

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.

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.

lew Target (Connection		- 🗆 🔀			
Connection D	efinitions					
Connection name	MyTarget via Socket		~			
Connection type	Socket Port		~			
MBTD file(s)	C:\trcdsc.mbtd;C:\sys_trcdsc.	mbtd	Browse			
	immediately connect to targ					
	assign a trace view with the r	new connection				
		🔷 New Target	Connection	n		
		Connection D Declaration of Sc				•
		Connection host	164.125.4.0			
?	< Back Next >	Connection port	20000			
L		3	< Back	Next >	Finish	Cancel

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

Image: Service of the service of th	🏶 MP3 Player Trace View [1] 🛛 🔅 🤹 😤 😤				
If: S8726 UI call to [RENDERER] RENDERER_ CONTROL_get_playlist() III: S8733 RENDERER call to [STORAGE],STORAGE_DATA_open() III: S8734 RENDERER call to [STORAGE],STORAGE_DATA_open() III: S8735 UI call to [RENDERER],RENDERER_CONTROL_start() III: S8736 DISPLAY DISPLAY_CONTROL_set_mode() III: call to [DISPLAY]_DISPLAY_CONTROL_set_mode() S8757 III: call to [DISPLAY]_CONTROL_set_mode() S8757 III: call to [REVPAD_CONTROL_set_mode() S8757 III: call to [DISPLAY_CONTROL_set_mode() S8758 DISPLAY Element: DISPLAY_CONTROL_set_mode() S8757 III: call to [RENDERER], RENDERER_CONTROL_set_content() S8768 III: call to [STORAGE], STORAGE_INFO_get_cover() S8816 III: call to [STORAGE], STORAGE_INFO_get_cover() S8859 III: call to [STORAGE], STORAGE_INFO_get_cover() S8859 III: call to [STORAGE], STORAGE_INFO_get_cover() S8866 S8865 DISPLAY Element: DISPLAY_CONTROL_set_content() III: S8865 DISPLAY<		🕔 timestamp	🚑 sender	🤌 details	^
If: 58733 RENDERER cal to [STORAGE]STORAGE_DATA_get_next_block() If: 58734 RENDERER cal to [STORAGE]STORAGE_DATA_get_next_block() If: 58736 DISPLAY DISPLAY_CONTROL_set_mode() reached If: 58736 DISPLAY DISPLAY_CONTROL_set_mode() reached If: 58756 UI cal to [DISPLAY_CONTROL_set_mode() reached If: 58757 VI cal to [REVPAD_CONTROL_set_mode() reached If: 58758 DISPLAY Element: DISPLAY_CONTROL_set_mode() If: 58758 DISPLAY Element: DISPLAY_CONTROL_set_mode() If: 58788 UI cal to [TSTORAGE]STORAGE_INFO_get_cover() If: 58788 UI cal to [STORAGE]STORAGE_INFO_get_cover() If: 58816 UI cal to [DISPLAY_CONTROL_set_content() If: 58829 UI cal to [DISPLAY_CONTROL_set_content() If: 58859 UI cal to [DISPLAY_CONTROL_set_content() If: 58866 DISPLAY Element: DISPLAY_CONTROL_set_content() If: 58868 UI cal to [STORAGE]STORAGE_INFO_get_cover() If	az	58725	UI	UI_CONTROL_key_pressed() reached	
Fr. 58734 RENDERER call to [STORAGE].STORAGE_DATA_get_next_block() 15 58735 UL call to [RENDERER].RENDERER_CONTROL_set_t() 25 58736 DISPLAY DISPLAY_CONTROL_set_mode() reached 26 58756 UL call to [DISPLAY_CONTROL_set_mode() 26 58757 UL call to [DISPLAY_CONTROL_set_mode() 26 58757 UL call to [DISPLAY_CONTROL_set_mode() 26 58758 DISPLAY Element: DISPLAY_CONTROL_set_mode() 27 88758 DISPLAY Element: DISPLAY_CONTROL_set_mode() 28 58758 DISPLAY Element: DISPLAY_CONTROL_set_mode() 28 58783 UL call to [DISPLAY_CONTROL_set_content() 28 58783 UL call to [STORAGE].STORAGE_INFO_get_cover() 28 58879 UL call to [DISPLAY_CONTROL_FILE_CONTENT_type 28 58866 <td>10 010</td> <td>58726</td> <td>UI</td> <td>call to [RENDERER].RENDERER_CONTROL_get_playlist()</td> <td></td>	10 010	58726	UI	call to [RENDERER].RENDERER_CONTROL_get_playlist()	
