Eclipse Plug-Ins und Notes Teil 1

Durch den Aufbau der Notes Produkte auf Eclipse steht den Anwendungsentwicklern die Option zur Verfügung, neben den bekannten Anwendungsentwicklungen im Notes Bereich mit dem Domino Designer Erweiterungen in Notes durch das Bereitstellen von Plug-Ins zur realisieren.

Plug-Ins sind eine Technologie, die nicht nur IBM dazu einsetzt, um Anwendungen und Funktionen schnell erweiterbar zu machen und damit die Flexibilität in den verschiedensten Bereichen steigern zu können.

Plug-Ins kennen wir heute in der IBM i Umgebung von den verschiedensten Produkten. WDSC und RDI sind als Entwicklungsumgebungen Plug-in basiert. Beide Produkte basieren, wie auch andere Rational Entwicklungswerkzeuge auf Eclipse.

Solche Plug-Ins bilden unter anderem auch die Basis für die gesamte Realisierung der Notes Funktionen (und auch der Darstellungsbereiche) auf Basis von Eclipse.

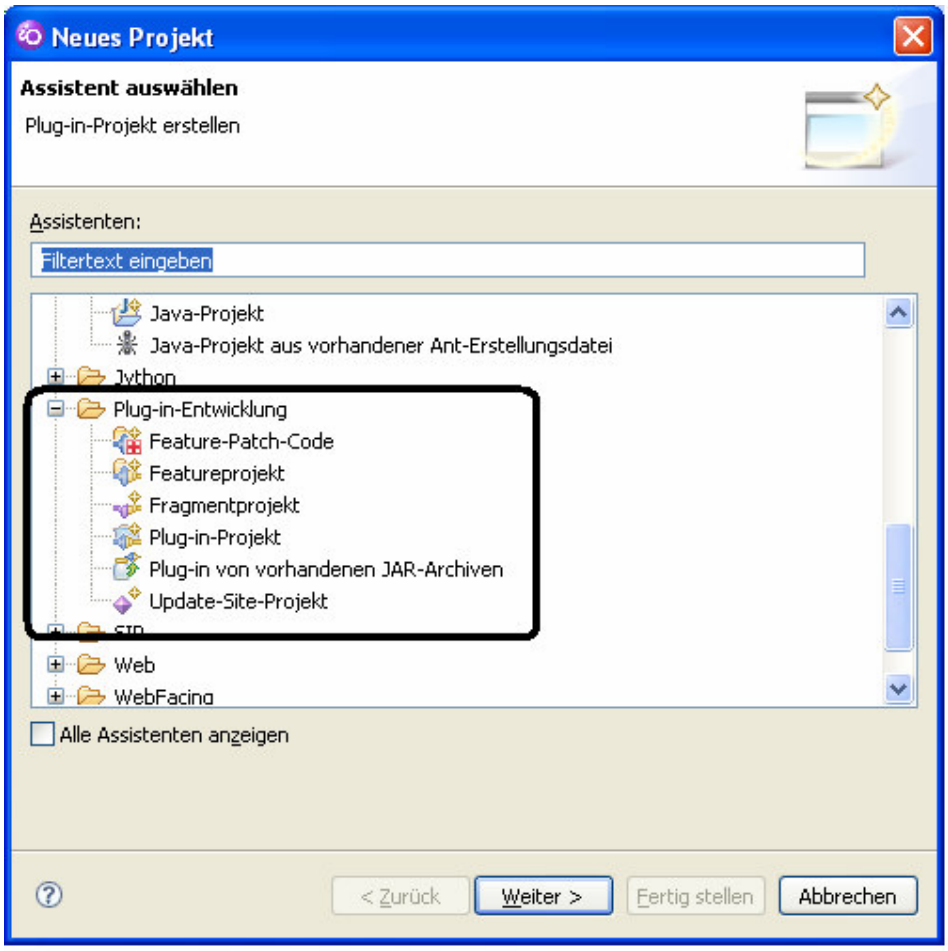
Ein Plug-In gestattet das individuelle Erweitern von bestehenden Anwendungen, ohne die gesamte Anwendung ersetzen zu müssen. Durch das selektive Hinzufügen von Erweiterungen lassen sich so geschickt bei Bedarf Erweiterungen an bestehenden Applikationen implementieren.

Plug-Ins müssen nicht von IBM zur Verfügung gestellt werden. Vielmehr stehen mit den Entwicklungswerkzeugen auch Funktionen zur Verfügung, die man – einen gewissen Kenntnisstand vorausgesetzt – zur Erstellung eigener Plug-Ins verwenden kann. Individuelle Erweiterungen von ansonsten Standardanwendungen wie zum Beispiel auch Lotus Notes lassen sich folglich ideal als Plug-Ins erstellen. Unabhängig davon, woher ein Plug-In kommt, kann dieses bei Bedarf (unter Berücksichtigung etwaiger Lizenzbestimmungen) auch angepasst werden.

Diese Erweiterungen werden in Eclipse in einem eigenen Plug-In Projekt beschrieben. Plug-in Projekte sind nicht an die Lotus Spezifika gebunden, sondern als allgemein anwendbare Werkzeuge und Funktionen für die Entwicklung und Verwaltung von Plug-Ins einsetzbar.

