Chapter 8: Mastering Link Dependency and Constraints

Every mission into space is a *project*, due to its unique objectives and challenges. Imagine that you are the Project Manager of such a space mission. Your schedule will be highly realistic and reflect the dynamics of the mission's execution. You will be required to cater to special sequencing requirements and hard dates on the timeline. Let's consider some examples of such tasks: only after a new solar battery has been plugged in should the older one be removed. Otherwise, the power could go down and jeopardize the entire mission. This is an example of a special task sequencing requirement. Similarly, within a certain precise date, certain mission experiments should be terminated, and other new ones initiated. This is an example of a special date constraint.

Microsoft Project provides tools and features to reflect such realism in your schedule, as we will explore in depth in this chapter. But the cost of high realism is complexity (of the schedule), and the project will incur some extra cost to retain effectiveness. You should use such schedule customization only on special occasions. We will discuss some best practices to counterbalance realism with effectiveness.

In this chapter, we're going to cover the following main topics:

- Customizing task dependencies and constraints
- Hands-on practice using different task dependencies with a running project
- Incorporating date constraints within your schedule
- Using dependency scheduling best practices with Project
- Let's get started!

Scheduling – a quick refresher

We can define a schedule as *a chronologically ordered listing of tasks or events*. The following screenshot shows an illustration of a schedule:

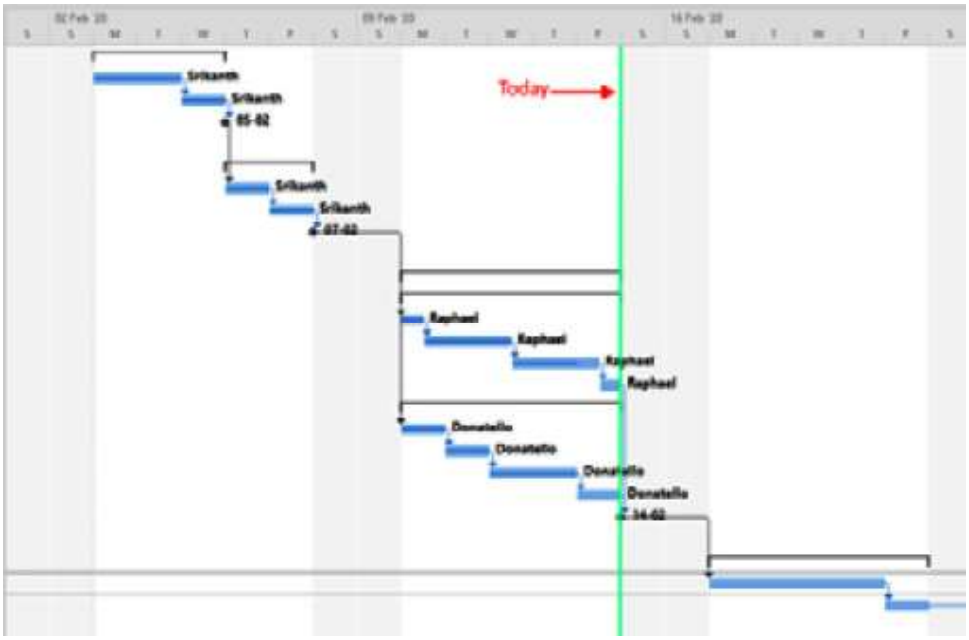

Figure 8.1 – A schedule represented as a Gantt chart

The timeline represents the canvas on which you paint the picture of your project. The single green vertical line represents today's date, and on every new day, this line marches one step ahead on the timeline, from start to finish with regard to the project. Here is a close-up view of the timeline:

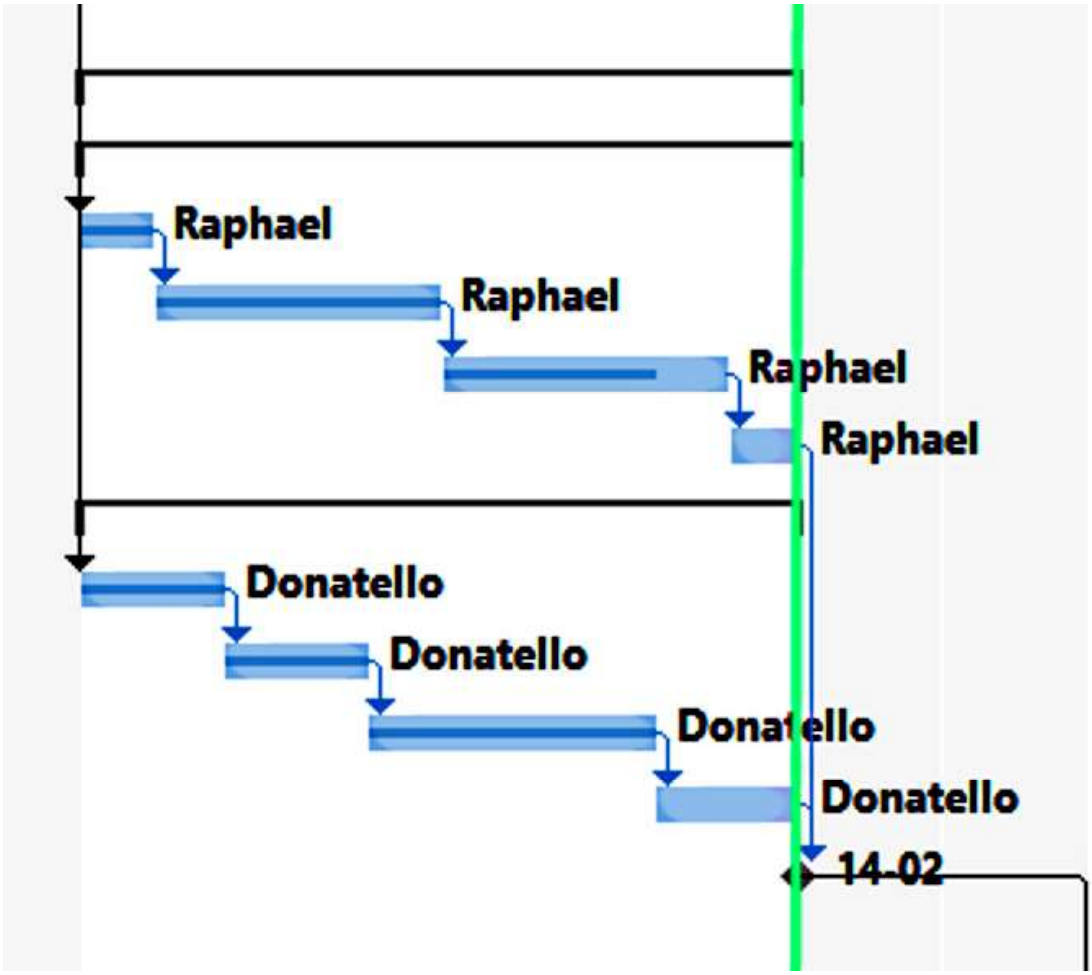

Figure 8.2 – Close-up of a timeline, represented as a Gantt chart

The thick horizontal bars represent tasks. The arrow lines connecting the tasks are their links (or dependencies). They perform another critical functionality – they show the direction of execution.

All the links in the preceding screenshot are of the same type, where the end of one task neatly triggers the start of the next task in line. While such simple links are the most common type, in real-life scenarios, you will come across tasks that have more complicated dependencies between them. We will explore all the different task dependencies in the next section.

Task dependencies and constraints

There are two major aspects that affect sequencing decisions in a schedule. These are as follows:

- The inherent relationship of dependencies between the project tasks
- The inherited set of constraints on the project

We will now investigate how these dependencies and constraints can be realistically reflected in our schedule using Project features.

Task relationships

As the famous saying goes, *no man is an island*, and similarly, within a schedule, no task should exist in isolation. Consider the following diagram, which denotes a relationship between two tasks:

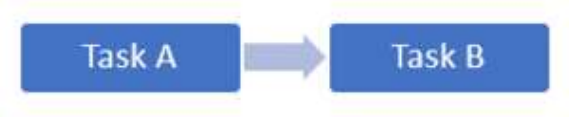

Figure 8.3 – Dependency relationship between two tasks

Tasks in a project derive their meaning only in relation to other tasks. For example, if you are building a house, it is mandatory to build the foundation before your build the walls, and only after you build the walls can the roof be constructed. The universal law of gravity mandates these particular dependencies. Similarly, in software development, an application can only be deployed after you have created it. Dependencies are often rooted in common sense and business logic.

In each of these examples, one task *drives* the other task. Based upon this observation, we can classify the tasks on either side of this relationship as follows:

- **Predecessor**: The controlling task that drives another task. In the preceding diagram, this is Task A.
- **Successor**: The task that is driven. In the preceding diagram, this is Task B.

The process of connecting the predecessor and successor is known as **linking** (or *sequencing* in some texts you may come across).

There are some important points for you to observe from this discussion:

- The definitions of predecessor and successor *only* have relevance in relation to the existence of a relationship between said tasks. These sequences of such predecessors are known as **driving predecessors**. We will encounter this feature in Project later in this chapter.
- A predecessor task in one relationship could be a successor task in a different relationship and vice versa. Terminology is only relevant to the relationship in question.
- If a task is a predecessor, it does not automatically mean that the task occurs earlier than the successor on the schedule's timeline! Rather, it means that the successor is driven (*activated* or *put in motion*) by the predecessor. We will see examples in the next section.
- A predecessor task will drive either the start or finish date of the successor task. Other date parameters of the successor will be driven subsequently by the task's duration.

