Robot Framework 7.0

Robot Framework 7.0 is a new major release with highly enhanced listener interface (#3296), native `VAR` syntax for creating variables (#3761), support for mixing embedded and normal arguments with library keywords (#4710), JSON result format (#4847) and various other enhancements and bug fixes.

Robot Framework 7.0 was released on Thursday January 11, 2024. Questions and comments related to the release can be sent to the `#devel` channel on Robot Framework Slack and possible bugs submitted to the issue tracker.

Installation

If you have pip installed, just run

```
pip install --upgrade robotframework
```

to install the latest available stable release or use

```
pip install robotframework==7.0
```

to install exactly this version. Alternatively you can download the package from PyPI and install it manually. For more details and other installation approaches, see the installation instructions.

Most important enhancements

> If you are interested to learn more about the new features in Robot Framework 7.0, join the RoboCon conference in February, 2024. Pekka Klärck, Robot Framework lead developer, will go through the key features briefly in the onsite conference in Helsinki and more thoroughly in the online edition.
>
> The conference has also dozens of other great talks, workshops and a lot of possibilities to meet other community members as well as developers of various tools and libraries in the ecosystem. All profits from the conference will be used for future Robot Framework development.

Listener enhancements

Robot Framework's listener interface is a very powerful mechanism to get notifications about various events during execution and it also allows modifying data and results on the fly. It is not typically directly used by normal Robot Framework users, but they are likely to use tools that use it internally. The listener API has been significantly enhanced making it possible to create even more powerful and interesting tools in the future.

Support keywords and control structures with listener version 3

The major limitation with the listener API has been that the listener API version 2 only supports getting notifications, not making modifications, and that the more powerful listener API version 3 has only supported suites and tests/tasks.

The biggest enhancement in the whole Robot Framework 7.0 is that the listener version 3 has been extended to support also keywords and control structures (#3296). For example, a listener having the following methods prints information about started keywords and ended WHILE loops:

```
from robot import result, running


def start_keyword(data: running.Keyword, result: result.Keyword):
    print(f"Keyword '{result.full_name}' used on line {data.lineno} started.")


def end_while(data: running.While, result: result.While):
    print(f"WHILE loop on line {data.lineno} ended with status {result.status} "
          f"after {len(result.body)} iterations.")
```

With keyword calls it is possible to also get more information about the actually executed keyword. For example, the following listener prints some information about the executed keyword and the library it belongs to:

```
from robot.running import Keyword as KeywordData, LibraryKeyword
from robot.result import Keyword as KeywordResult


def start_library_keyword(data: KeywordData,
                          implementation: LibraryKeyword,
                          result: KeywordResult):
    library = implementation.owner
    print(f"Keyword '{implementation.name}' is implemented in library "
          f"'{library.name}' at '{implementation.source}' on line "
          f"{implementation.lineno}. The library has {library.scope.name} "
          f"scope and the current instance is {library.instance}.")
```

As the above example already illustrated, it is even possible to get an access to the actual library instance. This means that listeners can inspect the library state and also modify it. With user keywords it is even possible to modify the keyword itself or, via the `owner` resource file, any other keyword in the resource file.

Listeners can also modify results if needed. Possible use cases include hiding sensitive information and adding more details to results based on external sources.

Notice that although listener can change status of any executed keyword or control structure, that does not directly affect the status of the executed test. In general listeners cannot directly fail keywords so that execution would stop or handle failures so that execution would continue. This kind of functionality may be added in the future if there are needs.

The new listener version 3 methods are so powerful and versatile that going them through thoroughly in these release notes is not possible. For more examples, you can see the acceptance tests using the methods in various interesting and even crazy ways.

Listener version 3 is the default listener version

Earlier listeners always needed to specify the API version they used with the `ROBOT_LISTENER_API_VERSION` attribute. Now that the listener version 3 got the new methods, it is considered so much more powerful than the version 2 that it was made the default listener version (#4910).

The listener version 2 continues to work, but using it requires specifying the listener version as earlier. The are no plans to deprecate the listener version 2, but we nevertheless highly recommend using the version 3 whenever possible.