Her 58734 RENDERER cal to [STORAGE].STORAGE_DATA_get_next_block() 11 Sar35 UI cal to [STORAGE].STORAGE_DATA_get_next_block() 12 Sar35 DISPLAY CONTROL_set_mode() reached 13 Sar56 DISPLAY CONTROL_set_mode() 14 Sar57 UI cal to [DISPLAY_CONTROL_set_mode() 15 Sar57 UI cal to [DISPLAY_CONTROL_set_mode() 16 Sar58 DISPLAY Element: DISPLAY_CONTROL_set_mode() 16 Sar57 UI cal to [DISPLAY_CONTROL_set_mode() 16 Sar58 DISPLAY Element: DISPLAY_CONTROL_set_mode() 17 Sar58 DISPLAY Element: DISPLAY_CONTROL_set_content() 16 Sar788 UI cal to [STORAGE].STORAGE_INFO_get_cover() 17 Sar799 UI cal to [DISPLAY_DISPLAY_CONTROL_set_content() 18 Sar74 DISPLAY Element: DISPLAY_CONTROL_set_content() 18 Sar799 UI cal to [DISPLAY_DISPLAY_CONTROL_set_content() 18 Sar799 UI cal to [DISPLAY_CONTROL_set_content() 19 Sar799 UI	10 010	58733	RENDERER	call to [STORAGE].STORAGE_DATA_open()	
92 58736 DISPLAY DISPLAY CONTROL_set_mode() reached 92 58736 DISPLAY mode set to 0 93 58756 UI call to [DISPLAY].DISPLAY_CONTROL_set_mode() 95 58757 UI call to [ISPLAY].DISPLAY_CONTROL_set_mode() 95 58757 UI call to [ISPLAY].DISPLAY_CONTROL_set_mode() 95 58758 DISPLAY Element: DISPLAY_CONTROL_set_content() 95 58758 UI call to [ISPLAY].DISPLAY_CONTROL_set_content() 95 58788 UI call to [STORAGE].STORAGE_INFO_get_content() 95 58799 UI call to [STORAGE].STORAGE_INFO_get_cover() 95 58824 DISPLAY Element: DISPLAY_CONTROL_set_content() 95 58859 UI call to [STORAGE].STORAGE_INFO_get_cover() 95 58859 UI call to [STORAGE].STORAGE_INFO_get_cover() 95 58968 UI call to [STORAGE].STORAGE_INFO_get_cover() 95 5897 UI call to [DISPLAY].CONTROL_FIELD_CONTENT_type 95 59033 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type <t< td=""><td></td><td>58734</td><td>RENDERER</td><td></td><td></td></t<>		58734	RENDERER		
92 58736 DISPLAY DISPLAY CONTROL_set_mode() reached 92 58736 DISPLAY mode set to 0 93 58756 UI call to [DISPLAY]_DISPLAY_CONTROL_set_mode() 94 58757 KEYPAD KEYPAD_CONTROL_set_mode() 95 58757 UI call to [DISPLAY_CONTROL_set_mode() 95 58758 DISPLAY Element: DISPLAY_CONTROL_set_content() 95 58788 UI call to [DISPLAY_CONTROL_set_content() 95 58788 UI call to [STORAGE].STORAGE_INFO_get_cover() 95 58816 UI call to [STORAGE].STORAGE_INFO_get_cover() 95 58824 DISPLAY Element: DISPLAY_CONTROL_set_content() 95 58829 UI call to [STORAGE].STORAGE_INFO_get_cover() 95 58859 UI call to [STORAGE].STORAGE_INFO_get_cover() 95 58866 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type 95 58917 UI call to [STORAGE].STORAGE_INFO_get_cover() 95 58968 UI call to [STORAGE].STORAGE_INFO_get_cover() 95 58968	10 010	58735	UI	call to [RENDERER].RENDERER_CONTROL_start()	
If: 58756 UI call to [DISPLAY].DISPLAY_CONTROL_set_mode() If: 58757 KEYPAD KEYPAD_CONTROL_set_mode() If: 58757 UI call to [KEYPAD].CONTROL_set_mode() If: 58758 DISPLAY Element: DISPLAY_CONTROL_set_content() If: 58788 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() If: 58788 UI call to [STORAGE].STORAGE_INFO.get_track_info() If: 58786 UI call to [STORAGE].STORAGE_INFO.get_cover() If: 58816 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() If: 58824 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type If: 58829 UI call to [STORAGE].STORAGE_INFO.get_cover() If: 58859 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() If: 58866 DISPLAY Element: DISPLAY_CONTROL_set_content() If: 58968 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() If: 59033 DISPLAY Element: DISPLAY_CONTROL_set_content() If: 59147 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() <td>æ</td> <td>58736</td> <td>DISPLAY</td> <td></td> <td></td>	æ	58736	DISPLAY		
