Automated Testing of a Kanzi-based HMI with TTA

MAGNA Telemotive GmbH and Rightware Oy

By Manuel Zimmermann, Senior Software Engineer HMI, Magna Telemotive,
Rebekka Haisch, Product Manager Software Solutions, Magna Telemotive,
and Derek Sellin, Vice President, Marketing, Rightware

Abstract

Rightware and Magna Telemotive cooperate in the development and validation of automotive human machine interfaces (HMI), especially for instrument clusters. Magna Telemotive has developed an innovative HMI framework and integrated it into the Kanzi tool chain. Before releasing the HMI software, it is validated with the test automation framework Telemotive Test Automation (TTA). The process from developing the HMI framework to connecting it to Kanzi and finally validating the graphical output is described in this whitepaper.

Keywords: HMI development, Kanzi, HMI framework, test automation, Telemotive Test Automation (TTA)

Contents

Introduction

In-vehicle display systems - particularly the instrument cluster - are increasingly becoming a service partner for the driver. MAGNA Telemotive develops and improves such display systems. Since 2012, many original equipment manufacturers (OEM) and component suppliers have used Rightware's Kanzi user interface (UI) software for the development of human machine interface (HMI) components. Magna Telemotive has identified an opportunity to complement the Kanzi offering with its proprietary test automation solution Telemotive Test Automation (TTA) that uses APIs available in Kanzi.

Magna Telemotive's existing proprietary HMI framework has been connected with Kanzi in order to develop a reference automotive HMI. At the same time, the HMI functions are validated with the test automation framework TTA. The development and validation process will be explained in the following.

The development of Magna Telemotive's reference HMI

To be able to develop various HMIs for various OEMs, Magna Telemotive has developed its own HMI framework with a code generator. This HMI has been developed as an innovative reference HMI in which many common HMI features are presented in a Magna Telemotive-specific design. One big advantage of the framework is that it can be connected to different graphic engines. It manages the module and signal structure via configuration files, which guarantees a good overview and maintainability.

signalname	type	min	max	step	default	invalid	description
VehicleStateIntern	uint8	0	3		0	4	0: off 1: welcome 2: driving 3: goodbye
IsElectro	bool	false	true		false	false	Electro (true) or Combustion (false) engine
GearId	uint8	0	4		0	4	0: P (park), 1: R (reverse), 2: N (neutral); 3: D (drive); 4: invalid/unknown
SpeedKmh	uint16	0	2400		0	0	speed in km/h, must be devided by 10
PowerCombustion	uint16	0	6000		0	0	number of U/min
PowerElectro	sint16	-1000	1000		0	0	positiv is power up to 100% (1000), negative is charge up to 100% (-1000)
ChargingLevel	uint16	0	1000		0	1000	1000 is 100%, fuel for combustion or battery for electro
TelltaleDirectionIndicatorLeft	bool	false	true		false	false	false: off, true: on
TelltaleDirectionIndicatorRigh	bool	false	true		false	false	false: off, true: on
TelltaleLightLongDistance	bool	false	true		false	false	false: off, true: on
TelltaleHeatingWindshield	bool	false	true		false	false	false: off, true: on
TelltaleWarningSeatbelt	bool	false	true		false	false	false: off, true: on
TelltaleWarningPressure	bool	false	true		false	false	false: off, true: on
TelltaleAbs	bool	false	true		false	false	false: off, true: on
TelltaleBatteryCar	bool	false	true		false	false	false: off, true: on
TelltaleHandbrake	bool	false	true		false	false	false: off, true: on
TelltaleCruiseControl	bool	false	true		false	false	false: off, true: on
TimeHour	uint8	0	23		0	0	0h .. 23h
TimeMinutes	uint8	0	59		0	0	0m .. 59m

Figure 1: Screenshot of the configuration file

The code generator verifies if the configuration files are valid and creates basic classes for the modules, which results in a minimal manual programming effort. It also visualizes the signal flow and thus shows which signals enter which modules and which widgets show something in the end. A part of the clear signal flow visualization can be seen in Figure 2.