Libraries can register themselves as listeners by using string `SELF`

Listeners are typically enabled from the command line, but libraries can register listeners as well. Often libraries themselves want to act as listeners, and that has earlier required using `self.ROBOT_LIBRARY_LISTENER = self` in the `__init__` method. Robot Framework 7.0 makes it possible to use string `SELF` (case-insensitive) for this purpose as well (#4910), which means that a listener can be specified as a class attribute and not only in `__init__`. This is especially convenient when using the `@library` decorator:

```
from robot.api.deco import keyword, library


@library(listener='SELF')
class Example:

    def start_suite(self, data, result):
        ...

    @keyword
    def example(self, arg):
        ...
```

Nicer API for modifying keyword arguments

Modifying keyword call arguments programmatically has been made more convenient (#5000). This enhancement eases modifying arguments using the new listener version 3 `start/end_keyword` methods.

Paths are passed to version 3 listeners as `pathlib.Path` objects

Listeners have methods like `output_file` and `log_file` that are called when result files are ready so that they get the file path as an argument. Earlier paths were strings, but nowadays listener version 3 methods get them as more convenient pathlib.Path objects.

Native VAR syntax

The new `VAR` syntax (#3761) makes it possible to create local variables as well as global, suite and test/task scoped variables dynamically during execution. The motivation is to have a more convenient syntax than using the `Set Variable` keyword for creating local variables and to unify the syntax for creating variables in different scopes. Except for the mandatory `VAR` marker, the syntax is also the same as when creating variables in the Variables section. The syntax is best explained with examples:

```
*** Test Cases ***
Example
    # Create a local variable `${local}` with a value `value`.
    VAR    ${local}    value

    # Create a variable that is available throughout the whole suite.
    # Supported scopes are GLOBAL, SUITE, TEST, TASK and LOCAL (default).
    VAR    ${suite}    value    scope=SUITE

    # Validate created variables.
    Should Be Equal    ${local}    value
    Should Be Equal    ${suite}    value

Example continued
    # Suite level variables are seen also by subsequent tests.
    Should Be Equal    ${suite}    value
```

When creating `${scalar}` variables having long values, it is possible to split the value to multiple lines. Lines are joined together with a space by default, but that can be changed with the `separator` configuration option. Similarly as in the Variables section, it is possible to create also `@{list}` and `&{dict}` variables. Unlike in the Variables section, variables can be created conditionally using IF/ELSE structures:

```
*** Test Cases ***
Long value
    VAR    ${long}
    ...    This value is rather long.
    ...    It has been split to multiple lines.
    ...    Parts will be joined together with a space.

Multiline
    VAR    ${multiline}
    ...    First line.
    ...    Second line.
    ...    Last line.
    ...    separator=\n

List
    # Creates a list with three items.
    VAR    @{list}    a    b    c

Dictionary
    # Creates a dictionary with two items.
    VAR    &{dict}    key=value    second=item

Normal IF
    IF    1 > 0
        VAR    ${x}    true value
    ELSE
        VAR    ${x}    false value
```

```
    END

Inline IF
    IF    1 > 0    VAR    ${x}    true value    ELSE    VAR    ${x}    false value
```

Mixed argument support with library keywords

User keywords got support to use both embedded and normal arguments in Robot Framework 6.1 (#4234) and now that support has been added also to library keywords (#4710). The syntax works so, that if a function or method implementing a keyword accepts more arguments than there are embedded arguments, the remaining arguments can be passed in as normal arguments. This is illustrated by the following example keyword:

```
@keyword('Number of ${animals} should be')
def example(animals, count):
    ...
```

The above keyword could be used like this:

```
*** Test Cases ***
Example
    Number of horses should be    2
    Number of horses should be    count=2
    Number of dogs should be    3
```

JSON result format

Robot Framework 6.1 added support to convert test/task data to JSON and back and Robot Framework 7.0 extends the JSON serialization support to execution results (#4847). One of the core use cases for data serialization was making it easy to transfer data between process and machines, and now it is also easy to pass results back.

Also the built-in Rebot tool that is used for post-processing results supports JSON files both in output and in input. Creating JSON output files is done using the normal `--output` option so that the specified file has a `.json` extension:

```
rebot --output output.json output.xml
```

When reading output files, JSON files are automatically recognized by the extension:

```
rebot output.json
rebot output1.json output2.json
```

When combining or merging results, it is possible to mix JSON and XML files:

```
rebot output1.xml output2.json
rebot --merge original.xml rerun.json
```

The JSON output file structure is documented in the `result.json` schema file.

The plan is to enhance the support for JSON output files in the future so that they could be created already during execution. For more details see issue #3423.

Argument conversion enhancements

Automatic argument conversion is a very powerful feature that library developers can use to avoid converting arguments manually and to get more useful Libdoc documentation. There are two important new enhancements to it.

