

# Entwicklung von Add-Ins für IBM Rational Rose

Dipl.-Inf. Manja Nelius  
Universität Rostock, Institut für Informatik  
Lehrstuhl Datenbanken und Informationssysteme

27. Mai 2005

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>3</b>
<b>2</b>	<b>Add-In-Typen</b>	<b>3</b>
<b>3</b>	<b>Entwicklung der Add-In-Komponenten</b>	<b>3</b>
3.1	Hauptmenü-Erweiterungen . . . . .	4
3.2	Datentypen . . . . .	4
3.3	Stereotypen . . . . .	4
3.4	Properties . . . . .	6
<b>4</b>	<b>Installation von Add-Ins</b>	<b>9</b>
4.1	Registry-Schlüssel . . . . .	9
4.2	Registry-Einträge . . . . .	9
4.3	Registry-Datei . . . . .	11
<b>5</b>	<b>Aktivierung und Deaktivierung von Add-Ins</b>	<b>11</b>

### **Zusammenfassung**

Dieser Artikel gibt einen kurzen Überblick darüber, wie Add-Ins für Rational Rose entwickelt und in das Programm eingebunden werden. Der Artikel ist angelehnt an das ausführliche Tutorial „[Using the Rose Extensibility Interface \(Version: 2001A.04.00\)](#)“ von IBM.

# 1 Einführung

Mit Hilfe von Add-Ins ist es möglich, Anpassungen und Automatisierungen von verschiedenen Rose-Features durch das Rose Extensibility Interface (REI) in einem Paket zu bündeln, das einfach in die Rose-Anwendungsumgebung integriert werden kann. Ein Add-In ist eine Kombination aus folgenden Komponenten:

- Hauptmenü-Einträge,
- Shortcut-Menü-Einträge\*,
- angepasste Spezifizierungen\*,
- Eigenschaften,
- Datentypen,
- Stereotypen,
- Online-Hilfe,
- kontextsensitive Hilfe\*,
- Event-Handling\*,
- Funktionalitäten durch Rose-Skripte oder Controls (OLE-Server)\*.

Ein Add-In bietet sich vor allen Dingen dann an, wenn man auf Rose-Events wie zum Beispiel `OnNewModel` und `OnAppInit` reagieren oder mit anderen Add-Ins interagieren möchte.

## 2 Add-In-Typen

Es gibt zwei Arten von Add-Ins in Rational Rose:

- **Basic**  
Ein Basic-Add-In definiert seine eigenen Reaktionen auf Events, um externe Skripte oder Programme auszuführen. Es nutzt nicht die Component View für die Code-Generierung. Ein Basic-Add-In kann nicht auf Events bezüglich der Code-Generierung reagieren.
- **Language**  
Ein Language-Add-In nutzt die Abbildung auf Komponenten aus, indem eine Zielsprache definiert wird. Es definiert außerdem seine eigenen Reaktionen auf Events bezüglich der Code-Generierung und der Round-Trip-Engineering-Integration. Die Ereignisse der Letztgenannten beinhalten `OnGenerateCode`, `OnBrowseBody`, und `OnBrowseHeader`. Weiterhin unterstützen Language-Add-Ins nutzerdefinierbare Datentypen und das Overriding von Default-Spezifikationen.

## 3 Entwicklung der Add-In-Komponenten

In den folgenden Abschnitten wird beschrieben, wie die einzelnen Komponenten eines Add-Ins entwickelt werden<sup>†</sup>. Dazu muss sich der Nutzer vorher darüber klar werden, welche Funktionalitäten sein Add-In bereitstellen soll, und danach die notwendigen Komponenten definieren.

---

\*Auf diese Add-In-Komponenten wird in diesem Artikel nicht näher eingegangen.

<sup>†</sup>Es werden nicht alle Komponenten vorgestellt. Für eine vollständige Dokumentation wird auf das Tutorial von IBM verwiesen.