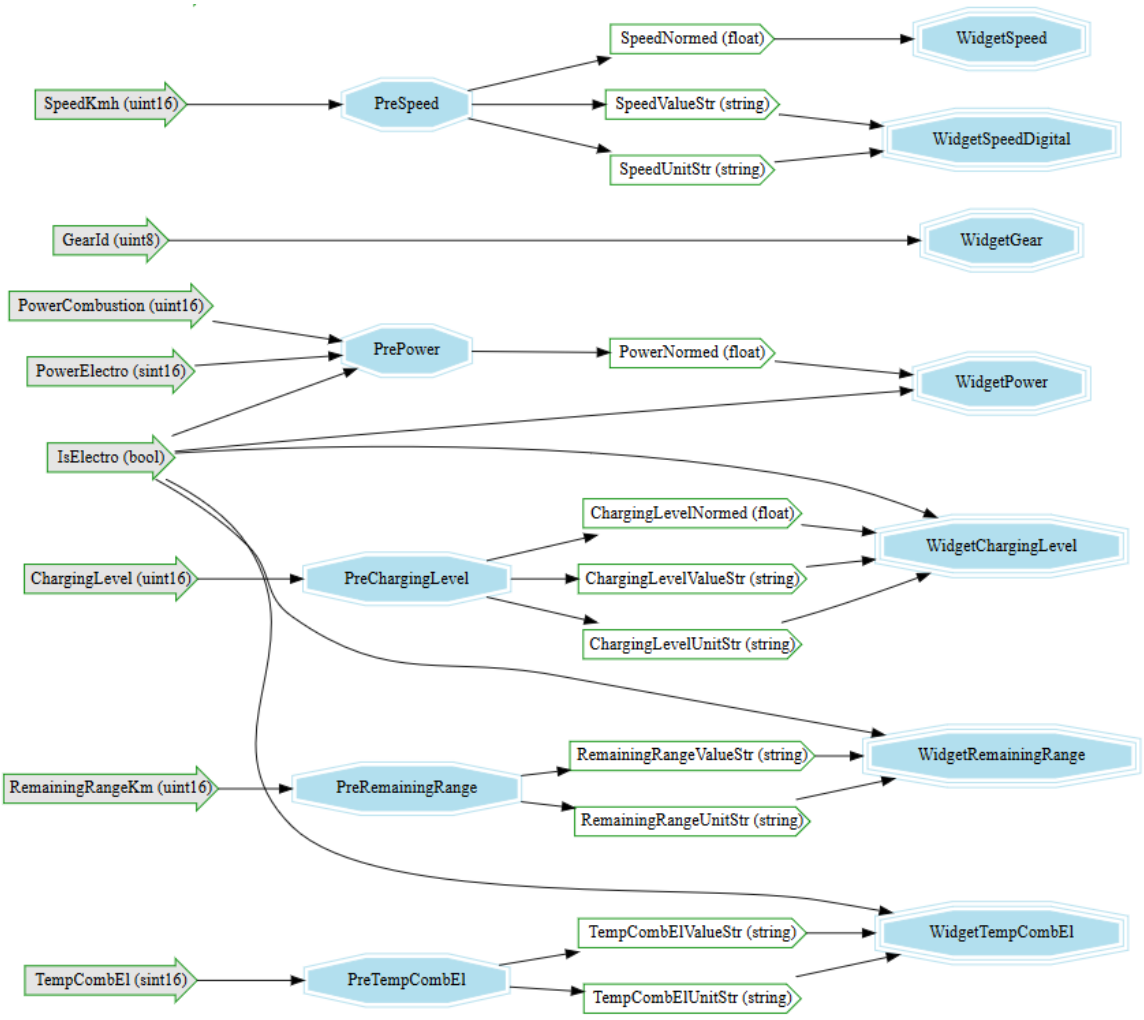


Figure 2: A part of the signal flow visualization

Moreover, code can be generated either for C++ (e.g. for visualization via Kanzi) or for C# (e.g. for visualization via Unity), thus ensuring that the framework is easily adaptable to various graphic engines.

For the input signals, the code generator creates a HTML5 simulation interface, which comprises a big advantage in the prototyping phase. There, the single feature states can be controlled by dragging a slider of different input signals. The information from the input signals is queried by the HMI framework and built up dynamically in HTML. Data storage only happens in the HMI framework.

