droidcon academy

Updating WidgetState

Piotr Prus

@piotr_prus

in/piotrprus

Section Overview

- Update state of widget(data)
- Trigger the recomposition of the widget
- Initial configuration of the widget
- Using work manager for periodic updates
- Referencing the widget in other context

droidcon academy

Update state of widget(data)

GlanceAppWidgetState.kt

```
suspend fun updateAppWidgetState(
    context: Context,
    glanceId: GlanceId,
    updateState: suspend (MutablePreferences) → Unit,
) {
    updateAppWidgetState(context, PreferencesGlanceStateDefinition, glanceId) {
        it.toMutablePreferences().apply {
            updateState(this)
        }
    }
}
```

droidcon academy

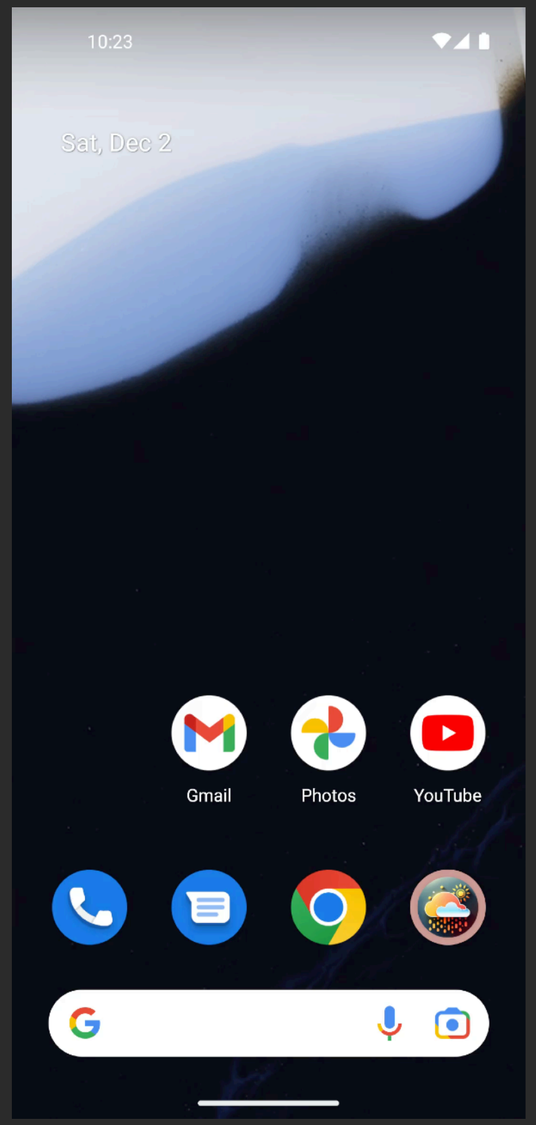
Trigger the recomposition of the widget

GlanceAppWidget.kt

```
suspend fun update(
    context: Context,
    id: GlanceId
) {
    require(id is AppWidgetId) { "Invalid Glance ID" }
    update(context, id.appWidgetId)
}
```

Initial configuration of the widget

Configuration activity




```
<activity
    android:name=".widget.ConfigurationActivity"
    android:exported="false">
    <intent-filter>
        <action android:name=
      "android.appwidget.action.APPWIDGET_CONFIGURE" />
    </intent-filter>
</activity>
```

```
data class PlaceItem(
    val name: String,
    val latitude: Double,
    val longitude: Double
)
```

