Starting from the names of the keywords which have been agreed on with the business experts, the test analyst needs to work out which steps are involved with that keyword. The documentation is performed as in a) above.

c) Test interface
Keywords can also be defined starting from the test interface. As the number of interface elements is limited and usually small, a limited number of keywords can be defined addressing these interface elements. And contrary to a) and b) above, which define high-level keywords on a domain layer, this approach will define low-level keywords on a test interface layer.

This approach can be rewarding if there is a focus on test automation, as the final limited set of keywords can be matched with keyword execution code. If a proper Keyword-Driven test framework is available, specified test cases can be available as automated tests almost immediately. The documentation is performed as in a) above.

d) Documented test procedures and test cases
Available test procedures and test cases can also be a valuable source of keywords. The actions in the test cases are examined. As a first step, each action can be treated as a new keyword. If two or more of these keywords turn out to refer to the same activities, they will be replaced by only one keyword, which best describes the activities.

If some of the keywords only occur in a certain sequence with others, they can be replaced either by one higher level keyword, or by a hierarchical keyword. The documentation is performed as for a) above.

It can be sufficient to use only one of these sources, but usually information from several sources is used.

In all cases, it can happen that the created pool of keywords is not sufficient to describe test cases, as there may be activities that were not recognized as requiring a keyword. These "gaps" will be filled by defining the missing keywords as test effort progresses.

C.3 Composing test cases

Once a basic set of keywords has been defined, these keywords can be used to describe test cases.

The test cases first need to be identified using test techniques as defined in ISO/IEC/IEEE 29119-4, and along with the process steps defined in ISO/IEC/IEEE 29119-2 (e.g. the activities of TD4 which are documented according to ISO/IEC/IEEE 29119-3).

While writing down the actions for the test cases, instead of describing the necessary activities in natural language, the predefined keywords are used.

If several test cases share the same sequence of actions or keywords, but their test data is different, they can be joined into one keyword test case along with different sets of test data.

Annex D
(informative)

Roles and Tasks

D.1 Overview – Roles and Tasks

A sound framework for Keyword-Driven Testing allows different tasks to be performed by different people, which can require different skills, such as test automation skills for the test interface layer, and business knowledge or test skills for the domain layer. This clause describes different roles in Keyword-Driven Testing.

NOTE 1 More roles can be involved in Keyword-Driven Testing or in the test process in general. In this clause, only those roles which are specific for the division of labour supported by Keyword-Driven Testing are discussed.

A single person can be assigned one, several or even all of these roles; but for best efficiency, and to reflect the different capabilities of team members, it can be advisable to assign the roles to different people. This is especially helpful in cases where a single person with all the required skills is not available.

NOTE 2 The following roles can be named differently in practice and the roles' activities can vary.

D.2 Domain expert

The domain expert is often the actual or future user of the test item. A domain expert has in-depth knowledge of how the test item should behave. This knowledge can be focussed on business cases, but can also reflect technical aspects. A person assuming this role ideally should have basic knowledge of test techniques and processes.

Tasks for the domain expert can include the following:

— Provide parts of the test basis by defining use cases, business cases, or paths through the application which need to be considered.

— Design test cases, and contribute to the test design specification.

— Identify business keywords.

The domain expert will closely cooperate with the test designer to perform these tasks.

D.3 Test designer

The test designer analyses complex use cases, requirements documents, and specifications. From these, the test designer derives test cases and subsequently the useful business level keywords. Therefore it is crucial to be able to distinguish relevant and irrelevant information.

The test designer should be in close dialogue with a subject matter expert in addition to analysing requirements or other product information (operation concepts, user guides, etc.) in order to derive useful business-level keywords.

Tasks for the test designer can include the following:

— Define keywords and their interfaces.

— Specify keywords composition and application in test cases.

— Create test cases based on the test basis.

— Rework any rough test cases from domain experts to create test cases and test procedure specifications.

The test designer will closely cooperate with the domain expert to make sure that keywords reflect the domain's language and the test cases are appropriate.

The test designer will closely cooperate with the test automation expert to make sure that the interfaces of the keywords are consistent and that the keyword execution code reflects the keywords in the right way.

