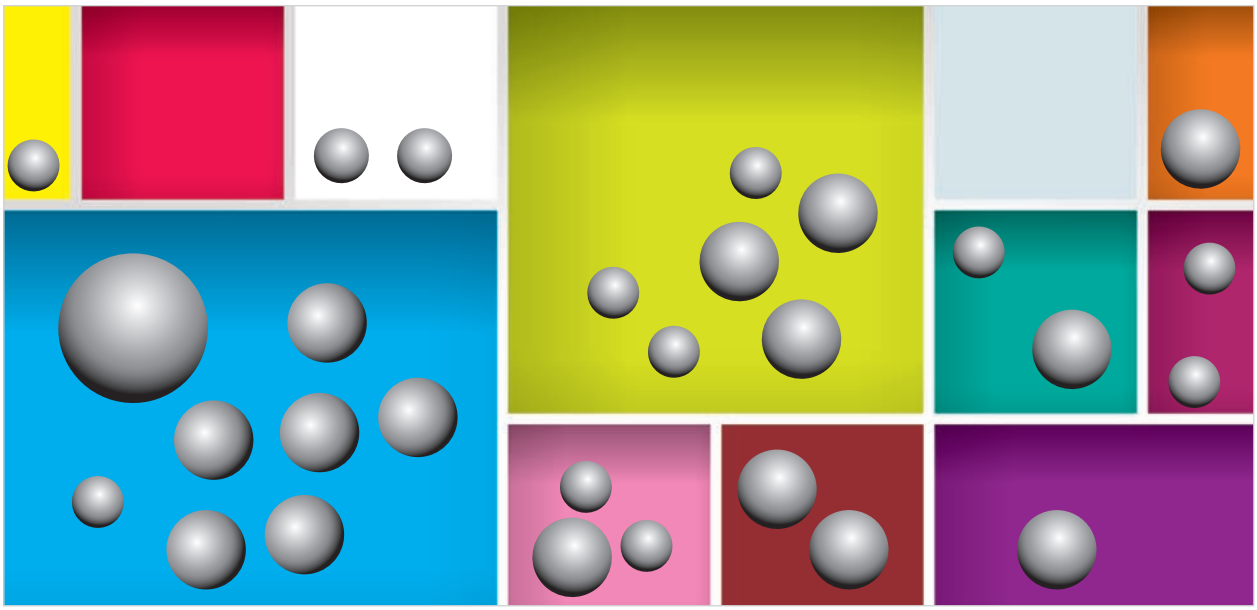

Das erste Major-Release unter Eclipse-Flagge ist releast

Nebula NatTable 1.0.0

NatTable ist ein SWT Control für komplexe, hochperformante Tabellen, Grids und Trees, das mit sehr großen Datenmengen umgehen kann und mit einem umfangreichen Featureset beeindruckt [1]. Dieses gibt es bereits seit mehreren Jahren und es wurde bis zur Version 2.3.2 auf Sourceforge gehostet. Vor gut einem halben Jahr wurde NatTable innerhalb des Nebula-Projekts in das Eclipse-Universum aufgenommen und hat aufgrund der Vorgaben im Eclipse Development Process [2] einen Versionsrücksprung auf 0.9.0 vollzogen. Die Aktivitäten in den letzten Monaten haben bewiesen, dass Nebula NatTable sowohl als Control als auch als Projekt reif ist, und somit ein Major-Release verdient hat [3]. Nachfolgend werden die wichtigsten Neuerungen vorgestellt.

von Dirk Fauth

Das Hauptaugenmerk bei der Entwicklung des 1.0.0-Releases lag auf den Themen Usability, Stabilisierung und Flexibilität. Zur Verbesserung der Usability wurden Arbeiten an der Projektinfrastruktur, dem API und der Dokumentation vorgenommen. Letzteres vor allem im Bereich Javadoc und der Veröffentlichung selbiger. Eine detaillierte Benutzerdokumentation ist in Arbeit. Im Rahmen der Stabilisierung wurden zahlreiche von der Community gemeldete Fehler behoben und teilweise kaum bekannte Features vervollständigt und korrigiert, wie z. B. das TickUpdate. Auch die Überarbeitung des API in Bezug auf das Editierverhalten war Teil der Stabilisierung. Was die Flexibilität angeht, so wurde an vielen Stellen die Sichtbarkeit von Methoden und Eigenschaften erweitert, um das Überschreiben zu ermöglichen. Außerdem wurden fixe Einstellungen konfigurierbar gemacht, so-

