Transforming into an Enterprise Multitool

with MDT PE 3.0-3.1 release

by Błażej Sytar, Sales & Support Leader, ASTEC

In the second half of the last year we have focused strongly on possible extensions to MDT, which would allow the entire development teams to benefit from it. In pursuit of this goal we have implemented a dedicated tool for testers and something for architects. With the support of Mylyn MDT can now serve for different tasks within a team. Of course that is not all, here's a short overview of the new stuff:

New Extensions

- **MUnit 3 Support plug-in** – a mechanism that allows for writing and running tests for applications written in Magik language. It is responsible for automatic generation of test cases, running tests and presenting their results. It is very helpful in discovering errors and mistakes already in code production phase.

- **UML2 Modler plug-in** – a tool to be used for modeling of whole systems with the use of UML2 language. Its main functionality is to model system architectures as well as dynamic generation of Magik source code according to previously created models. It helps to combine the work of architects working with UML language with the tasks given to Magik developers.

New Features

- **New Mixin, New Slotted Exemplar Wizards** – those wizards have been added for easy and semiautomatic creation of Magik elements such as Mixins and Slotted Exemplars. With their help you can now create entire blocks of code including automatic creation of method stubs, pragma statements and comments.

- **Type Overview** – a kind of read-only editor that allows you to see the source code of an entire class in case it is spread between many files. Type Overview collects all information directly from the session, so the shown code is always up-to-date. It groups the methods coming from the same files and marks them with different background color so they are easily distinguished.

Are You new to MDT?

Visit MDT online if you would like to know

- why MDT is the most intuitive IDE for Magik development,
- how it will increase your programming efficiency and
- why is it being used by companies all over the world.

Try out the 2-month trial, watch our video tutorials and subscribe to the newsletter.

Visit us today at [www.mdt.net](http://www.mdt.net)

Last year was full of little accomplishments. We have gained many new customers from different enterprises and continents. We have also heard many words of appreciation. This drives us further and keeps motivated to work even harder on improving Magik Development Tools.

In effect we are happy to place in your hands the 2012 Issue of MDT Magazine. What’s in it? First of all you can learn about what’s new in the recent releases 3.0 and 3.1 have brought in. And there is something to be proud of: MUnit 3 Support and UML2 Modeler plug-in have been published.

Further you can read about our newest endeavor, which is the MStyle plug-in - a tool that will help you improve the quality of your code in many different ways.

Next you can find an article about many handy Magik tricks that we are sure you will find useful in your work. They have been gathered over the years by one of our developers who has been working with Magik for a long time. With a little incentive he agreed to share them with You.

We also invite you to look into the interview section. One with a couple of representatives of our very good business partner – Ubisense Germany. They will share their opinion about MDT. The other is with Dr. Jun Gang from GE Smallworld. She will tell us a little about her impressions after attending our dedicated training: “Development techniques with the use of MDT”.

Best wishes,

MDT Team, ASTEC
MStyle plug-in – Coming soon!

by Wojciech Trocki, Software Developer, ASTEC

Introduction

Code reviews are essential to maintain solid code quality. Manual reviews are not efficient due to human limitations and the size of projects. To improve the overall code quality with little cost, every review should be automatic, especially in case of well-known problems typical to specific programming languages.

Large-scale projects always involve big numbers of programmers, many of them varying in skill level and often working remotely without direct contact with each other. This greatly affects the source code of end-user applications. Manual review of millions of code lines is a tedious job, though. Ultimately as exhausting as it must be thorough.

To meet those needs we were planning to develop a code-checking plug-in from some time. It was a hard task to accomplish, because rules specific for Magik language are neither commonly known nor widely accessible and also rarely used. Fortunately, in our travels to GE conferences, we came across a couple of representatives from Realworld Systems, a well-known and respected company within Smallworld community. They revealed to us that they have been using a simple, internal plug-in, called Code Definition Handler, that checks their code for specific violations. They agreed to share it with us so its source code and coding rules stored within could serve as foundations for our own MStyle plug-in.