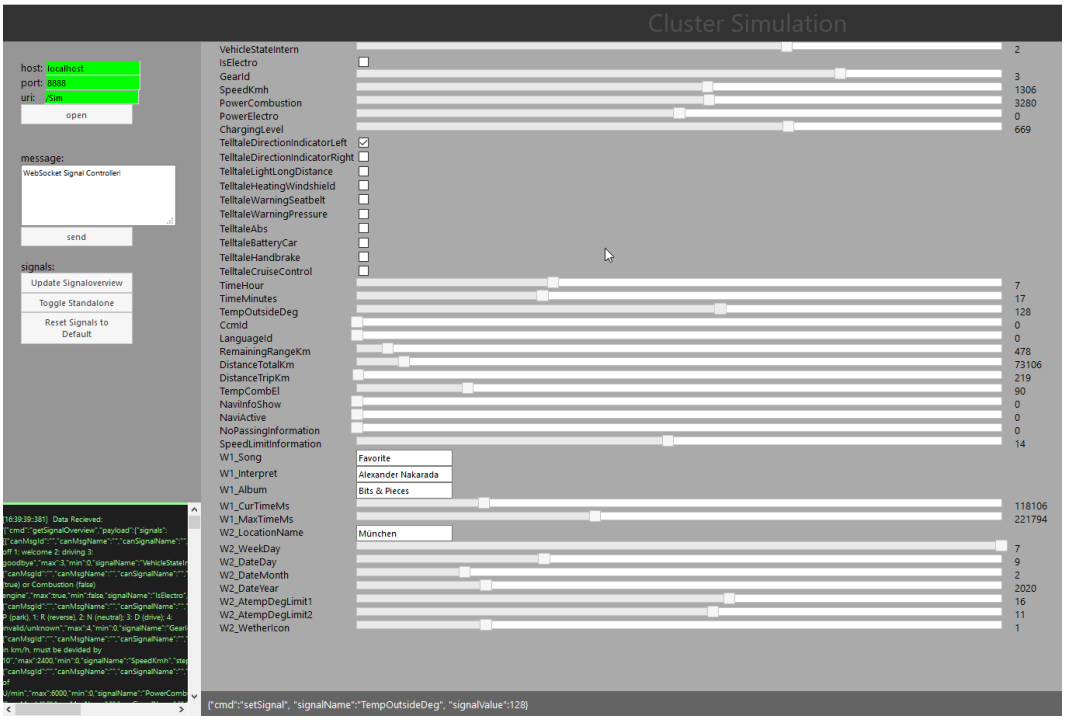


Figure 3: Screenshot of the HTML simulation interface

Moreover, a demo mode which animates all existing features is available. These are random animations, which are useful to get a quick overview of all implemented features and their behavior. In addition, the simulation interface creates an animation of input signals across different devices via a web socket connection. For example, a release on an Android device can be controlled from the computer if both devices are in the same network.

To sum up, the HMI has been developed in an innovative and open manner. It is largely independent of graphics engines and programming languages, i.e. it can be linked to new graphics engines (previously Kanzi and Unity) and ported to other programming languages (previously C++ and C#) with ease. Finally, the possibility to control the HMI framework via HTML5 is a big advantage in the prototyping phase.

The decision for Kanzi

Kanzi is a software tool chain developed by Rightware for designing and developing user interfaces, from early prototyping through to series production. It offers a high-performance rendering engine and UI framework with dedicated tools for designers and developers. Kanzi decouples design from application logic, allowing UI components, plugins, and application code to be reused across HMI projects and platforms. The Kanzi UI solution comprises Kanzi Studio, a PC-based UI editor with real-time preview, and

Kanzi Runtime, a cross-platform C++ API and UI framework for application development and platform integration.

