Generationswechsel mit .NET

Plug-ins für ACT! 7 mit .NET programmieren

von Mark Boland

Die Entwicklung von Erweiterungen für das CRM ACT! 7, das mit der aktuellen Version auf .NET 1.1 umgestellt wurde, am Beispiel einer einfachen Plug-in DLL mithilfe des ACT!-7-SDKs.

Im August 2004 ist in den USA eines der populärsten CRM-Systeme in die nächste Runde gegangen – ACT! 7 bzw. ACT! 2005. Ab Mai 2005 steht die deutsche Version der CRM-Software in den Regalen. Der Sprung von ACT! 6 zu ACT! 7 ist mehr als nur die Änderung der Versionsnummer. Fielen die Änderungen von Version 5 zu Version 6 kaum auf, wurde das Programm in Version 7 komplett neu auf Basis des .NET Framework 1.1 entwickelt. Als Datenbank-Backend dient nun der SQL Server 2000 bzw. die MSDE (je nach ACT!-Version). Da CRM-Software auch in Deutschland in den letzten Jahren immer mehr an Bedeutung gewonnen hat, rückt sie auch in den Fokus vieler Entwickler, die sich um die Anpassungsfähigkeit vorhandener CRM-Lösungen auf den Firmenbedarf Gedanken machen müssen.

Wie schon die vorherigen Versionen von ACT! ist auch die neue Version hochgradig anpassungsfähig und besitzt ein sehr umfangreiches SDK ([1] – US SDK 7.04). Aufgrund des neuen SDK, das sich komplett vom Vorgänger unterscheidet, und auch der relativ spärlichen Dokumentation und Beispiele des neuen SDK, wird vielen Entwicklern der Umstieg zur neuen Version nicht gerade leicht gemacht. ACT! enthält in der Version 7 eine echte Plug-in-Schnittstelle, mit der sich selbst entwickelte Plug-in DLLs integrieren lassen. Basis dieses Artikels ist die Entwicklung eines einfachen Plug-ins mit Visual Basic .NET und ACT! 7 in der deutschen Version 7.04, welche den Aufbau eines ACT!-Plug-ins verständlich machen soll und einfache SDK-Operationen zeigt. Halten Sie sich nicht mit Gedanken über die Zweckmäßigkeit der Beispiele auf, der Fokus liegt auf der Basis des Plug-ins und der Vorgehensweise bei der Integration in ACT!.

kurz & bündig

Inhalt

Erstellung von Plug-ins für ACT! 7. Benutzerdefinierte Schaltflächen und Register in das CRM-System ACT! 7 einbauen

Zusammenfassung

ACT! 7, das selber in C# entwickelt wurde, bietet eine erstklassige Schnittstelle für Plug-in-Entwickler

Quellcode

VB

Quellcode auf CD

Vorbereitung

Zur Entwicklung eines Plug-ins benötigen Sie mindestens vier DLLs, die aber bei einer Standardinstallation von ACT! 7 mitinstalliert werden. Das SDK selbst wird nicht benötigt – es beinhaltet nur eine Sammlung von Dokumentationen und Beispielen. Erstellen Sie ein neues Klassenbibliotheksprojekt unter Visual Studio. Als Verweis geben Sie folgende DLLs an: *Act.Framework.dll*, *Act.Shared.Collections.dll*, *Act.UI.dll* und *Act.UI.Core.dll* – diese befinden sich alle im GAC (*Global Assembly Cache*) und müssen aus diesem herauskopiert werden, um auf sie referenzieren zu können. Die Eigenschaft *Lokale Kopie* des Verweises muss auf *False* gesetzt werden. Zusätzlich wird noch ein Verweis auf *System.Windows.Forms* und *System.Drawing* benötigt. Um die Beispiele nachzuvollziehen, benötigen Sie natürlich ACT! 7. Eine 30 Tage lauffähige deutsche Testversion erhalten Sie unter [2].

Die vorhandene Klasse nennen Sie *SimplePlugin* – diese soll die Schnittstelle *Act.UI.IPlugin* implementieren. Nach Eingabe der Schnittstellendaten ergänzt die Entwicklungsumgebung Ihre Klasse automatisch um die beiden Subs *OnLoad* und *OnUnload*. In der Sub *OnLoad* wird ein *ActApplication*-Objekt übergeben. Mit Übergabe dieses Objekts sind Sie im

