# AppleScript

## Scripting Merlin Project

# AppleScript - as of June 2017

### Note for Merlin Project Express Users

The following functions are only available in Merlin Project.

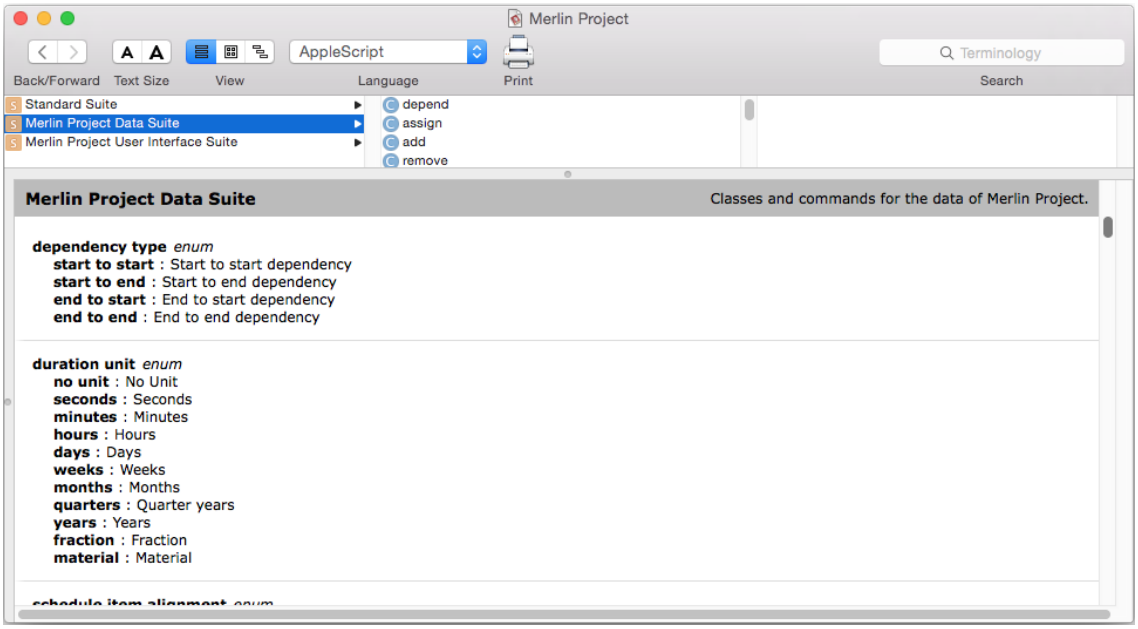
## About this document

With version 3.1 version, Merlin Project supports AppleScript. This document will be of interest for Merlin Project users wanting to write their own scripts or modify existing AppleScript samples.

## General

Merlin Project allows you to script project data and the user interface and thus automate frequently used workflows. Supported are both dialects, AppleScript and Javascript. This documentation is about the AppleScript dialect.

For a full reference of Merlin Project's dictionary just call in Script Editor *File > Open Dictionary* and browse the info provided there.




When scripting Merlin Project you will often use the *Dictionary* as a reference to check the classes and commands Merlin Project understands for accessing data of a currently opened project file, be it its data, be it its user-interface or settings. This document here offers snippets of code showing how to ask or set such data.

You will find more samples of scripting Merlin Project in the Questions & Answers area of our website.

## Project name

To get the name of a project you ask its *title*

```
title of root project of first document
```

For the script to work even if there isn't currently an opened document, you should put the whole code into a try block, thus

```
tell application "Merlin Project"
    try
        first document
        set doc to first document
        display dialog "This project is called  " & title of root project of doc
    end error
end tell
```

## Get selected items

You may simply ask the *selection*. If you know there are many windows or documents opened and you want to define a specific window or document, you may use *selection of first window*, or *selection of second document*. *First* is currently opened and on focus of the user. Following 3 lines are all correct and reference the same data.

```
set myselection to selection
set myselection to first window
set myselection to first document
```

Following sample iterates the items of the selection of current and second window or a second document.

```
tell application "Merlin Project"
    -- get selection of current window
    set myselection to selection
    repeat with act in myselection
        act
    end repeat
```

```
    -- get selection of a second window
    try
        set myselection to selection of second window
        repeat with act in myselection
            act
        end repeat
    on error
        -- only one window open
    end try
    -- get selection of second document
    try
        set myselection to selection of second document
        repeat with act in myselection
            act
        end repeat
    on error
        -- only one document open
    end try
end tell
```

## All activities of a project

To ask sub-activities of an activity you ask its *child activities*. To get all activities of a project, you ask the *root project* for its *child activities*

```
tell application "Merlin Project"
    set allActivities to child activities of root project of first document
    repeat with act in allActivities
        act
    end repeat
end tell
```

### Identify project row

You don't want to iterate on the project row (#0)? You may check the *outline level*. Row #0 has an *outline level* of 0.