Mehr zum Thema

Dieser Artikel geht auf die Neuerungen in NatTable 1.0.0 ein. In den Eclipse Magazinen 4.2011 und 3.2012 sind Artikel erschienen, die den Einstieg in das Thema NatTable beschreiben. www.eclipse-magazin.de

dass z. B. die Icons zum Auf-/Zuklappen von Baumknoten jetzt konfigurierbar sind. Aber auch eine Reihe neue Features hat es in die neue Version geschafft und erweitert, wie das bereits recht umfangreiche Featureset von NatTable.

Installation

Vor der Version 1.0.0 konnte NatTable nur manuell installiert werden, indem die notwendigen JAR-Dateien in das *dropins*-Verzeichnis der Eclipse-Installation gelegt wurden. Aufgrund der infrastrukturellen Änderungen ist es jetzt möglich, NatTable über eine p2 Update Site zu installieren, wodurch der manuelle Schritt entfällt. Aber nicht nur NatTable, sondern auch die Third-Party-Dependencies der NatTable Extensions, namentlich GlazedLists [4] in der Version 1.9 und Apache POI [5] in der Version 3.9, sind dank der Arbeit des Eclipse-Orbit-Teams [6] jetzt über eine Update Site verfügbar. Um die Installation von NatTable und der abhängigen Bundles zu ermöglichen, bietet es sich an, die Eclipse Orbit Update Site einzutragen, bevor NatTable installiert wird. Die Update Site kann über Window | Preferences | Install/Update | Available Software Sites | Add hinzugefügt werden (**Abb. 1**). Die aktuellste Update Site von Eclipse Orbit findet man unter [7].

Anschließend können NatTable und die NatTable Extensions installiert werden, wobei die notwendigen Abhängigkeiten automatisch aufgelöst werden. Den Update Manager startet man über Help | Install New Software. Anschließend wählt man die Nebula NatTable Update Site aus, die entweder zusammen mit der Eclipse Orbit Update Site hinterlegt wurde oder direkt in den Dialog eingetragen werden kann (**Abb. 2**).