How does it work?

MStyle plug-in is a tool for code analysis created to help developers write Magik code conform with coding standards specific to their tasks. With it team members are able to create internal, customizable and exchangeable code verification processes. MStyle automates the process of checking Magik code and helps to discover many well-known code violations already in code production phase. It makes it ideal for teams which want to enforce a specific coding standard and keep their code clean.

MStyle in an extension to MDT and it is based on its Magik grammar parser. To launch it you have to select any resource container (project, folder or file) to check its code using defined rules. Our plug-in can check many different aspects of source code like metric rules, some invalid/depreciated source code elements and run it on specified projects.

MStyle allows for introducing modern team standards to Magik projects through fully customizable profiles. Rules can be also imported/exported and easily shared in team environment. After running a check the list of violations is presented in separate view allowing quick access to them. After jumping to them violations are also marked in the opened editor.
Rules

MStyle plug-in consists of over 30 rules which check the code against different kinds of violations or so called “bad smells”. Here are some of the most interesting ones:

- **Unassigned Local Variable** – if a variable is declared without assigning it will take `_unset` value. In most cases those are junk declarations. Local variables must be first assigned to be later used in expressions.

- **Responsibility Class** – calculates metrics for class responsibility using weighted number of non-private class members and checks if metrics do not exceed value defined in preferences.

- **Wrong Magik File Name** – checks if the file name differs from the name of the containing exemplar. If file name contains more than one exemplar, this rule checks if one of them matches the file name.

- **Method Name Too Long / Exemplar Name Too Long** – long methods or exemplar names make the code difficult to read and understand. Long method names are sometimes related to too high responsibility of an element. For example method name: `calculate_and_restore_points` may be refactored into `calculate_points` and `restore_points`. The rule checks if length of method or exemplar name does not exceed defined value.

- **Nesting Depth** – a metric used for a method which counts the maximum number of encapsulated scopes inside the body of the method. Statements like `if`, `for`, or `loop` are recognized as nesting blocks. If nesting depth exceeds the defined value, next nested block should be extracted to a new method.

- **Chained Instructions** – checks if source code contains multiple chained instructions based on return values from invocations. Multiple invocations are not recommended and often may rise `does_not_understand` condition. Return values are not stored in local variables, so debugging such statements can be very difficult.

- **ValidPragma Statement** – pragma classify level and usage should always have proper values. If pragmas are being entered by hand some typographic errors may occur, what can cause the method to become unclassified.

- **Method With Class Name Invocation** – sometimes a check is needed to see if an object is an instance of a specific class. Checking its type by using `class_name` method does not guarantee unique results. With this rule we can easily track such invocations.

- **Invalid Comparison** – a developer can sometimes use improper operator while comparing values. This rule checks if elements are properly compared. Strings and numbers should be compared using the `=` and `<>` operators. Symbols should be compared using the `is` and `isnt`.

You can also define your own rules basing on provided regular expressions. MStyle reads all files with extensions linked to such rule and checks if the file contents match the provided pattern.

**On-the-fly checks or Continuous Integration**

MStyle can be used not only to check the code locally, on-the-fly within one IDE. With some additional mechanisms it can be also used for continuous integration supporting popular systems like Sonar. This allows for reports generation and fast tracking of code changes or quickly spotting of decreasing code quality within the entire project.

**Coming soon!**

MStyle plug-in is still under development. Internal releases have been already tested and the results are more than promising. Still, there is much more hard development work ahead of us. The official release is planned for the second half of 2012. We will keep you posted, so subscribe to our Newsletter at www.mdt.net. If you would like to participate in the beta testing program you are welcome to apply (write to support@mdt.net).
MDT in Ubisense

ASTEC has been a business partner of Ubisense Germany for many years. We have done many of Smallworld projects together and successfully implemented various applications to customers from different enterprises. They convinced themselves to start using MDT in Spring of 2011 and have been applying our tool to more and more tasks ever since. Below you can read through a short conversation between Blażej Sytar (ASTEC) and two of Ubisense’s developers: Hans-Peter Barth (Hape) and Michael Pahlke.