Support for `Literal`

In Python, the Literal type makes it possible to type arguments so that type checkers accept only certain values. For example, this function only accepts strings `x`, `y` and `z`:

```
def example(arg: Literal['x', 'y', 'z']):
    ...
```

Robot Framework has been enhanced so that it validates that an argument having a `Literal` type can only be used with the specified values (#4633). For example, using a keyword with the above implementation with a value `xxx` would fail.

In addition to validation, arguments are also converted. For example, if an argument accepts `Literal[-1, 0, 1]`, used arguments are converted to integers and then validated. In addition to that, string matching is case, space, underscore and hyphen insensitive. In all cases exact matches have a precedence and the argument that is passed to the keyword is guaranteed to be in the exact format used with `Literal`.

`Literal` conversion is in many ways similar to Enum conversion that Robot Framework has supported for long time. `Enum` conversion has benefits like being able to use a custom documentation and it is typically better when using the same type multiple times. In simple cases being able to just use `arg: Literal[...]` without defining a new type is very convenient, though.

Support "stringified" types like `'list[int]'` and `'int | float'`

Python's type hinting syntax has evolved so that generic types can be parameterized like `list[int]` (new in Python 3.9) and unions written as `int | float` (new in Python 3.10). Using these constructs with older Python versions causes errors, but Python type checkers support also "stringified" type hints like `'list[int]'` and `'int | float'` that work regardless the Python version.

Support for stringified generics and unions has now been added also to Robot Framework's argument conversion (#4711). For example, the following typing now also works with Python 3.8:

```
def example(a: 'list[int]', b: 'int | float'):
    ...
```

These stringified types are also compatible with the Remote library API and other scenarios where using actual types is not possible.

Tags set globally can be removed using `-tag` syntax

Individual tests and keywords can nowadays remove tags that have been set in the Settings section with `Test Tags` or `Keyword Tags` settings by using the `-tag` syntax with their own `[Tags]` setting (#4374). For example, tests `T1` and `T3` below get tags `all` and `most`, and test `T2` gets tags `all` and `one`:

```
*** Settings ***
Test Tags      all    most

*** Test Cases ***
T1
    No Operation
T2
```

```
    [Tags]    one    -most
    No Operation
T3
    No Operation
```

With tests it is possible to get the same effect by using the `Default Tags` setting and overriding it where needed. That syntax is, however, considered deprecated (#4365) and using the new `-tag` syntax is recommended. With keywords there was no similar functionality earlier.

Dynamic and hybrid library APIs support asynchronous execution

Dynamic and hybrid libraries nowadays support asynchronous execution. In practice the special methods like `get_keyword_names` and `run_keyword` can be implemented as async methods (#4803).

Async support was added to the normal static library API in Robot Framework 6.1 (#4089). A bug related to handling asynchronous keywords if execution is stopped gracefully has also been fixed (#4808).

Timestamps in result model and output.xml use standard format

Timestamps used in the result model and stored to the output.xml file used custom format like `20231107 19:57:01.123` earlier. Non-standard formats are seldom a good idea, and in this case parsing the custom format turned out to be slow as well.

Nowadays the result model stores timestamps as standard datetime objects and elapsed times as a timedelta (#4258). This makes creating timestamps and operating with them more convenient and considerably faster. The new objects can be accessed via `start_time`, `end_time` and `elapsed_time` attributes that were added as forward compatibility already in Robot Framework 6.1 (#4765). Old information is still available via the old `starttime`, `endtime` and `elapsedtime` attributes, so this change is fully backwards compatible.

The timestamp format in output.xml has also been changed from the custom `YYYYMMDD HH:MM:SS.mmm` format to ISO 8601 compatible `YYYY-MM-DDTHH:MM:SS.mmmmmm`. Using a standard format makes it easier to process output.xml files, but this change also has big positive performance effect. Now that the result model stores timestamps as datetime objects, formatting and parsing them with the available isoformat() and fromisoformat() methods is very fast compared to custom formatting and parsing.

A related change is that instead of storing start and end times of each executed item in output.xml, we nowadays store their start and elapsed times. Elapsed times are represented as floats denoting seconds. Having elapsed times directly available is a lot more convenient than calculating them based on start and end times. Storing start and elapsed times also takes less space than storing start and end times.