Pitfall

Do not have disjointed sections in a schedule. Disjointed sections will lose all the power of automatic scheduling when they're disconnected from the rest of the project. Refresh your old network theory class notes on this topic for a refresher. At least connect disjointed sections to the start and end nodes of a schedule. A simple example can be found in Chapter 7, Tasks – under the Microscope; please refer to Figure 7.16. A detailed discussion of several sequencing error patterns, including the disjointed sequencing of tasks, along with more screenshot examples, are provided in Chapter 18, Reviewing Projects and Creating Templates for Success, in the Sequencing error patterns section.

Four types of task dependencies

Continuing our discussion from earlier, we can extrapolate that there are four fundamental types of task dependencies. Let's understand each here.

Finish-to-Start (FS)

This is by far the most common type of dependency, and we have been using this dependency in all the examples and projects so far in this book. Refer to the following screenshot:

Figure 8.4 – Finish-to-Start (FS) dependency

In FS dependency, the predecessor is executed earlier than the successor. The predecessor completing triggers the start of the successor. For example, a project has to be approved (predecessor) before it is launched (successor).

Start-to-Start (SS)

The start of the predecessor task drives the start of the successor task. Compared to the previous example, the predecessor doesn't need to be completed before you launch the successor task. Refer to the following screenshot:

Figure 8.5 – Start-to-Start (SS) dependency

The obvious advantage of using this dependency relationship is that you can overlap the tasks to an extent, thereby potentially reducing project duration. For example, if your organization has several departments, often, it is possible to work with several departments at the same time.

Finish-to-Finish (FF)

The FF dependency is the symmetric opposite of the previous SS dependency. The successor task's completion is driven by the completion of the predecessor task:

Figure 8.6 – Finish-to-Finish (FF) dependency

Here, the floor's electrical layout planning is completed as soon as the floor's seating arrangement plan is completed for an office.

Once again, the possibility of overlapping occurring makes the FF relationship attractive, just like SS.

Pitfall

The task-overlapping nature of SS and FF can cause overallocation if both the associated tasks are associated with the same resource. So far in this book, we have learned about two techniques we can use to combat such overallocation: assign to different resources or

reduce the Resource Units assigned to the task (that is, say the resource only works for 4 hours/day on each task, thereby increasing the duration of the task).

Start-to-Finish (SF)

Now, we come to the strangest, least used, and least understood of all the relationships – the SF dependency. In the SF dependency, the start of the predecessor triggers the finish of the successor. This is the reverse of the most common FS dependency:

Figure 8.7 – Start-to-Finish (SF) dependency

Zero downtime is often achieved by the SF dependency. In the preceding example, consider the case of a running account of payable bills. As soon as we initiate the payables calculation for the existing infrastructure (predecessor), a new account should already be in place for the customer to track newly running payables (successor).

This type of SF dependency is rare, and you must carefully inspect your logic if you encounter this relationship in your project.

Tip

If you encounter and use any dependency other than the default FS in your project, be sure to document the logic. This will be helpful to you for future reviews, to the team members who will execute the task, and to any other project manager who will want to use your plan.

Now, let's look at **constraints**, which are the limitations involved in the project.

Understanding constraints

In *Chapter 1, Project Management - the Essential Primer,* we discussed how every project operates under the triple constraints of scope, cost, and time, just like an iron triangle. In the planning stage of the project, when you are grappling with task sequencing, these constraints translate as follows:

1. **Limited resources**: The **Project Manager** (**PM**) can only overlap a limited set of tasks to reduce overall project duration, and this limit is determined closely by the number of resources in the Project team. Exceeding this limit results in overallocation and should be avoided.

   ### *Important note*

   The FS dependency may often be used for no other reason than because the resource will work through their set of tasks one after another, even when there are no other explicit dependencies between the tasks. In fact, we have used task linking like this in all our project examples so far in this book.

2. **Hard dates**: When we use automatic scheduling, we do not set either the start or finish date of the sequenced task. However, the PM will encounter tasks that are tied to calendar dates. Such dates are known as *hard dates* for the task. Date constraints will be explained in the *Working with date constraints* section.

Next, we will learn how task dependency selection is an important aspect of project scheduling.

Selecting the correct task dependency

The vast majority of task dependencies you will encounter will be **Finish-to-Start** (**FS**). However, when your project presents special sequencing requirements, the question is, *How do you select the correct dependency?* A handy technique we can use to identify the dependency is a two-step process, as follows:

1. First, identify which task drives the relationship. By doing this, you will have established the predecessor-successor aspect of the relationship.

2. Next, identify whether the *start* or *finish* date is driven. By doing this, you will have identified the *Start-to* or *Finish-to* aspect of the relationship.

With these two initial steps, you will have enough clarity to correctly identify the exact task dependency. It is important for you to get all special dependencies reviewed by your team members, subject matter experts, and all other pertinent stakeholders. To cement all our learning so far, let's work on another hands-on project.