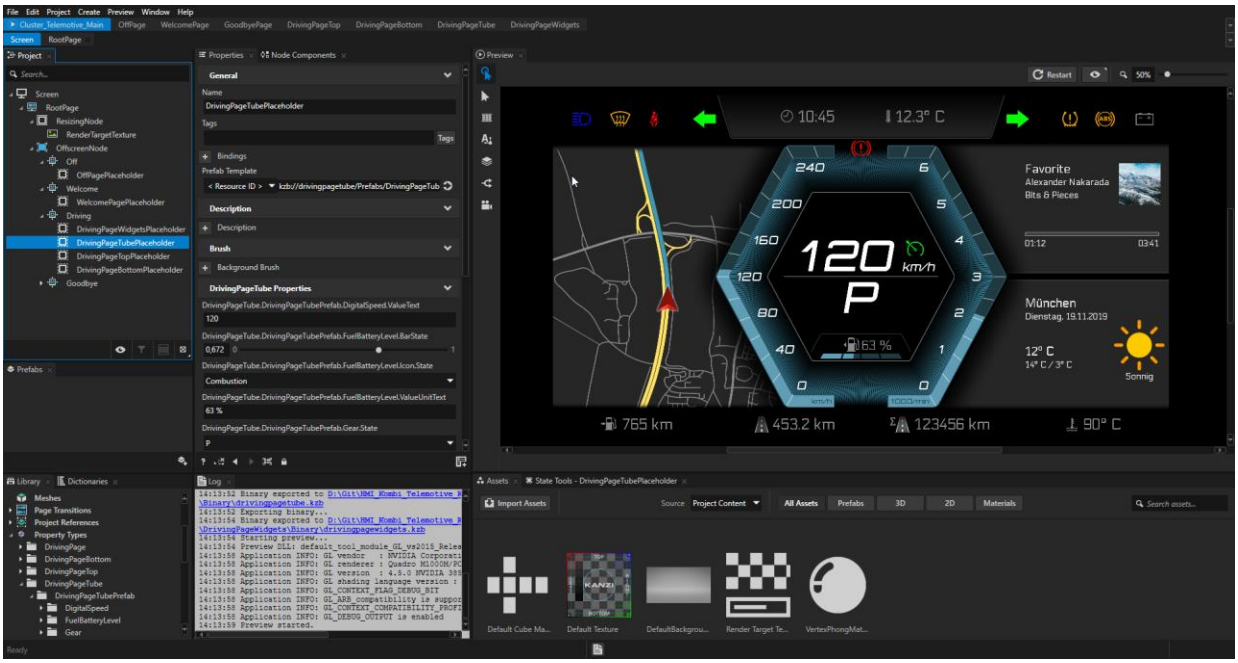


Figure 4: Overview of the Kanzi Studio application at Magna Telemotive

Magna Telemotive has connected the proprietary HMI framework to Kanzi Runtime, which allows for loading and accessing the pre-exported *.kzb files. The animations have been created in Kanzi Studio and the developed HMI graphics have been placed there. Then, an asset could be played out of Kanzi Studio (*.kzb), which contains the graphics as well as additional logic-like animations.

Kanzi has been chosen for HMI development because it is very simple to import graphical assets in standard formats (e.g. Photoshop, 3D models). The configurable state managers within Kanzi are used and the animations can be configured and played during the design process, based on the live preview capability built into Kanzi Studio.

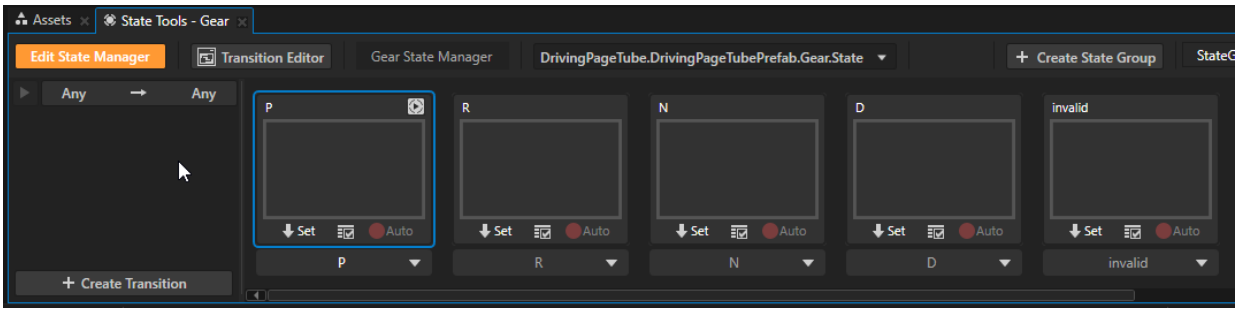


Figure 5: Definition of the different states for the example gear

Another advantage of Kanzi is that the content of the graphical assets can be easily controlled via the API.

Testing with TTA

The innovative HMI developed with Kanzi can be tested with the Python test automation framework Telemotive Test Automation (TTA), which allows the flexible and quick creation of test cases. With this tool, based on Python 3, developers can use open-source packages and benefit from elaborate reporting with clear visualization. TTA can be connected to any hardware or software, and thus facilitates the testing of embedded systems. Thanks to its open-source approach, tools for testing web interfaces and backends can be connected just as well. REST APIs can be used to connect not only the software under test in the development process but also any existing software tools. TTA can fill any gap in the automated tool chain. Writing test cases is child's play since TTA uses the Python syntax. Pre-assembled units, sample code, and easy-to-apply test case implementations support a straightforward learning process for new users as well as accelerated development cycles. Telemotive Test Automation is continuously improved to keep up with the market and can be customized for individual requirements.

Controlling the Kanzi-based HMI with TTA is fairly simple via the socket connection. For testing the Kanzi HMI, TTA reads back and verifies the graphical output. TTA checks if the HMI is displaying the correct signals and graphics with some special units such as image processing or the OCR unit. It therefore takes screenshots (or uses a frame grabber if a real head unit is being tested) of a specific part of the HMI and searches for specific symbols such as speed limit signs, as can be seen in Figure 6.

Figure 6: TTA recognizes the traffic sign

The tests always follow a schema:

1. Search for the predefined pixel area
2. Take a screenshot of this area
3. Compare the image with the expected result (reference image).