If: 58756 UI call to [DISPLAY].DISPLAY_CONTROL_set_mode() If: 58757 KEYPAD KEYPAD_CONTROL_set_mode() If: 58757 UI call to [KEYPAD].CONTROL_set_mode() If: 58758 DISPLAY Element: DISPLAY_CONTROL_set_content() If: 58788 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() If: 58788 UI call to [STORAGE].STORAGE_INFO.get_track_info() If: 58786 UI call to [STORAGE].STORAGE_INFO.get_cover() If: 58816 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() If: 58824 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type If: 58829 UI call to [STORAGE].STORAGE_INFO.get_cover() If: 58859 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() If: 58866 DISPLAY Element: DISPLAY_CONTROL_set_content() If: 58968 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() If: 59033 DISPLAY Element: DISPLAY_CONTROL_set_content() If: 59147 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() <td>æ</td> <td>58736</td> <td>DISPLAY</td> <td>mode set to 0</td> <td></td>	æ	58736	DISPLAY	mode set to 0	
92 58757 KEYPAD KEYPAD_CONTROL_set_mode() reached 11 58757 UI call to [KEYPAD_CONTROL_set_mode() 12 58758 DISPLAY Element: DISPLAY_CONTROL_Set_mode() 12 58758 DISPLAY Element: DISPLAY_CONTROL_Set_content() 12 58788 UI call to [DISPLAY]_DISPLAY_CONTROL_get_content() 13 58799 UI call to [STORAGE]_STORAGE_INFO_get_cover() 14 58816 UI call to [STORAGE]_STORAGE_INFO_get_cover() 15 58824 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type 16 58859 UI call to [DISPLAY]_DISPLAY_CONTROL_set_content() 16 58866 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type 16 58859 UI call to [DISPLAY]_DISPLAY_CONTROL_set_content() 16 58866 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type 16 59033 DISPLAY Element: DISPLAY_CONTROL_set_content() 16 59147 UI call to [DISPLAY]_DISPLAY_CONTROL_set_content() 16 59246 DISPLAY Element: DISPLAY_CONTROL_set_content() <td>10 010</td> <td>58756</td> <td>UI</td> <td>call to [DISPLAY].DISPLAY_CONTROL_set_mode()</td> <td></td>	10 010	58756	UI	call to [DISPLAY].DISPLAY_CONTROL_set_mode()	
S8759 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type S8783 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() S8783 UI call to [RENDERER].RENDERER_CONTROL_get_status() S8789 UI call to [STORAGE].STORAGE_INFO_get_track_info() S8789 UI call to [STORAGE].STORAGE_INFO_get_cover() S8824 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type S8859 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() S8859 UI call to [STORAGE].STORAGE_INFO_get_cover() S8859 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() S9042 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() S9042 UI call to [DISPLAY].DISPLAY_CONTROL_set_content()		58757	KEYPAD		