As the result of these changes, times are available in the result model and in output.xml in higher precision than earlier. Earlier times were stored in millisecond granularity, but nowadays they use microseconds. Logs and reports still use milliseconds, but that can be changed in the future if there are needs.

Changes to output.xml are backwards incompatible and affect all external tools that process timestamps. This is discussed more in Changes to output.xml section below along with other output.xml changes.

Dark mode support to report and log

Report and log got a new dark mode (#3725). It is enabled automatically based on browser and operating system preferences, but there is also a toggle to switch between the modes.

Backwards incompatible changes

Python 3.6 and 3.7 are no longer supported

Robot Framework 7.0 requires Python 3.8 or newer (#4294). The last version that supports Python 3.6 and 3.7 is Robot Framework 6.1.1.

Changes to output.xml

The output.xml file has changed in different ways making Robot Framework 7.0 incompatible with external tools processing output.xml files until these tools are updated. We try to avoid this kind of breaking changes, but in this case especially the changes to timestamps were considered so important that we eventually would have needed to do them anyway.

Due to the changes being relatively big, it can take some time before external tools are updated. To allow users to take Robot Framework 7.0 into use also if they depend on an incompatible tool, it is possible to use the new `--legacy-output` option both as part of execution and with the Rebot tool to generate output.xml files that are compatible with older versions.

Timestamp related changes

The biggest changes in output.xml are related to timestamps (#4258). With earlier versions start and end times of executed items, as well as timestamps of the logged messages, were stored using a custom `YYYYMMDD HH:MM:SS.mmm` format, but nowadays the format is ISO 8601 compatible `YYYY-MM-DDTHH:MM:SS.mmmmmm`. In addition to that, instead of saving start and end times to `starttime` and `endtime` attributes and message times to `timestamp`, start and elapsed times are now stored to `start` and `elapsed` attributes and message times to `time`.

Examples:

```
<!-- Old format -->
<msg timestamp="20231108 15:36:34.278" level="INFO">Hello world!</msg>
<status status="PASS" starttime="20231108 15:37:35.046" endtime="20231108 15:37:35.046"/>

<!-- New format -->
<msg time="2023-11-08T15:36:34.278343" level="INFO">Hello world!</msg>
<status status="PASS" start="2023-11-08T15:37:35.046153" elapsed="0.000161"/>
```

The new format is standard compliant, contains more detailed times, makes the elapsed time directly available and makes the `<status>` elements over 10% shorter. These are all great benefits, but we are still sorry for all the extra work this causes for those developing tools that process output.xml files.

Keyword name related changes

How keyword names are stored in output.xml has changed slightly (#4884). With each executed keywords we store both the name of the keyword and the name of the library or resource file containing it. Earlier the latter was stored to attribute `library` also with resource files, but nowadays the attribute is generic `owner`. In addition to `owner` being a better name in general, it also matches the new `owner` attribute keywords in the result model have.

Another change is that the original name stored with keywords using embedded arguments is nowadays in `source_name` attribute when it used to be in `sourcename`. This change was done to make the attribute consistent with the attribute in the result model.

Examples:

```
<!-- Old format -->
<kw name="Log" library="BuiltIn">...</kw>
<kw name="Number of horses should be" sourcename="Number of ${animals} should be" library="my_resource">...</kw>

<!-- New format -->
<kw name="Log" owner="BuiltIn">...</kw>
<kw name="Number of horses should be" source_name="Number of ${animals} should be" owner="my_resource">...</kw>
```

Other changes

Nowadays keywords and control structures can have a message. Messages are represented as the text of the `<status>` element, and they have been present already earlier with tests and suites. Related to this, control structured cannot anymore have `<doc>`. (#4883)

These changes should not cause problems for tools processing output.xml files, but storing messages with each failed keyword and control structure may increase the output.xml size.

Schema updates

The output.xml schema has been updated and can be found via https://github.com/robotframework/robotframework/tree/master/doc/schema/.

Changes to result model

There have been some changes to the result model that unfortunately affect external tools using it. The main motivation for these changes has been cleaning up the model before creating a JSON representation for it (#4847).

Changes related to keyword names

The biggest changes are related to keyword names (#4884). Earlier `Keyword` objects had a `name` attribute that contained the full keyword name like `BuiltIn.Log`. The actual keyword name and the name of the library or resource file that the keyword belonged to were in `kwname` and `libname` attributes, respectively. In addition to these, keywords using embedded arguments also had a `sourcename` attribute containing the original keyword name.