### 3.1 Hauptmenü-Erweiterungen

Jedes Add-In kann zusätzliche Menü-Einträge mittels einer Menü-Datei (\*.mnu) zum Rose-Hauptmenü hinzufügen. Folgendes einfache Beispiel fügt in das Tools-Menü von Rose eine Trennlinie gefolgt von einem Untermenü namens „My Add-In“ hinzu, welches zwei Menü-Einträge beinhaltet. Die erste Menü-Aktion ruft die Eingabeaufforderung auf, die zweite Aktion öffnet den Taschenrechner.

```
Menu Tools
{
  Separator
  menu "My Add-In"
  {
    option "Menu Entry 1" { exec cmd }
    option "Menu Entry 2" { exec calc.exe }
  }
}
```

Weitere Informationen über die Syntax von Menü-Dateien sind im IBM-Tutorial im Kapitel „[Customizing Rose Main Menus](#)“ zu finden.

### 3.2 Datentypen

Ein Language-Add-In kann eine Kollektion von Standarddatentypen bereitstellen, die dem Nutzer als Auswahl zur Typisierung von Attributen, Parametern, usw. in den Rose-Spezifikationsdialogen angezeigt werden sollen. Diese Datentypen werden in der Registry im entsprechenden Schlüssel des Add-Ins durch den Eintrag `FundamentalTypes` festgelegt\*.

### 3.3 Stereotypen

Stereotypen erlauben es, das Aussehen von verschiedenen Modell-Elementen auf deren Bedeutung bezüglich des Add-Ins anzupassen. Diese Anpassungen können im einfachsten Fall zusätzliche Zeichenketten (zum Beispiel `<<Special Class>>`) oder Icons für die Toolbar-Buttons, Browser-Buttons oder den Diagramm-Editor sein.

Ein Add-In kann nun eine Kollektion von Stereotypen und zusätzliche Icons bereitstellen, um diese Stereotypen zu repräsentieren. Diese selbstdefinierten Stereotypen werden bei der Aktivierung des Add-Ins zu den Standardstereotypen von UML hinzugefügt und in Rose verfügbar gemacht. Selbstdefinierte Stereotypen ersetzen dabei nicht die vordefinierten. Der Speicherort der definierten Stereotypen wird in der Registry durch den Eintrag `StereotypeCfgFile` festgelegt\*.

Zu einem Stereotyp können also Icons oder Text bereitgestellt werden. Stereotypen sind auf folgende Modell-Elemente<sup>†</sup> anwendbar:

- Association,
- Attribute,

---

\*Siehe Kapitel 4.

<sup>†</sup>In der Auflistung werden die englischen Begriffe verwendet, wie sie auch in der Rose-Anwendung erscheinen.

- Class,
- Component,
- Component package,
- Connection,
- Dependency,
- Device,
- Generalization,
- Logical package,
- Operation,
- Processor,
- Use case,
- Use-case package.

Die Stereotypen werden in einer `.ini`-Datei definiert. Diese Datei beinhaltet folgende Informationen:

**[General]**

Dieser Abschnitt beinhaltet Add-In-spezifische Angaben wie den Namen des Add-Ins und ob es sich um ein Language-Add-In handelt.

**[Stereotyped Items]**

Dieser Abschnitt ist eine Art Inhaltsverzeichnis für die Stereotypen. Er umfasst eine Liste von stereotypisierten REI-Objekten, zum Beispiel `Class:Control`, `Component:DLL`, `Operation:Set`.

**[REI Item:Stereotype name]**

Dieser Abschnitt beinhaltet Angaben zu jedem Stereotyp, einschließlich der optionalen Icon-Dateien und -Parameter.

Eine `stereotypes.ini`-Datei zur Definition von Text-Stereotypen ohne eigene Icons sieht zum Beispiel folgendermaßen aus:

**[General]**

`ConfigurationName=MyStereotypes`  
`IsLanguageConfiguration=Yes`

**[Stereotyped Items]**

`Class:Interface`  
`Component:DLL`  
`Component:ActiveX`  
`Component:Application`

**[Class:Interface]**

`Item=Class`  
`Stereotype=Interface`

```
[Component: DLL]
Item=Component
Stereotype=DLL
```

```
[Component: ActiveX]
Item=Component
Stereotype=ActiveX
```

```
[Component: Application]
Item=Component
Stereotype=Application
```