S8759 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type S8783 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() S8783 UI call to [RENDERER].RENDERER_CONTROL_get_status() S8789 UI call to [STORAGE].STORAGE_INFO_get_track_info() S8789 UI call to [STORAGE].STORAGE_INFO_get_cover() S8824 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type S8859 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() S8859 UI call to [STORAGE].STORAGE_INFO_get_cover() S8859 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() S9042 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() S9042 UI call to [DISPLAY].DISPLAY_CONTROL_set_content()	10	58757	UI		
Image: Series of the series			DISPLAY		
Image: Series of the series					
Image: Start Star					
Image: State in the intervent of the interv			UI		
Image: Set in the set of					
Image: state in the interval of the image: state interval of image: state in					
Image: Series of the series					
Bit 58866 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type Bit 58871 UI call to [DISPLAY].DISPLAY_CONTROL_set_content() Bit 58968 UI call to [STORAGE].STORAGE_INFO_get_cover() Bit 59033 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type Bit 59042 UI call to [DISPLAY].DISPLAY_CONTROL_Set_content() Bit 59147 UI call to [STORAGE].STORAGE_INFO_get_cover() Bit 59216 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type Bit 59242 UI call to [STORAGE].STORAGE_INFO_get_cover() Bit 59244 UI call to [DISPLAY_CONTROL_FIELD_CONTENT_type Bit 59245 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type Bit 59462 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type Bit 59469 UI call to [STORAGE].STORAGE_INFO_get_cover() Bit 59577 UI call to [DISPLAY].DISPLAY_CONTROL_FIELD_CONTENT_type Bit 59648 DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type Bit 59777 UI call to [S					
Image: State in the image: Sta					
Image: Sepsitive Sep					
Image: Second					
Image: Section of the sectin of the section of the section of the					
Image: Section of the section of th					
Image: System in the system					
Image: SpectrumUIcall to [DISPLAY].DISPLAY_CONTROL_set_content()Image: SpectrumSpectrumCall to [STORAGE].STORAGE_INFO_get_cover()Image: SpectrumDISPLAYElement: DISPLAY_CONTROL_FIELD_CONTENT_typeImage: SpectrumSpectrumCall to [DISPLAY].DISPLAY_CONTROL_set_content()Image: SpectrumSpectrumCall to [STORAGE].STORAGE_INFO_get_cover()Image: SpectrumCall to [STORAGE].STORAGE_INFO_get_cover()Image: SpectrumCall to [DISPLAY].DISPLAY_CONTROL_FIELD_CONTENT_typeImage: SpectrumCall to [DISPLAY].DISPLAY_CONTROL_Set_content()Image: SpectrumCall to [STORAGE].STORAGE_DATA_get_next_block()Image: SpectrumCall to [STORAGE].STORAGE_INFO_get_track_info()Image: SpectrumCall to [STORAGE].STORAGE_INFO_get_track_info()Image: SpectrumCall to [DISPLAY].DISPLAY_CONTROL_FIELD_CONTENT_typeImage: SpectrumCall to [DISPLAY].DISPLAY_CONTROL_FIELD_CONTENT_typeImage: SpectrumCall to [DISPLAY].DISPLAY_CONTROL_Set_content()Image: SpectrumCall to [DISPLAY].DISPLAY_CONTROL_set_content()Image: SpectrumCall to [DISPLAY].DISPLAY_CONTROL_set_content()Image: SpectrumCall to [RENDERER].RENDERER_CONTROL_register_status_listener()Image: SpectrumCall to [RENDERER].RENDERER_CONTROL_get_current_track()Image: SpectrumCall to [RENDERER].RENDERER_CONTROL_get_current_track()					
Image: Sparse in the sparse					