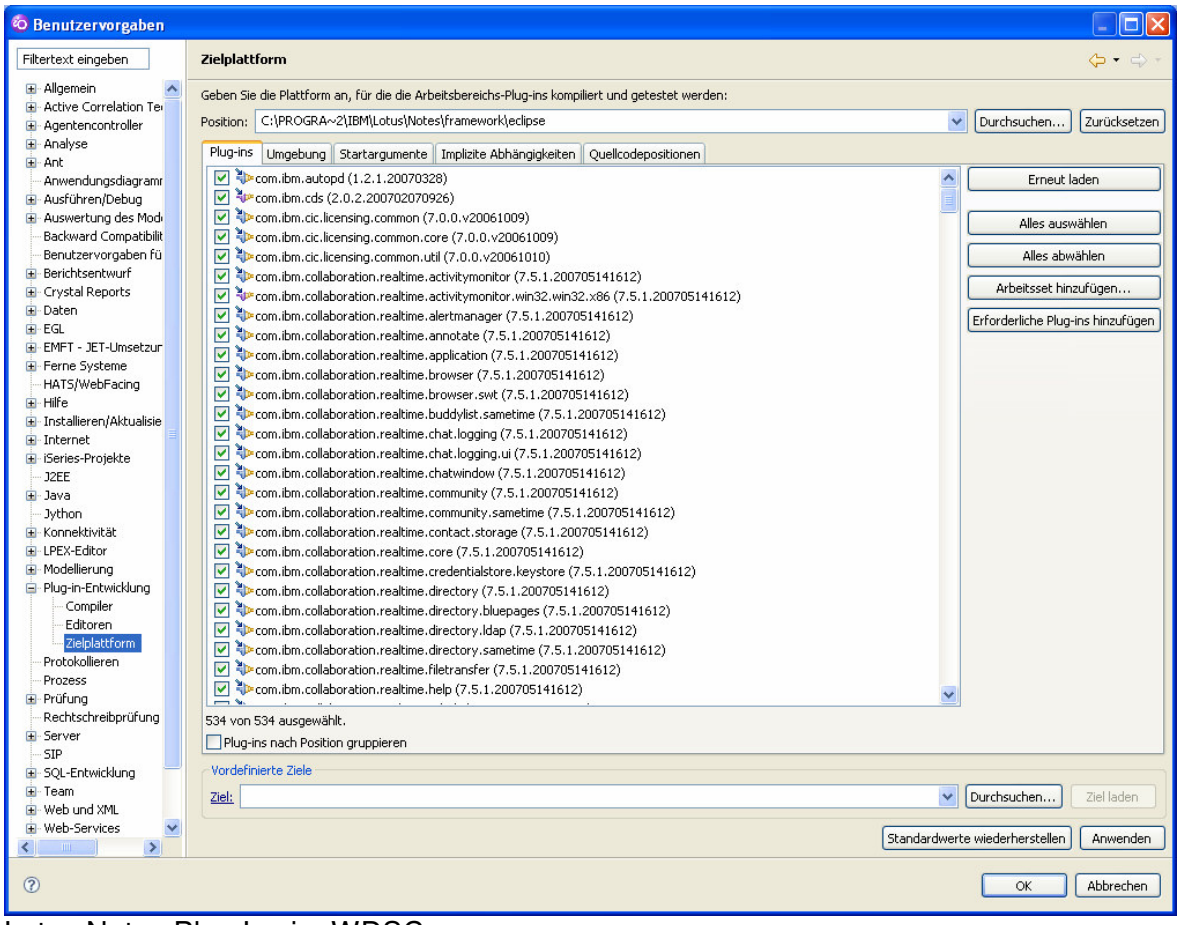
Ein solches Plug-In Projekt besteht neben dem administrativen Rahmen in Form des Projektes aus unterschiedlichen Komponenten, die neben dem Programmcode, welcher die neue Funktion darstellt, auch die Voraussetzungen und die Implementierung mittels der verschiedenen benötigten Komponenten beschreibt. Diese Bereiche werden mittels eines Manifests definiert.

Eclipse bietet bei dem Erstellen von Plug-Ins verschiedene Assistenten, mit deren Hilfe das Generieren neuer Funktionen erleichtert wird. Die folgende Abbildung zeigt einen Ausschnitt der Plug-In Assistenten im WDSC.



Plug-In Assistenten

Schaut man sich die Standard Plug-Ins an, welche in der Version 8 von Lotus Notes zur Verfügung stehen, dann hat man einen groben Überblick über die Mächtigkeit des Produkts – denn mit mehr als 530 Plug-Ins steht eine große Anzahl von Funktionen bereit. Diese Plug-Ins bestehen nicht nur aus reinen Lotus Notes Funktionen, sondern beinhalten auch Lotus Expeditor Funktionen wie auch Lotus Notes APIs.



Lotus Notes Plug Ins im WDSC

Diese lassen sich bei Bedarf um komplett eigene Plug-Ins (beispielsweise Anzeigen von Anwendungsprogrammen) erweitern. Außerdem besteht die Option, bestehende Plug-Ins auf eigene Bedürfnisse hin anzupassen.

In diesem Abschnitt möchte ich Ihnen zeigen, wie man die Mailfunktionen mit Hilfe von Eclipse Werkzeugen anpassen kann. Auf diese Weise lassen sich fehlende Funktionen relativ leicht in Notes abbilden. So beispielsweise auch Regeln, nach denen die Mailzuordnungen in Ordner automatisiert werden können – auch ohne einen Domino Server.

Diese Lotus Notes Erweiterung werden wir in Form eines eigenen Plug-Ins abbilden – also eine funktionale Ergänzung zu dem Standard-Umfang von Lotus Notes – nicht mit dem Domino Designer, sondern mit Eclipse (bzw. WDSC) umgesetzt.

WDSC oder auch dessen Nachfolger RDI (Rational Developer für System i) sind die von IBM empfohlenen Entwicklungswerkzeuge, die beispielsweise auch für die RPG Anwendungsentwicklung als Ersatz bzw. Nachfolger von SEU/PDM & Co. eingesetzt werden können. Das Ziel, eine Anwendungsentwicklungsumgebung für die

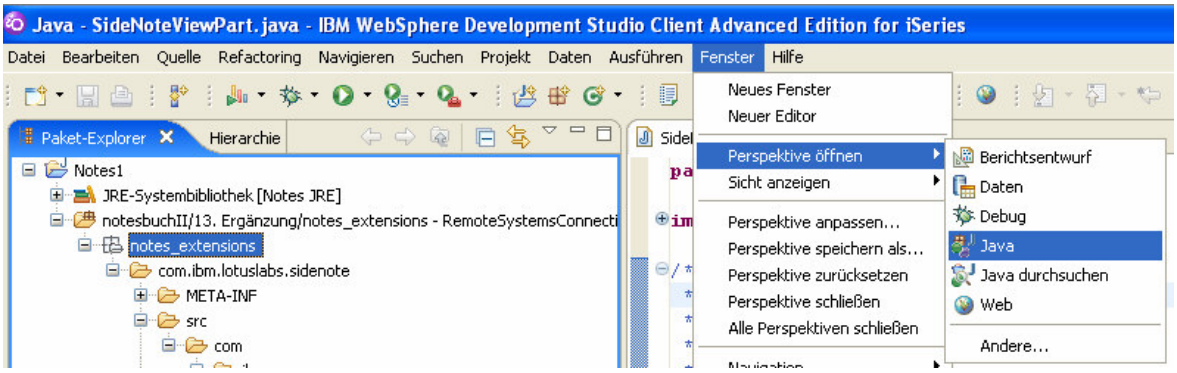
unterschiedlichsten Anforderungen zur Verfügung zu haben, ist damit auch in den Bereich der Notes-Anwendungsentwicklung vorgedrungen.

Leider hat IBM von dem Übergang der WDSC Produkte hin zu den unter Rational zusammengefassten Entwicklerwerkzeugen einige Funktionen aus dem Basispaket der Entwicklerwerkzeuge entfernt. Derzeit (Stand Herbst 2009) ist WDSC das umfangreichere Paket, mit dem Anwendungsentwickler relativ viele Funktionen nutzen können, die leider in der neuesten Version der Entwicklerwerkzeug im Rational Umfeld nicht mehr oder nur gegen entsprechende Gebühr zur Verfügung stehen.

Aus diesem Grund verwende ich in diesem Beispiel auch noch die WDSC Version und nicht die neuere RDI Funktionen.

