embenatics
foundation layer software

WHITE PAPER

mbLog Logging and Diagnosis Tool

This white paper gives an introduction to the logging and diagnosis tool of the embenatics tool suite, called mbLog. It shows views of a typical logging session with a high level description of the tool's features and usage.

mbLog an Introduction

The logging and diagnosis tool of the embenatics tool suite is called mbLog. It is an Eclipse™ platform-based tool, which can be used as a plug-in for an existing Eclipse™ workbench or as a stand-alone application. The Eclipse™ platform is Java™-based and therefore runs on multiple different PC operating systems like Windows™, Linux™ and MacOS™.

mbLog connects with the target system via different types of interfaces, like USB or a socket port. All logging events sent by the target system will be captured and logged by the tool. Different views supported by the tool allow online and offline investigation of the target behavior. Events captured by the tool can be stored, distributed and reloaded for further analysis. mbLog, therefore, acts as a lightweight tool for different purposes during the development cycle; these include diagnosis and inspection tasks of the running system even via a remote connection.

mbLog is not a hardware debugger or emulator for in-deep system debugging purposes. Instead, it relies on instrumenting the code by using a dedicated logging API and inter-process communication traces that are logged automatically by the embenatics foundation layer software. mbLog is the visualization tool of the embenatics tool chain to display target system traces, statistics and historical data of the underlying foundation layer.

© 2011

Connecting with the Target

Due to the fact that the connection type of the logging interface varies among different target hardware platforms, mbLog provides a connection manager which handles the specifics of the physical connection.

An initial connection with the target system is usually done by using the connection wizard of mbLog. A connection is classified by a set of parameters that describe the connection details. These depend on the physical type of the connection and may vary. Beside the physical connection parameters, the user should provide a name of the connection, and optionally, a set of trace description files for increased usability of the tool. Trace description files are generated by the generator mbGen. For further information on mbGen and the embenatics design methodology please see another document in this white paper series [**embenatics Design Methodology**].

Below is an example for a socket connection created using the connection wizard.

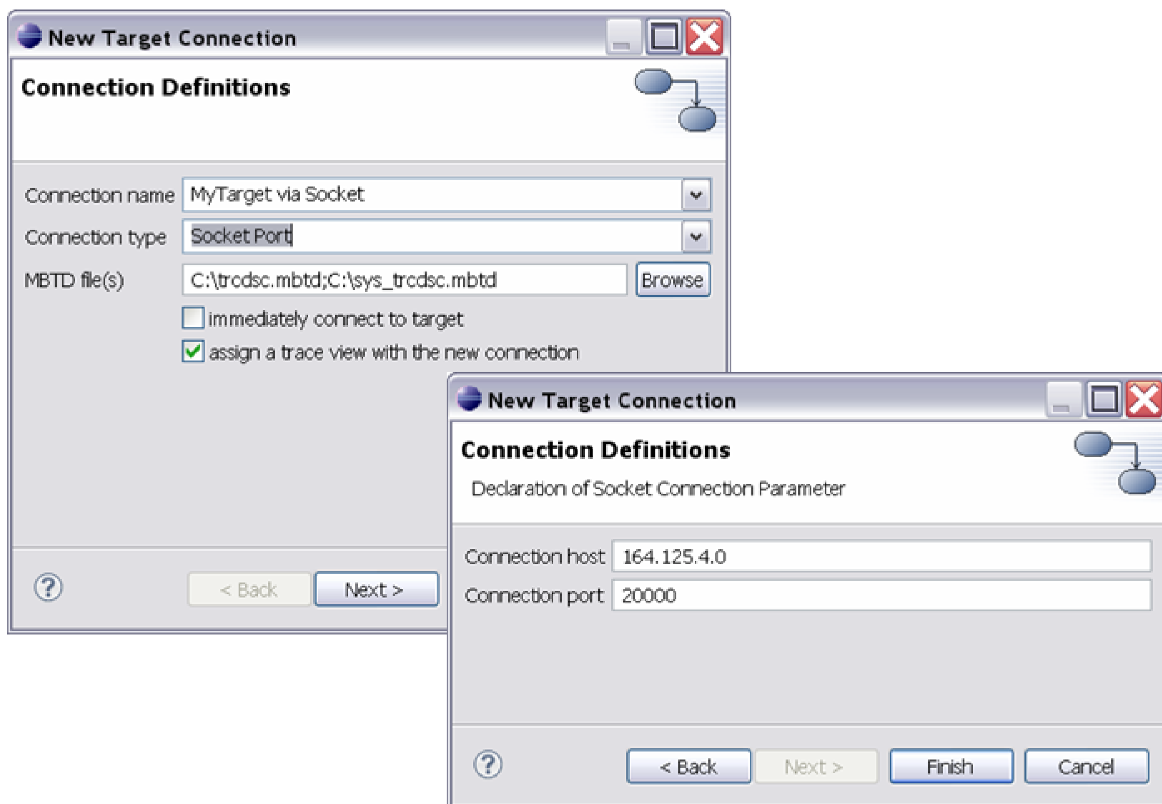

Figure 1 Connection Wizard

Once a connection has been created, all parameters are stored by mbLog and will be available for a restart of that connection during the next session.