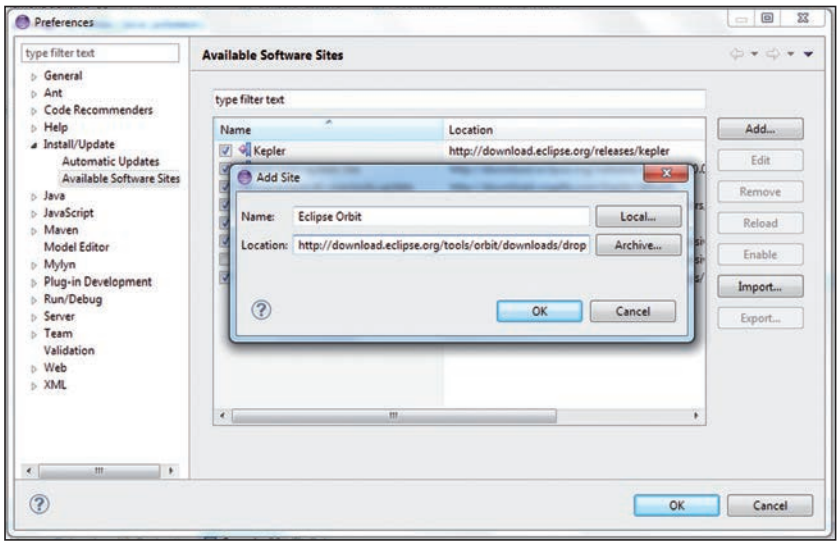

Abb. 1: Eclipse Orbit Update Site hinzufügen

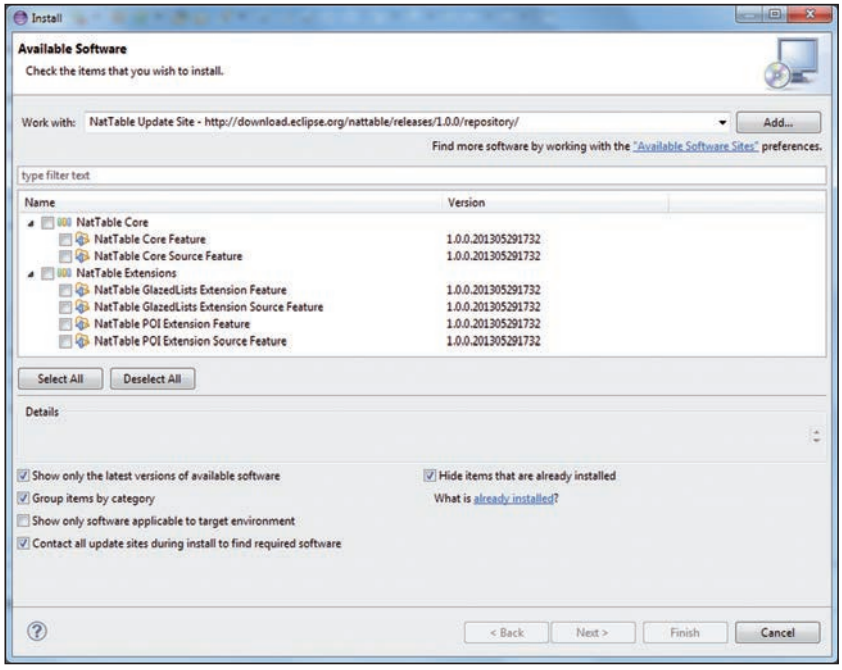

Abb. 2: NatTable installieren

EditConfigAttributes	Datentyp	Beschreibung
OPEN_IN_DIALOG	Boolean	Konfiguration, um festzulegen, ob ein Editor inline oder in einem Subdialog geöffnet werden soll. Standardmäßig ist dieser Wert auf *false* für Inline-Editing gesetzt.
OPEN_ADJACENT_EDITOR	Boolean	Konfiguration, um festzulegen, ob nach dem Bestätigen eines Werts in einem Editor der folgende Editor direkt geöffnet werden soll (z. B. nach Bestätigen mit Enter der Editor der darunterliegenden Zelle). Diese Konfiguration wird nur für Inline-Editing ausgewertet und ist standardmäßig auf *false* gesetzt.
SUPPORT_MULTI_EDIT	Boolean	Konfiguration, um festzulegen, ob der Editor multi-edit unterstützt. Standardmäßig auf *true* gesetzt, um beim Drücken von F2 für ausgewählte Zellen desselben Datentyps einen Subdialog für die Mehrfachbearbeitung zu öffnen.
EDIT_DIALOG_SETTINGS	Map<String, Object>	Konfiguration, um die Darstellung des Subdialogs festzulegen (single/multi-edit). Definiert werden können Fenstertitel, -icon, -größe, -Resizable und eine benutzerdefinierte Nachricht. Die innerhalb dieser Map unterstützten Keys sind im *ICellEditDialog*-Interface beschrieben.

Tabelle 1: Neue „EditConfigAttributes“

Da GlazedLists und Apache POI jetzt ebenfalls als OSGi Bundles verfügbar sind, konnte die Target-Platform NatTable entsprechend angepasst und die lokalen Build-Abhängigkeiten entfernt werden. Dadurch sind jetzt auch Snapshot Builds [8] möglich, die nach Änderungen im Repository automatisch gebaut werden. Benutzer haben somit die Möglichkeit, immer den aktuellsten Stand der Entwicklung zu nutzen und somit schneller von Bugfixes profitieren zu können.

Listing 1

```
// register a TextCellEditor for column two that commits on key up/down
// moves the selection after commit by enter
configRegistry.registerConfigAttribute(
    EditConfigAttributes.CELL_EDITOR,
    new TextCellEditor(true, true),
    DisplayMode.NORMAL,
    EditorExample.COLUMN_TWO_LABEL);

// configure to open the adjacent editor after commit
configRegistry.registerConfigAttribute(
    EditConfigAttributes.OPEN_ADJACENT_EDITOR,
    Boolean.TRUE,
    DisplayMode.EDIT,
    EditorExample.COLUMN_TWO_LABEL);

// configure a custom message for the multi edit dialog
Map<String, Object> editDialogSettings = new HashMap<String, Object>();
editDialogSettings.put(CellEditDialog.DIALOG_MESSAGE,
          "Please specify the lastname in here:");

configRegistry.registerConfigAttribute(
    EditConfigAttributes.EDIT_DIALOG_SETTINGS,
    editDialogSettings,
    DisplayMode.EDIT,
    EditorExample.COLUMN_TWO_LABEL);
```