Image: SystemDISPLAYElement: DISPLAY_CONTROL_FIELD_CONTENT_typeImage: SystemSystemUIcall to [DISPLAY].DISPLAY_CONTROL_set_content()Image: SystemSystemUIcall to [STORAGE].STORAGE_INFO_get_cover()Image: SystemDISPLAYElement: DISPLAY_CONTROL_FIELD_CONTENT_typeImage: SystemDISPLAYElement: DISPLAY_CONTROL_FIELD_CONTENT_typeImage: SystemSystemcall to [DISPLAY].DISPLAY_CONTROL_set_content()Image: SystemSystemcall to [STORAGE].STORAGE_DATA_get_next_block()Image: SystemSystemcall to [STORAGE].STORAGE_INFO_get_cover()Image: SystemUIcall to [STORAGE].STORAGE_INFO_get_track_info()Image: SystemDISPLAYElement: DISPLAY_CONTROL_FIELD_CONTENT_typeImage: SystemDISPLAYElement: DISPLAY_CONTROL_FIELD_CONTENT_typeImage: SystemUIcall to [DISPLAY].DISPLAY_CONTROL_set_content()Image: SystemSystemcall to [RENDERER].RENDERER_CONTROL_register_status_listener()Image: SystemCall to [UI].UI_CONTROL_key_pressed()Image: SystemCall to [RENDERER].RENDERER_CONTROL_get_current_track()					
Image: Space of the space of					
Image: System					
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() 50300 UI call to [RENDERER].RENDERER_CONTROL_get_current_track()					
Image: System System Image: System Call to [DISPLAY].DISPLAY_CONTROL_set_content() Image: System Call to [STORAGE].STORAGE_DATA_get_next_block() Image: System Call to [STORAGE].STORAGE_INFO_get_cover() Image: System Call to [STORAGE].STORAGE_INFO_get_cover() Image: System Call to [STORAGE].STORAGE_INFO_get_track_info() Image: System Call to [DISPLAY].DISPLAY_CONTROL_FIELD_CONTENT_type Image: System Call to [DISPLAY].DISPLAY_CONTROL_set_content() Image: System Call to [RENDERER].RENDERER_CONTROL_register_status_listener() Image: System Call to [UI].UI_CONTROL_key_pressed() Image: System Call to [RENDERER].RENDERER_CONTROL_get_current_track()					
Image: Synthetic status Renderer call to [STORAGE].STORAGE_DATA_get_next_block() Image: Synthetic status Synthetic status					
Image: Spinormal system UI call to [STORAGE].STORAGE_INFO_get_cover() Image: Spinormal system Call to [STORAGE].STORAGE_INFO_get_track_info() Image: Spinormal system Call to [DISPLAY]_CONTROL_FIELD_CONTENT_type Image: Spinormal system Call to [DISPLAY].DISPLAY_CONTROL_set_content() Image: Spinormal system Call to [RENDERER].RENDERER_CONTROL_register_status_listener() Image: Spinormal system Call to [UI].UI_CONTROL_key_pressed() Image: Spinormal system Call to [RENDERER].RENDERER_CONTROL_get_current_track()					
Image: System					
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()					
Image: System System System Call to [DISPLAY].DISPLAY_CONTROL_set_content() System System System Call to [RENDERER].RENDERER_CONTROL_register_status_listener() System System System Call to [UI].UI_CONTROL_key_pressed() Solo UI Call to [RENDERER].RENDERER_CONTROL_get_current_track()					
Image: System System Call to [RENDERER].RENDERER_CONTROL_register_status_listener() Image: System System Call to [UI].UI_CONTROL_key_pressed() Image: System Call to [RENDERER].RENDERER_CONTROL_get_current_track()					
Image: System KEYPAD call to [UI].UI_CONTROL_key_pressed() Image: System 60300 UI call to [RENDERER].RENDERER_CONTROL_get_current_track()					
B 60300 UI call to [RENDERER].RENDERER_CONTROL_get_current_track()					
	10	60310	UI	call to [STORAGE].STORAGE_INFO_get_track_info()	
					~

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.

