

# Tapi3 Service Provider

## 1. Overview

Tapi3 is a GJTAPI service provider for Microsoft Tapi 3.0.

A Swing application called Tapi3Gui accompanies Tapi3 and shows how to use this service provider.

This document describes both the Tapi3 service provider and the Tapi3Gui test program.

## 2. Limitations

Tapi3 provider supports only a limited subset of the methods of CCTpi and MediaTpi:

Package	Method	Supported	
CallControl	call()	YES	
	answer()	YES	
	hold()	YES	configurable through the <i>swapOnHold</i> parameter
	unhold()	YES	
	release()	YES	
	join()	no	
Media	allocateMedia()	no	
	freeMedia()	no	
	isMediaTerminal()	no	
	play()	no	
	record()	no	
	retrieveSignals()	YES	
	sendSignals()	YES	
	stop()	no	
	triggerRTC()	no	

Tapi3 provider associates only one terminal with each address. The terminal name is identical with the address name.

## 3. Installation and tuning

There is nothing to install in order to use the Tapi3 provider. Just make sure that all jars and needed props files (GenericResources.props, GenericCapabilities.props, Tapi3.props etc.) are in your classpath and that Tapi3Provider.dll is in your java.library.path.

The name of the Tapi3 provider, as declared in GenericResources.props, is "Tapi3":

```
PROVIDER_Tapi3 = Tapi3.props
```

So you can create a Tapi3 provider using:

```
provider = peer.getProvider("Tapi3");
```

System properties can be used in Tapi3.props using the syntax: `${propertyName}` (e.g. `${user.home}`).

The following paragraphs show how to control the Tapi3 provider behavior by changing some of the parameters of Tapi3.props.

## 3.1. Logging

### 3.1.1. Java logging

Tapi3 defines a very simple logging API through the `Logger` interface in package `net.sourceforge.gjtapi.raw.tapi3.logging`. It also offers a few implementations of this API:

- `PrintStreamLogger` - logs events to a `PrintStream`
- `ConsoleLogger` - logs events to `System.err`
- `NullLogger` - simply discards all log messages
- `Tapi3Logger` - delegates its job to a `log4j` `Logger` (used by `Tapi3Gui`)

You can easily write your own `Logger` implementation if none of these predefined loggers fit your needs.

You can configure a `PrintStreamLogger` that logs events to a file through the `tapi3.log.out` parameter. Example:

```
tapi3.log.out = C:\\tapi3.log  
or  
tapi3.log.out = ${user.dir}\\tapi3-${user.name}.log
```

The special value `"console"` can be used to configure a `ConsoleLogger`:

```
tapi3.log.out = console
```

The general method to configure a logger is to set the value of the `tapi3.log.class` parameter with the fully qualified name of the logger class. Example:

```
tapi3.log.class = net.sourceforge.gjtapi.raw.tapi3.logging.NullLogger  
or  
tapi3.log.class = net.sourceforge.gjtapi.test.tapi3.Tapi3Logger
```

The logger class must provide a constructor that takes no arguments.

### 3.1.2. DLL logging

`Tapi3Provider.dll` logs to the file specified by the parameter `tapi3.native.log.out`. Example:

```
tapi3.native.log.out = C:\\tapi3dll.log  
or  
tapi3.native.log.out = ${user.dir}\\tapi3dll.log
```

## 3.2. Hold and Unhold behaviour

Some telephony systems do not support the hold and unhold operations, but they may support the swap operation. You can ask Tapi3 provider to execute a swap instead of hold or unhold, by setting the `tapi3.native.swapOnHold` parameter to true:

```
tapi3.native.swapOnHold = true
```

In this case, a hold operation is possible only if the target terminal connection is in state `TALKING` and there is another terminal connection in state `HELD`.

Conversely, an unhold operation is possible only if the target terminal connection is in state `HELD` and there is another terminal connection in state `TALKING`.

### 3.3. Call handoff

You can instruct Tapi3 provider to perform a directed handoff of incoming calls to another application by setting the *tapi3.native.handoff* parameter with name of the target application. Example:

```
tapi3.native.handoff = dialer.exe
```

### 3.4. Native implementation

You will probably never want to change this parameter, but it will be described shortly for the sake of completeness.

Tapi3 provider defines an interface called Tapi3Native which provides an abstraction level that binds the Java implementation with the native implementation (e.g. Tapi3Provider.dll). The Tapi3Native implementation to be used can be specified through the *tapi3.impl.class* parameter:

```
tapi3.impl.class = net.sourceforge.gjtapi.raw.tapi3.Tapi3NativeImpl
```

(Why would someone need this parameter? During the developing of Tapi3 provider I have had only limited access to my telephony system, so I have implemented an emulator for it. Through the *tapi3.impl.class* parameter I have had the possibility to switch between the real telephony system and the emulator).