Listing 1

```
Public Class SimplePlugin
  Implements Act.UI.IPlugin

  'Variable um die Act Applikation Ereignisse
  'aufzunehmen und auszuwerten
  Private WithEvents ActApplication As Act.UI.
                                      ActApplication

  'Wird von der Schnittstelle ACT.UI.IPlugin angelegt.
  'Wird nach dem Start von Act ausgeführt
  Public Sub OnLoad(ByVal actApp As Act.UI.
      ActApplication) Implements Act.UI.IPlugin.OnLoad
    ActApplication = actApp
  End Sub

  'Wird von der Schnittstelle ACT.UI.IPlugin angelegt.
  'Wird vor dem Beenden von Act ausgeführt
  Public Sub OnUnLoad() Implements Act.UI.IPlugin.
                                      OnUnLoad
  End Sub
End Class
```

Spiel und haben die Möglichkeit, die Daten, die GUI, usw. der aktuellen Sitzung zu überwachen und zu manipulieren. Über die Ereignisse des *ActApplication*-Objekts ist es möglich, auf die meisten Aktionen in der Applikation zu reagieren. Aus diesem Grund sollten Sie zusätzlich eine *ActApplication*-Variable *WithEvents* in der Klasse definieren, sodass diese auf die Applikationsereignisse reagieren kann. Übergeben Sie den Verweis des *AppApplication*-Objekts in *OnLoad* an diese Variable (Listing 1).

Welche Aufgabe hat das Plug-in?

Jetzt haben Sie bereits eine generelle Basis für die Erstellung eines Plug-ins für ACT! 7 geschaffen. Aber was genau soll dieses Plug-in bieten? Es ist vorteilhaft, eine neue Schaltfläche in die Symbolleisten einzublenden, um dem Benutzer neue Funktionen anzubieten. Weiterhin kann es sinnvoll sein, ein eigenes neues Register in der unteren Detailansicht mit anzuzeigen, um dem Benutzer Detaildaten aus ACT! oder einer externen Datenquelle zu jedem Kontakt anzuzeigen. Ebenso soll der aktuell angezeigte Kontakt überwacht, und gegebenenfalls verändert oder eine Aktion eingeleitet werden, ohne dass der Benutzer interagiert. Diese immer wiederkehrenden Standardprobleme werden Sie in diesem Artikel Stück für Stück kennen und lösen lernen.

Einbau einer Schaltfläche

In Beispiel-Plug-in wird bei einem Klick auf die neue Schaltfläche die Sub *ButtonClick* gestartet. Als Beispielaktion wird ein Wert aus dem Kontakt ausgelesen (das Geburtstagsfeld), bearbeitet (Alter wird ermittelt) und der neue Wert in ein weiteres Feld (Benutzerfeld 10) zurückgeschrieben (Abbildung 1).

Abb. 1: Neue Schaltfläche in der Symbolleiste

Fügen Sie dem Projekt ein Icon als Ressource hinzu, welches Sie für die Schaltfläche benutzen möchten (im Beispiel ist dies *Face02.ico*). Setzen Sie in den Eigenschaften des Icons die *Build*-Aktion auf *Eingebettete Ressource*. Implementieren Sie die Ereignisse *AfterLogon* und *BeforeLogoff* des *ActApplication*-Objekts – diese bestimmen den Zeitpunkt des Ein- und Ausbaus der Schaltfläche in die Symbolleiste. Um die Schaltfläche in die Symbolleiste einzubauen, wird ein Icon benötigt, ein eindeutiger Name für die Schaltfläche, ein Text für den *ToolTip* (optional), der Name der Symbolleiste und die Adresse der Sub, die bei einem Klick ausgeführt werden soll (Listing 2) – die Sub muss dabei die Signatur des *Delegate Sub CommandHandler* besitzen.

Bei den Symbolleisten ist zu beachten, dass die ACT!-Version eingedeutscht wurde. Wenn Ihr Plug-in auch mit der amerikanischen ACT!-Version lauffähig bleiben soll, müssen Sie dafür sorgen, dass je nach ACT!-Version der deutsche oder englische Name für die Symbolleisten benutzt wird. Eine Auflistung der Hauptansichts-Symbolleisten in beiden Sprachversionen finden Sie in Tabelle 1.

Die SDK-Befehle in der Sub *ButtonClick* verwenden bereits die Standardzugriffsbefehle auf die Daten. Das Objekt *ActApplication.ApplicationState* enthält den aktuellen Status der laufenden Anwendung. Hier lässt sich überprüfen, welches der aktuell angezeigte Kontakt, die aktuelle Kontaktliste, aktuelle Firma, usw. ist. Jeder Elementtyp in ACT! besitzt im *Act.Framework* seine eigene Klasse – *Contacts.Contact*, *Companies*.

