Significant improvements in the newest MDT PE 1.1 release

by Dariusz Michura, ASTEC

We are happy to inform you about the new version of MDT. We have made a lot of improvements. Here are the most important ones:

• Improved debugger – more efficient debugging mechanism, improved breakpoint management and expression inspection.

• New session type called Command Line Session – enables users to use external commands to run Smallworld™ sessions. It allows you to run complex session configurations based on batch files or executable files. Just like other session types, command line sessions can be easily shared between developers within the team.

• Easier session management – set of new features which facilitates relations between Magik projects and Smallworld™ sessions.

• New Magik Editor Quick Outline – now users have access to the outline view directly from the editor. Depending on the style of work you can choose between static or quick outline.

• Complete ClearCase support has been fully tested and confirmed.

A list containing all new improvements and changes as well as more details about MDT 1.1 can be found on our website at www.mdt.net. Please also visit our forum and share your impressions about MDT PE 1.1 with the community. Of course we are always open to new suggestions that will make MDT even better, so please feel free to post about them, too.

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 it is being used by companies all over the world.

Check out the support forum for questions, read the documentation and subscribe to the newsletter.

Visit us today at www.mdt.net

Dear Readers,

Here we give into your hands the next issue of MDT Magazine. What you can find in it? At the beginning we shortly present changes and facilities implemented with the new version of MDT: MDT Professional Edition 1.1.

Optimizing the work of the whole team during leading the project – this matter is our leitmotiv and pushed us to invent and develop MDT. Our second article describes how to support the usage of MDT with Mylyn in order to enable programmers to focus on development tasks only, to help the project managers in the distribution of tasks and to control the development process within the whole team.

If the specific needs from your clients forced you to use solutions combining Smallworld™ technology and data from external software, the second article in the Technology section is for you. “Cross-platform communication in Magik via JMS” shows a few hints on how to create a connection between Smallworld™ and Java application components.

In Interview section Mr. Jan Kiefer from NIS AG tells about his experience with MDT and explains the reason of implementing MDT at his company. The issue is closed with Training section where you can find an overview of MDT trainings.

We hope the presented content is interesting and useful at your daily work.

ASTEC, MDT Team
Task focused development with MDT and Mylyn

by Michał Niewrzał and Wojciech Trocki, ASTEC

The proper organisation of every developer’s work is a key to success of all IT projects. Project managers face new challenges caused by the increasing size and complexity of projects, while the number of additional tasks assigned to each member of the team grows. Tools that allow to oversee assigned tasks, arising errors and other aspects of the project thus become very valuable.

When using tools integrated with development environment best practises can be introduced smoothly with no negative influence on the organisation of the teamwork. Making code development a central point of the work facilitates efficient communication and task control. Such innovations considerably improve the work environment of both the project manager and developer, resulting in better quality of created software.

A further part of the article presents the most important information about task focused development as well as an example of using MDT integrated with a task focused interface called Mylyn.

Task focused development

Task focused development is a software development methodology, marked by task driven developer’s work. The tasks are available directly in the development environment. The process of managing a task (or task portfolio) through a project’s life cycle includes: planning, testing, tracking and reporting. Task management can either help individuals to achieve their goals or groups of individuals to collaborate and share the knowledge needed to accomplish collective goals. The tasks are also differentiated by complexity and priority.

Task focused development defines a procedure allowing to execute the most important tasks in a set order and with suitable resources. The developer can focus on his task instead of tracking changes occurring in external task management tools. Task focused software development enables programmers to work independently on the same or different parts of a single coding project.

Mylyn

Mylyn is a commonly used task management solution for Eclipse. It can be integrated with MDT through a dedicated bridge. Due to Eclipse platform extensibility, such configuration can be simply extended with additional Eclipse plugins for issue tracking systems (Bugzilla, Trac, Jira, etc.), for teamwork (SVN/CVS, Subversion, Clear Case) and others. Mylyn’s interface allows defining and assigning tasks accessible only by the owner. Additionally such tasks can be presented to other team members in a remote system, e.g. Bugzilla. Mylyn saves the developer’s time by relieving him of numerous redundant actions. It filters, sorts, highlights, folds and manages tree expansion for numerous views within the Eclipse IDE based on the currently active task. It is possible to add comments, assign additional resources and monitor elapsed time.