😂 Set Color Mapping						
My Example Mapping	Add	Name:	My Example M	1apping		
SYSTEM E_TSK_SYSTEM	Remo	ve Message:	RPC_TRACE		~	
E_TSK_TaskIOS RENDERER	Remov	e All Sender:	UI		~	
KEYPAD UI		Receiver:	STORAGE		~	
DISPLAY		Service:	STORAGE_INF	=0_get_cover		
STORAGE						
		Color:				
	Se Se	ect Service				
MP3 Player Trace View	Filter:	STORAGE_INF	=0			
	Servi	ce Name		Comment		
Image: Second s	L L ST	ORAGE_INFO_ge	et_cover	get album cover data		
10291 DI	Į₽SI	ORAGE_INFO_ge		get music library inform	nation	
10313 UI		ORAGE_INFO_ge		get titles for a requeste		
10322 DI		ORAGE_INFO_ge		get the tracks for a rec	quested type	
10322 DI	- TI- ST	ORAGE_INFO_ge	et_track_info	get track information		
2 10347 ST						
🔁 658426 UI						
10 658427 UI						
🔁 658428 DIS	SPL				ок	Cancel
18 658448 UI						
🔁 658448 KE	YPAD	KETPAD_CONTI	ROE_set_mode	:Ureacheu		
🚻 658449 UI		call to [KEYPAD]].KEYPAD_CON	ITROL_set_mode()		
🔛 658450 DIS	SPLAY	Element: DISPL	AY_CONTROL_	FIELD_CONTENT_type		
🚻 658457 UI				NTROL_set_content()		
10 658635 UI		call to [STORAG				
🔛 658643 UI				NFO_get_track()		
10 658658 UI				NFO_get_cover()		
	SPLAY			FIELD_CONTENT_type		
658679 UI				NTROL_set_content()		
658707 UI				NFO_get_cover()		
	SPLAY			FIELD_CONTENT_type		
658719 UI				NTROL_set_content()		
658770 UI				NFO_get_cover()		
	SPLAY			FIELD_CONTENT_type		
8 658856 UI				NTROL_set_content()		
658922 UI				NFO_get_cover()		
658944 UI		call to [STORAG	EJ.STORAGE_I	NFO_get_cover()		¥



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.



🔷 Set	Trace Filter				
	String Traces		Add Name:	Element Trace DISPLAY_CONTROL_FIE	
	RPC Traces from K		Remove Messaç	e: DATA_TRACE	v
		ſ	Remove All Sender	:	~ I
			Elemer		
			😂 Select Elem	ent	
			Filter: DISPLAY	_CONTROL_FIELD	
			Element Name		Comment
<	Ш	>	B B CONTRACT CONTRACT CONTRACT CONTRACT CONTRACT CONTRACT CO	ze oordinate ze	field configuration of display
			DISPLA	:ONTROL_FIELD_CONTENT_type (_CONTROL_BITMAP_DATA_type bitmap_c (_CONTROL_TEXT_type text _jd	text and/or bitmap to be di lata
					OK Cancel
** N	MP3 Player Trace Vi	ew [1] 🕱		📰 uu 🗙 💥 🏹 🗖 🗖	OK Cancel
*			details	📮 🛛 🗶 ⊽ 🗖 🗖	OK Cancel
		🔄 sender			OK Cancel
	🕔 timestamp		Element: DISPLA	^	OK Cancel
190	() timestamp 103050	sender	Element: DISPLA Element: DISPLA		OK Cancel
	() timestamp 103050 103266	Sender DISPLAY DISPLAY	Element: DISPLA Element: DISPLA Element: DISPLA		OK Cancel
	() timestamp 103050 103266 103315	Sender DISPLAY DISPLAY DISPLAY	Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA	CONTROL_FIELD_CONTENT_type	OK Cancel
10 010 10 010 10 010	() timestamp 103050 103266 103315 103447	Sender DISPLAY DISPLAY DISPLAY DISPLAY	Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA		OK Cancel
	() timestamp 103050 103266 103315 103447 103631	Sender DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY	Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA	CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type	OK Cancel
	() timestamp 103050 103266 103315 103447 103631 103826	Sender DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY	Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA	CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type	OK Cancel
	() timestamp 103050 103266 103315 103447 103631 103826 104007 104192	Sender DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY	Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA	CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type	OK Cancel