D.4 Test automation expert

This role is only needed if test execution is to be automated.

The test automation expert needs to have experience in programming and knowledge about the test tools. The test automation expert needs to understand the scripting languages used in the framework which supports the automated test execution. Furthermore experience as a tester is beneficial and simplifies communication with other testers.

Tasks for the test automation expert can include the following:

— Implement the low-level keywords as executables and help ensure their functionality for test automation.

— Build the framework by selecting and combining appropriate tools on a technical level by adopting or implementing libraries.

— Together with the test designer, the test automation expert decides how to combine low-level keywords with higher level keywords and provides the technical means to support this from the automation side.

— Maintain test implementation scripts or the relevant parts of the scripts help ensure the availability and reliability of the test automation to avoid making a guarantee.

The test automation expert will closely cooperate with the test designer.

Annex E
(informative)

Basic keywords

E.1 Overview

This annex provides a basic list of keywords as an example. These keywords can be applied on a GUI as a test interface. The set of keywords is supposed to be generic and usable for most applications on this test interface. In practice, a test item may require more or different keywords than provided here. Therefore this set of keywords is extendable.

This set of keywords can be useful as an example for other test interfaces, and is offered as a quick start for using Keyword-Driven Testing.

E.2 Basic keywords for a GUI

In the following, basic keywords are listed for testing a GUI. A GUI typically has dialogs that are distinguished by unique titles. Within a dialog there are various GUI-objects. The dialog and its GUI-objects are identified by an identifier (id).

Keyword	Description
`clearContext (id)`	Removes the context from a component. **Parameters:** **id** (IN): id of the component
`click (id)`	Simple click with left mouse button on a given component. **Parameters:** **id** (IN): id of the target component
`clickWithOptions (id, MOUSE_BUTTON, times, MODIFIER, x, y)`	Extended click with additional options on a given component. **Parameters:** **id** (IN): id of the target component **MOUSE_BUTTON** (IN): one of the values which are defined in parameter MOUSE_BUTTON **times** (IN) OPTIONAL: how many times to click **MODIFIER** (IN) OPTIONAL: one of the values which are defined in parameter MODIFIER **x** (IN) OPTIONAL: x-coordinate relative in component **y** (IN) OPTIONAL: y-coordinate relative in component
`doubleClick (id)`	Double click with left mouse button on a given component. **Parameters:** **id** (IN): id of the target component
`drag (id, item, MOUSE_BUTTON, KEY)`	Drag. The parameter item is provided for selecting the drag source from a tree or list. Other components are normally not dragable. **Parameters:**

Keyword	Description
	id (IN): id of the target component **item** (IN) OPTIONAL: in case of a tree or list the id of the treeNode or the listItem **MOUSE_BUTTON** (IN): one of the values which are defined in parameter MOUSE_BUTTON **KEY** (IN): one of the values which are defined in parameter KEY
`drop (id, item)`	Drop. **Parameters:** **id** (IN): id of the target component **item** (IN) OPTIONAL: in case of a tree or list the id of the treeNode or the listItem
`getCaption (id, varCaptionValue)`	Writes the caption of target component (id) into varCaptionValue. **Parameters:** **id** (IN): id of the target component **varCaptionValue** (IN): variable with caption of component
`getProperty (id, PROPERTY_NAME, varPropertyValue)`	Writes the value of the given property (PROPERTY_NAME) of the target component (id) into varPropertyValue. **HINT:** In case of no hit the interaction fails and the parameter gets the value UNDEFINED. **Parameters:** **id** (IN): id of the target component **PROPERTY_NAME** (IN): one of the properties which are defined in parameter PROPERTY_NAME **varPropertyValue** (IN): variable with value of property
`getText (id, varText)`	Writes the text of the target component (id) into varText. **Parameters:** **id** (IN): id of the target component **varText** (IN): variable with text of component
`moveMouse (target_id, target_item)`	Move the mouse to the component with id target_id. **Parameters:** **target_id** (IN): id of the target component **target_item** (IN) OPTIONAL: in case of a tree or list the optional id of the treeNode or the listItem
`openContextMenu (id)`	Opens the context menu of the given component. **Parameters:** **id** (IN): id of the component
`pressKey (id, MODIFIER, KEY)`	Presses a key or key combination (with modifier). **Please note:** this interaction is intended for testing keyboard commands and shortcuts. For entering text into a text area or text field, please use the "setText" interaction instead. **Parameters:** **id** (IN): id of the target component or the value "UNUSED" in which case the key press happens on the currently focussed component **MODIFIER** (IN) OPTIONAL: a combination of one or more modifier keys (such as "shift" or "alt") **KEY** (IN): the key to be pressed