## New activity

To create new activities you need to know first in which project you'll have to insert them. As soon as you have a reference to the project, you call *make new child activity*

```
tell application "Merlin Project"
    set proj to root project of first document
    set aAct to make new child activity at end of child activities of root
activity of proj
    set title of aAct to "Do this"
end tell
```

## New resource

To create new resources you need a reference to the project in which you want to insert them. For creation of a new resource you call *make new resource*

```
tell application "Merlin Project"
    set proj to root project of first document
    set aRes to make new resource at end of resources of proj
    set title of aRes to "John Doe"
end tell
```

## Assignments

To assign a resource to a task you need a reference to the resource. Following sample assigns the second resource of your project to all activities currently selected.

```
tell application "Merlin Project"
    set proj to root project of first document
    set myselection to selection
    repeat with act in myselection
        try
            -- if there isn't a second resource in the project
            -- assigning a resource to it fails
            assign second item of resources of proj to act
        end try
    end repeat
end tell
```

Or you know the assignment of a one activity and want to use the same assignments to another? Following sample reads assignments on first item of the selection and sets the assignments to the second item of the selection.

```
tell application "Merlin Project"
    set proj to root project of first document
    set myselection to selection
    set TheResources to resources of assignments of item 1 of myselection
    repeat with aRes in TheResources
        if title of aRes is not "Default Resource" then
            -- assign only if first item was assigned to a real resource
            -- and not just to the "Default Resource"
            assign aRes to item 2 of myselection
        end if
    end repeat
end tell
```

## Dependencies

To define a dependency between two activities you need a reference to the two activities and the kind of dependency. Following types are valid: *start to start*, *start to end*, *end to start* and *end to end*.

Default dependency is *end to start*, so you don't need to explicitly set this dependency type.

```
tell application "Merlin Project"
    set myselection to selection
    set Predecessor to item 1 of myselection
    set Successor to item 2 of myselection
    try
        -- relate activities finish to start
        -- you don't need to define the dependency type
        set TheRelation to depend Successor upon Predecessor
        -- enter a positive lead, or negative lag
        set lag of TheRelation to "2d"
    end try
end tell
```

Following samples show how to define other kinds of dependencies:

```
-- relate activities start to start
set TheRelation to depend Successor upon Predecessor with type start to start
```

```
-- relate activities finish to finish
set TheRelation to depend Successor upon Predecessor with type start to end
```

## Work

Work is a record with following properties: **amount**, **unit**, **estimated**, **resolved amount**, **material unit**.

You may ask for *given work* (which may or may not contain a value). To set it, you need to set its properties.

```
tell application "Merlin Project"
    set myselection to selection
    repeat with act in myselection
        --get work
        given work of act
        -- set work
        set WorkRecord to given work of act
        set WorkUnit to unit of WorkRecord
        set given work of act to {amount:(amount of WorkRecord) * 2,
unit:WorkUnit}
    end repeat
end tell
```

You may ask for *actual work* or set it.

```
tell application "Merlin Project"
    set myselection to selection
    repeat with act in myselection
        --get actual work values
        actual work of act
        remaining work of act
        -- set work
        set WorkRecord to actual work of act
        try
            set WorkUnit to unit of WorkRecord
            set actual work of act to {amount:(amount of WorkRecord) * 2,
unit:WorkUnit}
        end try
    end repeat
end tell
```

## Custom fields

You may attach custom fields onto activities, resources, attachments, assignment rows.

Over applescript you can get and set their value. To reference them, you use *custom field value for* and the custom field title.

Referencing a custom field value is a case sensitive action.

