FAQ

Frequently Asked Questions

- MiG InfoCom AB -

Revised for v6.0

Product FAQ

What is MiG Calendar?
It is a calendar component that can be used to create any type of application that shows time based events such as flight times, appointments, TV schedules, school schedules, event itineraries, project plans, Gantt charts and alarm visualization.
The preferred way to use MiG Calendar is through the Java Beans in com.miginfocom.beans.* package.

You say the component is Flexible and Extensible, what do you mean?
We have made the component extensible in may ways so that the user of it should never end up in a dead end and not be able to proceed. There are short cuts so the component is really easy to use, but you also go a very custom way with almost everything. For instance the visualization of the events: You can with only one line swap in another AShape that should do the visualization, or you can exchange the whole rendering path and use whatever code you prefer to paint them. You can also map an AShape to the paintContext of the activity.
Same thing with the placement of the activities in the date grid, there are four different layout managers for this included in the component, but you can easily write you own since ActivityLayout is a normal Java Interface.

What's in for the future of the component, in what direction will it evolve? Do you have a product plan?
Yes we do, however it's not public for the moment.

If you have any recommendations on what to include in the next major version please don't hesitate to contact us at the forums or by email.

http://www.miginfocom.com/forums/
support@miginfocom.com

Is MiG Calendar a Visually editable Java Bean?
Yes. From v5.5 it is. It works in all Java Bean compatible IDEs. See below for IDE support. In 6.0 this approach has been improved further and the use of JavaBeans is now the recommended way to code against MiG Calendar.

Do you have a feature matrix?
The flexibility in the component makes is support almost any feature

thinkable. For instance we can take the Activity class, which outlines something that is time based. You can set any property on it and set any object type for that property. The visualization of that activity, which is done trough one or many ActivityViews, can depend on the value of that property. This makes for an unsurpassed, almost unlimited, flexibility. A feature matrix would not do this flexibility justice.

Can I use the component in an application that I sell?
Yes, as long as you don't expose the component's API. There are no runtime fees. You just have to buy one developer license for every developer working with the component.

What IDEs do your component support?
Any that is Java Bean compatible. Since the whole component comes in one .jar file it is very simple to include in your current environment. How you do this differs between the IDEs, but usually you will include it in the classpath and/or set it as an external library. Tests has been run against netBeans 4.1+, JFormDesigner 2.0+, JBuilder X, JBuilder 2006. (JBuilder 2005 has a severe bug the hinders it from working with custom beans) and Eclipse (Visual Designer and Swing Designer).

Which versions of Java do you support?
Java 1.4.0 and higher. Sub pixel (LCD) anti aliasing is even supported for JDK 6.0!

How many simultaneous activities can the component handle?
We have an internal test that creates 100.000 activities and shows 10% of them visually. It starts up and displays them in 8 seconds and consumes 95 MB of memory according to Windows Task Manager (which means the actual component is actually using much less). We use no special tweaking or command line arguments. Just the normal client VM and Java 1.5.0_01.

Do the component support recurrent events/activities?
Yes. We have used the standard iCalendar specification from RFC 2445 as a base and support all of the recurrence rules, including include and exclude boolean operations. It has been Java-fied though and are therefore *much* simpler to use, yet very powerful. It can even be used separately for recurrence calculation. If you are to use medium to advanced recurrence calculations that part in itself is probably worth the money for the component.

Do you provide any standard dialogs?
No.

What can the AShape framework do?
You can build an advanced shape framework with it, much like SVG

(Scalable Vector Graphics) only that it is more Java friendly and sports more a programmatic approach. It is also much smaller in size, the AShape component will probably be around a 100k .jar once release.

The shape hierarchy can be used almost as a brush and painted within any bounds, much like the Flyweight Rendering pattern. It is very powerful and for instance support almost Swing-like layout managers, only it's much faster and a lot more light weight. It also support animations and a flexible coordinate system, and more.

Which file format do you use to save AShapes and Themes?
We use the standard XML JavaBeans framework introduced in Java 1.4. This makes sure you can read them without resorting to any custom library, everything is included in all standard edition Java distributions. It also ensures that you will not suffer from vendor lock-in when using our component since the file format is public, free and easy to use.

Do you include any standard persistence for activities?
Yes. We have a plugin called DBConnect that automatically persists activities and/or categories to a number of different supported databases. You can read about it at the migcalendar product site. www.migcalendar.com.

For v6.0 we will also be releasing a Google Calendar plugin that automatically connects the calendar component to one or more Gcal account and handles all the synchronizarion automatically. It even have support for full on/off line use.

What Time Zone support do you have?
Every Activity can have its own time zone. The date areas can have their own as well. Time zones are complex in nature, especially if you factor in the daylight savings problem. There are even a few utility methods in the DateUtil class for handling time zones and daylight savings diffs. MiG Calendar works out of the box with the default time zone for the platform it's running on, but that is configurable. You can even have multiple date headers with different time zones.

Technical FAQ

What do I need to deploy applications with MiG Calendar?
When you *develop* you should use (have in the classpath) **migcalendarbean.jar**. It contains more information that makes it easier to develop against, such as variable names in method signatures. It also contains all custom property editors needed for visual development in an IDE.

When you deploy you should use **migcalendar.jar**. It is smaller and still contain all classes needed.

The Theme Editor, component API and the JavaBean properties all mention the concept of Repetition. How does it work?
Repetition is a way to express which instances of something belong to what index. For instance if you have two Colors that should be used to draw every second or every third grid line you need this kind of flexible system.