Listing 2

```
ComboBoxCellEditor comboBoxCellEditor = new ComboBoxCellEditor(
                    Arrays.asList(PersonService.getDrinkList()), -1);
    comboBoxCellEditor.setFreeEdit(true);
    comboBoxCellEditor.setMultiselect(true);
      comboBoxCellEditor.setUseCheckbox(true);
      comboBoxCellEditor.setIconImage(GUIHelper.getImage("plus"));
    configRegistry.registerConfigAttribute(
        EditConfigAttributes.CELL_EDITOR,
        comboBoxCellEditor,
        DisplayMode.EDIT,
        EditorExample.COLUMN_TWELVE_LABEL);

    configRegistry.registerConfigAttribute(
        CellConfigAttributes.CELL_PAINTER,
        new ComboBoxPainter(GUIHelper.getImage("plus")),
        DisplayMode.NORMAL,
        EditorExample.COLUMN_TWELVE_LABEL);
```

Smooth Scrolling

Eine besondere optische Neuerung ist die Anpassung des Scroll-Verhaltens. Bisher wurde beim Scrollen zellenweise gesprungen. Dadurch wirkte NatTable kantig und es war nicht möglich, innerhalb von Zellen zu scrollen, die größer waren als der verfügbare Viewport. Dies wurde umgestellt auf pixelbasiertes Scrollen, wodurch sich das Handling von NatTable deutlich weicher anfühlt. Das Scrollen innerhalb sehr großer Zellen wird dadurch ebenfalls ermöglicht. Codeseitig sollte diese Anpassung keine Auswirkungen auf bestehenden Code haben, außer es wurden Anpassungen am *ViewportLayer* direkt vorgenommen.

Edit Refactoring

Der Code für die *Bearbeiten*-Funktionalität in NatTable wurde an vielen Stellen modifiziert. Ziel war die Stabilisierung des API, Vereinfachung der Erstellung neuer Editoren und das Ermöglichen neuer Funktionalitäten. Hierfür wurden das *ICellEditor*-Interface und der *AbstractCellEditor* überarbeitet und erweitert, um klare Strukturen vorzugeben. Außerdem wurden weitere Konfigurationsattribute hinzugefügt, um steuern zu können, ob ein Editor inline oder in einem Subdialog geöffnet werden soll, ob der Editor Mehrfachbearbeitung unterstützt und ob nach dem Bestätigen des bearbeiteten Werts der daneben liegende Editor geöffnet werden soll. Eigene Editoren können dieses Verhalten direkt über die entsprechenden Methoden statisch definieren. Die mit NatTable gelieferten Editoren unterstützen die Konfiguration über die neuen in Tabelle 1 aufgelisteten *EditConfigAttributes*.

In Listing 1 wird beispielhaft der *TextCellEditor* als Editor für die Spalte mit dem Label *COLUMN_TWO_LABEL* gesetzt und zusätzlich konfiguriert, dass nach dem Bestätigen des Werts der nebenliegende Editor direkt geöffnet werden soll. Außerdem wird eine benutzerdefinierte Nachricht hinterlegt, die im Subdialog für die Mehrfachbearbeitung dargestellt werden soll.

Auf Basis des *TextCellEditors* wurde der MultiLineTextCellEditor in die Sammlung der Standardeditoren aufgenommen. Dieser ermöglicht das Editieren von Fließtexten, wobei sich über die *lineWrap*-Eigenschaft des Editors das Verhalten definieren lässt, ob der Text automatisch umgebrochen oder stattdessen horizontales Scrollen im geöffneten Editor eingeschalten werden soll.

Der *ComboBoxCellEditor* und das zugehörige Custom Control NatCombo wurden ebenfalls komplett überarbeitet. Dadurch ist es jetzt möglich zu konfigurieren, ob die ComboBox freies Editieren im Text-Control, Mehrfachselektion und Checkboxen im Drop-down-Control unterstützen soll. Außerdem lässt sich das Icon, welches das Control als ComboBox kennzeichnet, sowie die Anzahl der sichtbaren Elemente im Drop-down-Control einstellen. Um immer alle Elemente ohne Scrollbar darzustellen, kann

für die Anzahl der Wert -1 gesetzt werden. In Listing 2 wird beispielhaft ein *ComboBoxCellEditor* konfiguriert, der freies Editieren und Mehrfachselektion erlaubt, Checkboxen im Drop-down und das Plus-Icon als ComboBox-Icon darstellt. Außerdem werden immer alle Elemente auf einmal dargestellt.