Following sample reads/creates/sets a custom field called "my Field". If the field is not existing it creates a new custom field by this title and sets its value.

```
tell application "Merlin Project"
    set rootProj to root project of first document
    set theRoot to root activity of rootProj
    try
        -- get value
        get custom field value theRoot for "my Field"
    on error
        -- create new custom field
        make new custom field at rootProj with properties {title:"my Field",
extended class:activity}
    end try
    -- set custom field value
    set custom field value theRoot for "my Field" to "Hallo!"
    get custom field value theRoot for "my Field"
end tell
```

## Tags

You may tag Merlin Project activities, resources, attachments, assignment rows.

To set just one tag on an item over AppleScript, you need to reference an existing *tag* by its title, thus:

```
set tag of activity to "foo"
```

If you want to add a *tag* to the maybe existing list of *tags* already enabled onto an item, you'll need to get a reference to the tag first and then use the *add* command, thus:

```
tell application "Merlin Project"
    set rootProj to root project of first document
    -- get all tags and a reference to the "foo" tag
    get tags of rootProj
    set myTag to tags of rootProj whose title is "foo"
```

```
    set myselection to selection
    repeat with act in myselection
        add (item 1 of myTag) to act
    end repeat
end tell
```

To remove only a specific tag from the maybe existing list of tags already enabled, you can use the *remove* command:

```
try
    remove (item 1 of myTag) from act
end try
```

To insert a new *tag* in the list of tags which can be enabled or disabled for the activities of a project, you'll need to create the *tag* reference first and set its title in a second step, thus:

```
tell application "Merlin Project"
    set rootProj to root project of first document
    set fooTag to make new tag at end of tags of rootProj
    set title of fooTag to "foo"
    set myselection to selection
    repeat with act in myselection
        add fooTag to act
    end repeat
end tell
```

## Costs and finance values

Cost is most of the times a read only real value. Properties such as *expected cost*, *planned cost*, *actual cost* may deliver a value. If there are no pricing rates for the resources, base costs or attachment costs, there won't be any cost in the project and the properties will return *missing value*. Following sample asks *expected cost* in the selection.

```
tell application "Merlin Project"
    set myselection to selection
    repeat with act in myselection
        --get cost
        get expected cost of act
    end repeat
end tell
```

Some cost properties can be changed as for example *base costs* or *planned standard rate*.

### Base costs

*Base cost* is a class. Its properties are: **title**, **planned amount**, **actual amount**, **account**, **billable**, **delivering resource**, **holder**, **type**, **status**, **unique id**

You may ask for *planned base costs*. This totals all base costs defined on the item. You may not set it, if there are more than one base cost items already defined.

To ask for each base cost item, you ask for *bases costs of* and iterate the list items.

```
tell application "Merlin Project"
    set myselection to selection
    repeat with act in myselection
        -- get base cost items
        get base costs of act
        repeat with bc in base costs of act
            title of bc
        end repeat
    end repeat
end tell
```

To create a new *base cost* you need to define where this should be attached to, afterwards you can *make new base cost at*. When having a reference on a *base cost*, you can define its *amount*.

Following sample creates a *base cost* and sets its *title*, *planned amount* and *actual amount*.

```
tell application "Merlin Project"
    set myselection to selection
    repeat with act in myselection
        -- set base cost
        set bc to make new base cost at act
        set planned amount of bc to 10000
        set actual amount of bc to 12123
        set title of bc to "base cost over applescript"
    end repeat
end tell
```

### Rate

When defining costs over AppleScript you may want pricing rates for the resources. So you can ask and set *planned standard rate*, *planned overtime rate*, *actual standard rate* or *actual overtime rate*.

*Rate* is a record. Its properties are: **amount**, **denominator unit**, **resolved amount**, **material unit**.

Following sample asks and sets *planned standard rate* of selected resources. It assumes the selection is in a 'Resources' or 'Assignments' view and will fail otherwise. The sample

sets the *planned overtime rate* of a resource based on the value of *planned overtime rate* in case there is any defined.

```
tell application "Merlin Project"
    set myselection to selection
    repeat with resItem in myselection
        get title of resItem
```

```
        -- get standard rate of a selected Resource
        get planned standard rate of resItem
        set rateRecord to planned standard rate of resItem
        try
            get amount of rateRecord
            set planned overtime rate of resItem to {amount:(amount of
rateRecord) * 2, denominator unit:hours}
        end try
    end repeat
end tell
```

### Budget

*Budget* is a record. Its properties are: **amount**, **fractional**, **resolved amount**

