Alarm Voice Announcement Example

Announce active alarms by voice on the operator client.

Import the **VoiceAlarm** plugin from **Solution Import Plugin**.

Download, unzip and move it to "C:\Users\Public\Documents\FrameworX\Exchange": VoiceAlarm.zip (10.1.5e or later)

The plugin contains a single Script Class. It requires no tags, no device configuration and no database.

Summary

This feature speaks alarm messages aloud on the operator client, so an operator who is not looking at the screen still learns that something needs attention and how serious it is.

It announces the **highest-priority active alarm**, read from `@Alarm.PriorityItem`. It works on both client types from the same code:

- **Windows client** (Rich Client / Smart Client) — speaks through `System.Speech.Synthesis`, using the voices installed in Windows.
- **HTML5 / browser client** — speaks through the browser's built-in Web Speech API. Nothing to install and nothing to host.

Speech happens on the **client**, never on the server. A server running as a Windows Service has no audio device, and a request to speak there returns successfully and makes no sound.

Step 1 — Import the plugin

In the Designer, go to **Solution Import Plugin** and select the VoiceAlarm plugin. This adds one Script Class named `VoiceAlarm`, with its **Domain** set to **Client**.

The Client domain matters: it gives every connected station its own instance, so each operator station decides independently what to announce.

Step 2 — Call it from a display that is always open

The class does nothing on its own. It must be called repeatedly from the code-behind of a display that stays open for the whole session.

The **Header** is the recommended place, because it is part of the startup layout and is open on every screen. Any always-open display works — a footer or a permanently docked panel is equally valid.

Open the display, go to the **CodeBehind** tab, and add:

```
public void DisplayIsOpen()
{
    @Script.Class.VoiceAlarm.CheckAlarmToVoice();
}
```

Then set the display's **DisplayIsOpen Cycle** property to **250** (milliseconds). That is how often the check runs.

That is the whole installation. There is nothing else to configure to get spoken alarms.

How it behaves when several alarms are active

This is the part worth understanding, because the behaviour is deliberate and it is not what a naive implementation would do.

It always speaks the most important alarm first

The class never maintains its own list of alarms to read out. On every check it asks the platform a single question: *what is the highest-priority active alarm right now?*

While a phrase is playing, the check does nothing at all. When the phrase finishes, it asks the question again — and announces whatever is the top alarm at that moment.

Why it does not queue

Queueing announcements sounds harmless and behaves badly. Alarms in a cascade usually arrive least-important first, so a queue reads them in exactly the wrong order, and keeps reading long after the situation has changed.

With five alarms arriving inside 1.2 seconds, a queue produces this:

Time	Queued announcement
0–3 s	Motor D overload *(priority 2 — the least important)*
3–6 s	Valve B fault *(priority 4)*
6–9 s	High level on tank C *(priority 6)*
9–12 s	High discharge pressure on compressor E *(priority 8)*
12–15 s	High temperature on pump A *(priority 9 — the most serious, announced last)*

Fifteen seconds of narration for an event that lasted just over one, and the alarm that mattered most is spoken twelve seconds late.

The same burst with this feature:

Time	Announcement
0–3 s	Motor D overload *(the top alarm at that instant)*
0.3–1.2 s	four more alarms arrive — nothing is queued
3.25–6.25 s	High temperature on pump A *(now the top alarm)*

The alarms it did not name are not forgotten

They stay active, and each is announced when it becomes the highest-priority alarm — typically once the more serious alarm above it has been acknowledged or has cleared.

So that silence is never mistaken for an all-clear, each announcement ends with a count of the others:

> *"Attention. High temperature on pump A."*
> … short pause …
> *"Thirty-nine more alarms active."*

The count is spoken as a separate sentence after a configurable pause (`OthersDelay`), so it registers as its own piece of information rather than running on from the alarm text. Set `AnnounceOthers` to false to turn it off.

> During a large alarm flood, the useful message is *"this is the worst thing, and there are many"*. The alarm list on screen carries the full inventory; the voice channel carries urgency and priority.

Configuration

Every setting has a working default — the feature runs with no configuration at all. Set properties from the code-behind before calling `CheckAlarmToVoice()`, or from a startup script on the client.

Property	Default	Purpose
`Enabled`	true	Master on / off for this station.
`MinPriority`	0	Ignore alarms below this priority. 0 announces every priority.
`OnlyGroups`	(empty)	Comma-separated alarm group names to announce. Empty means all groups.
`SpeakOnlyOnce`	true	Announce each alarm occurrence once. An alarm that clears and re-triggers is a new occurrence and is announced again.
`Template`	Attention. {Message}.	The spoken sentence. Placeholders: `{Message}`, `{TagName}`, `{Group}`, `{Priority}`.
`Volume`	100	0 to 100.
`Rate`	0	Speaking rate, -10 (slowest) to 10 (fastest).

`Language`	en-US	Voice language. Must match a voice available on the client.
`AnnounceOthers`	true	Append the count of other active alarms.
`OthersDelay`	1500	Milliseconds of silence between the alarm and the count sentence.

Example — announce only critical alarms, in a calmer voice:

```
public void DisplayOpening()
{
    @Script.Class.VoiceAlarm.MinPriority = 8;
    @Script.Class.VoiceAlarm.OnlyGroups  = "Critical";
    @Script.Class.VoiceAlarm.Rate        = -2;
}
```

Because the class runs in the Client domain, these settings apply to **that station only**. A control-room station and a maintenance laptop can have different thresholds.

Testing

Call `TestSpeak` from a button to confirm audio is working without waiting for a real alarm:

```
@Script.Class.VoiceAlarm.TestSpeak("Voice alert test.");
```

Requirements and limitations

- **No tags are required.** The class reads `@Alarm.PriorityItem` and `@Alarm.TotalCount`, which are platform objects. It creates nothing in the Unified Namespace.
- **The client needs an audio device.** Announcements are produced on the operator station, not the server.
- **Browser clients need one user interaction first.** Browsers block audio until the user has clicked somewhere on the page — logging on normally satisfies this. Until then the browser stays silent.
- **Voices come from the client machine.** On Windows they are the installed SAPI voices; in a browser they come from the browser and operating system. If no voice matches `Language`, a default voice is used instead and the substitution is recorded in the trace log.
- **One speaking station per area.** Every client with the feature enabled announces independently. Two stations within earshot will talk over each other, so enable the voice on one station per room and leave it off elsewhere.
- **Announcements are paced by speech, not by alarms.** A phrase takes two to three seconds, so in a fast cascade only the alarms that reach the top of the priority list are named individually. This is intended — see the section above.

> ⚠ A single upset raising dozens of alarms is a sign that alarm rationalisation is incomplete. Standards such as ISA-18.2 and EEMUA 191 address this upstream, with techniques like state-based suppression and first-out logic, so that one designed alarm replaces the cascade of consequences. Voice announcement is a safety net for the operator, not a substitute for that work.

In this section...
