MPS User's Guide

You are viewing documentation of MPS 2018.2, which is not the most recently released version of MPS. Please refer to the documentation page to choose the latest MPS version.

MPS User Guide for Language Designers

Credits

Welcome to MPS. This User Guide is a complete reference documentation to MPS and it will navigate you through the many concepts and usage patterns that MPS offers and will give you a hand whenever you need to know more details about any particular aspect of the system.

Beginner's Fast Track to MPS

Here's our offer for new comers: Try our Fast Track to MPS tutorial, which was designed specifically for professionals, who are completely new to MPS and prefer a guided tour through the MPS landscape. You will walk the beaten path one step at a time, following clear marks that show you where to go next. The information is structured so that you progress from simpler concepts to the more involved ones and at the end of the journey you'll understand MPS and will be able to use it effectively on your projects.

Structure of the User Guide

1. First, the Introduction section will offer a high-level overview of the basic notions and their roles.
2. In the second section, named Using MPS, you'll get familiar with the interface through which you'll communicate with MPS. Although very small, there still are some differences between how you interact with MPS and how you typically use other common programming tools.
3. In the third section, called Defining Languages, we'll get to the meat of MPS. We'll show details on how to define the many aspects of your custom languages - their structure, editors, generators and type systems rules.
4. The IDE integration section will then provide some additional context necessary to help you improve the IDE aspect of your languages and integrate them nicely into MPS.
5. The Platform languages section gives you details on all languages bundled with MPS including the corner stone language of MPS - the BaseLanguage.
6. The Delivering languages to the users and Java IDE integration section covers the

build language and the process of building standalone IDEs as well as language plugins.
7. Don't forget to check out our tutorials and focused cookbooks listed in the Tutorials an d Cookbooks sections, to learn more about individual aspects of MPS
8. Whatever didn't fit the mentioned scheme was placed into the last Miscelaneous secti on.

For your convenience, you can also view t he user guide in PDF (created by Marco Lombardo) or in a single page version.

Alternative User Guides

Not a language designer yet? Try out one of our simplified MPS user guides

You do not need to design your own languages and DSLs to benefit from MPS. You may well enjoy using languages designed and developed by somebody else. These languages are typically distributed as language plugins for Java IDEs or they may come bundled with their own standalone IDEs. Since using existing languages is a much simpler tasks than creating them, we prepared dedicated user guides covering the topics relevant to the DSL users.

- Check out the MPS user guide for Java developers (IntelliJ IDEA) if you want to learn quickly how to use MPS for Java development.
- The MPS user guide for DSL users collects the essential knowledge required for developing in MPS-based DSLs using an MPS-based standalone IDE.

User guide for language designers

1. Introduction to MPS
    - Basic notions of MPS: nodes, concepts, and languages
    - Glossary
    - FAQ
    - MPS project structure
    - Default keymap reference
2. Using MPS
    - The first run - essential information for total beginners into how MPS is organised, how it works and how users are supposed to interact with it.
    - MPS Java compatibility
    - Commanding the editor
    - IDE configuration
    - Getting the module dependencies right
    - Resolving difficulties, understanding reported errors
    - Where to find language plugins

- Version control notes
- Using Debugger
- Migrations
- Using Migrations with branching
- Console

3. Defining Languages - defining a language involves specifying several aspects, each of which codifies some part of the language - the allowed AST structure, the appearance on the screen, type-system rules, etc.

Aspect name	Description	Core documentation	Other links
Structure	Defines the kinds of nodes (called Concepts) that may be used in user models. Each node in the program (model) refers to its concept. Concepts specify, which properties, children and references nodes may have. Concepts can extend other Concepts and implement ConceptInterfaces.	Structure	SModel language - programatic access to the model Open API - API reference for accessing models Quotations - building AST snippets Pattern - language for pattern matching nodes Using model & module dependencies from code FAQ Video - Introduction to JetBrains MPS, part 1: Projects Video - Introduction to JetBrains MPS, part 2: Structure
Constraints	Restricts the relationships between nodes as well as the allowed values for properties beyond the rules defined in Structure. Constraints typicaly define: - the target scope for references (a collection of allowed nodes a reference can point to) - situations, in which a node can be a child/parent/ancestor of another node - allowed values for properties - property accessor methods (getters and setters)	Constraints	Scopes Video - Introduction to JetBrains MPS, part 3: Constraints
Behavior	Just like classes in OOP hold methods, Concepts may define methods and static methods that can be invoked on nodes in a polymorphic way. Nodes thus carry behaviour alongside their properties and relationships.	Behavior	Video - Introduction to JetBrains MPS, part 4: Behavior

Editor	Instead of defining a parser that would translate code from an editable form (i.e. text) into the tree-like structure that a computer could manipulate, MPS offers the concept of projectional editor, which let's the user edit the AST directly. The Editor aspect enables language designers to create a UI for editing their concept concepts.	Editor Diagramming Editor Transformation Menu Language Context assistant Context actions tool	Editor cookbook Editor language generation API Video - Introduction to JetBrains MPS, part 5: Editor
Actions	The Actions aspect provides means to specify advanced editor behavior, such as copy/paste or node initialization.	Editor Actions	Video - Introduction to JetBrains MPS, part 6: Actions (obsolete)
Intentions	All modern IDEs assist developers with instant code manipulating action available under a handy key-shortcut (Alt + Enter in MPS). Language authors can define such little code transformations for their languages in the Intentions aspect.	Intentions	Video - Introduction to JetBrains MPS, part 7: Intentions
Generator	Models written in one or more languages get ultimately translated into runnable code in some target general-purpose language and platform, such as Java. Along the way models get gradually transformed so that repeatedly concepts get replaced with concepts from a lower level of abstraction until the bottom-line level is reached. The rules for translating concepts and their proper ordering is defined in the Generator aspect.	Generator Generator Plan	Generator cookbook Building an interpreter cookbook Video - Introduction to JetBrains MPS, part 8: Generator