Dabei bedeuten die einzelnen Angaben folgendes:

- **ConfigurationName**  
Dies ist der Name des Add-Ins oder der Name, der zur Wartung genutzt wird.
- **IsLanguageConfiguration**  
Yes, wenn das Add-In ein Language-Add-In ist, andernfalls No. Diese Information belegt die Stereotypen mit einer Bedingung, so dass diese nur erscheinen, wenn die Sprache des Modell-Elements in Rose mit dem Wert des **ConfigurationName** übereinstimmt.
- **Item**  
Das Modell-Element, für welches ein Stereotyp definiert wird.
- **Stereotype**  
Die Text-String des Stereotypen. Das ist der Text, der zwischen den eckigen Klammern angezeigt wird (<< >>).

Beispiele von komplexeren **ini**-Dateien zur Spezifikation von Stereotypen mit Icons sind im Tutorial von IBM zu finden.

### 3.4 Properties

Rose-Modell-Eigenschaften (Properties) ermöglichen die Erweiterung eines Rose-Modells durch Einführung zusätzlicher Eigenschaften mit ihren Werten. Man kann eigene Tools (wie zum Beispiel ein Tab im Spezifikationsdialog), Sets\* und Eigenschaften hinzufügen, um für das Add-In relevante Informationen zu jedem Modell-Element zu speichern. Diese Informationen können auch zur Bestimmung benutzt werden, wann Funktionalitäten in dem Add-In auftreten sollen.

Properties werden zu Rose-Items durch eine Property-Datei (**.pty**) hinzugefügt. Jedes Add-In kann seine eigene Property-Datei bereitstellen, die einen Namensraum für ihre Properties und einen Tab im Spezifikationseditor definiert. Ein Tab entspricht dabei einem Tool mit Sets und Properties. Zu jedem Add-In kann nur jeweils eine Property-Datei angelegt werden, aber es ist möglich, mehrere Tools, Sets und Properties innerhalb dieser Datei zu definieren. Diese Datei wird automatisch mit dem Add-In aktiviert und deaktiviert. Auch wenn die Property-Datei deaktiviert

---

\*Eine Menge von Eigenschaften zusammengefasst unter einem Namen.

ist, bleiben die definierten Properties im Modell gespeichert. Um ein Tab zu verbergen, kann der Nutzer das entsprechende Add-In in Rose deaktivieren.

Bei der Definition von Tools/Tabs muss darauf geachtet werden, dass deren Name eindeutig für jedes Add-In ist. Rose kann keine Konflikte erkennen. Außerdem muss für jedes definierte Tool ein `default-Set` existieren. Weiterhin kann man Properties durch das REI hinzufügen, löschen und klonen.\*

Der Inhalt einer Property-Datei muss dem nachfolgenden Format entsprechen. In der Darstellung werden dabei die vom Nutzer zu definierenden Angaben schräg dargestellt. Die einzelnen Elemente werden im Anschluss erklärt.

```
# Beginn der Versionsinformationen
(object Petal
  version      number
  _written     "add-in name"
  charSet     0)
# Ende der Versionsinformationen

# Beginn der ersten Tool-Definition
(list Attribute_Set

  # Tool-Konfiguration
  (object Attribute
    tool      "tool"
    name      "propertyID"
    value     "809135966")

  # Beginn der ersten Set- und Modell-Element-Liste
  (object Attribute
    tool      "tool"
    name      "set__model element"
    # Beginn der Eigenschaftsliste
    value     (list Attribute_Set
      # Definition der ersten Eigenschaft
      (object Attribute
        tool      "tool"
        name      "property"
        value     datatype)

      # Definition der nächsten Eigenschaft
      (...))
    )
  # Ende der Eigenschaftsliste
)
```

---

\*Weitere Erläuterungen dazu sind im IBM-Tutorial im Kapitel „Managing Default Properties“ zu finden.

```

# Ende der Set- und Modell-Element-Liste

# Beginn der nächsten Set- und Modell-Element-Liste
(...)
# Ende der nächsten Set- und Modell-Element-Liste
)
# Ende der ersten Tool-Definition

# Beginn der nächsten Tool-Definition;
(...)
# Ende der nächsten Tool-Definition

(...)