[Błażej] Good day Hape, Michael. Thank you for agreeing to answer some questions. For starters I would like to ask you how long have you been working as a developer/Magik developer?

[Hape] I’ve been working as Magik developer for 12 years.

[Michael] I’m a young professional. I have started my work in the beginning of the year 2011 and begun developing in Magik just half a year ago.

[Błażej] What do you think about how trends in software development have changed over the past years and how Magik programming goes by them?

[Hape] In general software development becomes more and more agile, but Magik programming is too inflexible to react on these changing needs because it’s a highly specific development environment and there is no big community that constantly improves it.

[Błażej] What was a bigger challenge in your case, convincing the board or changing the mindset of your colleagues and yourselves?

[Hape] I think the hardest part is to convince yourself. Once we (developers) were convinced that we will benefit from working with MDT, the board was willing to follow this decision. Additionally, the board realizes that it’s really hard for beginners (like people coming straight from university) to become familiar with Emacs development environment. Using MDT has a big advantage - it’s based on a well-known platform and therefore could speed-up the adjustment to the tasks of a Magik developer.

[Michael] I can only agree with Hape because I knew Eclipse from my studies. Emacs seems to be too old-fashioned to me and therefore I am really glad that there is a Magik plugin for Eclipse. I really get on well with MDT due to my experiences with Eclipse. The decision came just at the right time for me, since developing with Emacs would have been a tough nut to crack.

[Błażej] What exactly has changed your mind and convinced you to switch from Emacs to MDT?

[Hape] I hoped to benefit from the many available tools (code assistant, refactoring, debugger, etc.) that MDT offers within one integrated environment. Additionally, I thought it could be very helpful to familiarize with an environment which is used as standard environment for other languages as well.

[Błażej] What is the general attitude towards MDT today? Do you think you will be ready to make a full switch eventually?

[Hape] Honestly, my feeling is that I’m still not 100% comfortable using MDT. Especially, there are areas I’m just faster using familiar Emacs short keys. But I think this is just a matter of time and configuration. I’m convinced I will get used to it. What still prevents me from fully switching is the fact that preparing your runtime/environment seems to require more effort and is not as flexible as with Emacs.

[Błażej] Did you experience any challenges with configuration or adaptation? How long did it take you to feel comfortable using it?

[Hape] The only configuration I’ve done frequently was the configuration of short keys. The configuration seems very intuitive and very helpful.

[Michael] Since our development environment is deployed on a server, the installation and initial configuration was done by our configuration manager. I only did some small adaptions concerning short keys, comments, code assistant and stuff. Such configurations are done really quickly and you can export your adaptions to another MDT image, which is quite useful. It took me only a few days to feel comfortable using MDT.

[Błażej] You have been using MDT for around half a year. What would you say about it today, when you already have some perspective?

[Hape] MDT is a powerful tool and I still discover interesting and useful functionalities. The debug mode for instance is very helpful, because it allows to explicitly inspect problems and system behavior at runtime. Searching terms and methods or browsing source code is really intuitive. To sum it up MDT provides a quick, broad and also detailed view into your code.

[Błażej] You both have different areas of expertise. What sort of tasks is our tool most useful for?

[Hape] Since I usually do not develop new features from scratch MDT is most useful for me when it comes to tasks like validation of bugs. Especially the debugger is a powerful tool. MDT also helps me to keep the code consistent while restructuring it, e.g. by renaming classes.

[Michael] Due to the aforementioned features, I think that MDT is most useful when it comes to bug fixing and refactoring… and simply coding, of course.

[Błażej] You have also attended the training that was arranged in Ubisense, Darmstadt in summer last year. Please comment it in a few words.

[Hape] The training was a good starting point to the general concepts of MDT, especially if you are not familiar with Eclipse.

[Błażej] Would you recommend MDT to other Magik developers?

[Hape] Yes, I would definitely recommend MDT to other Magik developers.