droidcon academy

Item click

Update widget with location and address

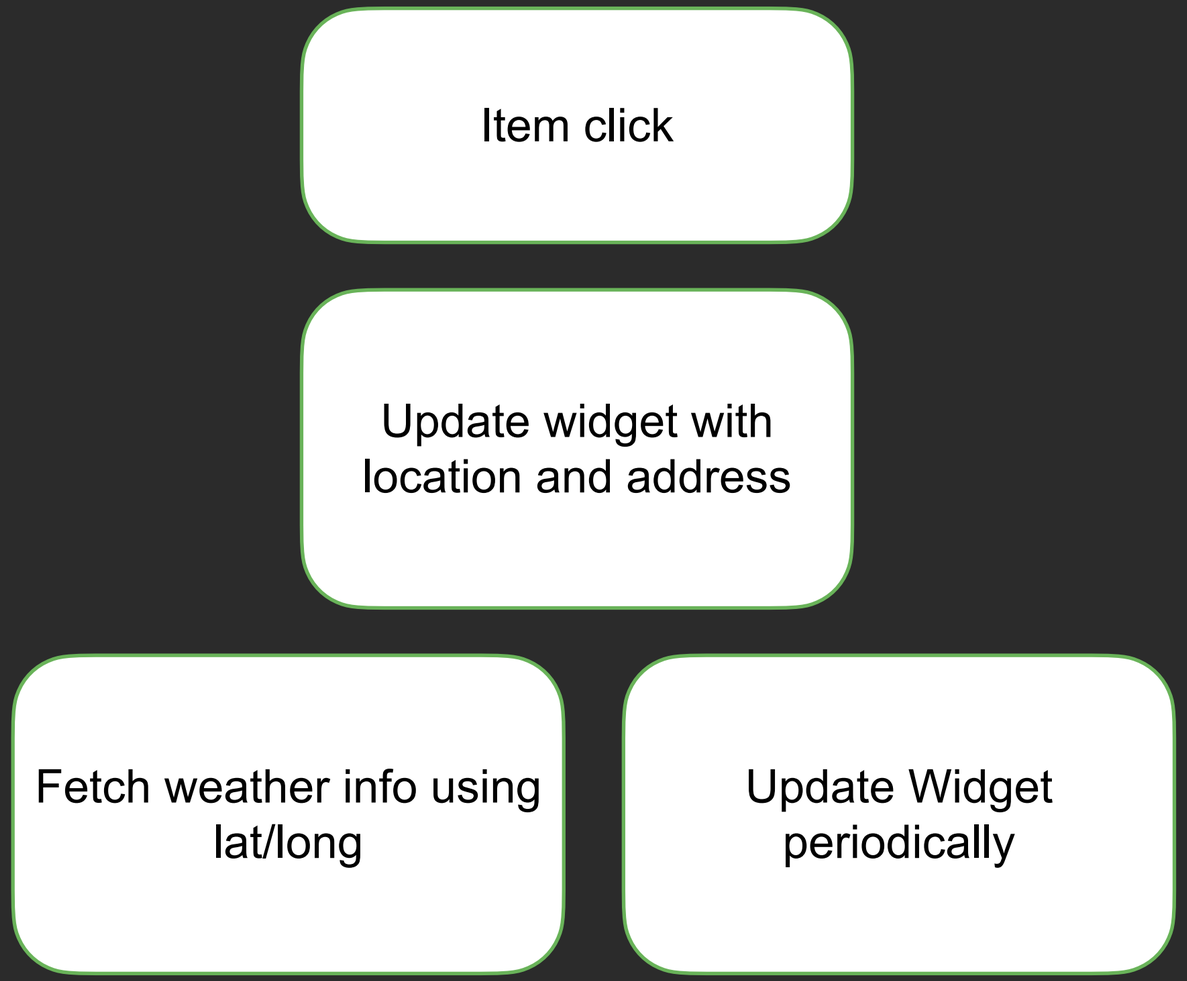
droidcon academy

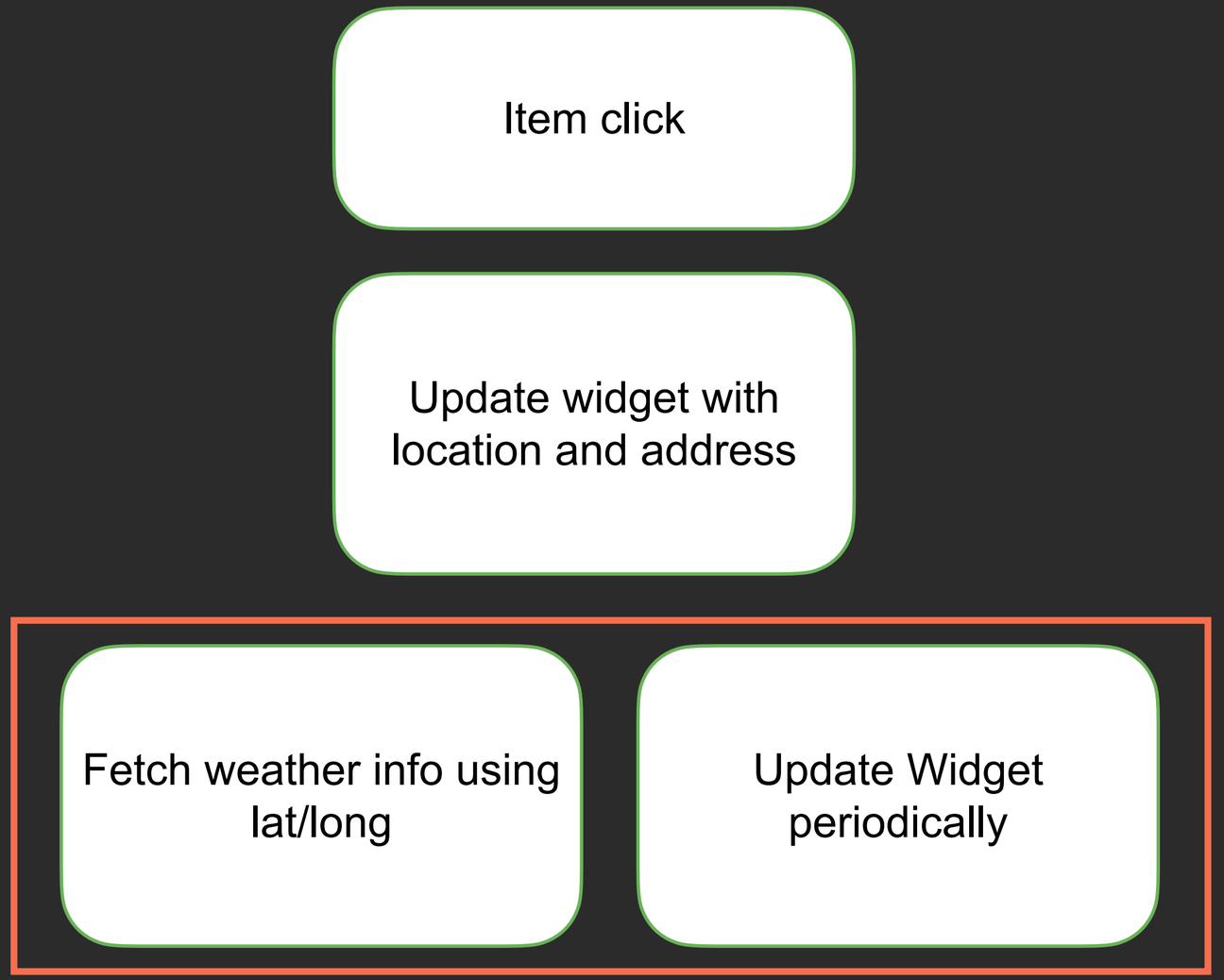
Item click

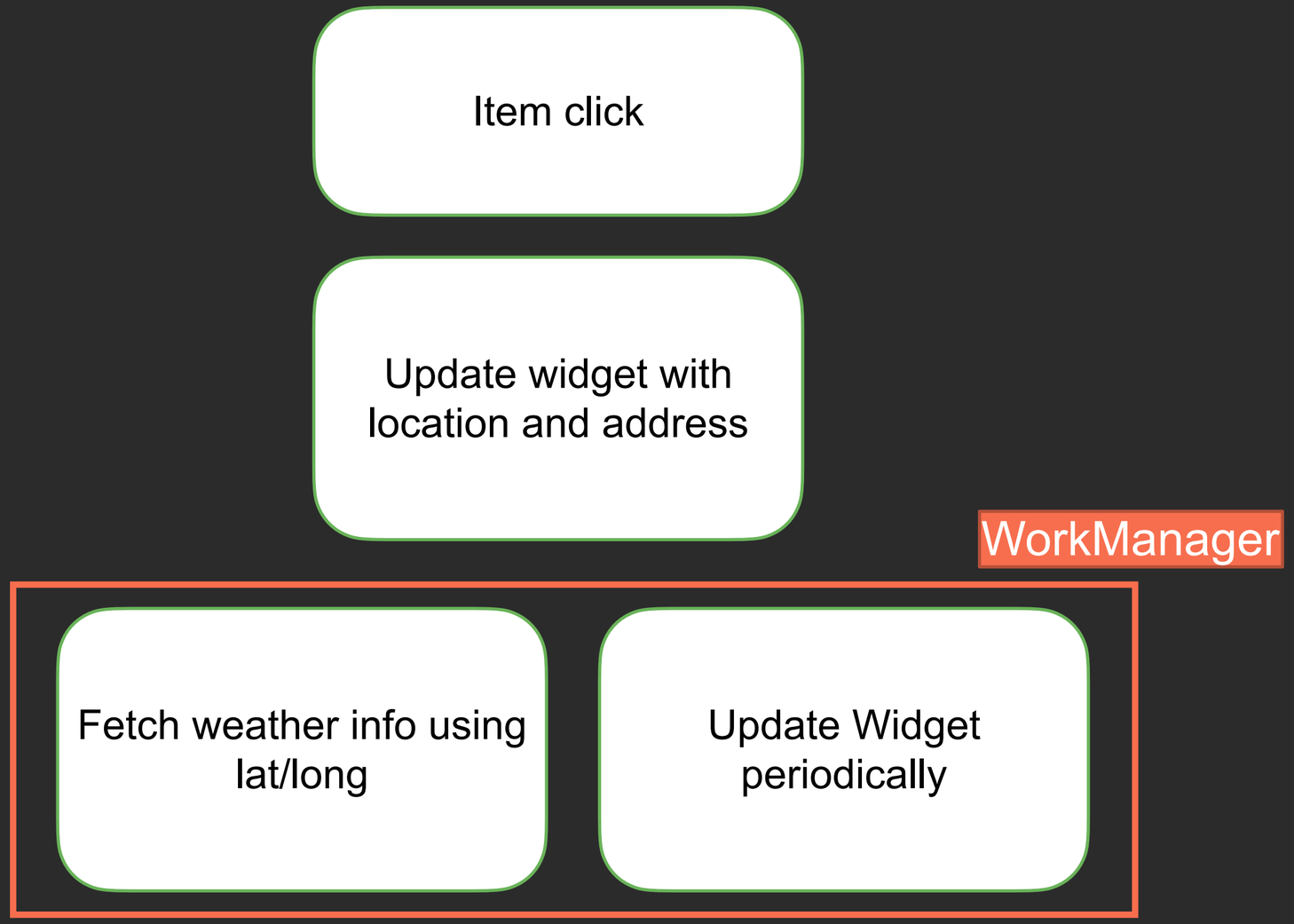
Update widget with location and address

Fetch weather info using lat/long

droidcon academy


Item click
Update widget with location and address
Fetch weather info using lat/long
Update Widget periodically


Item click
Update widget with location and address
Fetch weather info using lat/long
Update Widget periodically


Item click
Update widget with location and address
WorkManager
Fetch weather info using lat/long
Update Widget periodically

Using work manager for periodic updates

WeatherWidget.kt

```
fun Context.startWeatherWorker(latitude: Double, longitude: Double) {
    val networkConstraint =
        Constraints.Builder().setRequiredNetworkType(NetworkType.CONNECTED).build()
    val request = PeriodicWorkRequest
        .Builder(WeatherWidgetWorker::class.java, 15, TimeUnit.MINUTES)
        .setBackoffCriteria(BackoffPolicy.EXPONENTIAL, 5000L, TimeUnit.MILLISECONDS)
        .setInputData(
            WeatherWidgetWorker.buildData(latitude, longitude)
        )
        .setConstraints(networkConstraint)
        .build()
    val uniqueTag = WeatherWidget.UNIQUE_WORK_TAG + "_$latitude" + "_$longitude"
    WorkManager.getInstance(this)
        .enqueueUniquePeriodicWork(
            uniqueTag,
            ExistingPeriodicWorkPolicy.REPLACE,
            request
        )
}
```