Since a complex system can consist of more than one CPU core, mbLog's connection manager supports that fact and can handle several connections in parallel. All currently established

connections of a session are displayed in the Connection Status View. The different status of the connections can be supervised and modified by a couple of controls assigned to each connection status display. Below is a screenshot showing three active connections and their corresponding status. From top to bottom:

- A serial connection using COM port 1 interface. The logger is connected and the connection has been paused by the user
- A log file connection reading the contents of the logfile `myLogfile.mbLog`. The logger is connected and the connection is active.
- A socket connection with port 20000 of IP 164.250.4.0. The logger is currently disconnected and therefore the connection is inactive.

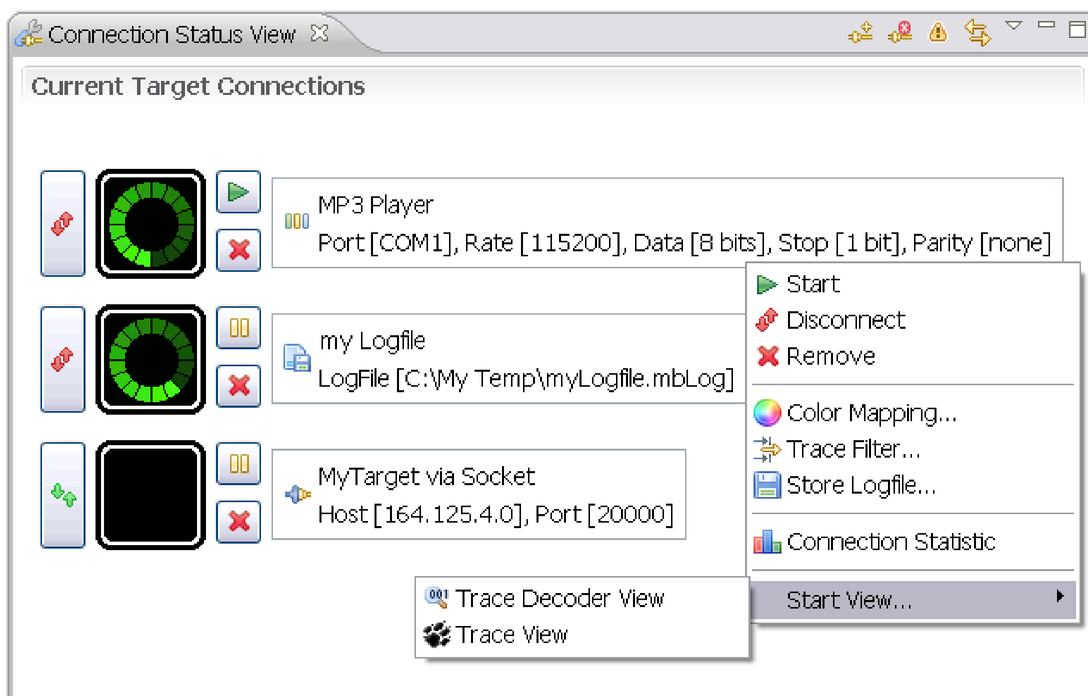

Figure 2 Connection Status View

The Connection Status View provides controls to create a new connection, remove an existing connection set-up and restore a previous connection when starting a new session. Each connection status display provides a context menu that allows the user to change the connection status, set and modify color mappings and trace filters, store the log file or start an associated view for the connection. See Figure 2 for more details.