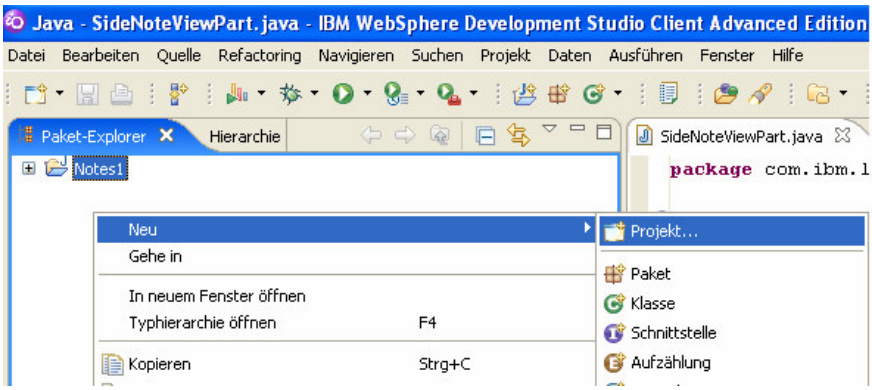
Die Voraussetzungen für das Erstellen eines Lotus Notes Plug-Ins beginnen mit dem Definieren der Zielplattform in den Benutzervorgaben von Eclipse – ein Beispiel dazu finden Sie in der vorhergehenden Abbildung.

Für das Erstellen eines neuen Lotus Notes Plug-Ins verwenden wir die Java-Perspektive. Sollten Sie diese nicht bereits eingestellt haben, wechseln Sie die Perspektive. Eclipse bietet dazu verschiedene Möglichkeiten – beispielsweise die Menüoption „Fenster / Perspektive öffnen / Java“.



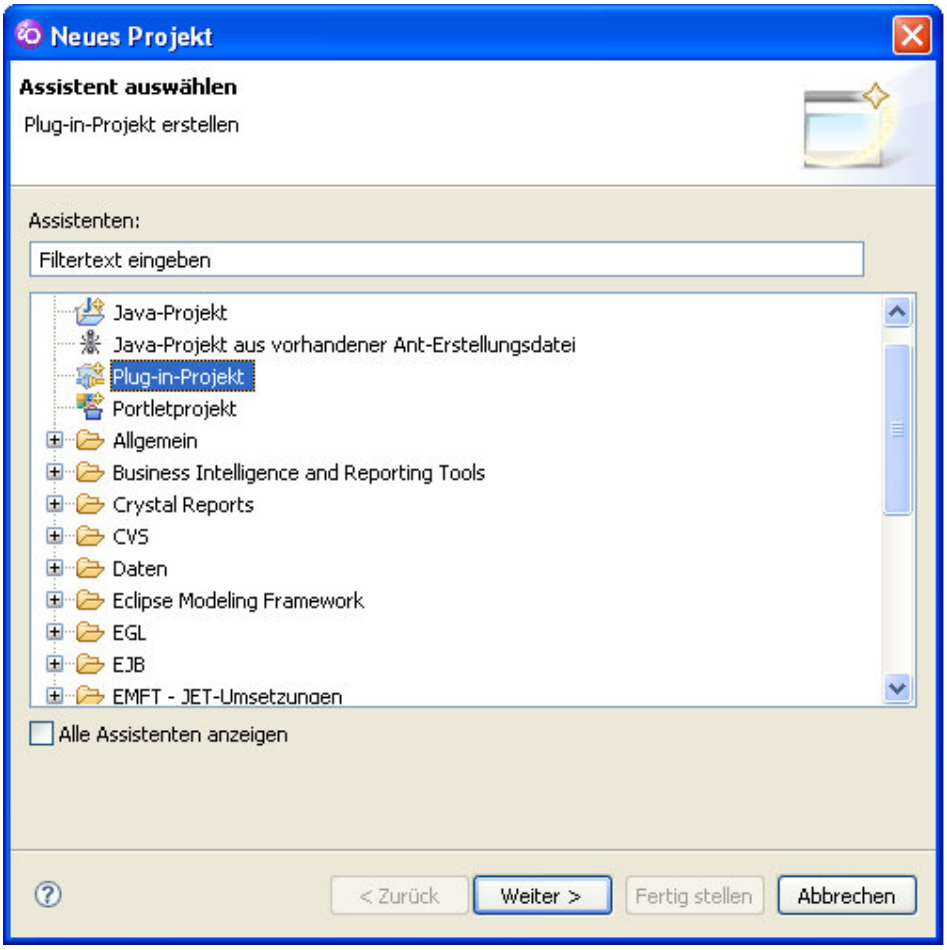
Perspektive öffnen

Die Basis für das Erstellen eines Notes Plug-Ins innerhalb von Eclipse bildet das Projekt. Ein solches Projekt stellt eine verwaltbare Einheit in Eclipse dar, in welcher alle Bestandteile dieses Projekts – in unserem Fall eines Plug-Ins – enthalten sind. Ein solches neues Projekt erstellen wir, indem wir in dem linken Fensterbereich „Paketexplorer“ mit der rechten Maustaste klicken und die Option „Neu / Projekt“ auswählen.



Neues Projekt erstellen

Eclipse bietet eine Vielzahl verschiedener Assistenten für das Erstellen von Programmen, Webseiten, Projekten und auch Plug-Ins. In der Folgeanzeige selektieren wir den Assistenten für die Erstellung von Plug-Ins oder genauer, für ein Plug-In Projekt.