	() timestamp 103050 103266 103315 103447 103631 103826 104007 104192 104386	Sender DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY	Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA	CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type	OK Cancel
	() timestamp 103050 103266 103315 103447 103631 103826 104007 104192 104386 104566	Sender DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY	Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA	CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type CONTROL_FIELD_CONTENT_type	OK Cancel
	() timestamp 103050 103266 103315 103447 103631 103826 104007 104192 104386 104566	Sender DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY	Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA	CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type	OK Cancel
	() timestamp 103050 103266 103315 103447 103631 103826 104007 104192 104386 104566 104755 104950	Sender DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY	Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA Element: DISPLA	CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_C	OK Cancel
	() timestamp 103050 103266 103315 103447 103631 103826 104007 104192 104386 104566 104755 104950 105129	Sender DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY DISPLAY	Element: DISPLA Element: DISPLA	CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_CONTENT_type (_CONTROL_FIELD_C	OK Cancel

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 called STORAGE_INFO_get_cover() and an element data type trace named DISPLAY_CONTROL_FIELD_CONTENT_type.



(*************************************	😽 🕀 🖻 💥 V 🗖 🗖
■ Im 18258: UI call to [STORAGE].STORAGE_INFO_get_cover()	
Image: Service: STORAGE_INFO_get_cover // get album cover data	
Parameter: S32 track_id // track identifier	
♦ Value: 1	
🗉 👄 Parameter: UBF-size // requested cover size	
♦ Value: 1 >> SMALL_COVER	
Parameter: UBF rows // requested rows of cover	
♦ Value: 10	
🗉 👄 Parameter: UBF offset // requested offset of rows	
♦ Value: 10	
Result: STORAGE_INFO_COVER_DATA_type * cover_data // return th	e cover data for the request
🖃 💊 ByRef -> 0x200077B0	
😑 🍃 union: STORAGE_INFO_COVER_DATA_type	
🗉 🗪 Tag:TAG_SMALL_COVER , Value: 1	
Element: STORAGE_INFO_SML_COVER_type * sml_cover	_bmp // small cover
ByRef -> 0×200077DC	
structure: STORAGE_INFO_SML_COVER_type // da	
	roe <u>* rows [Size+10 Max+32]</u> Expand Element
E III 18956: DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type	
structure: DISPLAY_CONTROL_FIELD_CONTENT_type // text and/or bitm structure: DISPLAY_CONTROL_FIELD_CONTENT_type // text and/or bitm structure: DISPLAY_CONTROL_FIELD_CONTENT_type // text and/or bitm	X Remove Trace
Element: DISPLAY_CONTROL_BITMAP_DATA_type bitmap_data // structure: DISPLAY_CONTROL_BITMAP_DATA_type // bitmap information	
 Element: U16 offset_x // offset x, y 	🕀 Expand all
♦ Value: 20	📄 Collapse all
 Value: 20 ■ ● Element: U16 offset_y 	💥 Remove All
♦ Value: 0	
Element: DISPLAY_CONTROL_BITMAP_PTR_type bitmap_ptr //	bitmap
structure: DISPLAY_CONTROL_BITMAP_PTR_type // bitmap	
■ ● Element: U16 size_x // size x, y	
♦ Value: 1	
🗉 🔍 Element: U16 size_y	
♦ Value: 32	
🗉 💿 Element: U16 * pixel_ptr	
🖃 💊 ByRef -> 0x200077EC	
♦ Value: 49960	
Element: DISPLAY_CONTROL_TEXT_type text >>Not Present<<	
🗉 🔍 Element: U8 field_id	
Value: 6 >> LIBRARY_MODE_LIST_2_FIELD	
٠	>

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 interprocess 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.