Inspecting the Trace View

Once a connection with the target is established and the system is up and running, diagnostic and trace information will be logged by the tool. All of this data can be inspected by opening one or several Trace Views for that particular connection. A Trace View can be opened via the context menu of the connection display in the Connection Status View, see Figure 2.
A Trace View shows trace data in a tabular fashion with different columns dedicated to particular parts of the trace information, like the timestamp, the sender and additional detailed information depending on the type of the trace.
Below is an example screenshot for a Trace Viewer

MP3 Player Trace View [1]

timestamp	sender	details
58725	UI	UI_CONTROL_key_pressed() reached
58726	UI	call to [RENDERER].RENDERER_CONTROL_get_playlist()
58733	RENDERER	call to [STORAGE].STORAGE_DATA_open()
58734	RENDERER	call to [STORAGE].STORAGE_DATA_get_next_block()
58735	UI	call to [RENDERER].RENDERER_CONTROL_start()
58736	DISPLAY	DISPLAY_CONTROL_set_mode() reached
58736	DISPLAY	mode set to 0
58756	UI	call to [DISPLAY].DISPLAY_CONTROL_set_mode()
58757	KEYPAD	KEYPAD_CONTROL_set_mode() reached
58757	UI	call to [KEYPAD].KEYPAD_CONTROL_set_mode()
58758	DISPLAY	Element: DISPLAY_CONTROL_FIELD_CONTENT_type
58783	UI	call to [DISPLAY].DISPLAY_CONTROL_set_content()
58788	UI	call to [RENDERER].RENDERER_CONTROL_get_status()
58799	UI	call to [STORAGE].STORAGE_INFO_get_track_info()
58816	UI	call to [STORAGE].STORAGE_INFO_get_cover()
58824	DISPLAY	Element: DISPLAY_CONTROL_FIELD_CONTENT_type
58829	UI	call to [DISPLAY].DISPLAY_CONTROL_set_content()
58859	UI	call to [STORAGE].STORAGE_INFO_get_cover()
58866	DISPLAY	Element: DISPLAY_CONTROL_FIELD_CONTENT_type
58871	UI	call to [DISPLAY].DISPLAY_CONTROL_set_content()
58968	UI	call to [STORAGE].STORAGE_INFO_get_cover()
59033	DISPLAY	Element: DISPLAY_CONTROL_FIELD_CONTENT_type
59042	UI	call to [DISPLAY].DISPLAY_CONTROL_set_content()
59147	UI	call to [STORAGE].STORAGE_INFO_get_cover()
59216	DISPLAY	Element: DISPLAY_CONTROL_FIELD_CONTENT_type
59224	UI	call to [DISPLAY].DISPLAY_CONTROL_set_content()
59393	UI	call to [STORAGE].STORAGE_INFO_get_cover()
59462	DISPLAY	Element: DISPLAY_CONTROL_FIELD_CONTENT_type
59469	UI	call to [DISPLAY].DISPLAY_CONTROL_set_content()
59577	UI	call to [STORAGE].STORAGE_INFO_get_cover()
59648	DISPLAY	Element: DISPLAY_CONTROL_FIELD_CONTENT_type
59656	UI	call to [DISPLAY].DISPLAY_CONTROL_set_content()
59727	RENDERER	call to [STORAGE].STORAGE_DATA_get_next_block()
59770	UI	call to [STORAGE].STORAGE_INFO_get_cover()
59787	UI	call to [STORAGE].STORAGE_INFO_get_track_info()
59800	DISPLAY	Element: DISPLAY_CONTROL_FIELD_CONTENT_type
59842	UI	call to [DISPLAY].DISPLAY_CONTROL_set_content()
59859	UI	call to [RENDERER].RENDERER_CONTROL_register_status_listener()
59860	KEYPAD	call to [UI].UI_CONTROL_key_pressed()
60300	UI	call to [RENDERER].RENDERER_CONTROL_get_current_track()
60310	UI	call to [STORAGE].STORAGE_INFO_get_track_info()
60312	DISPLAY	Element: DISPLAY_CONTROL_FIELD_CONTENT_type

Figure 3 Trace View

A Trace View window usually displays incoming traces for the respective connection immediately when they are logged by the tool. For a better inspection of the traces while the system is running, each Trace View display can be paused and scrolled to the desired position. The trace history displayed by the Trace Viewer can be purged at any time during the session to remove older or unnecessary traces from the view.

Currently two types of traces are supported. One of them is a text-based trace, that transports user-defined strings, which has been added to the application code by the programmer, utilizing the logging API of the embenatics foundation layer software. The other is a binary trace, which holds the data content of inter-process service calls or data type elements. Inter-process communication traces are generated automatically by the foundation layer software if enabled by the user. In addition, all other data type elements, that have been defined in interface description documents, can be logged via the logging API whenever needed and applicable for the running system. More details about how traces are generated by the target system and how the programmer is able to instrument the code with customized traces can be found in another publication of this series of white papers [mbLay Logging].