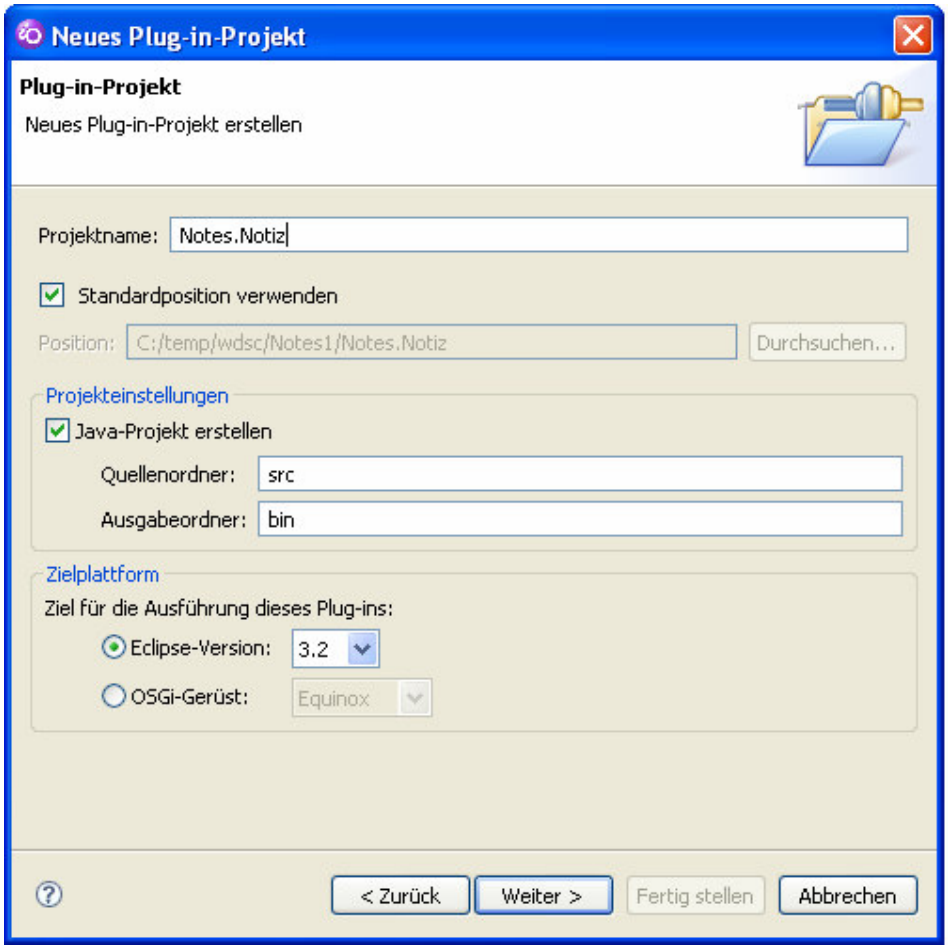
Plug-In Projekt

Die Auswahl wird mit einem Klick auf die Schaltfläche „Weiter“ bestätigt.

Als nächstes müssen wir einen Projektnamen angeben. Das Projekt nennen wir „Notes.Notiz“. Dies ist ein frei vergebbarer Name und kann bei Bedarf auch individuell angepasst werden.

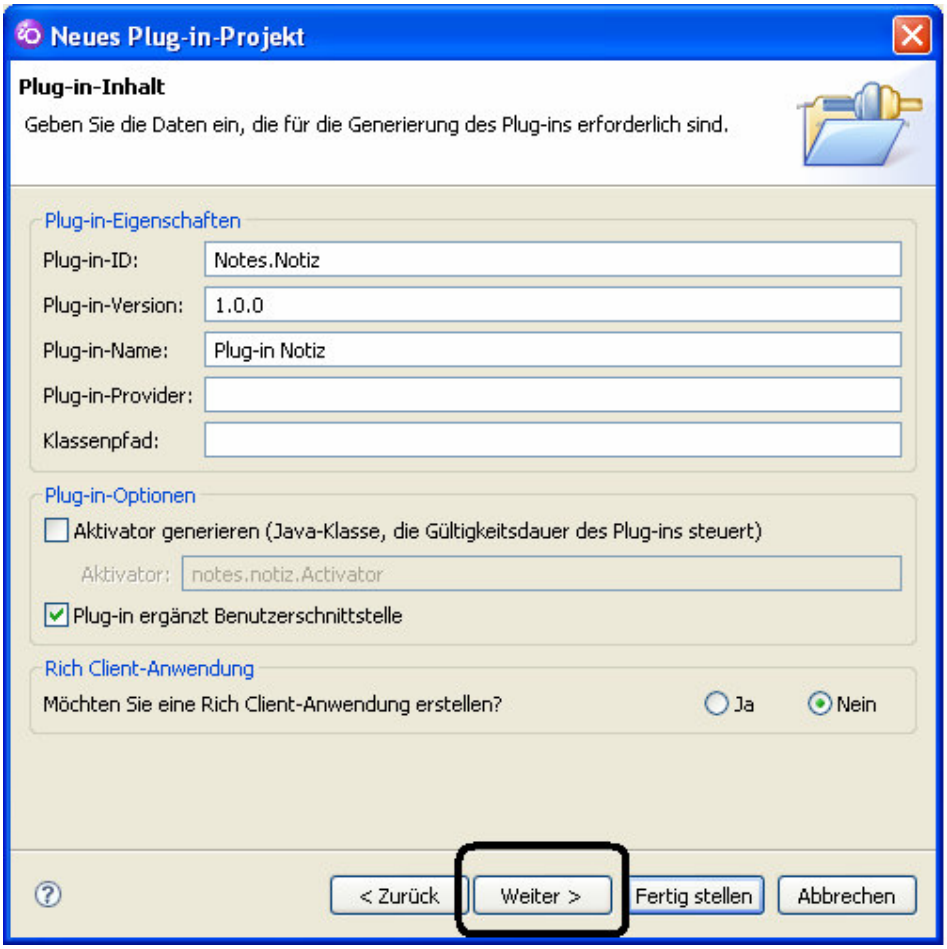
☝ Anmerkung

Wir werden hier ein Beispiel von IBM als Demo-Plug-in verwenden. Dies hat den Vorteil, dass wir uns nicht um den Inhalt des Programmcodes kümmern müssen. Dieser liegt in Form verschiedener Java Klassen vor und lässt sich relativ leicht in ein neues Projekt einbinden. Die dazu erforderlichen Klassen können Sie von den Webseiten der IBM herunterladen – finden diese aber auch als Kopie auf der mitgelieferten CD in dem Verzeichnis „



Projektnamen vergeben

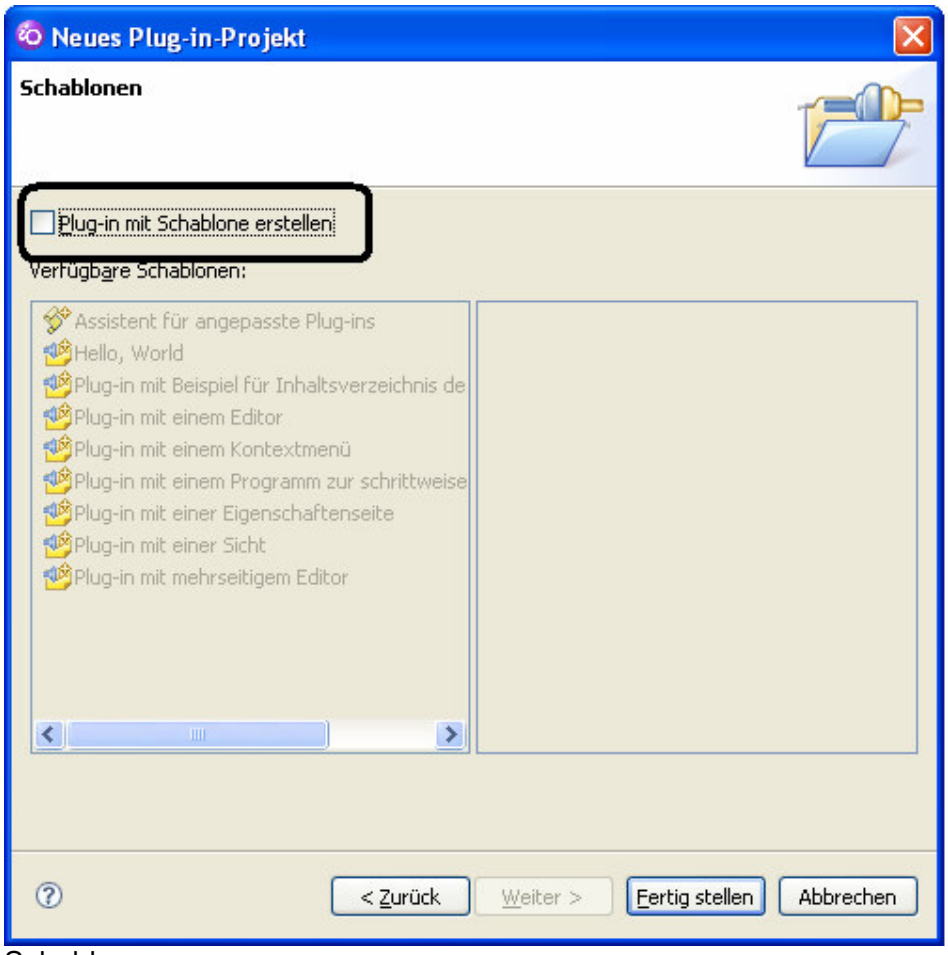
Mit einem Klick auf „Weiter“ gelangen wir in die Eigenschaftsanzeige des neuen Projektes.