Die Darstellung des Texts im Text-Control des *ComboBoxCellEditors* bei Mehrfachselektion kann ebenfalls über die entsprechenden *multiselect*-Eigenschaften der Klasse eingestellt werden. Die Javadoc der Klasse gibt hierzu entsprechende Hinweise.

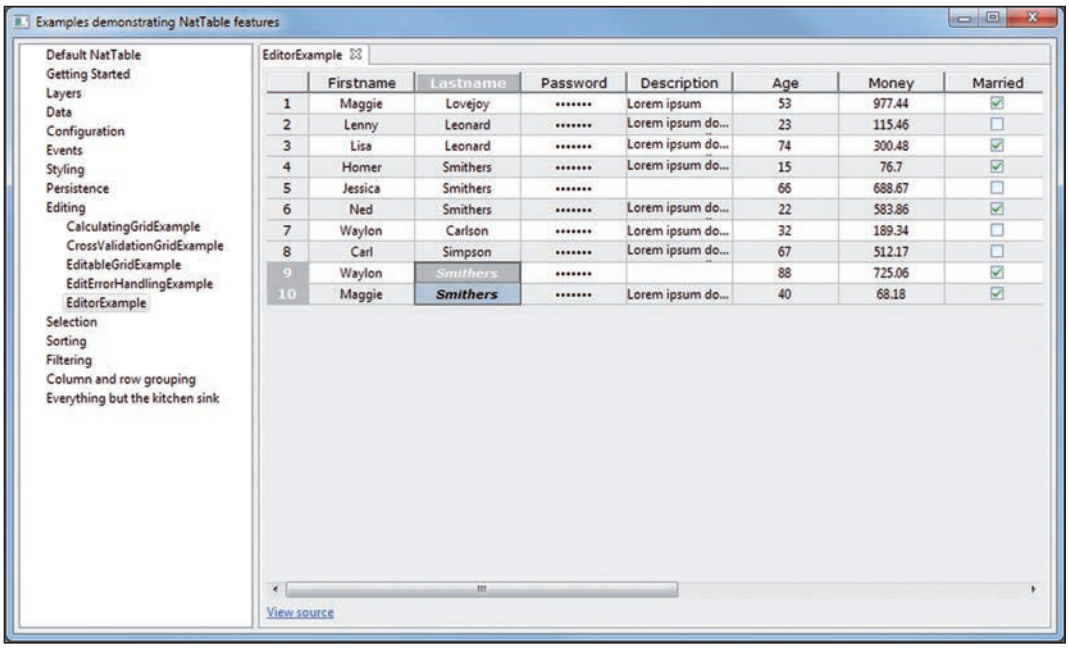

Abb. 3: NatTable EditorExample

Das Refactoring des Codes rund um das Editieren in NatTable ermöglicht nun außerdem die Definition von Editoren in NatTable die auf SWT/JFace-Dialogen aufsetzen, wie z. B. *FileDialog*. Um die Umsetzung zu vereinfachen, wurde der *AbstractDialogCellEditor* eingeführt und darauf aufbauend beispielhaft der *FileDialogCellEditor* implementiert.

Die verschiedenen Standardeditoren und deren Konfigurationsmöglichkeiten sind in der NatTable-Examples-App (zu finden unter [1] über den *Try it!*-Button) unter Editing | EditorExample dargestellt (**Abb. 3**). Über den *View source*-Link am unteren Ende des Beispiels kann der Quellcode eingesehen werden.

View-Management/Darstellungen verwalten

NatTable unterstützt seit Langem das Sichern und Wiederherstellen des Zustands über die Methoden *saveState(String, Properties)* und *loadState(String, Properties)*. Bisher wurden allerdings nicht alle Zustände gesichert, wie z. B. das Fixieren/Freeze von Zeilen und Spalten. Mit der aktuellen Version wurde darauf geachtet, dass alle Zustände gesichert und wiederhergestellt werden können. Außerdem wurde ein Dialog hinzugefügt, der es ermöglicht, unterschiedliche Zustände einer NatTable-Instanz zu verwalten. Hierzu muss der notwendige ILayerCommandHandler für das Öffnen des Dialogs registriert werden. Durch Ausführen des passenden ILayerCommand wird dem Handler dann mitgeteilt, dass der Dialog geöffnet werden soll. Hierfür kann beispielsweise ein Menüeintrag in das ColumnHeader-Menü hinzugefügt werden, der diese Aufgabe übernimmt. In Listing 3 sind diese beiden Aufgaben exemplarisch implementiert.