Mylyn in practice

To start using Mylyn in Smallworld™ projects the following products have to be installed: Eclipse, MDT plugin, Mylyn plugin and MDT Mylyn Bridge (available at www.mdt.net). As a next step, the task repository (e.g. Bugzilla) has to be chosen, then tasks can be created or retrieved. The added tasks are being synchronised periodically and the changes made by other users are being reported.

Once the tasks are integrated and one of them is active, Mylyn monitors work activity to identify relevant information and uses this task context to focus the user interface on the task-at-hand. This puts the information at the fingertips and improves productivity by reducing searching, scrolling, and navigation. By making task context explicit, Mylyn facilitates multitasking, planning, reusing past efforts, and sharing expertise.

Reviewer can take advantage of incorporating Mylyn into software development process. Each task can be associated with related source code. Additional binding with description provides information about reasons, assumptions and background of the task. This approach allows sharing knowledge among team members.

Additional functionality of Mylyn is registration of task’s duration without inconvenient noting, counting and reporting.

Summary

With tools like MDT and Mylyn it is possible to develop Smallworld™ solutions in very convenient and well organised way. Applying task focused development makes multi-tasking work easy, reduces information overload and improves team productivity.

More information about Mylyn and task focused development in Eclipse can be found on Mylyn project official web site: www.eclipse.org/mylyn.
Software solutions are becoming more complex. Modern applications are no longer standalone systems but create heterogeneous software infrastructure. Therefore it is more and more often required to cooperate with technologies from outside Smallworld™ ecosystem.

As an example we can mention a solution managing telecommunication network infrastructure based on Smallworld™ Physical Network Inventory supplied with data from external source, such as Customer Relationship Management software. In such case it is essential to provide an efficient system of data exchange through a common interface. Smallworld™ CST supports multiple standard interfaces enabling such cooperation, e.g. CORBA, TICS or JMS. In this article we would like to present Sun’s Java Message Service (JMS) and show step by step how to implement the interface to enable the connection between Smallworld™ and Java application components.

In programming of disparate systems it is crucial to choose an appropriate development environment (IDE). Eclipse - one of the newest IDEs - provides an advanced graphical interface as well as considerable number of plugins enabling an efficient development in many technologies. A convenient way of using Eclipse for work with both Smallworld™ and Java technologies is to use MDT for Magik part and additional JDT plugin for Java part.

About JMS

The JMS API is a Messaging Oriented Middleware and an integral part of the Java 2 Enterprise Edition (J2EE). It is very useful for enterprise applications integration (EAI) thanks to providing enterprise messaging in Java and a framework for building message driven software. The JMS defines a simple set of instructions helpful at development stage. JMS-based solutions are very flexible and can support both synchronous and asynchronous modes of communication. JMS clients are the most commonly Java-based applications, but can be written in any modern programming language. Some vendors support great variety of clients for many platforms and languages.

Requirements

In order to use messaging functionality the eai_messaging module must be loaded. It can be found under “sw_core” branch in Module Dialog. It is also necessary to install J2EE and a message server e.g. Sun’s Java Message Queue (JMQ). There are two Magik classes: jms_user and jms_channel which can be used to implement a program that will communicate with other JMS-clients. The first one provides connectivity and sending facilities, the second one can open and close a topic or a queue and receive messages.
Opening a connection is as simple as invoking method new() on jms_user with appropriate arguments.

```plaintext
_jlocal jmq_home, classpath, acp_command, conn_class, conn_args, connection
jmq_home  << "C:\Program Files\JavaMessageQueue1.1"
classpath << "u:\swdev\users\peters\jms\swjms.jar;" + "+" + "C:\j2sdkeel.2.1\lib\j2ee.jar" + "+" + "+" + "jmq_home + "+" + jmq_home + ";"
acp_command << "java -classpath " + classpath + " com.gesmallworld.extdb.ExtdbAcp"
conn_class << "com.gesmallworld.jms.jmq.JmqConnection"
conn_args << {"server", "test_server"}
connection << jms_user.new(acp_command, conn_class, _scatter conn_args)
```