Plug In Eigenschaften

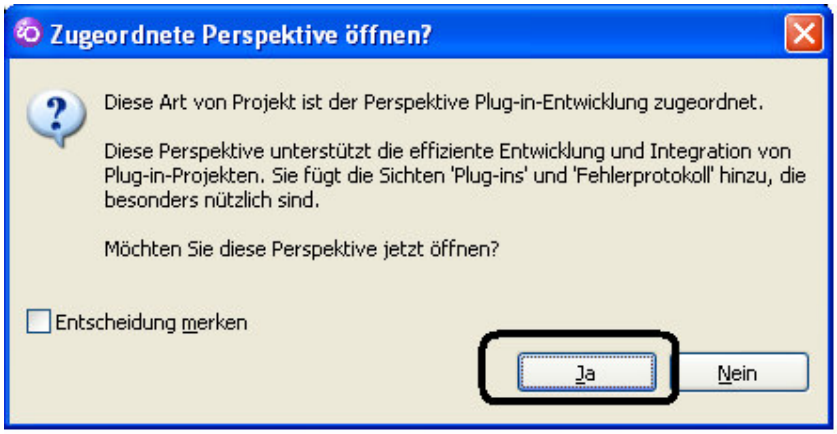
Übernehmen Sie die voreingestellten Werte und klicken Sie erneut auf „Weiter“.

Der Eclipse Assistent bietet für das Erstellen von Plug-Ins verschiedene Schablonen als Vorlagen zur Auswahl an. Deaktivieren Sie die Voreinstellung „Plug-In mit Schablone erstellen“, wie es die nachfolgende Abbildung zeigt.



Schablonen

Schließen Sie den Projekterstellungsassistenten mit einem Klick auf „Fertig stellen“.
In Abhängigkeit der verwendeten Eclipse Version wird unter Umständen der Wechsel in eine andere Perspektive – die Plug-In Entwicklungsperspektive- vorgeschlagen.
Bestätigen Sie die Auswahl mit einem Klick auf „Ja“.

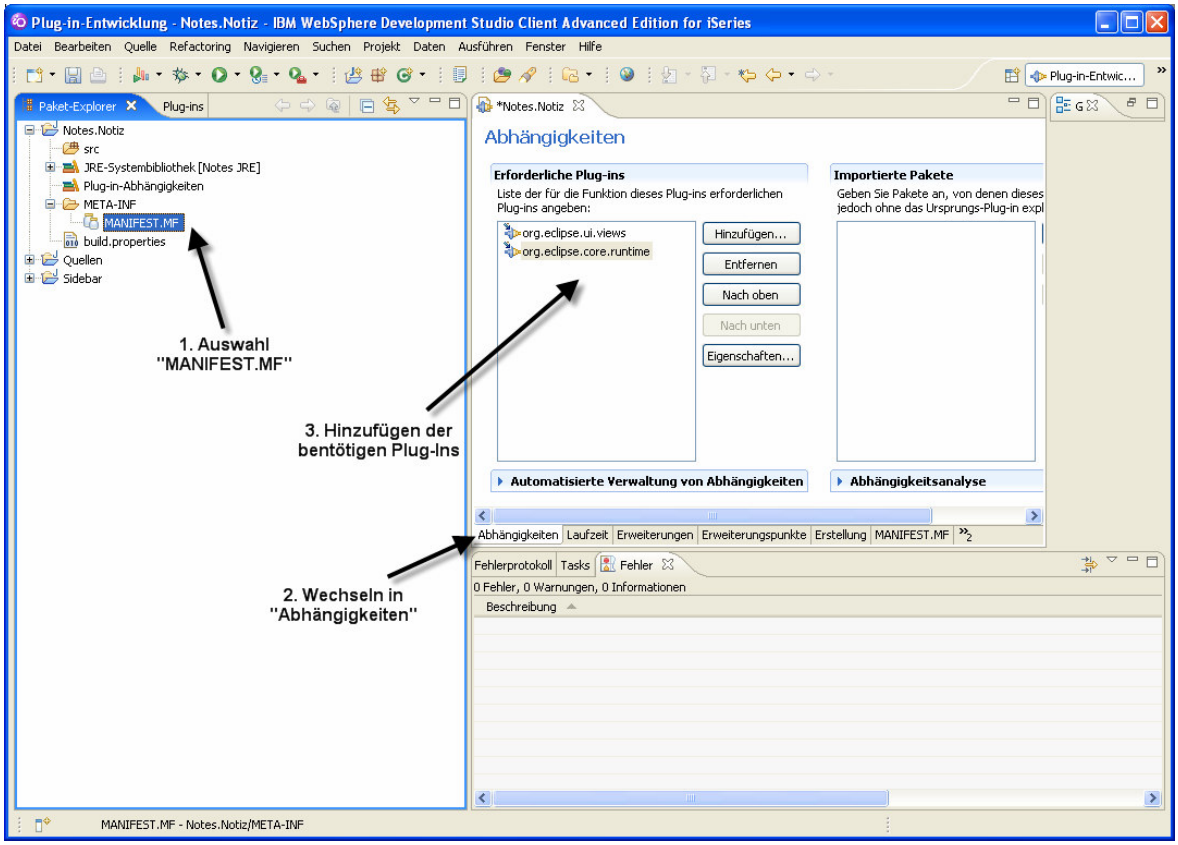


Perspektive wechseln

Damit wird das neue Plug-In Projekt erstellt. Diese besteht per Standard aus verschiedenen Bereichen:

- JRE Systembibliothek
- Plug-In Abhängigkeiten, welche in einer Plugin.XML Datei beschrieben werden. Die Plugin.xml Datei beinhaltet die Funktionen und die verwendbaren Plattformen in einem XML Format. Mit Hilfe einer eindeutigen ID wird die Plugin.xml Datei identifiziert.
- META-INF
  - Bei den Manifest Definitionen handelt es sich um die Zusammenstellung der Plug-In Informationen. Diese Angaben lassen sich in Eclipse mit Hilfe eines Multipage Editors zentral verwalten.