[Michael] I can certainly recommend it as well, because it is a powerful and convenient tool!
Magik Tricks

by Michał Bielewicz, Software Developer, ASTEC

I have been working for ASTEC for around six years. Being involved in several different projects gave me an opportunity to broaden my knowledge and learn many new things. In my work very often not only programming skills but also some “magic tricks” were needed in order to successfully complete tasks I was given. Now, I would like to share with you a handful of such tricks that I have learned and gathered over the years. Some of them may seem pretty trivial but I hope they will be useful to you as they have been to me.

1. How to draw a trail on the map using a given (pseudo) geometry?

This trick can be used to observe the exact location of wanted geometry on the map. Hint: Do not forget to set the right value for !current_coordinate_system! dynamic variable in the Magik console.

```
local my_map << my_app.get_service_provider(:map_manager).current_map
local pg << pseudo_point.new_at(coordinate.new(1, 1))
my_map.set_trail_from_geometry(pg)
```

2. How to fetch a record that is currently selected on the map?

First get a set of currently selected geometries on the map:

1. Using a databus

```
current_map_selection << my_app.databus.request_data(:map_selection)[2]
```

2. Directly from the map

```
local my_map << my_app.get_service_provider(:map_manager).current_map
current_map_selection << my_map.current_selection
```

Second we take the first geometry from the set and fetch the record that it belongs to.

```
my_rec << current_map_selection.an_element().rwo
```

3. How to log in as a different user without the need of closing the application first?

```
gis_program_manager.authorisation_view.login("login", "pass")
```

4. How to measure the length of the geometry on the map?

The returned value is expressed in units defined in application’s coordinate system or in database units if !current_coordinate_system! is not set.

```
local my_map << app.plugin(:maps).current_map
local current_sel << my_map.current_selection.an_element().rwo
local unit << !current_coordinate_system!.default(current_sel.source_view.world).unit_name
write(current_sel.route.line_length, %space, unit.write_string)
```

Example Console Output:

```
105.3 m
```

5. How to force messages to reload?

It is useful when we change messages in many modules and want to see the result without the need to compile each module’s messages separately.

```
sw_module_manager.register_messages()
```
6. How to check which method displayed a pop-up?

Example: message window pops up in the application. You would like to know what part of code is responsible for displaying this message and what was the execution flow that led to this pop-up.

```ruby
thread.named(:events).traceback()
```

7. How to speed up the network analyzer?

Disable the drawing of visited geometries to speed up the analyzing process. Where _self is a custom plugin that makes use of `network_analysis_component` hold in :network_analyser slot

```ruby
_self.network_analyser.engine.remove_dependent(_self.network_analyser, :link_encountered)
```

8. How to write out elements from any container with a given separator?

```ruby
_local numbers << {1,2,3}
write(write_string_with_separator(numbers, %-))

Console Output:
1-2-3
```

9. How to use the `evaluate()` method?

```ruby
_local sum << "9+1"
sum.evaluate()

Console Output:
10
```

10. How to use `magik_text` to define any procedure on-the-fly?

```ruby
_local mt << magik_text.new()
mt.add_last("_proc (num)"
mt.add_last("_return num * num"
mt.add_last("_endproc"
_local p2 << mt.evaluate()
write( p2(3) )

Console Output:
9
```

11. How to remove a message that is already defined in the image?

This trick is extremely useful when submitting changes as patches.

```ruby
read_message_patch(:my_exemplar)
$
:my_useless_message :|REMOVE|
$
```

12. How to check RGB components for a specific color?

```ruby
colour.called(:indigo)

Console Output:
colour(R=0.1175,G=0.000,B=0.3076)
```

13. How to debug a method without having its source code?

It sometimes happens that we have no source code for a method we are interested in. So how can we add a new functionality to this method that could be useful during debugging? I’ve learnt there are two ways:

1. Redefining the method:

First create a copy of the original method:

```ruby
orig_meth << basic_window.method(:show_alert).value

Write your own body for this method