Due to reasons explained in #4884, the following changes have been made in Robot Framework 7.0:

- Old `kwname` is renamed to `name`. This is consistent with the execution side `Keyword`.
- Old `libname` is renamed to generic `owner`.
- New `full_name` is introduced to replace the old `name`.
- `sourcename` is renamed to `source_name`.
- `kwname`, `libname` and `sourcename` are preserved as properties. They are considered deprecated, but accessing them does not cause a deprecation warning yet.

The backwards incompatible part of this change is changing the meaning of the `name` attribute. It used to be a read-only property yielding the full name like `BuiltIn.Log`, but now it is a normal attribute that contains just the actual keyword name like `Log`. All other old attributes have been preserved as properties and code using them does not need to be updated immediately.

Deprecated attributes have been removed

The following attributes that were deprecated already in Robot Framework 4.0 have been removed (#4846):

- `TestSuite.keywords`. Use `TestSuite.setup` and `TestSuite.teardown` instead.
- `TestCase.keywords`. Use `TestCase.body`, `TestCase.setup` and `TestCase.teardown` instead.
- `Keyword.keywords`. Use `Keyword.body` and `Keyword.teardown` instead.
- `Keyword.children`. Use `Keyword.body` and `Keyword.teardown` instead.
- `TestCase.critical`. The whole criticality concept has been removed.

Additionally, `TestSuite.keywords` and `TestCase.keywords` have been removed from the execution model.

Changes to parsing model

There have been some changes also to the parsing model:

- The node representing the deprecated `[Return]` setting has been renamed from `Return` to `ReturnSetting`. At the same time, the node representing the `RETURN` statement has been renamed from `ReturnStatement` to `Return` (#4939).

To ease transition, `ReturnSetting` has existed as an alias for `Return` starting from Robot Framework 6.1 (#4656) and `ReturnStatement` is preserved as an alias now. In addition to that, the `ModelVisitor` base class has special handling for `visit_ReturnSetting` and `visit_ReturnStatement` visitor methods so that they work correctly with `ReturnSetting` and `ReturnStatement` with Robot Framework 6.1 and newer. Issue #4939 explains this in more detail and has a concrete example how to support also older Robot Framework versions.

- The node representing the `Test Tags` setting as well as the deprecated `Force Tags` setting has been renamed from `ForceTags` to `TestTags` (#4385). `ModelVisitor` has special handling for the `visit_ForceTags` method so that it will continue to work also after the change.
- The token type used with `AS` (or `WITH NAME`) in library imports has been changed to `Token.AS` (#4375). `Token.WITH_NAME` still exists as an alias for `Token.AS`.
- Statement `type` and `tokens` have been moved from `_fields` to `_attributes` (#4912). This may affect debugging the model.

Changes to Libdoc spec files

The following deprecated constructs have been removed from Libdoc spec files (#4667):

- `datatypes` have been removed from XML or JSON spec files. They were deprecated in favor of `typedocs` already in Robot Framework 5.0 (#4160).
- Type names are not anymore written to XML specs as content of the `<type>` elements. The name is available as the `name` attribute of `<type>` elements since Robot Framework 6.1 (#4538).
- `types` and `typedocs` attributes have been removed from arguments in JSON specs. The `type` attribute introduced in RF 6.1 (#4538) needs to be used instead.

Libdoc schema files have been updated and can be found via https://github.com/robotframework/robotframework/tree/master/doc/schema/.

Changes to selecting tests with `--suite`, `--test` and `--include`

There are two changes related to selecting tests:

- When using `--test` and `--include` together, tests matching either of them are selected (#4721). Earlier tests need to match both options to be selected.
- When selecting a suite using its parent suite as a prefix like `--suite parent.suite`, the given name must match the full suite name (#4720). Earlier it was enough if the prefix matched the closest parent or parents.

Other backwards incompatible changes

- The default value of the `stdin` argument used with `Process` library keyword has been changed from `subprocess.PIPE` to `None` (#4103). This change ought to avoid processes hanging in some cases. Those who depend on the old behavior need to use `stdin=PIPE` explicitly to enable that.
- When type hints are specified as strings, they must use format `type`, `type[param]`, `type[p1, p2]` or `t1 | t2` (#4711). Using other formats will cause errors taking keywords into use. In practice problems occur if the special characters `[`, `]`, `,` and `|` occur in unexpected places. For example, `arg: "Hello, world!"` will cause an error due to the comma.
- `datetime`, `date` and `timedelta` objects are sent over the Remote interface differently than earlier (#4784). They all used to be converted to strings, but nowadays `datetime` is sent as-is, `date` is converted to `datetime` and sent like that, and `timedelta` is converted to a `float` by using `timedelta.total_seconds()`.
- Argument conversion support with `collections.abc.ByteString` has been removed (#4983). The reason is that `ByteString` is deprecated and will be removed in Python 3.14. It has not been too often needed, but if you happen to use it, you can change `arg: ByteString` to `arg: bytes | bytearray` and the functionality stays exactly the same.