## 4. Non-standard features

Tapi3 provider can send PrivateCallEv events containing the caller name, caller number, called name and called number as a private data object of type Tapi3PrivateData. The following idiom retrieves this information from a PrivateCallEv event:

```
PrivateCallEv privCallEv = ...;
Object privateData = privCallEv.getPrivateData();
if(privateData instanceof Tapi3PrivateData) {
    Tapi3PrivateData tapi3PrivateData = (Tapi3PrivateData)privateData;
    String callerName = tapi3PrivateData.getCallerName();
    String callerNumber = tapi3PrivateData.getCallerNumber();
    String calledName = tapi3PrivateData.getCalledName();
    String calledNumber = tapi3PrivateData.getCalledNumber();
    ...
    ...
    ...
}
```

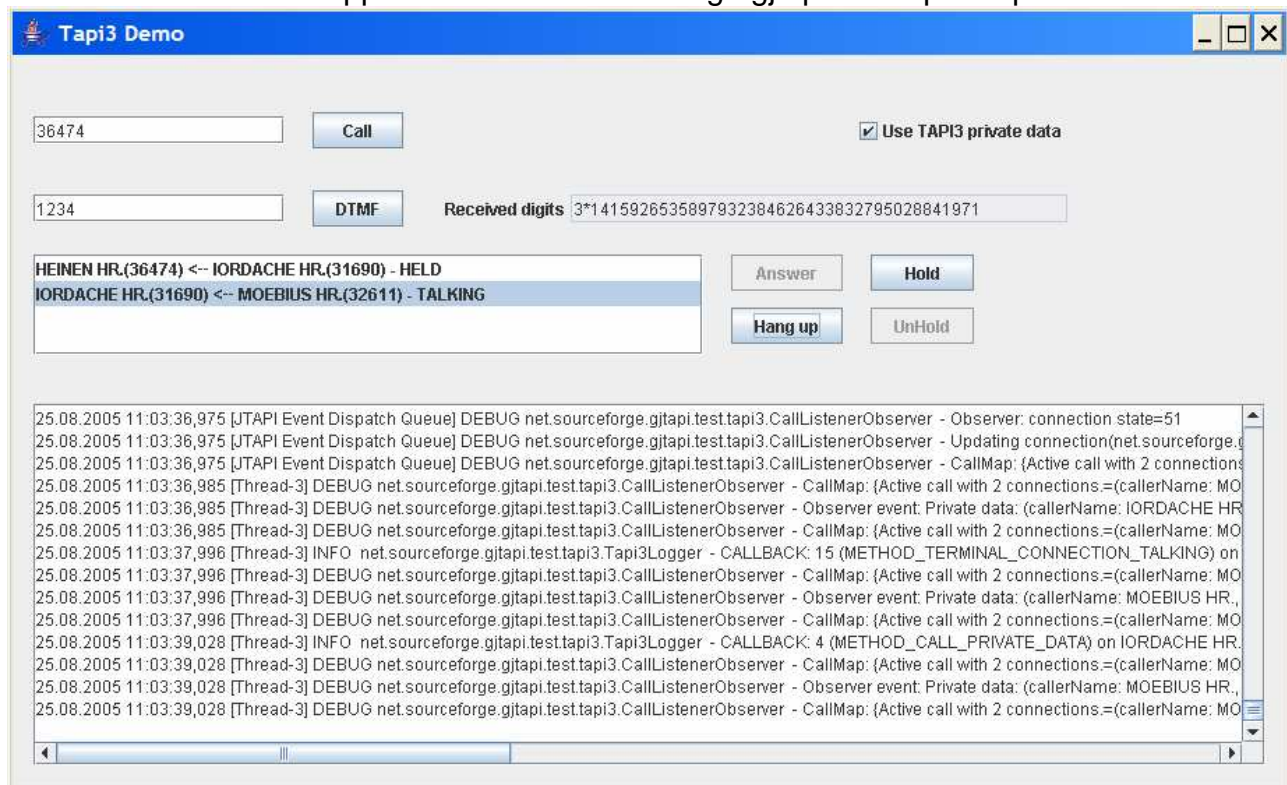
Since this is a Tapi3 specific feature, you should try to avoid it if you don't really need the information provided by Tapi3PrivateData. In most cases, the data retrieved using the standard CallControlCall's methods: getCallingAddress(), getCalledAddress() and getCallingTerminal() will be enough for you.

## 5. The Tapi3Gui application

Tapi3Gui is a Swing application that shows the use of the Tapi3 service provider. You can force the program to use another service provider by specifying its name as argument. Tapi3Gui allows you to make calls, receive calls and hold or unhold calls. You can also send and receive DTMF digits to and from the selected terminal.

The checkbox "Use TAPI3 private data" determines whether the information in call list is retrieved using CallControlCall's getCallingAddress() and getCalledAddress() or based on the non-standard TAPI3 private data. Of course, the last case makes no sense if the application uses another service provider than Tapi3.

The main class of this application is net.sourceforge.gjtapi.test.tapi3.Tapi3Gui.



## 5.1. Logging

Tapi3Gui uses the log4j based Tapi3Logger, so make sure that a recent version of the log4j jar (<http://logging.apache.org/site/binindex.cgi>) is in your classpath.

By default, Tapi3Gui has only one log4j appender that logs all messages to the trace area (placed at the bottom part of the GUI).

You can configure additional appenders if you use a log4j configuration file. For example, you can create a file named log4j.properties with the following content:

```
log4j.rootLogger=INFO, CON
log4j.appender.CON=org.apache.log4j.ConsoleAppender
log4j.appender.CON.layout=org.apache.log4j.PatternLayout
log4j.appender.CON.layout.ConversionPattern=%-5p - %m%n
```

Start the program setting the value of the *log4j.configuration* system property to the URL of your log4j.properties file:

```
java -Dlog4j.configuration=file:/C:/gjtapi/log4j.properties ...
```

Now, Tapi3Gui will log messages both to the trace area and the console. This can be very useful to trace problems that arise before the main window (and hence the trace area) is displayed. Only messages with level INFO or higher will be logged using the configuration file above. (Note that this also affects the messages logged to the trace area).