WeatherWidget.kt

```
fun Context.startWeatherWorker(latitude: Double, longitude: Double) {
    val networkConstraint =
        Constraints.Builder().setRequiredNetworkType(NetworkType.CONNECTED).build()
    val request = PeriodicWorkRequest
        .Builder(WeatherWidgetWorker::class.java, 15, TimeUnit.MINUTES)
        .setBackoffCriteria(BackoffPolicy.EXPONENTIAL, 5000L, TimeUnit.MILLISECONDS)
        .setInputData(
            WeatherWidgetWorker.buildData(latitude, longitude)
        )
        .setConstraints(networkConstraint)
        .build()
    val uniqueTag = WeatherWidget.UNIQUE_WORK_TAG + "_$latitude" + "_$longitude"
    WorkManager.getInstance(this)
        .enqueueUniquePeriodicWork(
            uniqueTag,
            ExistingPeriodicWorkPolicy.REPLACE,
            request
        )
}
```

WeatherWidget.kt

```
fun Context.startWeatherWorker(latitude: Double, longitude: Double) {
    val networkConstraint =
        Constraints.Builder().setRequiredNetworkType(NetworkType.CONNECTED).build()
    val request = PeriodicWorkRequest
        .Builder(WeatherWidgetWorker::class.java, 15, TimeUnit.MINUTES)
        .setBackoffCriteria(BackoffPolicy.EXPONENTIAL, 5000L, TimeUnit.MILLISECONDS)
        .setInputData(
            WeatherWidgetWorker.buildData(latitude, longitude)
        )
        .setConstraints(networkConstraint)
        .build()
    val uniqueTag = WeatherWidget.UNIQUE_WORK_TAG + "_$latitude" + "_$longitude"
    WorkManager.getInstance(this)
        .enqueueUniquePeriodicWork(
            uniqueTag,
            ExistingPeriodicWorkPolicy.REPLACE,
            request
        )
}
```

WeatherWidget.kt

```
fun Context.startWeatherWorker(latitude: Double, longitude: Double) {
    val networkConstraint =
        Constraints.Builder().setRequiredNetworkType(NetworkType.CONNECTED).build()
    val request = PeriodicWorkRequest
        .Builder(WeatherWidgetWorker::class.java, 15, TimeUnit.MINUTES)
        .setBackoffCriteria(BackoffPolicy.EXPONENTIAL, 5000L, TimeUnit.MILLISECONDS)
        .setInputData(
            WeatherWidgetWorker.buildData(latitude, longitude)
        )
        .setConstraints(networkConstraint)
        .build()
    val uniqueTag = WeatherWidget.UNIQUE_WORK_TAG + "_$latitude" + "_$longitude"
    WorkManager.getInstance(this)
        .enqueueUniquePeriodicWork(
            uniqueTag,
            ExistingPeriodicWorkPolicy.REPLACE,
            request
        )
}
```

WeatherWidget.kt

```
fun Context.startWeatherWorker(latitude: Double, longitude: Double) {
    val networkConstraint =
        Constraints.Builder().setRequiredNetworkType(NetworkType.CONNECTED).build()
    val request = PeriodicWorkRequest
        .Builder(WeatherWidgetWorker::class.java, 15, TimeUnit.MINUTES)
        .setBackoffCriteria(BackoffPolicy.EXPONENTIAL, 5000L, TimeUnit.MILLISECONDS)
        .setInputData(
            WeatherWidgetWorker.buildData(latitude, longitude)
        )
        .setConstraints(networkConstraint)
        .build()
    val uniqueTag = WeatherWidget.UNIQUE_WORK_TAG + "_$latitude" + "_$longitude"
    WorkManager.getInstance(this)
        .enqueueUniquePeriodicWork(
            uniqueTag,
            ExistingPeriodicWorkPolicy.REPLACE,
            request
        )
}
```