Keyword	Description
`setContext (id)`	Sets the context 'passively' for the given component (programming construct). **IMPORTANT:** This is in contrast to setWindowActive which brings a window / dialog 'actively' to the foreground. **Parameters:** **id** (IN): id of the component
`setFocus (id)`	Sets the focus on the given component. **IMPORTANT:** For cells, tree items, list items and menu items it isn't possible to set the focus. The focus can only be set for tables, trees, lists and menus, respectively. **Parameters:** **id** (IN): id of the component
`setText (id, text)`	Sets or clears text in the given component. **Parameters:** **id** (IN): id of the component **text** (IN): the text
`verifyCaption (id, OPTION_PATTERN_MATCHING, expectedCaptionValue)`	Verifies the expected caption (expectedCaptionValue) of the target component (id) regarding search algorithm (OPTION_PATTERN_MATCHING). **Parameters:** **id** (IN): id of the target component **OPTION_PATTERN_MATCHING** (IN): specifies the format of search algorithm **expectedCaptionValue** (IN): expected caption
`verifyProperty (id, PROPERT_NAME, OPTION_PATTERN_MATCHING, expectedPropertyValue)`	Verifies the expected property value (expectedPropertyValue) of the target component (id) regarding search algorithm (OPTION_PATTERN_MATCHING). **Parameters:** **id** (IN): id of the target component **PROPERTY_NAME** (IN): one of the properties which are defined in parameter PROPERTY_NAME **OPTION_PATTERN_MATCHING** (IN): specifies the format of search algorithm **expectedPropertyValue** (IN): variable with value of property
`verifyText (id, OPTION_PATTERN_MATCHING, expectedCaptionText)`	Verifies the expected text (expectedText) of the target component (id) regarding search algorithm (OPTION_PATTERN_MATCHING). **Parameters:** **id** (IN): id of the target component **OPTION_PATTERN_MATCHING** (IN): specifies the format of search algorithm **expectedCaptionText** (IN): expected caption
`waitForExist (id, maxTimeToWait)`	Waits until a component exist. **Parameters:** **id** (IN): id of the target component **maxTimeToWait** (IN): waiting period in milliseconds
`waitForNotExist (id,`	Waits until a component does not exist.

Keyword	Description
`maxTimeToWait)`	**Parameters:** **id** (IN): id of the target component **maxTimeToWait** (IN): waiting period in milliseconds

Table E.1 — Example of generic basic keywords

There are further GUI objects that require specific, specialized or extensible keywords.

Examples can be found in the table below:

Keyword	Description
`selectMenuItem(id, OPTION_NUMBER_OR_NAME, menuItem)`	Selects a menu item (depends on value of OPTION_NUMBER_OR_NAME). **Parameters:** **id** (IN): id of the target menu **OPTION_NUMBER_OR_NAME** (IN): defines whether the following parameter is defined by its name or number **menuItem** (IN): name or number of menu item dependent on value of OPTION_NUMBER_OR_NAME
`selectListItem (id, MODIFIER, OPTION_NUMBER_OR_NAME, listItem)`	Selects one list item. MODIFIER allows to define a multiselect interaction by repeating this interaction. The list items can be given by name or number (dependent on the value of OPTION_NUMBER_OR_NAME) . A list can be a list view, a combobox, a dropdown list, radio button or tabcard. **Parameters:** **id** (IN): id of the target list **MODIFIER** (IN) OPTIONAL: one of the values which are defined in parameter MODIFIER **OPTION_NUMBER_OR_NAME** (IN): defines whether the following parameter is defined by its name or number **listItem** (IN): name or number of list item dependent on value of OPTION_NUMBER_OR_NAME
`startApplication (currentClientID, propertyFile)`	Starts the application, using a property file for application settings. The property file is expected to be in the directory properties. **Parameters:** **currentClientID** (IN): id of the application **propertyFile** (IN): filename of the property file