```

Die Property-Datei besteht aus folgenden Elementen:

- **Kommentare**  
Ein # am Anfang der Zeile zeigt an, dass es sich um eine Kommentarzeile handelt.
- ***number***  
Die Petal-Versionsnummer, die der Rose-Version entspricht, für die das Add-In entwickelt wird.\*
- ***add-in name***  
Der Name des Add-Ins.
- ***tool***  
Name des Tools. Es können mehrere Tools innerhalb einer Property-Datei definiert werden, die alle einen eindeutigen Namen bekommen müssen.
- ***value***  
Für jedes Tool sollte der gleiche Wert (*809135966*) benutzt werden. Wenn Probleme auftauchen, sollte der Wert um 1 erhöht werden.
- ***set\_model element***  
Name des Sets und des Modell-Elements, zum Beispiel *default\_Class* oder *CompilerV2.0\_Project*. Es können mehrere Sets und Modell-Elemente pro Tool definiert werden. Gültige Modell-Elemente sind:
  - Association,
  - Attribute,
  - Category,
  - Class,
  - Has,
  - Inherit,

---

\*Um herauszufinden, welche Nummer dies ist, öffnet man eine Modell-Datei (.mdl), die mit der gleichen Rose-Version erstellt wurde, in einem Texteditor.

- Module-Spec,
  - Module-Body,
  - Operation,
  - Param,
  - Project,
  - Role,
  - Subsystem,
  - Uses.
- *property*  
Der Name der Eigenschaft, zum Beispiel `minCount`.
  - *datatype*  
Der Default-Wert für den Datentyp der Eigenschaft, zum Beispiel, wenn die Eigenschaft
    - ein Integer-Wert ist, dann könnte der Default-Wert 0 sein.
    - ein String ist, dann könnte der Default-Wert "" oder "unbekannt" sein.
    - ein Wahrheitswert ist, dann könnte der Default-Wert TRUE sein.

Diese in der `.pty`-Datei definierten Tools und Properties werden in der Registry durch den Eintrag `PropertyFile*` in der Rose-Anwendung verfügbar.

## 4 Installation von Add-Ins

In den folgenden Abschnitten wird erläutert, wie ein Add-In in die Rose-Anwendung eingebunden wird. Dazu müssen zunächst alle dazu nötigen Komponenten entwickelt und dann über verschiedene Registry-Einträge registriert werden.