When defining budgets one should first request some to be able to change its state later on to *approved*

Following sample asks and sets *requested budget* and *approved budget* amount. It assumes there are expected costs in the project. It will fail otherwise.

```
tell application "Merlin Project"
    set myselection to selection
    repeat with act in myselection
        --get cost
        get expected cost of act
        set costValue to expected cost of act
        -- get budget
        get requested budget of act
        set requested budget of act to costValue
        -- to use the complete requested amount switch the status
        set budget status of act to approved
        -- to use another amount...
        set approved budget of act to {amount:costValue * 0.8}
        -- set the resource requesting or approving the budget
        set budget approving resource of act to resource 2 of root project of
first document
    end repeat
end tell
```

## Attachments

*Attachments* can be linked to the project, a child activity, resource or assignment.

To ask for attachments on an item, you ask *attachments of* and iterate the items list.

To ask for all attachments defined in the project, you need to ask *attachments of root project* thus:

```
tell application "Merlin Project"
    set attachedItems to attachments of root project of first document
    repeat with itemA in attachedItems
        tell itemA
            title
            holder
        end tell
    end repeat
end tell
```

To create an attachment just insert it at the end of attachments and define its properties later on:

```
tell application "Merlin Project"
    set proj to root project of first document
    set aRes to make new resource at end of resources of proj
    set title of aRes to "John Doe"
    -- create a new issue at end of attachments
    set newIssue to make new issue at end of attachments of root activity of
proj
    tell newIssue
        set title to "my Issue"
        set assigned to to last resource of proj
    end tell
end tell
```

## Dates

A *date* is a record with following properties: **timestamp**, **estimated**

Some date properties are read only. Properties like *given start*, *given earliest start*, *actual start* can be asked and set. They throw an applescript error when they don't have a value. Others like *expected start*, *planned start* are read only.

Following sample asks for *given earliest start* of the selection:

```
tell application "Merlin Project"
    set myselection to selection
    repeat with act in myselection
        try
            -- ask for given value
            given earliest start
        on error
            -- unrestricted start, ask for planned start which is a calculated
value
            planned start of act
        end try
    end repeat
end tell
```

To set an editable date property you need to have a date value.

```
tell application "Merlin Project"
    -- to have some dates to use
    set today to (current date)
    tell application "Finder" to set tomorrow to today + (1 * days)
    set proj to root project of first document
    set aAct to make new child activity at end of child activities of root
activity of proj
    tell aAct
        set title to "new task"
        set given earliest start to short date string of today
        set actual start to short date string of tomorrow
    end tell
end tell
```

## Status Date

Should you need to do any calculations referring to the *Status Date*, following sample asks this project setting:

```
tell application "Merlin Project"
    tell root project of first document
        get status date
    end tell
end tell
```

*Status date* always returns a value.

If you need to set the *Status date*, you'll set the *given status date* thus:

```
tell application "Merlin Project"
    tell root project of first document
        get given status date
        if given status date is missing value then set given status date to date
"Thursday, June 28, 2017 at 12:00:00 AM"
    end tell
end tell
```

## Currency symbol

There are two currency related properties of the root project: **currency symbol** and **currency symbol position**.

Following sample asks and sets the *currency symbol* and the *currency symbol position* properties.

```
tell application "Merlin Project"
    -- in order to get some variety
    set currencyList to {"¥", "£", "$", "€"}
    set positionList to {before amount with space, after amount with space}
    set randomNumber to (random number from 1 to count of currencyList)
    set ranCurrency to item randomNumber of currencyList
    tell root project of first document
        -- get values
        get currency symbol
        get currency symbol position
        -- set values
        set currency symbol to ranCurrency
        if randomNumber < 3 then
            set currency symbol position to item 1 of positionList
        else
            set currency symbol position to item 2 of positionList
        end if
    end tell
end tell
```

## Unit for expected durations

When calculating with duration, you would need to make sure all values are set to the same unit. You may set the calculation unit over the user interface in *Settings > General > Duration > Calculation Unit*, but if this is not the case you need to set it before starting any calculations.

To ask for the unit, you call *derived durations unit of root project of first document*. If it is not set, it returns *missing value*.