Color Mapping

For a better overview and separation of traces it is possible to map individual color settings to particular trace attributes. This could simply involve all traces from a specific sender, or, at a level of greater complexity, manage all calls of a specific RPC service that are performed between a dedicated sender and receiver. Color mappings are stored together with the connection settings; for this reason, every time a connection is restored in mbLog, the corresponding color mappings will be applied. To define color mappings for a particular connection, the color mapping dialog can be opened via the context menu of the connection display in the Connection Status View, see Figure 2. The dialog itself is shown below.

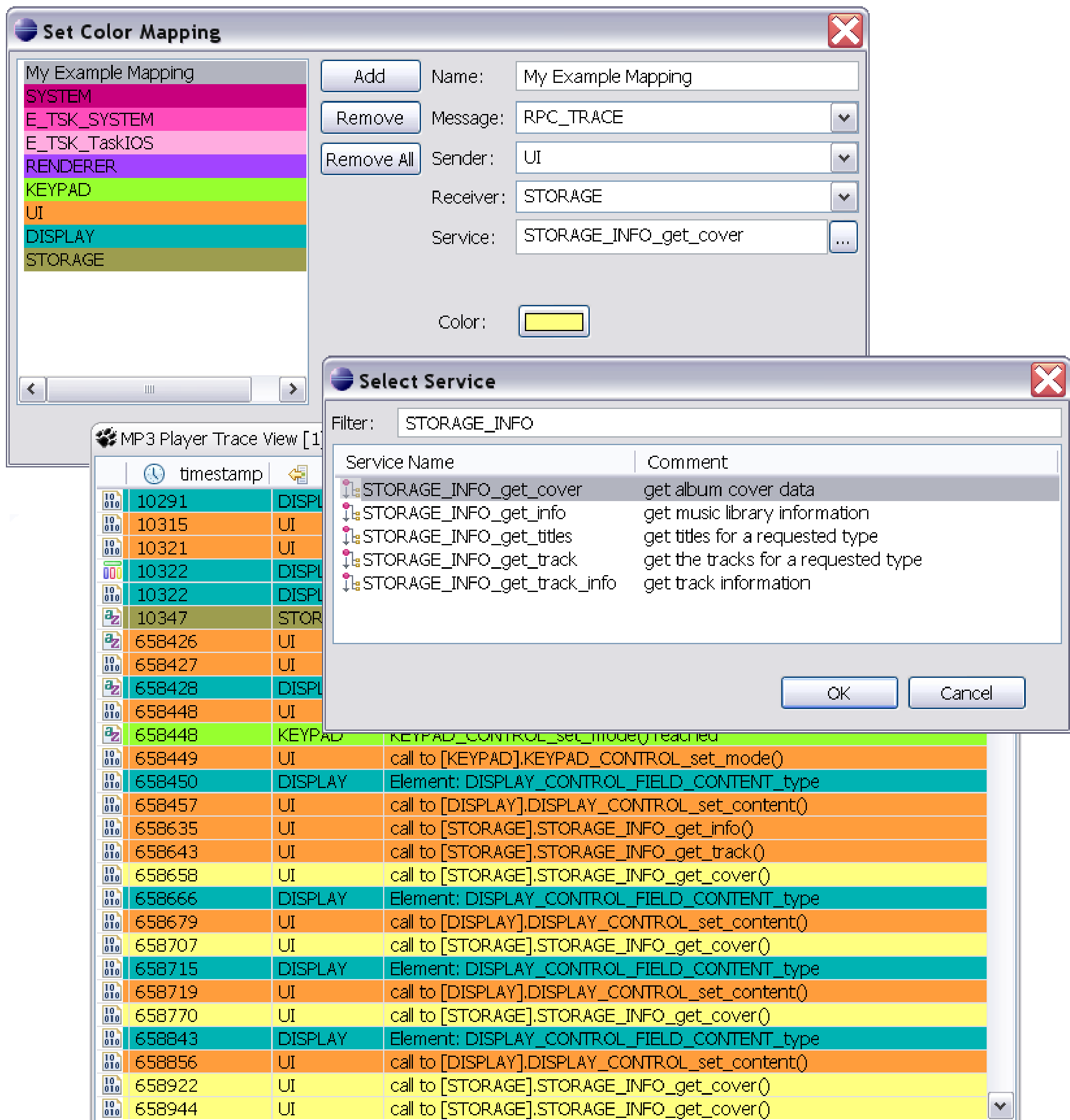

Figure 4 Color Mapping Dialog

Figure 4 illustrates a fairly complex setting for a color mapping as an example. In this display, a yellow color should be assigned to traces which are a combination of the following attributes:

- A binary RPC service trace
- The sender or better caller of the service is the thread with the name UI
- The receiver or callee for the service is the thread with the name STORAGE
- The particular RPC service that is called is `STORAGE_INFO_get_cover()`

For the selection of the particular RPC service another dialog window is available, which lists all services that are provided by the trace description files. These have been assigned to the connection during its set-up, as already mentioned above. The color matching rules for a connection will be processed from top to bottom. The first match will assign the corresponding color for that trace. Therefore, the order of the rules is important and can be sorted via drag and drop of the table elements on the left of the color mapping dialog.
The Trace View in the back of Figure 4 shows the outcome of the mapping and the yellowish RPC traces for `STORAGE_INFO_get_cover()`.
Color mapping is a simple but powerful feature of mbLog to detect certain patterns in target loggings that could easily be judged as normal or abnormal behaviour of the system under observation.

Trace Filter

Besides the above-mentioned color mappings, trace filters are another way to organize the traces that have been logged by the tool. Each Trace View has an individual set of active filters that decide which traces are allowed to pass and to be displayed by the viewer. These filters only affect the traces that have already been logged by mbLog. A trace filter, therefore, creates a subset of all available traces logged by the tool and can be changed by the user whenever needed.
Once a filter has been created, it is available for all Trace Viewers of a connection. Trace filters can be enabled or disabled per viewer which allows the user is able to create different filter combinations for organizing separate perspectives of the trace data in several Trace Views.

To define filter settings for a particular connection, the trace filter dialog can be opened via the context menu found in the connection display as part of the Connection Status View (see Figure 2), or by clicking the filter icon in the toolbar of the Trace Viewer.

The dialog itself is shown below.