Über Rechtsklick auf den Tabellenkopf öffnet sich so ein Menü mit einem Menüeintrag, um Darstellungen zu verwalten. Nach Auswahl dieses Menüeintrags öffnet sich der in **Abbildung 4** dargestellte Dialog.

Listing 3

```
// register the command handler for opening the view management dialog
DisplayPersistenceDialogCommandHandler handler =
        new DisplayPersistenceDialogCommandHandler();
gridLayer.registerCommandHandler(handler);

// add a menu entry to the column header menu for opening the dialog
natTable.addConfiguration(
  new HeaderMenuConfiguration(natTable) {
    @Override
    protected PopupMenuBuilder createColumnHeaderMenu(NatTable natTable) {
      return super.createColumnHeaderMenu(natTable)
      .withStateManagerMenuItemProvider();
  }
});
```

Listing 4

```
ComboBoxFilterRowHeaderComposite filterRowHeaderLayer =
  new ComboBoxFilterRowHeaderComposite(
    bodyLayerStack.getFilterList(),
    bodyLayerStack.getGlazedListsEventLayer(),
    bodyLayerStack.getSortedList(),
    columnPropertyAccessor,
    columnHeaderLayer,
    columnHeaderDataProvider,
    configRegistry);
```

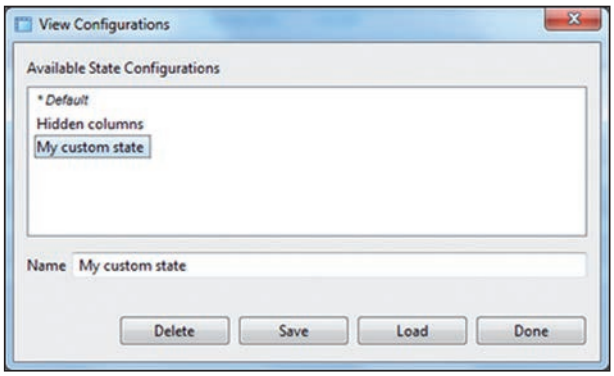

Abb. 4: Darstellungen verwalten Dialog

Excel like filter row

Ein neues Feature in NatTable ist die Excel-ähnliche Filterzeile. Sie besteht aus einer Combobox je Spalte, in der die Werte enthalten sind. Über Checkboxen können die Werte abgewählt werden, die in der Tabelle gefiltert werden sollen. Über das *ComboBoxFilterRowHeaderComposite* kann eine ähnliche Filterzeile in NatTable eingebunden werden (Listing 4), die dann wie in **Abbildung 5** dargestellt wird. Zur Erzeugung werden zwei Elemente benötigt: die *FilterList*, auf der die Filteroperationen ausgeführt werden sollen, und der *GlazedListsEventLayer*, um die ComboBoxen zur Laufzeit aktualisieren zu können, wenn Daten in der NatTable editiert werden. Daneben sind noch weitere Parameter notwendig, um die Erstellung des *ComboBoxFilterRowHeaderComposite* zu ermöglichen. Eine genaue Beschreibung der unterschiedlichen Konstruktoren und deren Parameter können der Javadoc entnommen werden. Der Code in Listing 4 geht davon aus, dass ein eigener *bodyLayerStack* implementiert wurde, der die notwendigen Informationen bereitstellt.

Dadurch erhält man die in **Abbildung 5** dargestellte Filterzeile.

Um die ComboBoxen zu befüllen, muss der Inhalt der gesamten Tabelle gescannt werden. Dies kann unter Umständen initial sehr viel Zeit in Anspruch nehmen, weshalb die Verwendung dieser speziellen Filterzeile nicht für sehr große Datenmengen empfohlen wird.

Das vollständige Beispiel ist im NatTable-Git-Repository unter dem Namen _563_ExcelLikeFilterRowExample zu finden. Da aktuell Arbeiten an der Dokumentation und den Beispielen vorgenommen werden, ist dieses Beispiel noch nicht in der auf der unter [1] zu findenden Beispielanwendung zu finden.