Following sample checks the unit and sets it based on the user's choice.

```
tell application "Merlin Project"
    get derived durations unit of root project of first document
    if derived durations unit of root project of first document is missing value
then
        set myDurations to {days, weeks, months, years}
```

```
        try
            set selectedUnit to item 1 of (choose from list myDurations)
            set derived durations unit of root project of first document to
selectedUnit
        end try
    end if
    set theUnit to derived durations unit of root project of first document
    display dialog "Calculation Unit: " & theUnit buttons {"Ok"} default button
{"Ok"}
end tell
```

## Calendars

To get the base calendar, you ask *every calendar* of the project and get the first item of the list:

```
tell application "Merlin Project"
    get title of item 1 of every calendar of root project of document 1
end tell
```

### Exception rules

To get the *exception rules* is easy as soon as you have a reference to the base calendar.

```
tell application "Merlin Project"
    set proCal to item 1 of every calendar of root project of document 1
    get exception rules of proCal
    repeat with aRule in exception rules of proCal
        title of aRule
        start date of aRule
    end repeat
end tell
```

Following sample creates a new *exception rule*

```
tell application "Merlin Project"
    set today to current date
    tell application "Finder" to set tomorrow to today + (3 * days)
    set baseCal to item 1 of every calendar of root project of document 1
    set excRule to make new exception rule at baseCal with properties {start
date:today, end date:tomorrow}
end tell
```

### Weekday rule

You may ask the regular working days of the base calendar. They are items of the *weekday rules* of the base calendar.

Following sample asks the regular working days.

```
tell application "Merlin Project"
    set proCal to item 1 of every calendar of root project of document 1
    repeat with aRule in weekday rules of proCal
        tell aRule
            if time intervals is not missing value then weekday
        end tell
    end repeat
end tell
```

## User interface

Applescript is basically for manipulating data, so you usually use it to modify data or to create new data based on existing information to automate actions which wouldn't be otherwise possible or need many clicks to achieve. Should you however need to check or manipulate the current view of your Merlin Project file, Merlin Project also offers a 'Merlin Project User Interface Suite' of AppleScript commands and classes.

Every *window* contains a *configuration* with *panes*. First pane is the main view, second is the Details view which is sometimes closed.

## Views

### Current view type

To check the currently shown view type of your project file of your main window:

```
tell application "Merlin Project"
    get active view type of pane 1 of configuration of window 1
end tell
```

You may ask or set for the active view type of pane 2 even if this is currently closed:

```
tell application "Merlin Project"
    get active view type of pane 2 of configuration of window 1
    set active view type of pane 2 of configuration of window 1 to net plan view
type
end tell
```

Changing the active view type of a currently collapsed pane will make Merlin Project to show it.

### Active view

To check and even set the current view of a window:

```
tell application "Merlin Project"
    tell window 1
        tell configuration
            tell pane 1
                get title of active view
                if active view type is not net plan view type then set active
view type to net plan view type
```

```
                -- views contain a list of all view configurations
                -- in the curent active view
                get views
                -- to set the view to "Mind Map"
                set active view to item 1 of (views whose title is "Mind Map")
            end tell
        end tell
    end tell
end tell
```

### Details view

To open or close the Details view you need to address the *collapse state* of *views splitter state*.

```
tell application "Merlin Project"
    tell window 1
        tell configuration
            get views splitter state
            set views splitter state to {collapse state:second view collapsed}
            set views splitter state to {collapse state:both views visible}
        end tell
    end tell
end tell
```

Usually the Details view will appear in the bottom area of your window. Over the user interface a Merlin Project user can switch it however to the appear on the right-hand side.

To set the orientation of the Details view over Applescript, you also address the *orientation* of *views splitter state*:

```
tell application "Merlin Project"
    set views splitter state of configuration of window 1 to {orientation:split
vertical, collapse state:both views visible}
end tell
```

### Inspector

You may open or close the inspector over Applescript. To do so you need to address the *inspector splitter state*.

```
tell application "Merlin Project"
    set inspector splitter state of configuration of window 1 to {collapse
state:both views visible}
end tell
```

## Window frame

If you need to ask or set the size and position of the current Merlin Project window you need to access the *frame* of *configuration* of the window:

```
tell application "Merlin Project"
    get frame of configuration of window 1
    set frame of configuration of window 1 to {x:250.0, y:500.0, width:1000.0,
height:780.0}
end tell
```

## More about Applescript

More Information about Applescript can be found in the Mac Developer Library on Apple's developer server.