Table E.2 — Example of specialized basic keywords

E.3 Example application of basic keywords

The following example shows a keyword test case structured in three layers: low-level keywords are used at the test interface layer, an intermediate layer combines the low-level keywords to an application-related vocabulary, and business keywords use these keywords from the intermediate layer at the domain layer.

Each keyword is written in a function-like style, i.e. it consists of a unique name followed by none, one or more parameters placed in braces. The parameters used at the domain layer are passed through the intermediate layer to the test interface layer.

This test of a car's configuration program addresses the use case for configuring a car with some accessories. As a final action, the calculated price will be verified.

In practice, in this example a test case would be written using only domain layer keywords. The other layers are provided for a better understanding.

domain layer	intermediate layer	test interface layer
startCarConfigurator (“login”, “password”, “english”)		
	startCarConfiguratorCmdLine()	
		startApplication (“carConfigurator”, “E:\CarConfigurator.ini”)
	login (“login”, “password”)	setText (“userField”, “login”)
		setText (“pwdField”, “password”)
		click (“loginBtn”)
	setLanguage (“english”)	
		selectMenuItem (“mainMenu”, NAME, “Language”)
		selectMenuItem (“menuLanguage”, NAME, “english”)
selectVehicle (“Rolo”, “red”)		
	selectTabcard (“Cars”)	
		setContext (“carConfig”)
		selectListItem (“tabbedPane”, UNUSED, NAME, “Vehicles”)
	selectVehicleByNameAndColour (“Rolo”, “red”)	
		selectListItem (“vehicleList”, UNUSED, NAME, “Rolo”)
		selectListItem (“colourList”, UNUSED, NAME, “red”)
selectAccessories (“[Steering wheel, brown, leather]”, “[Mats, black, textile]”)		
	selectTabcard (“Accessories”)	
		setContext (“carConfig”)
		selectListItem (“tabbedPane”, UNUSED, NAME, “Accessories”)
	selectAccessoriesByNameColourMaterial (“Steering wheel”, “brown”, “leather”)	selectListItem (“accessoryNameList”, UNUSED, NAME, “Steering wheel”)

domain layer	intermediate layer	test interface layer
		selectListItem (“accessoryColourList”, UNUSED, NAME, “brown”)
		selectListItem (“accessoryMaterialList”, UNUSED, NAME, “leather”)
		click (“addAccessoryBtn”)
	selectAccessoriesByNameColourMaterial (“Mats”, “black”, “textile”)	
		selectListItem (“accessoryNameList”, UNUSED, NAME, “Mats”)
		selectListItem (“accessoryColourList”, UNUSED, NAME, “black”)
		selectListItem (“accessoryMaterialList”, UNUSED, NAME, “textile”)
		click (“addAccessoryBtn”)
verifyVehiclePrice (“20.000”, “$”)		
	verifyCalculatedPrice (“20.000”, “$”)	
		verifyProperties (“calculatedPrice”, “InnerText”, “=”, “20.000”)
		verifyProperties (“priceCurrency”, “InnerText”, “=”, “$”)
closeCarConfigurator ()		
	closeApplication (“CarConfigurator”)	
		selectMenuItem (“mainMenu”, NAME, “File”)
		selectMenuItem (“fileMenu”, NAME, “Exit”)

Table E.3 — Example test case using basic keywords as part of composite keywords

Annex F
(informative)
Examples

F.1 Overview

This annex provides supplemental examples on the application of Keywords. The examples are provided using different style and granularity to give an idea of the possible variety of different implementations of Keyword-Driven Testing.

F.2 Example: test procedure from ISO/IEC/IEEE 29119-3

This example shows how the test procedure example from ISO/IEC/IEEE 29119-3, Annex K.2 could be changed to be keyword based.

