

# 8

## Services, WorkManager, and Notifications

### Activity 8.01 – A reminder to drink water

#### Solution

The following steps will help you complete the activity:

1. Create an empty activity project and name your app My Water Tracker. Make sure that its package name is `com.example.mywatertracker`.
2. Add the required work manager dependency to your project:

```
androidx-work-runtime = {  
    group = "androidx.work",  
    name = "work-runtime",  
    version.ref = "androidxWorkRuntime"  
}
```

3. Add the data sync foreground service and post notifications permissions to your `AndroidManifest.xml` file:

```
<uses-permission  
    android:name="android.permission.FOREGROUND_SERVICE_DATA_SYNC" />  
<uses-permission  
    android:name="android.permission.POST_NOTIFICATIONS" />
```

4. Add the `SystemForegroundService` service to the manifest:

```
<service
    android:name="androidx.work.impl.foreground.SystemForegroundService"
    android:foregroundServiceType="dataSync"
    tools:node="merge" />
```

5. Create a new `CoroutineWorker` class named `WaterConsumptionWorker`:

```
class WaterConsumptionWorker(
    context: Context,
    parameters: WorkerParameters
) : CoroutineWorker(context, parameters) {
    override suspend fun doWork(): Result {
        return Result.success()
    }
}
```

6. Define a static variable in your worker to track the water level. Use `AtomicInteger` to ensure thread safety. Note that to keep the needed precision of fractions of milliliters, this variable should store the water level in microliters. A microliter is 1,000th of a milliliter. In other words, 1,000 microliters is equal to 1 milliliter. Add a function called `increaseBalance` that takes a float value in milliliters and adds it to the balance:

```
class WaterConsumptionWorker(
    context: Context,
    parameters: WorkerParameters
) : CoroutineWorker(context, parameters) {
    ...
    companion object {
        private val waterBalance: AtomicInteger =
            AtomicInteger(0)
        fun increaseBalance(amountInMilliliters: Float) {
            waterBalance.addAndGet((amountInMilliliters *
                1000f).toInt())
        }
    }
}
```

7. Define constants for a notification ID and for an extra Data work data key:

```
class WaterConsumptionWorker(
    context: Context,
    parameters: WorkerParameters
) : CoroutineWorker(context, parameters) {
    ...
    companion object {
        private const val NOTIFICATION_ID = 0x3A7E12
        private val waterBalance: AtomicInteger = AtomicInteger(0)
        const val DATA_KEY_WATER_BALANCE = "WaterLevel"
        fun increaseBalance(amountInMilliliters: Float) { ... }
    }
}
```

8. Set up the creation of the notification in the worker:

```
private fun getNotification() = getNotificationBuilder()
    .setContentText(String.format("Water level is %.3f",
        waterBalanceInMilliliters()))
    .build()

private fun getNotificationBuilder(): NotificationCompat.Builder {
    val pendingIntent = getPendingIntent()
    return NotificationCompat.Builder(
        applicationContext, channelId
    ).setTitle("Water balance")
        .setSmallIcon(
            R.drawable.ic_launcher_foreground
        ).setContentIntent(pendingIntent)
        .setTicker(
            "Tracking water balance"
        ).setOngoing(true)
        .setForegroundServiceBehavior(
            FOREGROUND_SERVICE_IMMEDIATE
        ).setOnlyAlertOnce(true)
}

private fun createNotificationChannel(): String = if (
```

```

        Build.VERSION.SDK_INT >= VERSION_CODES.O
    ) {
        val newChannelId = "WaterTracking"
        val channelName = applicationContext.getString(
            R.string.channel_name_water_tracking
        )
        val channel = NotificationChannel(newChannelId, channelName,
            IMPORTANCE_HIGH)
        val service = requireNotNull(
            ContextCompat.getSystemService(applicationContext,
                NotificationManager::class.java)
        )
        service.createNotificationChannel(channel)
        newChannelId
    } else {
        ""
    }
}

```

9. Create a `ForegroundInfo` instance and return it from the `getForegroundInfo()` function of the worker:

```

override suspend fun getForegroundInfo(): ForegroundInfo {
    val notification = getNotification()

    return if (Build.VERSION.SDK_INT >= VERSION_CODES.Q) {
        ForegroundInfo(
            NOTIFICATION_ID, notification,
            FOREGROUND_SERVICE_TYPE_DATA_SYNC
        )
    } else {
        ForegroundInfo(NOTIFICATION_ID, notification)
    }
}

```

10. Make the `Worker` object a foreground worker by calling `setForeground(ForegroundInfo)` from the `doWork()` function:

```

override suspend fun doWork(): Result {
    setForeground(getForegroundInfo())
    return Result.success()
}

