MPS user guide for Java developers (IntelliJ IDEA)

You are viewing documentation of MPS 3.1, which is not the most recently released version of MPS. Please refer to the documentation page to choose the latest MPS version.

One of the primary goals for MPS is to serve Java developers so that they could seamlessly combine DSLs with Java code as an integrated part of their Java projects. Various database queries, business rules, hardware-specific code, system configuration or any other code in languages developed by language vendors as well as those developed yourself can participate in Java projects and be used from Java IDEs directly. This user guide describes MPS from the Java developer's perspective and provides the essential information needed to jump-start coding with DSLs in IntelliJ IDEA.

Before you start

Other user guides

- You may alternatively check out the Standalone IDE user guide, which covers the topic of using MPS-based standalone IDEs.
- Once your ambitions grow and you decide to create languages of your own, the User Guide for Language Designers will provide you with the relevant material.

MPS glossary

Abstract Syntax Tree (AST)	a logical representation of code in memory (and disk) in the shape of a tree forest that describes hierarchies of nodes. These nodes have a notion of a parent-child relationship. Additionally, two nodes can be mutually connected with explicit references that go across the hierarchy structure.
BaseLanguage	a projectional clone of Java 6. I follows the Java specification and is 1:1 compatible with Java 6. Additionally, MPS provides several handy extensiont to BaseLanguagem such dates, collections, closures and many others.
Code generation	the process of transfromation code from one model (AST) into another model. For example, code describing a set of business rules can be transformed into plain Java so that it can be compiled with javac and run as part of an enterprise application.
DevKit	A package of related languages that have been grouped for user convenience.
Domain Specific Language (DSL)	a language dedicated to a particular problem domain, typically created with the aim of simplicity and greater expressivity compared to a general purpose language.
Language plugin	a packaged library (a zip file) containing all the required elements in order to use a language either inside either IntelliJ IDEA or MPS.
Projectional editor	an editor that allows the user to edit the AST representation of code directly, while mimicing the behavior of a text editor to some extent. The user sees text on teh screen and edits it, however, in reality the text is only an illusion (projection) of an AST.
Module	The top-level organization element of an MPS project that typically groups several models together. It can have three basic types: Solution, Language and DevKit and may depend on other modules and models.
Model	A lower-level organizational element grouping individual concepts. It may depend on other models.
Structure	A language aspect defining all types (concepts) of AST nodes that can be used in the language together with their relationships.
Concept	A definition that describes the abstract structure of a syntax element. E.g. the IfStatement concepts says that an if holds a boolean Expression and up-to two StatementLists.
Constraints	A language aspect holding additional restrictions on concepts, their properties and relationships.
Behavior	Allows the language designer to define behavior of the language concepts.
Editor	Holds vizualization definitions of individual language concepts. Since the way concepts are viewed and edited on the screen can be customized, the editors specify how the user will interact with the language.

Scope	The set of elements that are visible and applicable to a particular position within a program. Typically only a sub-set of all elements of a particular kind can be used at any given program location.
Typesystem	A set of rules that validate and infer types of concepts in a program.
Actions	User-invoked commands that may perform changes to the code. Actions can be attached to keyboard shortcuts or menu items.
Intention actions	Context-sensitive actions offered to the language user through a small pop-up window triggered by the Alt + Enter key shortcut. These actions typically perform a relatively local refactoring to the code under carret or a selected block of code.
Surround With intention actions	Intentions applicable to a selected block of code that wrap the block by another concept. E.g. Surround with Try-Catch.
Refactoring	A potentially substantial automated change in code structure triggered by a user action.

Frequently Asked Questions (FAQ)

Check out the FAQ document to get some of your questions answered before you even ask them.

Resolving difficulties, understanding reported errors

The Finding your way out of a problem page should give you a hand whenever you run into a problem.

Setting things up

To get started, you first need to install the essential MPS plugins into IntelliJ IDEA:

- MPS core
- MPS BaseLanguage support (in order to edit BaseLanguage in IDEA)
- MPS Version Control support (in order to use VCSs for MPS models in IDEA)

Both Community and Ultimate editions of IntelliJ IDEA will work. Once you have done that, pick a DSL that you want to use and install it into IntelliJ IDEA, as well. This will allow you to use MPS-based DSLs inside IntelliJ IDEA and interoperate between the MPS code od the rest of your project.

We've described all the steps required to get up and running at the Using MPS inside IntelliJ IDEA page. Check it out for details.

Editor specialities

The clever projectional editor in MPS is slightly different from what you may be used to. Since the editor is the most visible element, which you'll get to know as soon as you start interacting with your code, we prepared a short introductory document - Comanding the editor.

Where to find language plugins

Languages come packaged as ordinary zip files, which you unzip into the IntelliJ IDEA plugin directory and which IDEA will load upon restart. Additionally, many of the plugins have been shared through the IntelliJ IDEA plugin repository and so can be comfortably downloaded through the IDEA's plugin manager. The Using MPS inside IntelliJ IDEA page further details the necessary steps.

As examples, the BaseLanguage and Build language have been uploaded to the IntelliJ IDEA plugin repository for you to try.

- BaseLanguage is a projectional clone of Java 6, which allows you to write code in a language that you are familiar with, yet benefit from many advantages that MPS brings.
- Build language provides a comfortable and fully declarative build abstraction over Ant. This is the language we use internally to build MPS. You can use it, for example, to create build scripts for your Java projects or to compile and package your MPS-based languages. Check out the documentation for details.

Platform Languages

On top of Base Language, which is merely a copy of Java 6, MPS provides several useful language additions that aim at making Java development much more efficient and enjoyable:

- Closures

- Collections language
- Tuples
- Dates language
- Regexp language
- Unit test language
- Type Extension Methods
- Builders
- XML language
- Other languages

IDE integration - Java interoperability, cross-navigation, joint compilation

The MPS IDEA plugin integrates your MPS code tightly into the rest of the Java project. MPS code will seamlessly participate in IDEA's build and make, you can cross-navigate to usages or definitions between Java and BaseLanguage as well as refactorings will correctly include all the sources.

VCS integration

Being projectional in the world that has been built on text brings about some challenges. Integration with VCS systems is one of them. In order to be able to standard VCS systems, such as Subversion or Git, MPS provides an add-on that hooks into the important VCS events and gives you a projectional way to merge code and resolve conflicts. With this add-on installed IntelliJ IDEA will give you the experience you expect from a VCS for your projectional code. When comparing versions or resolving conflicts in MPS models, you can rely on the MPS structural diff/merge tool, just like in MPS. The tool will render the model content in a domain specific rather then the persistence-specific way.

Check out the details at the Version Control page.

Debugger

It is not yet possible to directly run a program represented by an MPS node - the generated code should be used to execute the program instead. However, we've already implemented the ability to set a breakpoint directly inside the DSL code and so the IntelliJ IDEA debugger will be stopped in the appropriate place allowing you to explore the stack trace and the variables.