a solution that contains a so-called *build script*. A build script is written in MPS' build language which is optimized for building RCP applications (as well as IntelliJ plugins and in the future, Eclipse plugins). When running the generator for this build script, MPS generates an `ant` file that creates the actual RCP distribution.

3 Building an example RCP build

In this document we describe the development of an RCP build script for the mbeddr project. mbeddr is a set of languages for embedded software development based on MPS. The project is available at `http://mbeddr.com`. It is Open Source, so all the code, including the RCP build script is available from the respository at `https://github.com/mbeddr/mbeddr.core/`. The project that contains the build script can be found in `/code/rcp-build/MPSIDE-build.mpr`.

3.1 Creating the Solution and the Build Script

In an arbitrary project create a new solution with a new model inside. In the model, configure as used languages

- `jetbrains.mps.build`
- `jetbrains.mps.build.mps`

Also import the `jetbrains.mps.ide.build` model. You can now create a new `build project` in the model (the name is confusing: it is not actually a new build *project*, just a build script). Fig. 1 shows the resulting, empty build script.

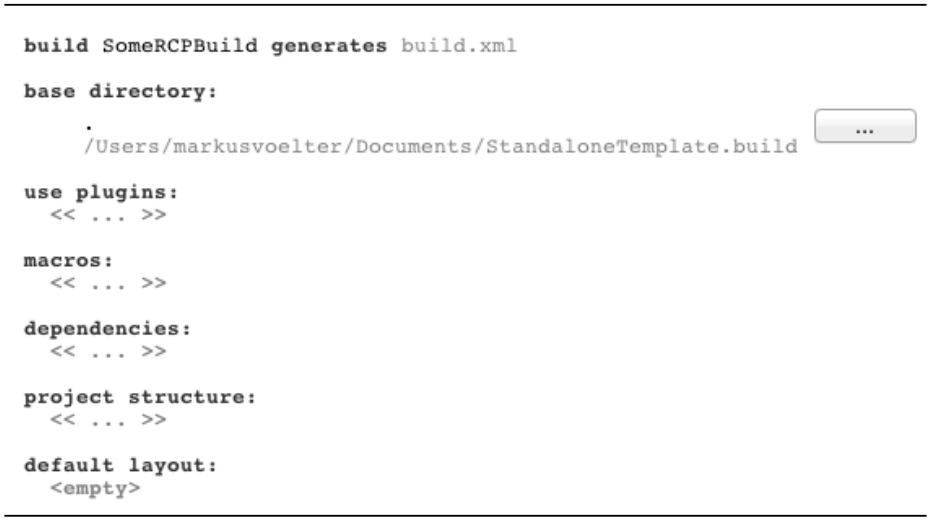

Fig. 1. An empty build script after directly after creation.

Let us look at the various sections of a build script:

base directory The base directory defines an absolute path relative to which all other paths are specified. By default, this is the directory in which the resulting `ant` file will be generated, and in which it will be executed.

use plugins The build language itself can be extended via plugins[1]. These plugins contribute additional build language syntax. Typically, the `java` and `mps` plugins are required. We will use syntax contributed by these plugins below.

[1] Note that these are *not* the plugins that make up MPS itself; those will be configured later.

macros Macros are essentially name-value pairs (similar to `${something} in ant`). In the Macros section, these names are defined and values are assigned. In the remainder of the build script these macro variables will be used. MPS supports two kinds of Macros: `var` macros are strings and can be assigned any value. `folder` represents paths, relative to the base directory defined above. Note that MPS provides code completion for the path components in `folder` macros.

dependencies This section defines dependencies to other build scripts. MPS bundles a set of build scripts (e.g. `buildStandalone`, `buildWorkbench` or `buildMPS`). By establishing a dependency to any one of them, the structures defined in that references build script can be used in the referencing build script. For example, the macros defined in the referenced build scripts can be used.

project structure This section defines the actual contents of the to-be-built RCP application. Such contents may be JAR files, plugins, branding or MPS languages.

default layout This section creates the directory, file and JAR structure of the new RCP distribution. It references and includes the artifacts defined in the project structure section.

4 Building the script for mbeddr

```
her is
some code
```