Als nächstes passen wir nun die Manifest Beschreibung an. Dazu erweitern wir in dem linken Navigationsbereich den Eintrag „META-INF“ und selektieren den Eintrag „MANIFEST.MF“, wie es die nachfolgende Abbildung zeigt.



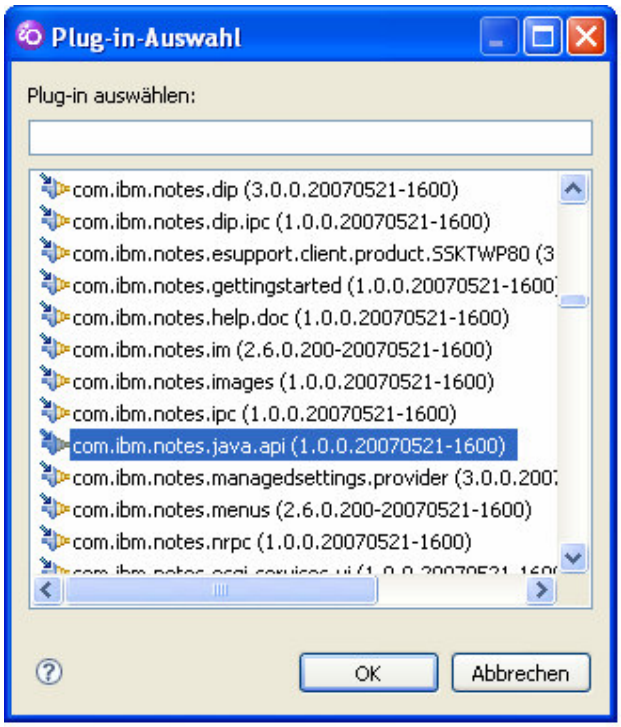
Das neue Plug In Projekt

Damit die Plug-In Entwicklung auf den bereits bestehenden Plug-Ins aufsetzen kann, müssen wir die Abhängigkeiten anpassen. Dazu wählen wir in dem Hauptfensterbereich den Abschnitt „Abhängigkeiten“, wie es die vorhergehende Abbildung zeigt.

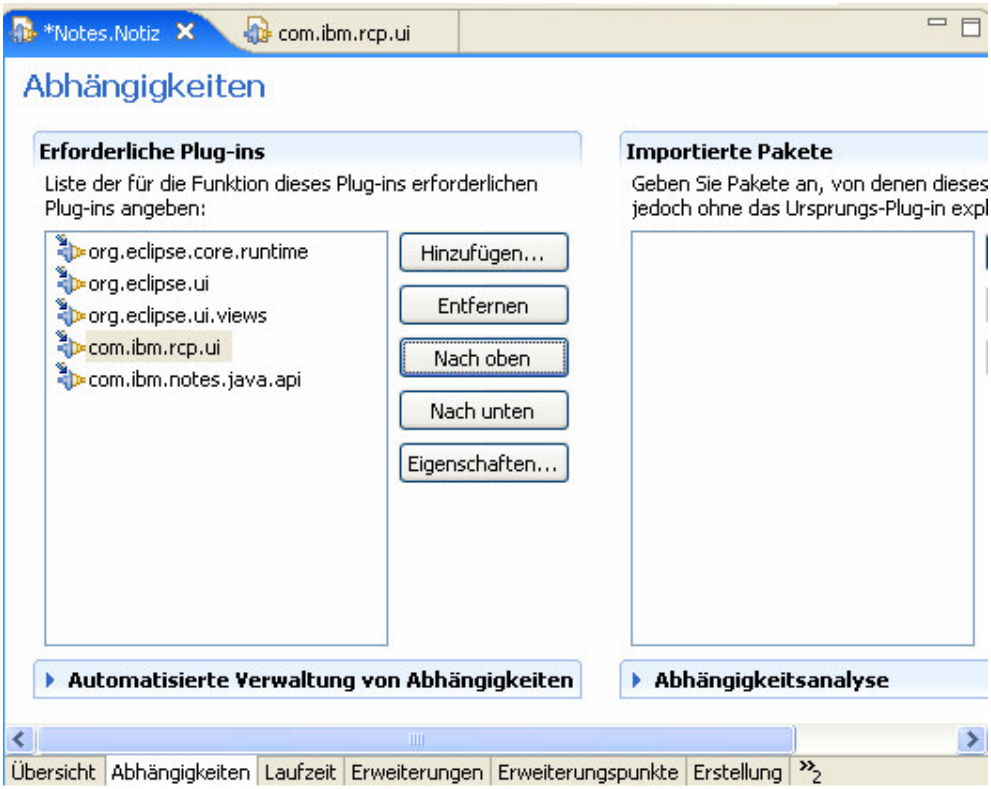
Als erforderliche Plug-Ins fügen wir dem neuen Plug-In Projekt nun die folgenden Plug-ins hinzu:

- org.eclipse.core.runtime
- org.eclipse.ui
- org.eclipse.ui.views
- com.ibm.rcp.ui
- com.ibm.notes.java.api

Klicken Sie dazu jeweils auf die Schaltfläche „Hinzufügen“. Damit gelangen Sie in eine Auswahlliste der per Definition verfügbaren Plug-ins. Wählen Sie in der Auflistung die gewünschten Plug-Ins für Ihr Projekt aus.



Plug-In Auswahl



Erforderliche Plug-Ins angepasst

Achten Sie bei der Definition der Plug-Ins auf die Reihenfolge. Aufgrund der hierarchischen Angabe werden die Plug-Ins auch entsprechende verarbeitet. Stellen Sie für das Beispiel sicher, dass die Reihenfolge genauso angegeben ist, wie es die vorhergehende Abbildung zeigt. Anpassungen lassen sich durch das Markieren des gewünschten Plug-Ins und des Verschieben mittels der Schaltflächen „Nach oben“ bzw. „Nach unten“ durchführen.

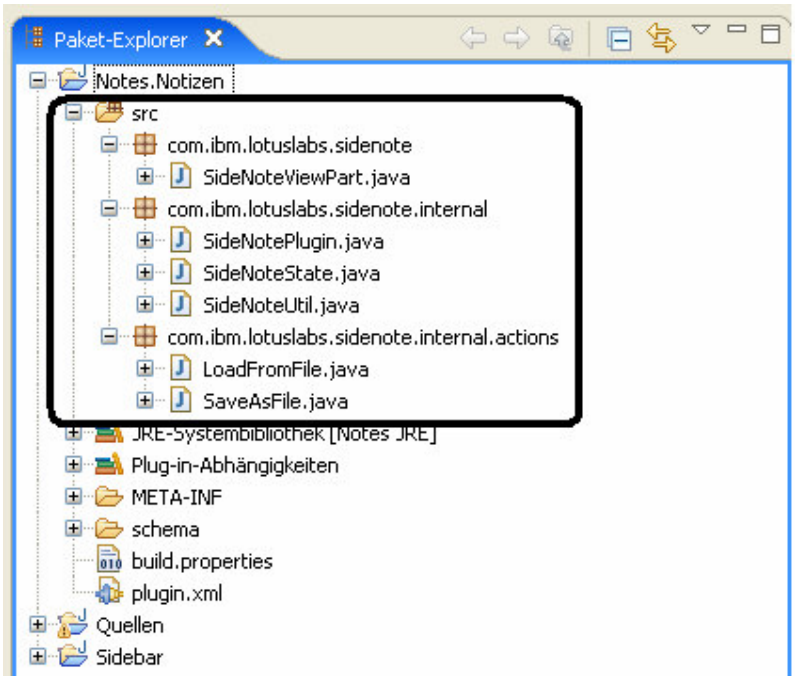
Speichern Sie nach den Anpassungen die Änderungen an der MANIFEST.MF ab.