WeatherWidget.kt

```
fun Context.startWeatherWorker(latitude: Double, longitude: Double) {
    val networkConstraint =
        Constraints.Builder().setRequiredNetworkType(NetworkType.CONNECTED).build()
    val request = PeriodicWorkRequest
        .Builder(WeatherWidgetWorker::class.java, 15, TimeUnit.MINUTES)
        .setBackoffCriteria(BackoffPolicy.EXPONENTIAL, 5000L, TimeUnit.MILLISECONDS)
        .setInputData(
            WeatherWidgetWorker.buildData(latitude, longitude)
        )
        .setConstraints(networkConstraint)
        .build()
    val uniqueTag = WeatherWidget.UNIQUE_WORK_TAG + "_$latitude" + "_$longitude"
    WorkManager.getInstance(this)
        .enqueueUniquePeriodicWork(
            uniqueTag,
            ExistingPeriodicWorkPolicy.REPLACE,
            request
        )
}
```

WeatherWidget.kt

```
fun Context.startWeatherWorker(latitude: Double, longitude: Double) {
    val networkConstraint =
        Constraints.Builder().setRequiredNetworkType(NetworkType.CONNECTED).build()
    val request = PeriodicWorkRequest
        .Builder(WeatherWidgetWorker::class.java, 15, TimeUnit.MINUTES)
        .setBackoffCriteria(BackoffPolicy.EXPONENTIAL, 5000L, TimeUnit.MILLISECONDS)
        .setInputData(
            WeatherWidgetWorker.buildData(latitude, longitude)
        )
        .setConstraints(networkConstraint)
        .build()
    val uniqueTag = WeatherWidget.UNIQUE_WORK_TAG + "_$latitude" + "_$longitude"
    WorkManager.getInstance(this)
        .enqueueUniquePeriodicWork(
            uniqueTag,
            ExistingPeriodicWorkPolicy.REPLACE,
            request
        )
}
```

WeatherWidget.kt

```
fun Context.startWeatherWorker(latitude: Double, longitude: Double) {
    val networkConstraint =
        Constraints.Builder().setRequiredNetworkType(NetworkType.CONNECTED).build()
    val request = PeriodicWorkRequest
        .Builder(WeatherWidgetWorker::class.java, 15, TimeUnit.MINUTES)
        .setBackoffCriteria(BackoffPolicy.EXPONENTIAL, 5000L, TimeUnit.MILLISECONDS)
        .setInputData(
            WeatherWidgetWorker.buildData(latitude, longitude)
        )
        .setConstraints(networkConstraint)
        .build()
    val uniqueTag = WeatherWidget.UNIQUE_WORK_TAG + "_$latitude" + "_$longitude"
    WorkManager.getInstance(this)
        .enqueueUniquePeriodicWork(
            uniqueTag,
            ExistingPeriodicWorkPolicy.REPLACE,
            request
        )
}
```

WeatherWidget.kt

```
fun Context.startWeatherWorker(latitude: Double, longitude: Double) {
    val networkConstraint =
        Constraints.Builder().setRequiredNetworkType(NetworkType.CONNECTED).build()
    val request = PeriodicWorkRequest
        .Builder(WeatherWidgetWorker::class.java, 15, TimeUnit.MINUTES)
        .setBackoffCriteria(BackoffPolicy.EXPONENTIAL, 5000L, TimeUnit.MILLISECONDS)
        .setInputData(
            WeatherWidgetWorker.buildData(latitude, longitude)
        )
        .setConstraints(networkConstraint)
        .build()
    val uniqueTag = WeatherWidget.UNIQUE_WORK_TAG + "_$latitude" + "_$longitude"
    WorkManager.getInstance(this)
        .enqueueUniquePeriodicWork(
            uniqueTag,
            ExistingPeriodicWorkPolicy.REPLACE,
            request
        )
}
```

WeatherWidgetWorker.kt

```
class WeatherWidgetWorker(
    private val repository: WeatherRepository,
    private val appContext: Context,
    private val workerParameters: WorkerParameters
) : CoroutineWorker(appContext, workerParameters) {
override suspend fun doWork(): Result {
   WidgetStateHelper.setLoading(true)
   repository.getData(latitude, longitude)
      .onSuccess { item →
          WidgetStateHelper.save(item)
           return Result.success()
      }
      .onFailure { throwable →
           WidgetStateHelper.setLoading(false)
          return Result.retry()
      }
}
```