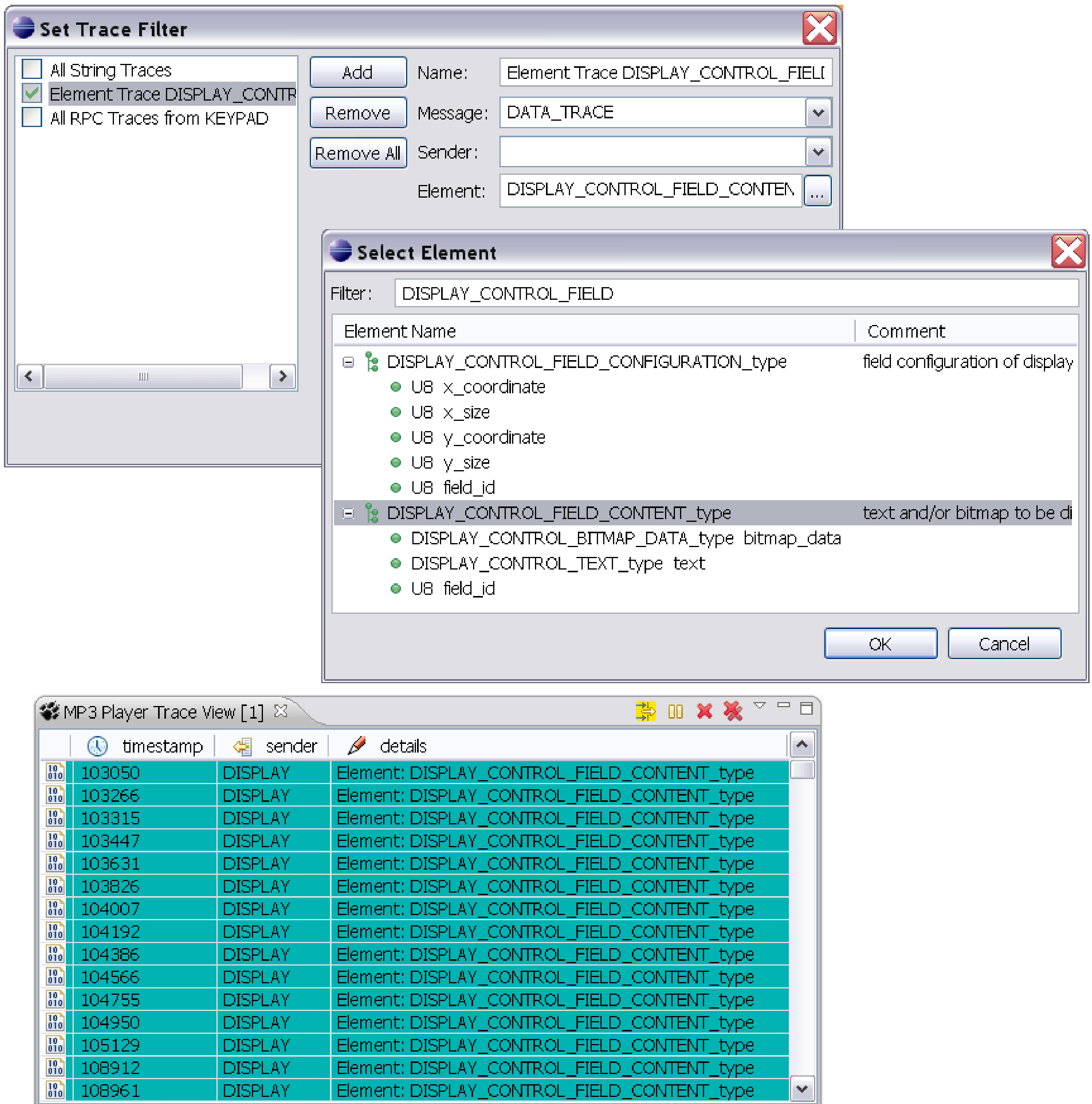

Figure 5 Trace Filter Dialog

Figure 5 shows a more specific filter setting as an example. In this case, only binary traces for a specific data type element are filtered and displayed by the view. The filter is built by a combination of the following attributes:

- A binary data trace
- The sender of the trace could be unspecified
- The particular data type element to be filtered is `DISPLAY_CONTROL_FIELD_CONTENT_type`

For the selection of a particular data type element, another dialog window is available, which lists all data type elements that are provided by the trace description files. These have been assigned to the connection at the time it was set up, as already mentioned above.
The Trace View in the back of Figure 5 shows the outcome of the applied filter and a set of traces for `DISPLAY_CONTROL_FIELD_CONTENT_type`. Filters can be activated and deactivated by using the checkbox in front of the filter name. Active filters for a Trace View are indicated by an orange background color of the trace filter icon in the toolbar of the respective view.

Filters are a powerful feature to focus on a particular trace or group of traces that need to be selected from the data stream for a more detailed inspection. Filters can be switched on and off in a flexible way on a case-by-case basis and are stored together with the connection settings.

Decoding Binary Traces

Binary traces are one type of trace that will be logged by mbLog. There are two kinds of binary traces. One is the RPC service call, that consists of the binary data that has been exchanged when performing the call, namely the service parameters and the return value. The other one is the data type element and all binary data it consists of. Data type elements comprise structures, unions and their sub elements that have been defined by the corresponding interface description documents (MBID). More details about interface description documents can be found in another publication of this series of white papers [embenatics Interface Description].

The Trace Viewer is dedicated to display the basic trace information, namely the timestamp and sender; and, in case of a service call the receiver and the name of the service or type. In order to decode and visualize the payload data of a binary trace, another viewer is available, the Decoder View.
One or more Decoder Views can be started in a similar fashion as already described for Trace Views, by using the context menu of the respective connection display. See figure 2 for more details.
A Decoder View is bound to the corresponding connection and only binary traces for that connection can be processed. To decode a binary trace, one simply drags and drops the trace in the Decoder View window area. The binary trace is marked by a headline similar to the one shown by the Trace Viewer and below that is the decoded data represented by a tree structure. Figure 6 illustrates an example Decoder View with two decoded traces that were introduced in the previous chapter while explaining the details of the Trace View. It shows an RPC service call named `STORAGE_INFO_get_cover()` and an element data type trace named `DISPLAY_CONTROL_FIELD_CONTENT_type`.