### 4.1 Registry-Schlüssel

Nachdem ein Add-In entwickelt wurde, sind mehrere Registry-Einträge notwendig, um das Add-In zu aktivieren. Diese Einträge sind unterhalb eines Schlüssels angesiedelt, der den Namen des Add-Ins repräsentiert. Das folgende Beispiel registriert ein Add-In namens „MyAddIn“:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\Rose\AddIns\MyAddIn]
```

Der Schlüssel zu einem Add-In sollte wie im Beispiel im Ordner `Rose\AddIns` angelegt werden.

### 4.2 Registry-Einträge

Die folgenden Einträge in der Registry sind verfügbar, um ein Add-In mit seinen Komponenten zu deklarieren<sup>†</sup>:

---

\*Siehe Kapitel 4.

<sup>†</sup>Es werden nur die Einträge aufgeführt, die für die in Kapitel 3 vorgestellten Komponenten notwendig sind. Eine vollständige Auflistung bietet das Tutorial von IBM.

- **Active**  
Gibt an, ob das Add-In aktiviert oder deaktiviert ist. Kann in der Rose-Anwendung auch über den Add-In-Manager geändert werden. Default-Wert: "Yes".
- **Company**  
Name der Firma, die das Add-In entwickelt hat. Beispiel: "Custom Software, Inc.".
- **Copyright**  
Gibt das Copyright-Datum des Add-Ins an. Beispiel: "©2000-2001".
- **FundamentalTypes**  
Eine String-Liste von Datentypen, die in der Auswahlliste für Attribute erscheinen sollen, wenn das Add-In aktiviert ist. Diese Angabe wird für alle Language-Add-Ins benötigt. Bei den Angaben wird zwischen Groß- und Kleinschreibung unterschieden. Beispiel: "LOGICAL;CHAR;REAL".
- **InstallDir**  
Verzeichnis, in dem das Add-In installiert ist. Beispiel: "d:\MyAddIn".
- **LanguageAddIn**  
Gibt an, ob es sich um ein Language-Add-In handelt, das das Komponenten-Abbildungs-Feature benutzen möchte. Beispiel: "Yes".
- **MenuFile**  
Name der Menü-Datei (.mnu), die die Erweiterungen des Rose-Menüs definiert. Diese Datei muss sich im Installationsverzeichnis des Add-Ins befinden. Beispiel: "myaddin.mnu".
- **PropertyFile**  
Name der Property-Datei (.pty) für das Add-In. Diese Datei muss sich im Installationsverzeichnis des Add-Ins befinden. Der Eintrag ist notwendig, wenn das Add-In neue Eigenschaften einbinden soll. Beispiel: "myprops.pty".
- **StereotypeCfgFile**  
Spezifiziert eine Konfigurationsdatei (.ini) für neue Stereotypen. Diese Datei muss sich im Installationsverzeichnis des Add-Ins befinden. Der Eintrag ist notwendig, wenn das Add-In eigene Stereotypen bereitstellen soll.
- **ToolDisplayName**  
Spezifiziert den Tool-Namen des Add-Ins, der im Properties-Tab und in der Sprachenauswahlliste von Rose angezeigt werden soll. Dieser Name kann sich von dem unterscheiden, der in der .pty-Datei benutzt wurde. Dies ist kein notwendiger Eintrag. Wenn dieser Eintrag nicht spezifiziert ist, wird der Wert von ToolName im Properties-Tab und in der Sprachenauswahlliste angezeigt. Wenn der Eintrag spezifiziert wird, ist dies der Name, der einer Komponente zugewiesen wird. Beispiel: "myLanguage" ist der ToolName für das Add-In, aber "My Proprietary Language" ist der ToolDisplayName.
- **ToolList**  
Dieser Eintrag gibt eine Liste von zusätzlichen Tools oder Property-Seiten an, die vom Add-In

definiert sind. Der Eintrag ist nur notwendig, wenn das Add-In mehr als eine Property-Seite bereitstellt. Beispiel: "Tool1;Tool2".

- **ToolName**  
Legt den Tool-Namen des Add-Ins fest, der dem Tool-Namen (der Name, der einer Komponente zugewiesen wird) in der .pty-Datei des Add-Ins entsprechen muss. Wird durch den ToolDisplayName überschrieben.
- **Version**  
Versionsnummer des Add-Ins (nicht die von Rose). Beispiel: "1.0.2".

### 4.3 Registry-Datei

Die vorgestellten Einträge können zur Registry durch eine Registrierungsdatei (.reg) hinzugefügt werden, die einfach mit einem Texteditor erstellt werden kann. Ein Beispiel für eine Registrierungsdatei sieht folgendermaßen aus:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\Rose\AddIns\MyAddIn]

"Active"="Yes"
"LanguageAddIn"="Yes"
"Version"="1.0"
"PropertyFile"="myProperties.pty"
"MenuFile"="myMenu.mnu"
"StereotypeCfgFile"="myStereotypes.ini"
"InstallDir"="d:\ProgramFiles\Rational\Rose\MyAddIn"
"ToolName"="My Add-In"
```

Die Registry wird dann durch einen Doppelklick auf die .reg-Datei aktualisiert.

## 5 Aktivierung und Deaktivierung von Add-Ins

Nachdem ein Add-In installiert wurde, kann es im aktivierten oder deaktivierten Zustand sein. Dieser Zustand kann über den Add-In-Manager in Rose geändert werden.

Wenn ein Add-In deaktiviert ist, ist es nicht deinstalliert, aber alle vom Add-In hinzugefügten Komponenten (zum Beispiel Menü-Einträge, Property-Tabs, Stereotypen, usw.) werden aus der Rose-Anwendung entfernt.