```

11. In MainActivity, add the code to request the notification permissions if they are required:

```
var showNotificationPermissionRationale by remember {
    mutableStateOf(false)
}
var waterConsumptionWorkId: UUID? by remember {
    mutableStateOf(null)
}
val requestPermissionLauncher = rememberLauncherForActivityResult(
    contract = RequestPermission(),
    onResult = { hasPermission ->
        if (hasPermission) {
            waterConsumptionWorkId = startConsumption()
        } else {
            showNotificationPermissionRationale =
                shouldShowNotificationPermissionRationale()
        }
    }
)
fun requestNotificationsPermission() {
    requestPermissionLauncher.launch(POST_NOTIFICATIONS)
}
if (showNotificationPermissionRationale) {
    PermissionRationaleDialog(
        onConfirmClick = {
            requestNotificationsPermission()
        },
        onDismissRequest = {
            showNotificationPermissionRationale = false
        }
    )
}
@Composable
@RequiresApi(VERSION_CODES.TIRAMISU)
fun PermissionRationaleDialog(
    onConfirmClick: () -> Unit,
    onDismissRequest: () -> Unit
) {
```

```

        AlertDialog(
            title = {
                Text(text = stringResource(R.string.permission_
rationale_dialog_title))
            },
            text = {
                Text(text = stringResource(R.string.permission_
rationale_dialog_text))
            },
            confirmButton = {
                TextButton(
                    onClick = {
                        onConfirmClick()
                    }
                ) {
                    Text(stringResource(
                        R.string.permission_rationale_dialog_confirm_
button_label
                    ))
                }
            },
            onDismissRequest = {
                onDismissRequest()
            }
        )
    }

```

12. Add the functionality to start the WaterConsumptionWorker worker:

```

private fun startConsumption(): UUID {
    val workRequest =
        OneTimeWorkRequestBuilder<WaterConsumptionWorker>()
            .setExpedited(RUN_AS_NON_EXPEDITED_WORK_REQUEST)
            .build()
    workManager.enqueue(workRequest)
    return workRequest.id
}

```

13. Add a button labeled **Start** to start tracking the user's water balance. Make sure to check the permissions and execute the `startConsumption` function if no permission is needed and if permission was granted. Request the permission if required:

```

    TextButton(onClick = {
        if (isNotificationPermitted()) {
            waterConsumptionWorkId = startConsumption()
        } else {
            requestNotificationsPermission()
        }
    }) {
        Text(text = stringResource(R.string.button_label_start))
    }

```

14. Add a label and a mutable state variable for storing and presenting the water level:

```

    var waterBalance by remember {
        mutableFloatStateOf(0f)
    }
    Text(text = stringResource(R.string.label_water_balance,
        waterBalance))

```

15. Add code to track the progress of work and update the water level on the UI:

```

suspend fun collectWorkInfo(requestId: UUID) {
    workManager.getWorkInfoByIdFlow(requestId)
        .collect { info ->
            when {
                info == null -> return@collect
                info.state == WorkInfo.State.CANCELLED -> {
                    Toast.makeText(
                        this, "Monitoring stopped.", LENGTH_SHORT
                    ).show()
                }
                info.progress != Data.EMPTY -> {
                    waterBalance =
                        info.progress.getFloat(DATA_KEY_WATER_
BALANCE, waterBalance)
                }
            }
        }
}

```

```

        }
    }

    LaunchedEffect(key1 = waterConsumptionWorkId) {
        waterConsumptionWorkId?.let { collectWorkInfo(it) }
    }

```

16. In the worker, set the water balance to decrease by 0.144 ml every 5 seconds:

```

override suspend fun doWork(): Result {
    setForeground(getForegroundInfo())
    while (true) {
        waterBalance.getAndAdd(-144)
        delay(5000)
    }
    return Result.success()
}

```

17. Update the progress and the notification as the water balance is updated:

```

override suspend fun doWork(): Result {
    setForeground(getForegroundInfo())
    while (true) {
        waterBalance.getAndAdd(-144)
        setProgress(workDataOf(DATA_KEY_WATER_BALANCE to
            waterBalanceInMilliliters()))
        updateNotification()
        delay(5000)
    }
    return Result.success()
}

```

18. Add a button to the main activity layout with a **Drank a glass of water** label. When the user taps the button, increase the value of the static AtomicInteger variable storing the water balance by 250 ml:

```

TextButton(onClick = {
    WaterConsumptionWorker.increaseBalance(amountInMilliliters =
        250f)
})

```



```
    }) {  
        Text(text = stringResource(R.string.button_label_drunk_glass))  
    }  
}
```

19. Add a third button, **Stop**, to the app. Have it cancel any ongoing work when tapped:

```
    TextButton(onClick = {  
        waterConsumptionWorkId?.let(workManager::cancelWorkById)  
    }) {  
        Text(text = stringResource(R.string.button_label_stop))  
    }  
}
```



#### Important note

The solution to this activity can be found at <https://github.com/PacktPublishing/How-to-Build-Android-Apps-with-Kotlin-Third-Edition/tree/main/Chapter08/Activity08.01>.