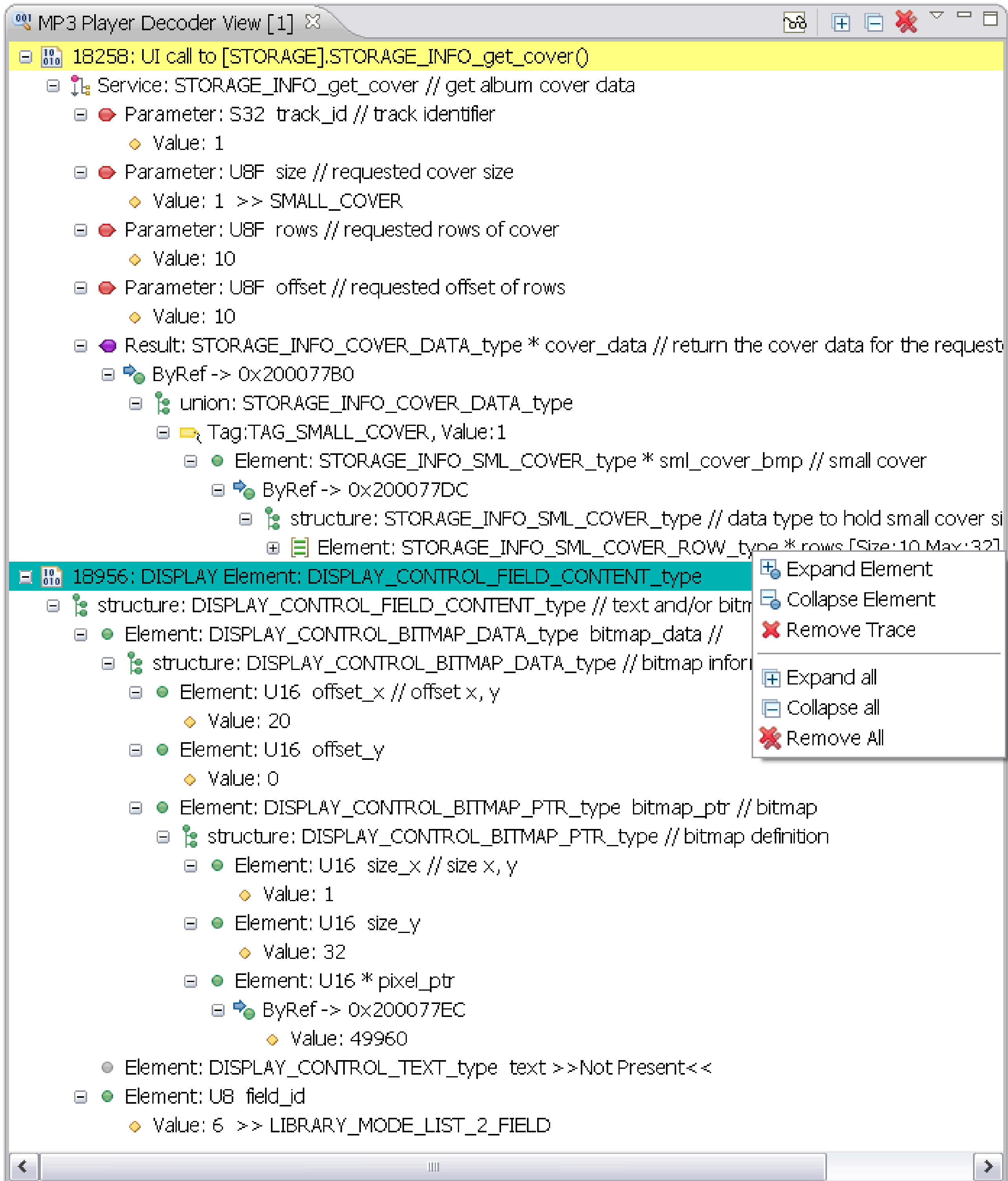

Figure 6 Decoder View

To customize and further inspect the decoded trace, a context menu is provided that allows to expand and collapse parts of the trace as well as to remove traces that are no longer needed.
A Decoder View is a powerful feature to verify data content that has been exchanged via inter-process communication or has been explicitly traced by instrumenting the code at specific locations for diagnostic purposes. All data type elements, as well as the service parameters, are displayed as

they have been defined in the interface description documents (MBID), including the comments and corresponding mappings if applied. The more information the user has put into the task of interface definition, the more benefits he/she has when interpreting the data using the Decoder View.

The Workbench

All of the mbLog views mentioned above are organized by a workbench. In addition to a preconfigured perspective, the user can freely rearrange the position of the views as preferred. Windows and areas can be resized and moved to achieve the optimal layout for inspecting the system internals that have been logged by the tool.
Below is a sample screenshot showing the different views in action.