When a connection is made the client can start sending messages. Simple message contains body and queue or topic, which is mandatory. Additionally, headers, properties, and other parameters can be used for more sophisticated communication.

```plaintext
_local body, topic
body       << "Hello World!"
topic      << "helloworld.topic"
connection.send( hash_table.new_with( :body, body), :topic, topic)
```

To read a message, a channel (topic or queue) must be opened. It can be achieved by invoking open() method on jms_user with at least one argument to specify whether the channel is a topic or a queue.

```plaintext
_local topic, timeout, channel
topic      << "helloworld.topic"
timeout    << 10000
channel << connection.open( :topic, topic, :timeout, timeout)
```

Messages can be read in synchronous or asynchronous manner. The method receive() of jms_channel returns next message from the channel or “unset” if there were no massages or the wait time is up (“timeout” parameter).

```plaintext
_local message
message << channel.receive()
_if message _isnt _unset
  _then
    write(message[:,body])
  _else
    write("No message received")
endif
```

There is another, more flexible way of receiving messages. It gives possibility to use a custom receiver implementing method on_message() which will be called every time a message is read from the channel. A sample receiver class can be implemented as follows.

```plaintext
Pragma(classify_level=basic, topic={eai_messaging})
def_slotted_exemplar(message_listener, {})

Pragma(classify_level=basic, topic={eai_messaging}, usage={internal})
_method message_listener.on_message(message)
  write(message[:,Body])
_endmethod
```

The next step is to invoke listen() on jms_channel with an instance of receiver as the argument. In this case a receiver of type “listener” is used.

```plaintext
_local message
message << channel.listen(message_listener.new())
```

Before terminating the program, all the connections and channels must be closed. It can be done by invoking close() on jms_channel and then disconnect() on jms_user.

```plaintext
channel.close()
connection.disconnect()
extdb_java_acp.close_all()
```
Smallworld™ JMS client is a powerful integration technology proved in practice. The example described in the article presents integration of IT systems within EAI architecture. It allows enterprise solutions to be interconnected in order to deliver maximum business benefit to the customer.

The most convenient tool for creating this kind of software is an IDE for both Magik and Java. MDT enables development and management of the source code in both languages. It is also possible to switch easily between different language-dependent perspectives. This approach allows to save some coding time and simplifies developer's work.

### Java Part

Sample Java code is presented here. The following section presents how to send a message.

```java
Context initialContex = new InitialContext();
ConnectionFactory connectionFactory = (ConnectionFactory)
    initialContex.lookup("helloworldCF");
Topic topic = (Topic) initialContex.lookup("helloworld.topic");
initialContex.close();
Connection connection = connectionFactory.createConnection();
Session session = connection.createSession(true, 0);
MessageProducer producer = session.createProducer(null);
TextMessage message = session.createTextMessage();
message.setText("Hello World!");
producer.send(topic, message);
session.commit();
connection.close();
```

This code shows how to receive a message.

```java
Context initialContex = new InitialContext();
ConnectionFactory connectionFactory = (ConnectionFactory)
    initialContex.lookup("helloworldCF");
Topic topic = (Topic) initialContex.lookup("helloworld.topic");
initialContex.close();
Connection connection = connectionFactory.createConnection();
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
MessageConsumer consumer = session.createConsumer(topic);
connection.start();
TextMessage message = (TextMessage) consumer.receive();
System.out.println(message.getText());
connection.close();
```

### Summary

Smallworld™ JMS client is a powerful integration technology proved in practice. The example described in the article presents integration of IT systems within EAI architecture. It allows enterprise solutions to be interconnected in order to deliver maximum business benefit to the customer.
Interview with Mr. Jan Kiefer

Mr. Kiefer, you have been one of the first developers interested in MDT. Already in 2006 we collected a lot of hints and remarks from you, which were very useful during the development of MDT. Why are you interested in this tool?