Basically it's an ordered list of objects that each contains information about what indexes (e.g. rows or columns) it should "hit". These objects are then asked by the framework one at a time if a row index is a hit. The first object in that list that accepts that index is chosen. The framework then starts at the top of the object to get the which object applies to the next index, and so on.

The class DefaultRepetion or one of its more specific sub classes is normally used for this and it supports an offset and modulo integer to denote for instance "every third index, starting with number two" or "every fifth, starting with zero". It also contains an optional minimum and maximum index to accept.

Using these DefaultRepetion objects you can thus form complex patters like "OOOOOXOOOXOOOXOOOXOOOOOOOOO" or "XOXOXOXOXXXXXXOXOXOXOX" without hard coding it.

This technique is very handy when you want to specify otherwise hard to specify statements like: "Every even grid line should be black and every odd grid line should be gray. Except the first five and last five which should be light gray". For a finite and known number of grid lines this can be quite simple without a Repetition approach, but if the grid line count is flexible or not known at design time you need a powerful algorithm like this.

The Repetition approach is used in quite a few places in the component even though you normally don't have to use the class directly. You might be using one of it's more specific sub classes or be configuring it visually in the Theme Editor or through a JavaBean property editor. Nevertheless it is very important to understand how it works.

Can you run the Theme Editor with another AShape? I'd like to test it with my custom created one.
Yes. Here are the command line switches for the Theme Editor:

```
ThemeEditor [theme filename] [-s DemoShapeFilename] [-e
PropertyEditorFactoryClass] [-e PropertyEditorFactoryClass]
```

You can load a custom AShape that has previously been saved to a file (with `AShapeUtil.saveShape(...)`).

I can't see any Activities in the component, what can be wrong?
Check that you have turned on activity support with:

`defaultDateArea.setActivitiesSupported(true)`; or the key `"Generic/activitiesSupported"` with the Theme Editor if you are using a Themed approach.

If you are using the manual approach see the Getting Started Guide, there's a complete explanation there. You basically have to install a `Decorator` that draws them and an `ActivityLayout` that tells the where to go. Both are included in the component of course, but they have to be added to the `DateArea`.

Do you have any examples on how to create AShapes?
Under the demo source installed with the component there are quite a few. See the `AShapeCreator.java` for instance, but a few of the the demo tabs also have their own.

How do you support the XML JavaBeans framework introduced in Java 1.4?
We have written delegates for all important classes in the component. Look in the `IOUtil` class. There you can get a delegate for a specific class or have them automatically installed on a `XMLEncoder` instance. We use this framework internally for saving and loading Themes and AShapes.

How can you show different looks for different Activities. For instance a different background color for recurring ones.

There are many ways to do this. One is to create your own `ActivityViewRenderer` and set that on a `DefaultDateArea`. Then you have total control but have to do everything manually, including mouse handling support and such.

Another way that can be used if there are different looks (e.g. colors) depending on the `Category` the activities belongs to (an activity can belong to any number of categories) is this:

```
CategoryDepository.setOverride(<categoryID>, <targetSubshapeName>,
<propertyName>, <new value>);
```

for example:

```
CategoryDepository.setOverride(susanID, "background", AShape.A_PAINT,
```

```
Color.PINK);
```

A more generic and flexible way is the new static generic overriders introduced in v5.2.

For instance to set a yellow border on all recurring events:

```
ActivityInteractor.setStaticOverride("outline", AShape.A_PAINT,
   new OverrideFilter() {
      public Object getOverride(Object subject, Object defaultObject) {
         if (((ActivityView) subject).getModel().isRecurrent()) {
            return Color.YELLOW;
         } else {
            return defaultObject;
         }
    }
});
```

Since you have the `Activity` at hand you can base your override on anything you like, such as for instance a custom property in the Activity.

One of things you need to know for the above to work is the name of the subshape to target. If you create you own `AShape` that's no problem since you are in control of everything. The default shape created by the framework if nothing else is provided have constants for the names. They are unlikely to change.

```
public static final String DEFAULT_CONTAINER_SHAPE_NAME
public static final String DEFAULT_BACKGROUND_SHAPE_NAME
public static final String DEFAULT_TITLE_TEXT_SHAPE_NAME
public static final String DEFAULT_MAIN_TEXT_SHAPE_NAME
public static final String DEFAULT_OUTLINE_SHAPE_NAME
public static final String DEFAULT_SHADOW_SHAPE_SHAPE_NAME
public static final String DEFAULT_SHADOW_SHAPE_NAME
```

They are defined in the class `AShapeUtil` and the source code for the shape is available here:
`http://www.miginfocom.com/forum/viewtopic.php?t=18`

Yet another way is to provide different `AShape`s (or rather `RootAShape`s..) for different `paintContexts`. This is automatically handled by the framework. You simply register a `RootAShape` for a `paintContext` by:

Statically for all `DefaultAShapeProvider`s:

```
DefaultAShapeProvider.setShapeGlobally(<rootAShape>, <paintContext>)
```

or for a specific `DefaultAShapeProvider` instance:

```
provider.setShape(<rootAShape>, <paintContext>)
```

Then you just set the `paintContext` for the activities to match one of the registered contexts and the renderer will use the appropriate shape to paint it. `null` is the default `paintContext` used for the default shape.

```
activity.setPaintContext(<paintContext>);
```

A much more advanced solution, but one that give you total control, is to write new `Interactor`s and `InteractionBroker`s and set them on the `RootAShape` in the render stage. That is not a route we recommend for simpler customizations though.

From v6.0 you can also install mouse listeners on the AShapes. See the AShape Developers Guide for how to do this. It makes interactions with the mouse much simpler.