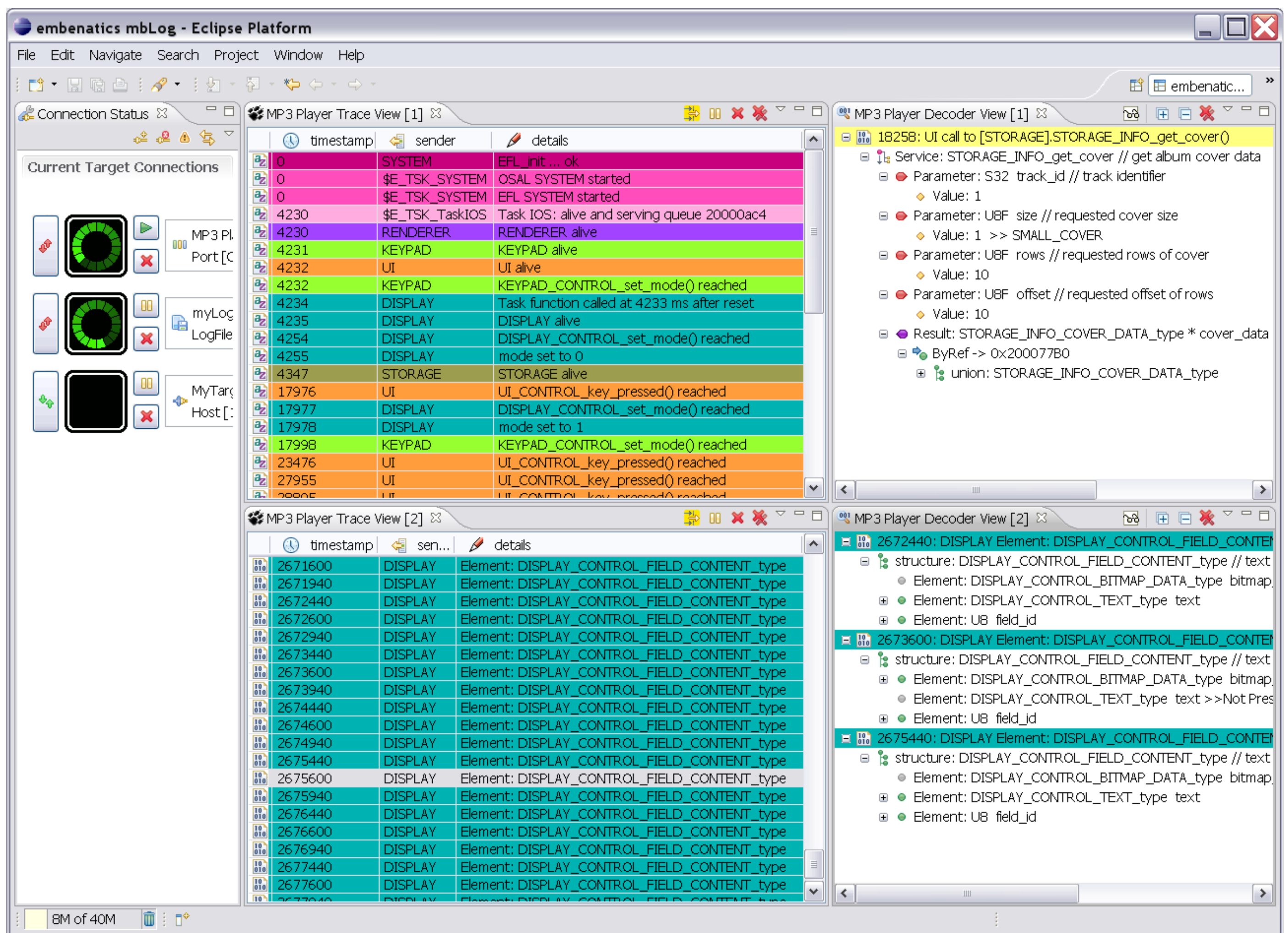

Figure 7 mbLog Workbench

Conclusion

mbLog is the logging and diagnostic tool of the embenatics tool chain. It is very useful when inspecting the activities and events of the running system based on the embenatics foundations layer mbLay. Inter-process communication services, data types and definitions declared in interface description documents during the design phase can be displayed and visualized for diagnostic and test purposes when bringing the embedded system to life. All events coming from the target system are logged and can be stored for an on-the-fly inspection or post analysis activities.

About Us

embenatics is a new company that entered the market in 2010. Our focus is on embedded software development; as such we offer a software foundation layer and tool suite that supports your development team in designing embedded software in an efficient, portable and maintainable way. Based on our wide and varied experience in embedded systems design and development, we know that future product requirements are hard to predict. Our goal is, therefore, to provide you with our technology to make the design of your products as flexible and adaptable as possible. Our approach allows your company to concentrate on the core competencies that differentiate your valuable product from those of your competitors.

Before embenatics was founded, we worked with well-known international companies over two decades and gained valuable experience in the embedded software business. While working as software developers and architects, we encountered the various challenges of the embedded software development life cycle. This wide range of experiences is the backbone of the software foundation products that are offered by embenatics.

Our business philosophy is to establish a close and trustful relationship with our customers in order to successfully promote and support projects over a long time period. For further information please contact

Joachim Pilz
Beerenstraße 29
14163 Berlin

info@embenatics.com
www.embenatics.com

Phone +49 30 26 34 75 28
Mobile +49 176 96 98 46 07

embenatics
foundation layer software