- Paths passed to result file related listener version 3 methods like `output_file` and `log_file` have been changed from strings to `pathlib.Path` objects (#4988). Most of the time both kinds of paths work interchangeably, so this change is unlikely to cause issues. If you need to handle these paths as strings, they can be converted by using `str(path)`.
- `robot.utils.normalize` does not anymore support bytes (#4936).
- Deprecated `accept_plain_values` argument has been removed from the `timestr_to_secs` utility function (#4861).

Deprecations

`[Return]` setting

The `[Return]` setting for specifying the return value from user keywords has been "loudly" deprecated (#4876). It has been "silently" deprecated since Robot Framework 5.0 when the much more versatile `RETURN` setting was introduced (#4078), but now using it will cause a deprecation warning. The plan is to preserve the `[Return]` setting at least until Robot Framework 8.0.

If you have lot of data that uses `[Return]`, the easiest way to update it is using the Robotidy tool that can convert `[Return]` to `RETURN` automatically. If you have data that is executed also with Robot Framework versions that do not support `RETURN`, you can use the `Return From Keyword` keyword instead. That keyword will eventually be deprecated and removed as well, though.

Singular section headers

Using singular section headers like `*** Test Case ***` or `*** Setting ***` nowadays causes a deprecation warning (#4432). They were silently deprecated in Robot Framework 6.0 for reasons explained in issue #4431.

Deprecated attributes in parsing, running and result models

- In the parsing model, `For.variables`, `ForHeader.variables`, `Try.variable` and `ExceptHeader.variable` attributes have been deprecated in favor of the new `assign` attribute (#4708).
- In running and result models, `For.variables` and `TryBranch.variable` have been deprecated in favor of the new `assign` attribute (#4708).
- In the result model, control structures like `FOR` were earlier modeled so that they looked like keywords. Nowadays they are considered totally different objects and their keyword specific attributes `name`, `kwname`, `libname`, `doc`, `args`, `assign`, `tags` and `timeout` have been deprecated (#4846).
- `starttime`, `endtime` and `elapsed` time attributes in the result model have been silently deprecated (#4258). Accessing them does not yet cause a deprecation warning, but users are recommended to use `start_time`, `end_time` and `elapsed_time` attributes that are available since Robot Framework 6.1.
- `kwname`, `libname` and `sourcename` attributes used by the `Keyword` object in the result model have been silently deprecated (#4884). New code should use `name`, `owner` and `source_name` instead.

Other deprecated features

- Using embedded arguments with a variable that has a value not matching custom embedded argument patterns nowadays causes a deprecation warning (#4524). Earlier variables used as embedded arguments were always accepted without validating values.
- Using `FOR IN ZIP` loops with lists having different lengths without explicitly using `mode=SHORTEST` has been deprecated (#4685). The strict mode where lengths must match will be the default mode in the future.

- Various utility functions in the `robot.utils` package that are no longer used by Robot Framework itself, including the whole Python 2/3 compatibility layer, have been deprecated (#4501). If you need some of these utils, you can copy their code to your own tool or library. This change may affect existing libraries and tools in the ecosystem.
- `case_insensitive` and `whitespace_insensitive` arguments used by some Collections and String library keywords have been deprecated in favor of `ignore_case` and `ignore_whitespace`. The new arguments were added for consistency reasons (#4954) and the old arguments will continue to work for the time being.
- Passing time as milliseconds to the `elapsed_time_to_string` utility function has been deprecated (#4862).

Acknowledgements

Robot Framework development is sponsored by the Robot Framework Foundation and its over 60 member organizations. If your organization is using Robot Framework and benefiting from it, consider joining the foundation to support its development as well.