Listing 5

```
// first wrap the base list in a GlazedLists EventList and a FilterList
// so it is possible to filter
EventList<Person> eventList =
    GlazedLists.eventList(PersonService.getPersons(10));
FilterList<Person> filterList = new FilterList<Person>(eventList);

// use the GlazedListsDataProvider for some performance tweaks
final IRowDataProvider<Person> bodyDataProvider =
    new GlazedListsDataProvider<Person>(
      filterList,
        new ReflectiveColumnPropertyAccessor<Person>(propertyNames));

// create the IRowIdAccessor that is necessary for row hide/show
final IRowIdAccessor<Person> rowIdAccessor = new
IRowIdAccessor<Person>() {
  @Override
  public Serializable getRowId(Person rowObject) {
    return rowObject.getId();
  }
};

DataLayer bodyDataLayer = new DataLayer(bodyDataProvider);

// add a DetailGlazedListsEventLayer event layer that is responsible
// for updating the grid on list changes
DetailGlazedListsEventLayer<Person> glazedListsEventLayer =
    new DetailGlazedListsEventLayer<Person>(bodyDataLayer, filterList);

GlazedListsRowHideShowLayer<Person> rowHideShowLayer =
      new GlazedListsRowHideShowLayer<Person>(
        glazedListsEventLayer, bodyDataProvider, rowIdAccessor, filterList);
```

Row hide/show/reorder

Bisher war es nur möglich, Spalten per Drag and Drop in NatTable zu verschieben oder manuell ein-/auszublenden. Um die Reihenfolge von Zeilen zu beeinflussen, gab es nur die Sortierung, für das Ein-/Ausblenden das Filtern. Um an dieser Stelle das Set an Funktionalitäten zu komplettieren, wurde entsprechend dem ColumnHideShowLayer und dem ColumnReorderLayer

Registrierung eigener Editoren

Eigene NatTable-Editoren, die für frühere Versionen entwickelt wurden, müssen an das neue API angepasst werden. Diese Anpassungen sollten aufgrund der klaren Struktur und der Codedokumentation relativ einfach von der Hand gehen. Bei der Registrierung der eigenen Editoren wurde ebenfalls eine Vereinfachung hinzugefügt. Um einen Editor in NatTable zu öffnen, muss zum einen das Editieren generell eingeschalten sein, zum anderen muss auch ein Editor für die angeklickte Zelle konfiguriert sein. Bisher musste jeder einzelne Editor in einer Konfiguration über den *BodyCellEditorMouseEventMatcher* registriert werden. Dieser wurde deprecatet und durch den *CellEditorMouseEventMatcher* ersetzt, der nicht mehr den Typ, sondern nur das Vorhandensein eines Editors prüft. Durch die Konfiguration dieses Matchers in den *DefaultEditBindings* ist keine eigene Configuration mehr notwendig.

der RowHideShowLayer und der RowReorderLayer den Standardlayern in NatTable hinzugefügt. Werden diese beiden Layer dem eigenen Layerstack hinzugefügt, können auch Zeilen per Drag and Drop verschoben und über das Headermenü ein-/ausgeblendet werden.

Im Falle einer auf GlazedLists aufgebauten NatTable sollte für das Ein-/Ausblenden von Zeilen der *GlazedListsRowHideShowLayer* verwendet werden (Listing 5). Dieser arbeitet auf Basis von Zeilen-IDs und filtert die Zeilen intern. Die Definition der Zeilen-ID muss dabei über einen IRowDataProvider definiert werden, der für Benutzer des RowSelectionModels bereits bekannt sein sollte. Der ebenfalls neu hinzugefügte *DetailGlazedListsEventLayer* sorgt dafür, dass die ListEvents bei Änderungen in der GlazedLists transformiert und in die NatTable-Tabelle weitergeleitet werden, wobei alle Detailinformationen beibehalten werden (z. B. dass Zeile x gelöscht wurde). Der bereits seit Längerem existierende *GlazedListsEventLayer* im Gegensatz sammelt die Events aus der GlazedLists in Intervallen von 100 ms und verschluckt die Detailinformationen bei der Transformation in ein NatTable-Event. Beide Layer haben weiterhin Relevanz und müssen je nach Use Case in den Layerstack eingebunden werden, um die Events aus GlazedLists prozessieren zu können.