Image recognition is used to check if the traffic signs, here the speed limit sign of 30 km/h, are recognized correctly and correspond to the template. Subsequently, a detailed test report is provided, as shown in Figure 7.

Steps		ID	Start Time	Duration	Result
step_01_check_for_30_speed		None	13:14:09.486136	15.75s	Passed
Description:	Check if speed is recognized correctly				
Expected Result:	True (30km/h shield matches over 0.99)				
Actual Result:	True				
found:					
threshold:	0.989648282527924				
Assert:	True is True (self.assertTrue(res[2] > 0.95))				
img:					
template:					

Figure 7: Test report for verification of 30 km/h speed sign

The OCR unit has implemented an open-source Python OCR package, which checks if a text is displayed correctly. The text in this screenshot is read out and compared to the expected result.

In the example in Figure 8, the location is checked if it shows the current location “Munich”.

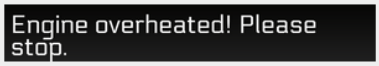
Steps		ID	Start Time	Duration	Result
step_01_get_location		None	13:12:01.379082	15.51s	Passed
Description:	Check if location matches expected value				
Expected Result:	$text == ['Munich']				
Actual Result:	['Munich'] == ['Munich'] is True				
img:					
location:					
Assert:	text = ['Munich'] self.assertActual(text=loc_text)				
step_03_set_speed		None	13:12:20.220257	12.12s	Passed
Description:	Ramp up speed to 200 and revs to 5000				
Expected Result:	$speed == 200				
Actual Result:	200 == 200 is True				
img:					
kmh_img:					
Assert:	speed = 200 self.assertActual(speed=kmh_value)				
step_05_check_control_en		None	13:12:37.919252	4.79s	Passed
Description:	Check if Check Control Message matches expected value.				
Expected Result:	Engine overheated! Please stop.				
Actual Result:	Engine overheated! Please stop.				
img:					
cc_img:					
Assert:	Engine overheated! Please stop. == Engine overheated! Please stop. (self.assertEqual('Engine overheated! Please stop.', " ".join(cc_text)))				

Figure 8: OCR is reading out the location, speed and control message

Then TTA ramps up the current speed to 200 km/h and checks the result. Moreover, the control messages at the top of the HMI are read out and compared to the expected result, which should be “Engine overheated! Please stop.” TTA has implemented several language packs, thus it can be used in any part of the world.

Conclusion

With Rightware’s Kanzi software, Magna Telemotive has found an easy-to-integrate solution for developing a reference automotive HMI. Compared to Unity, a big advantage of Kanzi is that it is a graphics engine designed for HMI development in the automotive industry. Unity, on the other hand, is a game engine that may spread into the automotive sector in the next few years, but has so far been used more for prototypes or concepts than for series development.

With Magna Telemotive’s HMI framework, the programming effort is kept to a minimum and a clear overview and easy maintainability are guaranteed due to the configuration files and the signal flow visualization. Conveniently, writing test cases in Telemotive Test Automation likewise requires minimal programming effort, since the TTA syntax corresponds to the Python syntax. Moreover, it also provides an elaborate reporting system with clear visualization and enables intuitive handling for testers with the HTML5 GUI.

Finally, it is important to mention that Magna Telemotive’s solutions are not bound to the automotive industry, but can be used in any digitized machine.

About the authors:

Manuel Zimmermann is a graduate computer scientist (M. Sc.) with focus on image processing and computer graphics. As a senior software engineer at MAGNA Telemotive, he has many years of experience in the field of software architecture for automotive HMI development. Another focus is the development of image processing algorithms and programming in the augmented reality (AR) environment.

Rebekka Haisch is a graduate industrial engineer (M. Sc.) and Product Manager of the Software Solutions division at MAGNA Telemotive. She is responsible for the strategic orientation of the division and the further development of Telemotive Test Automation and other products based on market requirements. Her goals also include digitizing companies, for example, through automated rather than manual testing. She holds the Professional Scrum Master certification and has experience in both Agile Development and Agile Management.

Derek Sellin is a graduate aerospace engineer (M. Sc.) with additional master and doctoral level business studies qualifications. In his capacity as Vice President of Marketing at Rightware, he leads product marketing as well as ecosystem development through the Kanzi Partner Program. His technology-industry background includes strategic planning, ecosystem management, and marketing leadership at Intel, Nokia, Symbian, and Broadcom in the domains of telecommunications networks, semiconductor platforms, operating systems, and UI frameworks.

DRIVING **EXCELLENCE.**
INSPIRING **INNOVATION.**