WeatherWidgetWorker.kt

```
class WeatherWidgetWorker(
    private val repository: WeatherRepository,
    private val appContext: Context,
    private val workerParameters: WorkerParameters
) : CoroutineWorker(appContext, workerParameters) {
override suspend fun doWork(): Result {
   WidgetStateHelper.setLoading(true)
   repository.getData(latitude, longitude)
      .onSuccess { item →
          WidgetStateHelper.save(item)
           return Result.success()
      }
      .onFailure { throwable →
           WidgetStateHelper.setLoading(false)
          return Result.retry()
      }
}
```

droidcon academy

WeatherWidgetWorker.kt

```
class WeatherWidgetWorker(
    private val repository: WeatherRepository,
    private val appContext: Context,
    private val workerParameters: WorkerParameters
) : CoroutineWorker(appContext, workerParameters) {
override suspend fun doWork(): Result {
    WidgetStateHelper.setLoading(true)
   repository.getData(latitude, longitude)
      .onSuccess { item →
          WidgetStateHelper.save(item)
           return Result.success()
      }
      .onFailure { throwable →
           WidgetStateHelper.setLoading(false)
          return Result.retry()
      }
}
```

WeatherWidgetWorker.kt

```
class WeatherWidgetWorker(
    private val repository: WeatherRepository,
    private val appContext: Context,
    private val workerParameters: WorkerParameters
) : CoroutineWorker(appContext, workerParameters) {
override suspend fun doWork(): Result {
   WidgetStateHelper.setLoading(true)
   repository.getData(latitude, longitude)
      .onSuccess { item →
          WidgetStateHelper.save(item)
           return Result.success()
      }
      .onFailure { throwable →
           WidgetStateHelper.setLoading(false)
          return Result.retry()
      }
}
```

WeatherWidgetWorker.kt

```
class WeatherWidgetWorker(
    private val repository: WeatherRepository,
    private val appContext: Context,
    private val workerParameters: WorkerParameters
) : CoroutineWorker(appContext, workerParameters) {
override suspend fun doWork(): Result {
   WidgetStateHelper.setLoading(true)
   repository.getData(latitude, longitude)
      .onSuccess { item →
          WidgetStateHelper.save(item)
           return Result.success()
      }
      .onFailure { throwable →
           WidgetStateHelper.setLoading(false)
          return Result.retry()
      }
}
```

WeatherWidgetWorker.kt

```
class WeatherWidgetWorker(
    private val repository: WeatherRepository,
    private val appContext: Context,
    private val workerParameters: WorkerParameters
) : CoroutineWorker(appContext, workerParameters) {
override suspend fun doWork(): Result {
   WidgetStateHelper.setLoading(true)
   repository.getData(latitude, longitude)
      .onSuccess { item →
          WidgetStateHelper.save(item)
           return Result.success()
      }
      .onFailure { throwable →
           WidgetStateHelper.setLoading(false)
          return Result.retry()
      }
}
```

droidcon academy

WeatherWidgetWorker.kt

```
class WeatherWidgetWorker(
    private val repository: WeatherRepository,
    private val appContext: Context,
    private val workerParameters: WorkerParameters
) : CoroutineWorker(appContext, workerParameters) {
override suspend fun doWork(): Result {
   WidgetStateHelper.setLoading(true)
   repository.getData(latitude, longitude)
      .onSuccess { item →
          WidgetStateHelper.save(item)
           return Result.success()
      }
      .onFailure { throwable →
          WidgetStateHelper.setLoading(false)
          return Result.retry()
      }
}
```

Preferences

GlanceStateDefinition

GlanceId

Referencing the widget in other context

droidcon academy

GlanceAppWidgetManager.kt

```
/**
 * Manager for Glance App Widgets.
 *
 * This is used to query the app widgets currently installed on the system, and some of their
 * properties.
 */
class GlanceAppWidgetManager(private val context: Context) {
. . .
/**
 * Returns the [GlanceId] of the App Widgets installed for a particular provider.
 */
suspend fun <T : GlanceAppWidget> getGlanceIds(provider: Class<T>): List<GlanceId> {}
}
```

droidcon academy

Code in Android Studio

droidcon academy

Handle widget click actions

Up Next

droidcon academy