Damit haben wir die Voraussetzungen für das Erstellen unseres ersten eigenen Plug-Ins geschaffen.

Jetzt geht es darum, den neuen Code in Form von beispielsweise Java Klassen zu erfassen.

Ich möchte an dieser Stelle nicht auf die Details der Erstellung von Java-Anwendungen eingehen, sondern mich auf das Erstellen von Plug-Ins beschränken. Deshalb bedienen wir uns der zuvor bereits erwähnten Beispielanwendung, welche IBM für die ersten Schritte mit Eclipse Plug-Ins auf ihren Webseiten zum Download bereitstellt.

Kopieren Sie den entsprechenden Eintrag in Ihren Arbeitsbereich. Darin enthalten ist unter anderem auch der Quellcodebereich „SRC“ welcher verschiedene Java Klassen beinhaltet, mit denen die Notizfunktionen realisiert werden.



Der Quellcodebereich

Kopieren Sie sich den Quellcodebereich in Ihr Projekt oder setzen Sie alternativ die Arbeit mit dem gerade in Ihren Arbeitsbereich kopierten Plug-In Projekt fort. Dieses beinhaltet die gesamten Einstellungen für die Implementierung des Plug-Ins.

In der Regel wird man bei dem Erstellen eines neuen Plug-Ins natürlich den Programmcode selbst schreiben. Diesen platzieren wir in der Regel genau in den Bereich „SRC“, wie es in dem gelieferten Beispiel auch der Fall ist.

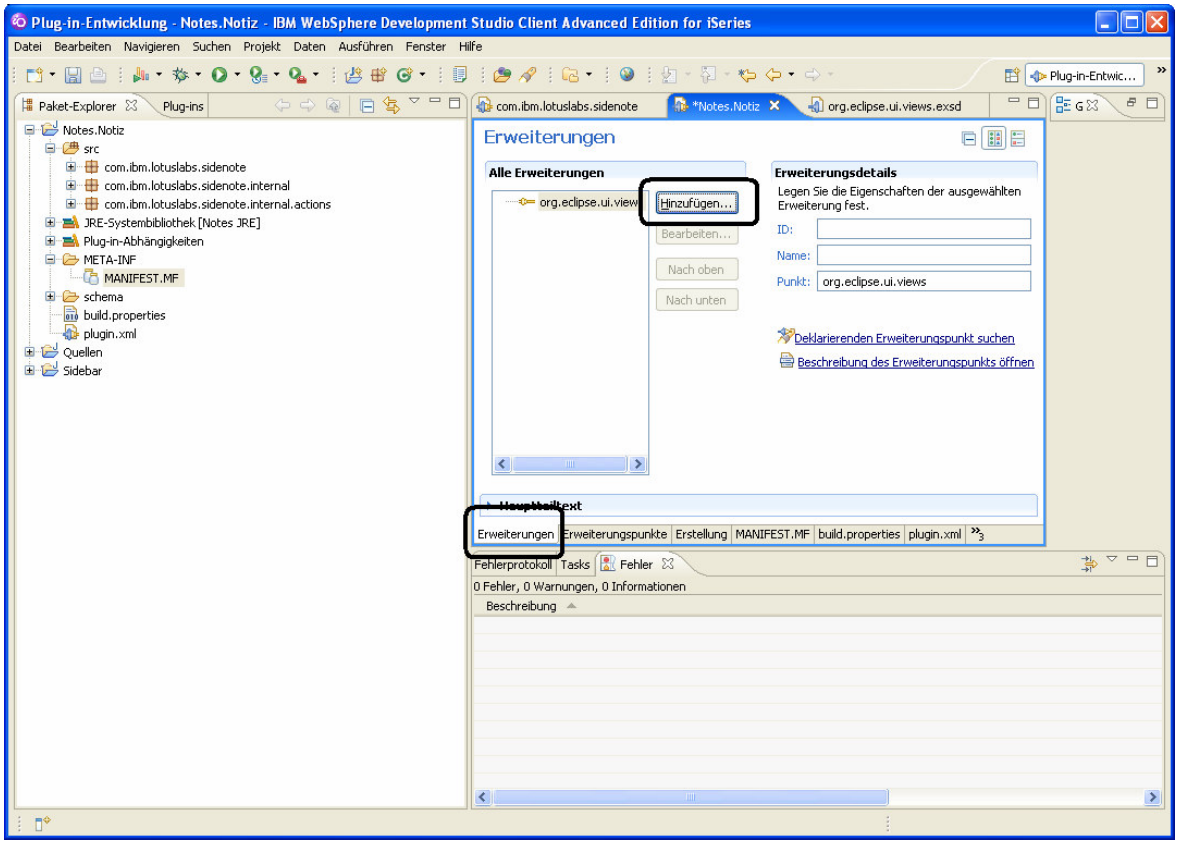
Nach dem Erstellen des Programms muss dieses für die Verwendung in einem Plug-In vorbereitet werden. Dazu verwenden wir Erweiterungspunkte, die innerhalb der Plug-In Manifest Definition angegeben werden.

Bevor wir die eigentlichen Erweiterungspunkte definieren, legen wir die Erweiterungen in dem gleichnamigen Tabellenbereich fest.

Sollten Sie mit dem mitgelieferten Plug-In Projekt arbeiten wollen, dann brauchen Sie die folgenden Schritte nicht ausführen. Die benötigen Einstellungen sind bereits in dem Projekt enthalten. Schauen Sie sich dennoch die möglichen Konfigurationsdefinitionen an.