Robot Framework 7.0 team funded by the foundation consists of Pekka Klärck and Janne Härkönen (part time). In addition to work done by them, the community has provided some great contributions:

- Ygor Pontelo added async support to the dynamic and hybrid library APIs (#4803) and fixed a bug with handling async keywords when execution is stopped gracefully (#4808).
- Topi 'top1' Tuulensuu fixed a performance regression when using `Run Keyword` so that the name of the executed keyword contains a variable (#4659).
- Pasi Saikkonen added dark mode to reports and logs (#3725).
- René added return type information to Libdoc's HTML output (#3017), fixed `DotDict` equality comparisons (#4956) and helped finalizing the dark mode support (#3725).
- Robin added type hints to modules that did not yet have them under the public `robot.api` package (#4841).
- Mark Moberts added case-insensitive list and dictionary comparison support to the Collections library (#4343).
- Daniel Biehl enhanced performance of traversing the parsing model using `ModelVisitor` (#4934).

Big thanks to Robot Framework Foundation, to community members listed above, and to everyone else who has tested preview releases, submitted bug reports, proposed enhancements, debugged problems, or otherwise helped with Robot Framework 7.0 development.

See you at RoboCon 2024 either onsite or online!

Pekka Klärck
Robot Framework lead developer

Full list of fixes and enhancements

ID	Type	Priority	Summary
#3296	enhancement	critical	Support keywords and control structures with listener version 3
#3761	enhancement	critical	Native VAR syntax to create variables inside tests and keywords
#4294	enhancement	critical	Drop Python 3.6 and 3.7 support

#4710	enhancement	critical	Support library keywords with both embedded and normal arguments
#4847	enhancement	critical	Support JSON serialization with result model
#4659	bug	high	Performance regression when using `Run Keyword` and keyword name contains a variable
#4921	bug	high	Log levels don't work correctly with `robot:flatten`
#3725	enhancement	high	Support dark theme with report and log
#4258	enhancement	high	Change timestamps from custom strings to `datetime` in result model and to ISO 8601 format in output.xml
#4374	enhancement	high	Support removing tags set globally by using `-tag` syntax with `[Tags]` setting
#4633	enhancement	high	Automatic argument conversion and validation for `Literal`
#4711	enhancement	high	Support type aliases in formats `'list[int]'` and `'int | float'` in argument conversion
#4803	enhancement	high	Async support to dynamic and hybrid library APIs
#4808	bug	medium	Async keywords are not stopped when execution is stopped gracefully
#4859	bug	medium	Parsing errors in reStructuredText files have no source
#4880	bug	medium	Initially empty test fails even if pre-run modifier adds content to it
#4886	bug	medium	`Set Variable If` is slow if it has several conditions
#4898	bug	medium	Resolving special variables can fail with confusing message

#4915	bug	medium	`cached_property` attributes are called when importing library
#4924	bug	medium	WHILE `on_limit` missing from listener v2 attributes
#4926	bug	medium	WHILE and TRY content are not removed with `--removekeywords all`
#4945	bug	medium	`TypedDict` with forward references do not work in argument conversion
#4956	bug	medium	DotDict behaves inconsistent on equality checks. `x == y` != `not x != y` and not `x != y` == `not x == y`
#4964	bug	medium	Variables set using `Set Suite Variable` with `children=True` cannot be properly overwritten
#4980	bug	medium	DateTime library uses deprecated `datetime.utcnow()`
#4999	bug	medium	XML Library: Double namespace during Element To String
#5005	bug	medium	`Log Variables` should not consume iterables
#3017	enhancement	medium	Add return type to Libdoc specs and HTML output
#4103	enhancement	medium	Process: Change the default `stdin` behavior from `subprocess.PIPE` to `None`
#4302	enhancement	medium	Remove `Reserved` library
#4343	enhancement	medium	Collections: Support case-insensitive list and dictionary comparisons
#4375	enhancement	medium	Change token type of `AS` (or `WITH NAME`) used with library imports to `Token.AS`

#4385	enhancement	medium	Change the parsing model object produced by `Test Tags` (and `Force Tags`) to `TestTags`
#4432	enhancement	medium	Loudly deprecate singular section headers
#4501	enhancement	medium	Loudly deprecate old Python 2/3 compatibility layer and other deprecated utils
#4524	enhancement	medium	Loudly deprecate variables used as embedded arguments not matching custom patterns
#4545	enhancement	medium	Support creating assigned variable name based on another variable like `${${var}} = Keyword`
#4667	enhancement	medium	Remove deprecated constructs from Libdoc spec files
#4685	enhancement	medium	Deprecate `SHORTEST` mode being default with `FOR IN ZIP` loops
#4708	enhancement	medium	Use `assing`, not `variable`, with FOR and TRY/EXCEPT model objects when referring to assigned variables
#4720	enhancement	medium	Require `--suite parent.suite` to match the full suite name
#4721	enhancement	medium	Change behavior of `--test` and `--include` so that they are cumulative
#4747	enhancement	medium	Support `[Setup]` with user keywords
#4784	enhancement	medium	Remote: Enhance `datetime`, `date` and `timedelta` conversion
#4841	enhancement	medium	Add typing to all modules under `robot.api`