Over ten years ago I began development with Smallworld™ Magik using the good old Emacs. Like every classic Magik developer I became very familiar with the Emacs but I never got rid of a like-to-have feeling when I took a look at IDEs available for other modern programming languages. Before MDT there had been one try to develop an IDE for Magik called Magician which was associated with a very powerful Magik Studio Tools (MST) Debugger running with Smallworld™ CST 3.2.1. In contrast to the MST Debugger, the Magician Code editor was difficult to use within this suite. When Smallworld™ CST 4 was released, these tools - unfortunately including the debugger - were no longer supported, so I had to work with plain Emacs again for some years. When I discovered the first versions of MDT in 2006, I was very happy that someone was spending resources on bringing a modern graphical IDE to Smallworld™ Magik development again. I took a look at the first version. Because it didn’t support GIS aliases, I gave a hint to ASTEC and deferred the use of the tool until a new version came out. Since then I consistently increased use of MDT and decreased the use of Emacs during my daily work.

Now you are an advanced MDT user and working for NIS AG you use MDT every day. Why did you choose this new environment?

When I started working for NIS AG I had already approximately two years of experience working with MDT. As even the early versions had increased work productivity quite remarkable, I encouraged the whole company to use MDT. Fortunately, at that time general changes to the development system have been planned and scheduled. We did the change and MDT was the choice to replace Emacs. The MDT features, Eclipse’s own powerful features, the support of other languages like Java, and its ease of integration with hundreds of plugins available made it possible to shape a really smart, modern and intuitive development environment for the whole team. Today we, at NIS AG, are proud that we’ve been the first company which has officially switched to MDT.

What was the biggest challenge when switching from Emacs to MDT?

To move from Emacs to MDT wasn’t really a big deal apart from the licensing mechanism. At workstations we had some trouble connecting to license server to get licenses activated. However, in cooperation with ASTEC support team we managed to activate the licences. The switch of the development environment was planned as an internal project and planned resources gave time to developers to become familiar with MDT. So the tool was accepted very fast throughout the whole NIS AG team and everybody were very motivated to switch.

What is the most effective aspect of using an integrated development environment like MDT?

As mentioned before, generally MDT brings a modern environment to Magik development. The most effective aspect - from my point of view - is the enormous increase in efficiency and productivity, of course depending on the task to be solved. For example during our Smallworld™ CST 4.1.1 upgrade work, where we did a lot of code consolidation, refactoring and code cleanup, we brought our task processing time down to 50% roughly, compared to Emacs.

In what way can the multiplatform ability of Eclipse facilitate everyday work?

To be honest, I’m mainly working on Magik since I work for NIS AG, so this multiplatform ability isn’t too important for me. For my previous employer I did some work on SIAS and MDT provided a very easy way to work on Magik and Java with the same tool. Today, if I take a look on my teammates in charge of development and support of SIAS solutions here at NIS AG, they seem to be very happy that ASTEC eased their job.

Does MDT correspond with your needs concerning team work?

Yes. With the integration of our version control system and the possibility of exchanging session definitions our needs concerning team working are fulfilled. At the moment we are not using the features for team work provided by the Mylyn integration and I can’t tell whether we’ll use it in future.

You are working on Smallworld™ version CST 4.1 which includes a debugger. Do you consider this functionality helpful?

Of course, it’s mandatory! Nowadays, every modern piece of software is developed using modern tools including a debugger. As I said before, working on Smallworld™ CST 3.2.1 I was using Magik Studio Tools Debugger heavily and when this tool was discontinued with CST 4.x, I missed a debugger again for years. The MDT debugger is quite good and usable, although there are some issues to be improved to make it a perfect tool, particularly for bug fixing. I provided a list of suggestions for enhancements and fixes to ASTEC and - based on our collaboration in the past - I’m quite confident that those features will be implemented in future versions.

What is the number one tip you would give to Smallworld™ developers?

