Migration from 2.2 to 2.3

Information

- All external dependencies have been upgraded to the latest version:
  - Kotlin: `1.4.21`
  - Coroutines: `1.4.2`
  - Koin: `2.2.1`
  - Moshi: `1.11.0`
  - ExoPlayer: `2.12.2`
  - Constraint Layout: `2.0.4`
- Implementations based on `AbsVideoPlayerWrapper` may need to adjust error reporting. See Error Handling for more details.
- Everything which worked in 2.2.x should work in 2.3.0 as well without any code changes.

Detailed Steps

To make use of new features like closed captions or error handling some changes may be required. Some imports have been changed. They are marked as deprecated and can be fixed by using "quick-fix" feature of IntelliJ.

ClosedCaptions

- No changes are required for `SxAdView` or `InstreamExoWrapper` based implementations.
- Implementations based on `AbsVideoPlayerWrapper` need to add the `closedCaptions` parameter to the `loadAd` method:

```
override fun loadAd(
    url: String,
    closedCaptions: List<SxClosedCaption>
) {
    super.loadAd(url, closedCaptions)
    ...
}
```

- Building an ExoPlayer `MediaSource` with the provided `SxClosedCaption` is shown in the following example:

```
fun buildMediaSource(
    mediaUrl: String,
    closedCaptions: List<SxClosedCaption>
): MediaSource {
    val videoSource = ProgressiveMediaSource.Factory(dataSourceFactory)
            .setLoadErrorHandlingPolicy(DefaultLoadErrorHandlingPolicy(0))
            .createMediaSource(MediaItem.fromUri(Uri.parse(mediaUrl)))
    val language = Locale.getDefault().language
    val subtitleSources = closedCaptions.firstOrNull { it.language == language
}?.let {
        SingleSampleMediaSource.Factory(dataSourceFactory).createMediaSource(
                MediaItem.Subtitle(Uri.parse(it.fileURL.withProtocol),
                        MimeTypes.TEXT_VTT, language, Format.NO_VALUE),
C.TIME_UNSET
        )
    }
    return (subtitleSources?.let { MergingMediaSource(videoSource, it) } ?:
videoSource)
}
```

Visibility Threshold

- The minimum relative amount of the `SxAdView` that needs to be visible to the user until playback starts/resumes can now be defined with the `SxConfiguration.visibleThreshold` property.
- Accepts a `Double` within a range of `0.0` (player not visbible at all) to `1.0` (complete player visible).
- The default is set to `0.5` (was `0.75` in `2.2.x`).

Error Handling

- To get notified for all errors that are propagated by the SDK, set a listener using the `addOnErrorListener` method, which is available in `SxSequencer` and `SxAdView` instances.
- No changes are required for `SxAdView` or `InstreamExoWrapper` based implementations.
- Implementaions based on `AbsVideoPlayerWrapper` need to report any error produced by the used video player to the wrappers `error` property. To achieve this, transform caught video player exceptions to a `SxMediaError` instance. ExoPlayer exceptions can be transformed using the provided `Throwable.buildMediaError()` method. When dealing with another video player, you can orientate yourself on the `Throwable.buildMediaError()` method to implement your own:

```
fun Throwable.buildMediaError(): SxMediaError = SxMediaError(when (this) {
                is ExoPlaybackException -> when (type) {
                    TYPE_TIMEOUT,
                    TYPE_SOURCE,
                    TYPE_REMOTE -> MEDIA_ERR_NETWORK
                    TYPE_RENDERER,
                    TYPE_OUT_OF_MEMORY -> MEDIA_ERR_DECODE
                    TYPE_UNEXPECTED -> MEDIA_ERR_SRC_NOT_SUPPORTED
                    else -> MEDIA_ERR_SRC_NOT_SUPPORTED
                }
                is HttpDataSource.HttpDataSourceException,
                is Loader.UnexpectedLoaderException,
                is RawResourceDataSource.RawResourceDataSourceException ->
MEDIA_ERR_NETWORK
                else -> MEDIA_ERR_DECODE
            }, message ?: "")
```

After creating a `SxMediaError` instance, set it to the `error` property of the `AbsVideoPlayerWrapper`. The following example shows the error reporting of the ExoPlayer used by the SDK:

```
class ExoWrapper : AbsVideoPlayerWrapper() {

    ...

    private val exoListener = object : EventListener {
        override fun onPlayerError(
            exception: ExoPlaybackException
        ) {
            error = exception.buildMediaError()
        }
        ...
    }

    init {
        player.addListener(exoListener)
    }

    fun loadMedia(...) {
        val mediaSource = buildMediaSource(mediaUrl, closedCaptions).apply {
                    addEventListener(Handler(Looper.getMainLooper()), object :
MediaSourceEventListener {

                        override fun onLoadError(windowIndex: Int,
                                                 mediaPeriodId:
MediaSource.MediaPeriodId?,
                                                 loadEventInfo: LoadEventInfo,
                                                 mediaLoadData: MediaLoadData,
                                                 error: IOException,
                                                 wasCanceled: Boolean) {
                            this@ReferenceVideoPlayerWrapper.error =
error.buildMediaError()
                        }
                    })
                }
        }
```

Timeouts

- All timeouts used by the SDK can now be defined in the `SxConfiguration` class using the `timeouts` property. See the KDoc of `SxTimeoutConfig` for details.
- Implementations based on `AbsVideoPlayerWrapper` need to configure the used video player with the `mediaConnect` and `mediaRead` timeouts. For the ExoPlayer it can be done like this:

```
val dataSourceFactory: DataSource.Factory =
    DefaultHttpDataSourceFactory(
        getExoUserAgentRawAndAppendOwn(configuration.userAgent),
        configuration.timeouts.mediaConnect * 1000,
        configuration.timeouts.mediaRead * 1000, true)
```

GDPR Consent

- The base64-encoded cookie value of IAB GDPR consent info that will expand the `[GDPRCONSENT]` macro should now be defined with the `gdprConsent` property of the `SxConfiguration` class.

Playback State

- The sdk became more strict regarding the `PlaybackState.PLAYING` state. You only should set this, when the position of the video is advancing. See the following code for an example:

```
fun buildPlaybackState(): PlaybackState = when (exoPlayer.playbackState) {
    Player.STATE_BUFFERING -> PlaybackState.BUFFERING
    Player.STATE_READY -> if (exoPlayer.isPlaying) PlaybackState.PLAYING else
PlaybackState.PAUSED
    Player.STATE_ENDED -> PlaybackState.ENDED
    else -> PlaybackState.IDLE
}
```