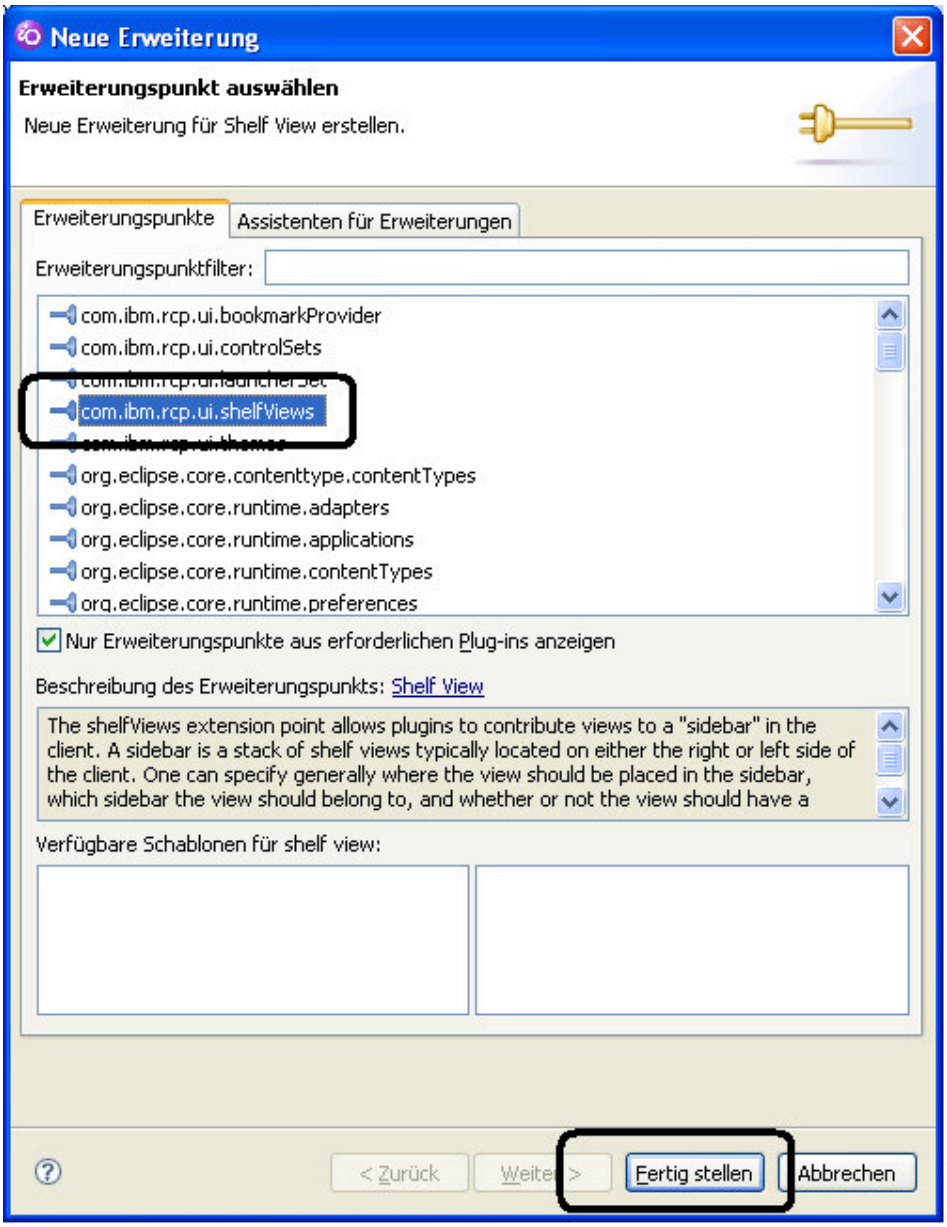
Geben Sie dort jeweils mit einem Klick auf „Hinzufügen“ die folgenden Erweiterungen an:

- org.eclipse.ui.views
- com.ibm.rcp.ui.shelfViews
- org.eclipse.ui.viewActions



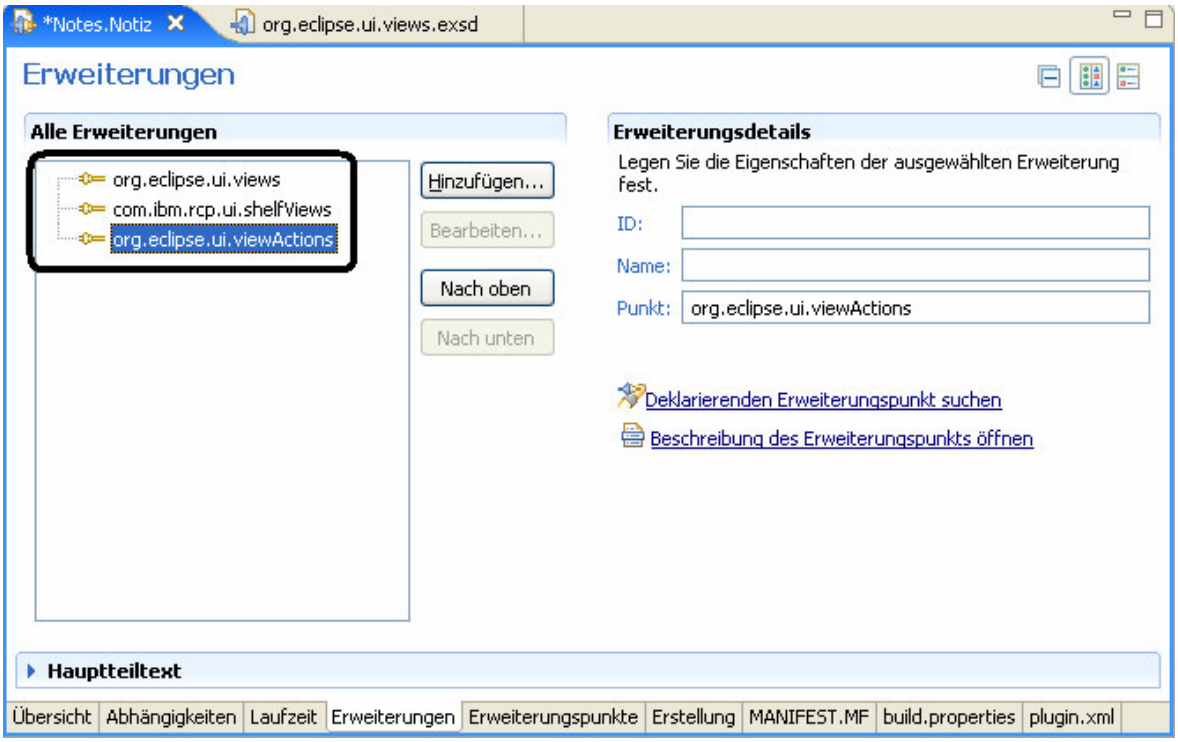
Erweiterungen erfassen

Jede Erweiterung kann aus der Auflistung selektiert werden. Deaktivieren Sie die Auswahl „Nur Erweiterungspunkte aus erforderlichen Plug-Ins anzeigen und markieren Sie in der Auflistung das gewünschte Plug-In.



Erweiterungspunkte auswählen

Ein solcher Erweiterungspunkt ist genau genommen eine Art Schnittstelle, mit der Erweiterungen definiert werden.



Die vollständigen Erweiterungen des Projektes

Nachdem alle Erweiterungen definiert wurden, sollte die Darstellung so aussehen, wie es die vorhergehende Abbildung zeigt.

Mit diesen ersten Schritten haben wir nun die Voraussetzungen für die Erstellung eines Plug-Ins geschaffen. In der nächsten Ausgabe werden wir die weiteren notwendigen Schritte behandeln, die für das Plug-In noch benötigt werden.