Buy, install and use MDT and reserve some time to get into it. Quite fast you’ll feel very comfortable using it and simplify your daily work. Finally, MDT won’t solve your problems but it will make you faster having more fun on solving them!

Thank you for the interview!
Choose the smart way to learn MDT

by Bartłomiej Łączkowski, ASTEC

In the modern competitive world, a growing number of employers need staff with high level skills. Users of programming tools gradually develop skills that enable them to perform their tasks more efficiently. Such process may take some time. Teams often decide to make use of trainings to accelerate learning. Trainings entail more efficient programming, and less time spent reading manuals and making novice errors.

The trainings are prepared by ASTEC and mainly intended for Magik developers. However, some elements of the presented information can be very useful for projects managers and software architects. The following trainings feature instructor-led classroom teaching, labs and practical exercises.

Our first course Moving up to MDT is a set of presentations and exercises for the standard training type. The main goal of this course is to teach the students how to get started with MDT. The contents of this training are focused on the moving up to the MDT Professional Edition environment in most effective way.

Development techniques with the use of MDT is an advanced training type, which is an extension to the introductory course. Students will acquire knowledge about most effective ways of creating and managing Smallworld™ sessions and projects. The participants will also learn how to migrate existing and complex configurations of the sessions from Emacs to MDT environment. The main goal of the second part of the training is to learn how to use the crucial features of MDT in line with advanced development techniques to increase the efficiency of Magik code development.

Advanced aspects of MDT environment is an additional course which is an extension to the advanced training type. The main goal of this training is to present the possibilities of extending MDT development environment with the set of very useful plug-ins which can boost the productivity and speed up the development process.

All trainings can be carried out on customer site or in our training center.

---

## Moving up to MDT

<table>
<thead>
<tr>
<th>DAY</th>
<th>SUBJECT</th>
</tr>
</thead>
</table>
| Day 1 | 1. Introduction to MDT  
2. MDT architecture  
3. Major features of MDT  
4. Installing Eclipse and MDT  
5. Creating and managing projects  
6. Creating and managing sessions  
7. Getting started with Magik code development  
8. Overall discussion |
| Day 2 | 1. Runtime, Session and Session Link  
2. MDT vs. Emacs session configuration ideology  
3. Managing runtimes and session  
4. Working with Magik projects  
5. Magik code development techniques  
6. Transmitting Magik elements to the session  
7. Working with the console  
8. Using additional tools supporting development |
| Day 3 | 1. Working with MDT’s Class Browser  
2. Using Magik Search  
3. Using Magik File Search  
4. Browsing perspectives  
5. Code navigation with the use of Magik Hyperlinks  
6. Using Magik Debugger  
7. Best practices summary  
8. Overall discussion |

## Development techniques with the use of MDT

<table>
<thead>
<tr>
<th>DAY</th>
<th>SUBJECT</th>
</tr>
</thead>
</table>
| Day 1 | 1. Runtime, Session and Session Link  
2. MDT vs. Emacs session configuration ideology  
3. Managing runtimes and session  
4. Working with Magik projects  
5. Magik code development techniques  
6. Transmitting Magik elements to the session  
7. Working with the console  
8. Using additional tools supporting development |
| Day 2 | 1. Working with MDT’s Class Browser  
2. Using Magik Search  
3. Using Magik File Search  
4. Browsing perspectives  
5. Code navigation with the use of Magik Hyperlinks  
6. Using Magik Debugger  
7. Best practices summary  
8. Overall discussion |

## Advanced aspects of MDT environment

<table>
<thead>
<tr>
<th>DAY</th>
<th>SUBJECT</th>
</tr>
</thead>
</table>
| Day 1 | 1. Extending MDT and Eclipse  
2. Installing additional plug-ins  
3. Team work in MDT  
4. Integration with Issue Tracking System  
5. Task-oriented development  
6. Composing multiplatform environment  
7. Using additional useful plug-ins  
8. Overall discussion |

---

Publisher ASTEC Sp. z o.o.  
© 2010 ASTEC Sp. z o.o.  
All rights reserved.
Powerful solutions through advanced technologies.

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

www.astec.net