TextGen	During code generation after the Generator has reached the bottom-line AST representation, the TextGen phase kicks in and translates all nodes in the model into their textual representation and saves the resulting textual source files on disk.	TextGen	Video - Introduction to JetBrains MPS, part 9: TextGen
Dataflow	The ability to understand the flow of values and the flow of control through language constructs helps languages report issues such as unreachable code or potential null-pointer error. Language designer can leverage the Dataflow aspect to define the flow for each concept, which MPS will then use to calculate the dataflow for the whole program.	Dataflow	Dataflow cookbook Video - Introduction to JetBrains MPS, part 10: Dataflow
Typesystem	Language that need to type-check their code need to provide type-system rules. The MPS type-system engine will evaluate the rules on-the-fly, calculate types for nodes and report errors, wherever the calculated type differs from the expectations. So called checking rules may additionally be defined to verify non-typesystem assertions about the model.	Typesystem	Typesystem cookbook Using the typesystem Debugging the typesystem Video - Introduction to JetBrains MPS, part 11: Type-system
Refactoring	Modern IDEs allow the developer to seamlessly and flawlessly change the structure of their code through refactoring. MPS allows the language designers to prepare such refactorings and make them part of their languages.	Refactoring	

Migrations	When a new version of a language is released to the public, projects that use the previous version of the language must be migrated so that they use the new language constructs. Migration scripts, prepared by the language authors, will manipulate user code and automatically update it to the most recent version of the language.	Migrations	Migrations with branching
Testing	Various aspects of language definition can be automatically tested. Language authors may create tests that will verify that the editor, actions, type-system, data flow or constraints of their languages behave according to the specifications.	Testing	
Scripts	TODO	Scripts	
Accessories	The Accessories Models can be stored at two places - either as an aspect of a language (recommended), or as a regular model under a solution. In both cases, the model needs to be added to the Langua ge Runtime Language Settings s o as it could be used. A typical use case would be a default library of Concept instances to be available at any place the language is used.	Accessories	

4. Languages for IDE Integration - how to customise MPS, add language-specific visual extensions, use different persistence format, etc.
    - Generic placeholders and generic comments
    - Commenting out nodes
    - Custom language aspects
    - Icon description - describing icons by text
    - UI Plugin - extending the UI (menus, tool windows, tabs,

preferences, etc.)
    - Find Usages - customising the way users discover nodes
    - Suppressing Errors
    - Debugger
    - Make
    - Extension support
    - Custom Persistence Cookbook
    - MPS and Ant
    - MPS and Git
    - HTTP support plugin
5. IDE tools - tools that MPS offers you to manipulate the languages
    - Dependencies Analyzer - analyze model dependencies (Analyze model dependencies)
    - Module Dependencies Tool (Analyze module dependencies)
    - Run Configurations
    - Changes highlighting
    - Default Keymap Reference
    - Module Cloning
6. Platform Languages - out-of-the-box languages ready for use
    - Base Language
        - Base Language Extensions Style Guide
        - MPS Java compatibility
        - Concept Functions
    - Closures
    - Collections language
    - Tuples
    - Lightweight DSL
    - Dates language
    - Regexp language
    - Type Extension Methods
    - Builders
    - Logging
    - XML language
    - Other languages
7. Delivering languages to the users and Java IDE integration - building languages from the command line, Ant integration, continuous integration, creating and using plugins, obfuscating code
    - Build Language
    - Building IntelliJ IDEA language plugins
    - Using MPS inside IntelliJ IDEA
    - Building MPS language plugins
    - Building standalone IDEs for your languages
    - Extending the user interface
    - Removing sources from generated code (Implementation stripping)
8. Tutorials (pdf variant)
    - The Fast Track to MPS tutorial
    - Shapes - an introductory MPS tutorial
    - JetBrains MPS Calculator tutorial
    - Generator demos tutorial
    - How to Add JARs to a Jetbrains MPS Project by Federico Tomasseti
    - MindMaps by Antoine Gagnon
    - Markus Voelter's tutorial
    - The beginners screen-cast series on JetBrains TV
    - The MPS screen-casts page
    - The MPS channel on JetBrains TV
9. Cookbooks - quick how-to guides (pdf variant)
    - Common language patterns - a

how-to guide covering recurring language design patterns
    - Getting the dependencies right
    - Editor cookbook
    - Generator cookbook
    - Building an interpreter cookbook
    - Description comments - a cookbook showing how to leverage attributes, scopes and error suppression to add support for description comments on arbitrary code elements of your languages
    - Type System
    - Dataflow cookbook
    - Regular expressions
    - Open API - accessing models from code
    - Custom Persistence Cookbook
    - Custom Language Aspect Cookbook
    - Finding your way out - a brief collection of guidelines that should help you move forward when you get stuck somewhere.
10. Miscellaneous
    - Glossary
    - FAQ
    - Dependencies and Classpath in MPS
    - Contributing to JetBrains MPS - guidelines for MPS contributors and developers on how to get around the MPS source code
    - Copyrights

On-line help

MPS comes with bundled help, which shows context-sensitive help information when you hit the F1 key. As an alternative you can view the help pages on-line:

- An on-line variant of the MPS help

Did not find the answers?
Post your questions to our discussion forum