Used keywords:

Keyword	Parameter	Description
StartUp		Set the apparatus ready for sampling analysis.
PlaceNcsSample	<value>	Place NCS samples with the value in parameter <value> in the carousel
StartAnalysis		Start analysis procedure
WaitForAnalysis	<time>	Wait for analysis to be completed; maximum wait time: <time> seconds
CheckResult	<value > <testcase>	Check that the analysis result equals parameter <value>; log number of test cases as provided in parameter <testcase>
Shutdown		Turn off the apparatus, remove the samples and clean up any spillage.

Table F.1 — Keywords applied to Annex K.2 of ISO/IEC/IEEE 29119-3

NOTE The rest of Annex F.2 uses the headings and numberings as were used in ISO/IEC/IEEE 29119-3 Annex K.2

Test procedure specification

2. Test Sets

This section describes the test sets to be executed in the first execution cycle. The sets are ordered by feature set.

2.1 (FS1) Setup of the system

.
.
.

2.3 (FS2) Identification of compounds

ID	Objective	Priority	Contents
I-3	Measuring range	Medium	Test cases 17-1 to 17-5 incl.
	…		

.
.
.

3.3 Test procedures

Test Procedure ID	**Objective and Priority**	**Estimated Duration:**
I-3	The purpose of this test procedure is to test the way the system handles the defined measuring ranges for NCS. Priority: Medium	

Relationships to other procedures: None

Test Log			
Date:	Initials:	Test item:	OK / Not OK

Comments:

Procedure

Step no.	**Keyword**	**Parameter 1**	**Parameter 2**	**Remark**
1	StartUp			
2	PlaceNcsSample	1		
3	PlaceNcsSample	2		
4	PlaceNcsSample	56		
5	PlaceNcsSample	315		
6	PlaceNcsSample	316		
7	StartAnalysis			
8	WaitForAnalysis	1		
9	CheckResult	"Invalid sample"	17-1	
10	WaitForAnalysis	1		
11	CheckResult	"OK"	17-2	
12	WaitForAnalysis	1		
13	CheckResult	"OK"	17-3	
14	WaitForAnalysis	1		
15	CheckResult	"OK"	17-4	
16	WaitForAnalysis	1		
17	CheckResult	"Invalid sample"	17-5	
18	Shutdown			

F.3 Example: Test of shopping procedure with low-level keywords

The following example shows a keyword test case composed from low-level keywords on a test interface layer.

It is written in a function-like style. The keywords are derived from the user interface and take parameters which are placed in brackets.

This test of a shopping cart would address a Use Case such as “Choose a product and place it in the shopping cart”. The selected product is referred to by the name "PRODUCT", which is assumed to have the

value "ISO/IEC/IEEE 29119-5 Keyword-Driven Testing". If this test case was iterated with different values for "PRODUCT", the test case would become data-driven:

enterValue(“SearchField”, “Keyword-Driven Testing”)

selectObject(“Button”, “Search”)

selectObject(“ResultList”, PRODUCT)

selectObject(“Button”, “AddToShoppingCart”)

selectObject(“Button”, “ShowShoppingCart”)

verifyObject(“ShoppingCart#CONTAINS”, ”PRODUCT”)

placeItemInShoppingCart(“PRODUCT”)

verifyItemInShoppingCart(“PRODUCT”)

F.4 Example for calculator with low-level keywords

This example shows a keyword test case for a calculator, composed from low-level keywords on a test interface layer. As opposed to the example in F.3, this keyword test case is laid out in a table with one column for the keywords, and further column for the parameters, of which the first parameter (e.g., object identifier) is distinguished as it defines the target of the operation.

Keyword	Object Identifier	Parameter
ClickButton	C	
ClickButton	5	
ClickButton	Multiplication	
ClickButton	9	
ClickButton	Equal	
Verify	ResultBox	45

Table F.2 — Example for low-level keywords

F.5 Example for calculator with domain level keywords

This example is similar to the example in F.4 as it addresses the same test item and uses the same layout.

But it uses domain level Keywords, which may be composed from low-level keywords (e.g. test interface layer) or may refer to a more complex set of actions.