Weitere Neuigkeiten

Neben den erwähnten Punkten gibt es noch weitere zahlreiche Erweiterungen und Bugfixes. So wurden zum Beispiel der TextPainter und der VerticalTextPainter erweitert, um definieren zu können, ob die Höhe, die Breite oder beides anhand des Inhalts automatisch berechnet werden sollen. In diesem Zuge wurde auch der AutomaticRowHeightTextPainter eingeführt, der die Zellen nicht nur wachsen, sondern auch wieder schrumpfen lässt, wenn sich der verfügbare Platz in einer Zelle verändert (z. B. durch Größenänderung des Fensters). Das Beispiel hierzu finden Sie in der NatTable-Examples-App unter Configuration| AutomaticRowHeight-Example.

TickUpdates

Ein Feature, das bisher kaum bekannt ist und daher auch einige Fehler hatte, ist das so genannte TickUpdate. Dieses ist standardmäßig durch die *DefaultTickUpdateConfiguration*, die über die *DefaultSelectionLayerConfiguration* hinzugefügt wird, eingeschalten. Per Tastendruck auf die Plus- und Minus-Tasten des Numpads lassen sich numerische Werte in NatTable erhöhen oder verringern. Die Standardkonfiguration erhöht bzw. verringert den Wert in der Zelle dabei um 1. Über die Implementierung und Registrierung eines eigenen *ITickUpdateHandler* gegen das *TickUpdateConfigAttributes.UPDATE_HANDLER* lässt sich das Standardverhalten anpassen.

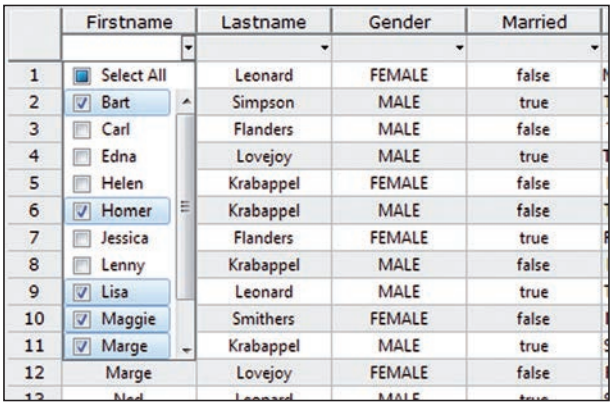

Abb. 5: Excel-ähnliche Filterzeile

Für die Definition von Header-Menüs wurde die AbstractHeaderMenuConfiguration eingeführt, die die Erzeugung vereinfachen soll, wobei auch ein Corner-Header-Menü berücksichtigt wurde. Somit können nun relativ einfach Header-Menüs in jeden Header eines Grids eingefügt werden. Daneben gibt es noch weitere zahlreiche kleinere Anpassungen, Erweiterungen und Bugfixes. Eine Beschreibung jedes einzelnen Punkts würde den Umfang dieses Artikels sprengen. Da aus Qualitätsgründen und Gründen der Nachvollziehbarkeit jede Anpassung im Code in einem Bugzilla Ticket getrackt wird, kann darüber eine detaillierte Beschreibung eingesehen werden. Auf der New-&-Noteworthy-Seite von NatTable [9] ist ein Link hinterlegt, der die konkrete Suchanfrage ausführt, sollten weitere Detailinformationen zum 1.0.0-Release gewünscht sein.

Dirk Fauth ist Senior Consultant Java/JavaEE/Eclipse bei der Be-One Stuttgart GmbH und seit mehreren Jahren im Bereich der Java-Entwicklung tätig. Er ist aktiver Committer und Co-Project Lead im Nebula-NatTable-Projekt und aktiver Contributor im Eclipse-4-Umfeld.

Links & Literatur

[1] http://eclipse.org/nattable/

[2] http://www.eclipse.org/projects/dev_process/development_process_2011.php#6_Development_Process

[3] http://wiki.eclipse.org/Development_Resources/Process_Guidelines/What_is_Incubation

[4] http://www.glazedlists.com/

[5] http://poi.apache.org/

[6] http://www.eclipse.org/orbit/

[7] http://download.eclipse.org/tools/orbit/downloads/

[8] http://download.eclipse.org/nattable/snapshots/

[9] http://eclipse.org/nattable/nandn/nandn_100.php