#4846	enhancement	medium	Result model: Loudly deprecate not needed attributes and remove already deprecated ones
#4872	enhancement	medium	Control continue-on-failure mode by using recursive and non-recursive tags together
#4876	enhancement	medium	Loudly deprecate `[Return]` setting
#4877	enhancement	medium	XML: Support ignoring element order with `Elements Should Be Equal`
#4883	enhancement	medium	Result model: Add `message` to keywords and control structures and remove `doc` from controls
#4884	enhancement	medium	Result model: Enhance storing keyword name
#4896	enhancement	medium	Support `separator=<value>` configuration option with scalar variables in Variables section
#4903	enhancement	medium	Support argument conversion and named arguments with dynamic variable files
#4905	enhancement	medium	Support creating variable name based on another variable like `${${VAR}}` in Variables section
#4910	enhancement	medium	Make listener v3 the default listener API
#4912	enhancement	medium	Parsing model: Move `type` and `tokens` from `_fields` to `_attributes`
#4930	enhancement	medium	BuiltIn: New `Reset Log Level` keyword for resetting the log level to the original value

#4939	enhancement	medium	Parsing model: Rename `Return` to `ReturnSetting` and `ReturnStatement` to `Return`
#4942	enhancement	medium	Add public argument conversion API for libraries and other tools
#4952	enhancement	medium	Collections: Make `ignore_order` and `ignore_keys` recursive
#4960	enhancement	medium	Support integer conversion with strings representing whole number floats like `'1.0'` and `'2e10'`
#4976	enhancement	medium	Support string `SELF` (case-insenstive) when library registers itself as listener
#4979	enhancement	medium	Add `robot.result.TestSuite.to/from_xml` methods
#4982	enhancement	medium	DateTime: Support `datetime.date` as an input format with date related keywords
#4983	enhancement	medium	Type conversion: Remove support for deprecated `ByteString`
#5000	enhancement	medium	Nicer API for setting keyword call arguments programmatically
#4934	---	medium	Enhance performance of visiting parsing model
#4621	bug	low	OperatingSystem library docs have broken link / title
#4798	bug	low	`--removekeywords passed` doesn't remove test setup and teardown
#4867	bug	low	Original order of dictionaries is not preserved when they are pretty printed in log messages
#4870	bug	low	User keyword teardown missing from running model JSON schema

#4904	bug	low	Importing static variable file with arguments does not fail
#4913	bug	low	Trace log level logs arguments twice for embedded arguments
#4927	bug	low	WARN level missing from the log level selector in log.html
#4967	bug	low	Variables are not resolved in keyword name in WUKS error message
#4861	enhancement	low	Remove deprecated `accept_plain_values` from `timestr_to_secs` utility function
#4862	enhancement	low	Deprecate `elapsed_time_to_string` accepting time as milliseconds
#4864	enhancement	low	Process: Make warning about processes hanging if output buffers get full more visible
#4885	enhancement	low	Add `full_name` to replace `longname` to suite and test objects
#4900	enhancement	low	Make keywords and control structures in log look more like original data
#4922	enhancement	low	Change the log level of `Set Log Level` message from INFO to DEBUG
#4933	enhancement	low	Type conversion: Ignore hyphens when matching enum members
#4935	enhancement	low	Use `casefold`, not `lower`, when comparing strings case-insensitively
#4936	enhancement	low	Remove bytes support from `robot.utils.normalize` function
#4954	enhancement	low	Collections and String: Add `ignore_case` as alias for `case_insensitive`

#4958	enhancement	low	Document `robot_running` and `dry_run_active` properties of the BuiltIn library in the User Guide
#4975	enhancement	low	Support `times` and `x` suffixes with `WHILE` limit to make it more compatible with `Wait Until Keyword Succeeds`
#4988	enhancement	low	Change paths passed to listener v3 methods to `pathlib.Path` instances

Altogether 88 issues. View on the issue tracker.