🥏 embenatics mbLog - Eclipse Platform File Edit Navigate Search Project Window Help 📸 • 📄 🕞 🖉 • 🗄 🖢 • 🖗 • 📑 🖪 embenatic... 🚴 Connection Status 🛛 🦳 🗖 📽 MP3 Player Trace View [1] 🖄 🚼 🕕 🗶 😤 🖱 🗖 🔍 MP3 Player Decoder View [1] 🛛 🔪 18 E E 💥 B 18258: UI call to [STORAGE].STORAGE_INFO_get_cover() 🚕 🦀 🕭 🚖 🏱 🕔 timestamp 🔄 sender 🥒 details ^ 🗉 🛼 Service: STORAGE_INFO_get_cover // get album cover data Current Target Connections 😑 👄 Parameter: S32 track_id // track identifier \$E_TSK_SYSTEM OSAL SYSTEM started **ව**ැ 0 වැ 4 Value: 1 SYSTEM FEL SYSTEM started \$E TSK TaskIOS Task IOS: alive and serving queue 20000ac4 ■ ● Parameter: U8F size // requested cover size 4230 MP3 Pl 32 32 RENDERER Value: 1 >> SMALL_COVER 100 4231 KEYPAD **KEYPAD** alive Parameter: UBF rows // requested rows of cover Port [C Value: 10 KEYPAD_CONTROL_set_mode() reached 4232 **KEYPAD** ■ ● Parameter: U8F offset // requested offset of rows Task function call ed at 4233 ms after re myLoc Value: 10 P az az LogFile Result: STORAGE_INFO_COVER_DATA_type * cover_data CONTROL_set_mode() reached □ ♣ BvRef -> 0x200077B0 🗉 🍖 union: STORAGE INFO COVER DATA type STORAGE STORAGE alive 2 4347
 3 1797 MyTarç ↓ ↓ Host [: PAD_CONTROL_set_mode() reach ₽2 ₽2 UI_CONTROL_key_pressed() reached 23476 UI 27955 UI_CONTROL_key_pressed() reached < > 🛼 💷 🗶 🏹 🗖 🗖 🤍 MP3 Player Decoder View [2] 🛛 7-7-8 🕀 🕀 🗮 📽 MP3 Player Trace View [2] 🛛 **H** 100 🕔 timestamp 🔄 sen... 🖉 details ~ 🔓 structure: DISPLAY_CONTROL_FIELD_CONTENT_type // text Element: DISPLAY CONTROL BITMAP DATA type bitmap 10 010 10 010 10 010 010 ● Element: DISPLAY_CONTROL_TEXT_type text DISPLA) ⊕ ● Element: U8 field id DISPLA' DISPLAY 🔓 structure: DISPLAY_CONTROL_FIELD_CONTENT_type // text 10 • Element: DISPLAY_CONTROL_BITMAP_DATA_type bitmap DISPLAY EIELD Element: DISPLAY_CONTROL_TEXT_type text >>Not Press 10 🗉 🔍 Element: U8 field_id 10 010 10 010 三 18 26 🗉 🍃 structure: DISPLAY_CONTROL_FIELD_CONTENT_type // text 10 Element: DISPLAY_CONTROL_BITMAP_DATA_type bitmap DISPLAY Element: DISPLAY_CONTROL_FIELD_CONTENT_type 2675600 10 DISPLAY 🗉 🔍 Element: U8 field id 10 DISPLA 10 DISPLAY PLAY CONTROL DISPLAY ISPLAY CONTR < > 8M of 40M 🗊 i 🖬

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