```ruby
_method basic_window.show_alert(message, _optional yes_message, no_message, default, mode)
  ## Redefined method
  !traceback!(:terminal!)
  _endmethod
$

When the debugging is finished restore the original method

```ruby
basic_window.method(:show_alert).value << orig_method

The only disadvantage of this approach is that the original functionality gets omitted for the sake of debugging.

2. Defining a synonym for the method:

Clone the original method and save it under a different name

```ruby
basic_window.define_method_synonym(:|clone_show_alert()|, :|show_alert()|)

Write your own body and call the cloned method.

```ruby
_method basic_window.show_alert(message, _optional yes_message, no_message, default, mode)
  ## Redefined method with preserved original functionality
  !traceback!(:terminal!)
  _return self.clone_show_alert(message, yes_message, no_message, default, mode)
  _endmethod
$

In result the functionality of `basic_window.show_alert()` method will stay the same and you will be able to modify its contents. In those two approaches `!traceback!(:terminal!)` has been added in order to display the execution flow that has led to the alert window.
Training Feedback

An interview with Dr. Jun Gang from GE Energy, Cambridge. She was one of the participants of a training session which was organized in UK in Summer of 2011. Now she agrees to answer a couple of questions and recall her impressions from the training.

[ASTEC] Hello Jun. You work for GE Energy and I am sure you have attended a great deal of training sessions for various purposes. Do you consider this kind of teachings effective?

[Jun] Definitely. When a block of time is allocated for training on a specific subject, and with an experienced and knowledgeable trainer, it is very effective. With some preparation beforehand and support after the training, the learning can be very efficient and productive.

[ASTEC] You used MDT for some time before you attended the training. What was your attitude towards it?

[Jun] I started doing Magik development using Emacs two years ago when I joined GE. I was already familiar with Emacs at the time. Emacs had been extensively extended and enhanced for Magik development and was the default tool to use.

When I started using Eclipse for Java development, I realized the benefit of Eclipse, and started to think that I could also benefit from using MDT and having one tool for both Java and Magik development. I downloaded it and attempted to configure it for a couple of times, but it was not very successful. I tried to get some help but at that stage, I did not really have the right concepts in mind, such as sessions and runtimes, to ask the right questions and get useful answers.

[ASTEC] What was the most helpful part of it? Did you like the training?

[Jun] Yes. I enjoyed the training. The most helpful part was learning how to configure the runtimes and sessions, getting to understand the three different kinds of session and also how environment variables configured at different places work together. After the training I was able to use MDT for development work. I found the workbook very useful with step-by-step guides and screenshots.

[ASTEC] The trainer’s personality also plays an important role. How was the communication and atmosphere in the class?

[Jun] The trainer was very knowledgeable and attentive, and the atmosphere was interactive.

[ASTEC] Do you consider the training a must-have to get started with MDT or rather a useful addition for organizing and extending your skill?

[Jun] I think attending the training was a must-have for me, as I would not have been able to start developing Magik in MDT without it.

[ASTEC] After this time that you have spent working with MDT do you see yourselves going back to Emacs?

[Jun] After my MDT training, I used MDT for most of my Magik development work. MDT offers (more or less) the same benefit as the Eclipse workbench, such as code assistants, perspectives for coding and debugging, easy access to files, various plugins, running multiple sessions and so on. The “GIS Commands Managing” perspective shows all command history grouped for all sessions used, which is very useful indeed. Through the “Hierarchy View” I can access the files defining exemplars by a simple mouse click. Building the image is also straightforward with the “External Tools” menu option.

I used to go back to Emacs to create patches and change notes. However, I was recently made aware of the new Patching System plugin which allows me to do this through MDT, so now I rarely have a reason to go back to Emacs.

If you wish to learn more about the training contents, availability or cost write to us at sales@mdt.net. Allow yourself to be better, get your training from the experts!
Powerful solutions through **advanced** technologies.

As a strong GE Energy authorized Solutions Provider we offer software development services for Smallworld™ users from utilities and telecommunication industry sectors.