Deutsche Symbolleisten	Englische/US Symbolleisten
Symbolleiste Kontaktdetail	Contact Detail Toolbar
Gruppendetail Symbolleiste	Group Detail Toolbar
Firmenlistedetail Symbolleiste	Company Detail Toolbar
Symbolleiste Kontaktliste	Contact List Toolbar
Gruppenliste Symbolleiste	Group List Toolbar
Firmenliste Toolbar	Company List Toolbar

Tabelle 1: Liste der Hauptsymbolleisten

Listing 2

```
Private Sub ActApplication_AfterLogon(ByVal sender As Object, ByVal e As System.
                                      EventArgs) Handles ActApplication.AfterLogon
  'Nach dem DB Login die Schaltfläche in die Symbolleiste der Kontaktdetail-Ansicht
  'einbauen
  AddButton("Symbolleiste Kontaktdetail", "SimplePlugin", "Simple Plugin",
            "Simple Plugin", GetIconFromRessource("FACE02.ICO"), AddressOf ButtonClick)
End Sub

Private Sub ActApplication_BeforeLogoff(ByVal sender As Object, ByVal e As System.
                                        EventArgs) Handles ActApplication.BeforeLogoff
  'Vor dem Logoff die Schaltfläche wieder aus der Symbolleiste entfernen
  RemoveButton("Symbolleiste Kontaktdetail", "SimplePlugin")
End Sub

Public Sub AddButton(ByVal ToolBar As String, ByVal UniqueName As String,
       ByVal Caption As String, ByVal ToolTip As String, ByVal Icon As System.Drawing.Icon,
                                        ByVal ButtonClick As Act.UI.CommandHandler)
  Dim NewButton As Act.UI.Core.CommandBarButton
  'Wenn Command bereits vorhanden, dann raus damit
  If (Not (ActApplication.Explorer.CommandBarCollection(ToolBar).ControlCollection
                                                  (UniqueName) Is Nothing)) Then
    RemoveButton(ToolBar, UniqueName)
  End If

  'Neuen Button erstellen
  NewButton = New Act.UI.Core.CommandBarButton(Caption, ToolTip, Nothing,
                                                  UniqueName, Icon)
  'Nur Image anzeigen
  NewButton.DisplayStyle = Act.UI.Core.CommandBarControl.ItemDisplayStyle.ImageOnly
  'Button registrieren
  ActApplication.RegisterCommand(UniqueName, ButtonClick, Act.UI.RegisterType.Shell)
  'Und an die jeweilige CommandBar anhängen
  ActApplication.Explorer.CommandBarCollection(ToolBar).ControlCollection.Add
                                                  (NewButton)
End Sub
```

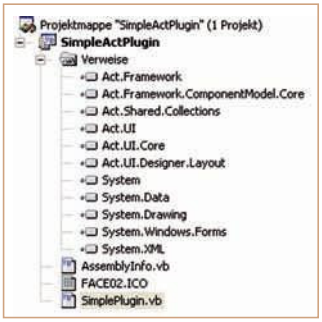

Abb. 2: Die benötigten Verweise

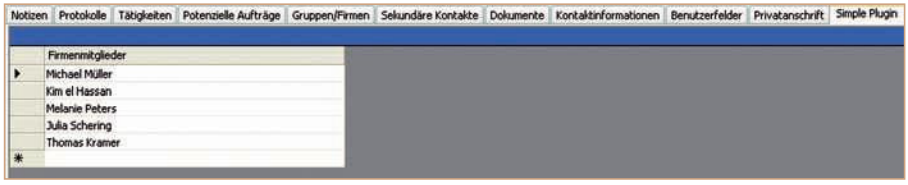

Abb. 3: Das eigene Register in Aktion

Company, *Groups.Group*, usw. Um damit auf die Daten zuzugreifen, gibt es die Möglichkeit, bei gängigen Datenfeldern diese direkt aus einer Eigenschaft auszulesen – z.B. *Contact.Fullname*. Bei einem Zugriff auf selten genutzte oder benutzerdefinierte Felder erfolgt der Zugriff über die Collection *Contact.Fields()*. Im Plug-in-Beispiel wird der Zugriff allerdings über einen *FieldDescriptor* durchgeführt. Der *FieldDescriptor* wird über seinen Namen (setzt sich aus Tabellen und Spaltennamen zusammen – die Liste der Felder liegt im SDK vor) herangezogen und liefert nicht nur den Wert über *GetValue* zurück, sondern enthält auch Daten wie Feldname, Feldtyp, usw. Über *SetValue* wird ein Wert in einen Elementtyp geschrieben – z.B. in einen *Contact*. Jetzt fehlt noch das *Contact.Update*, um die Änderung in die Datenbank zu schreiben, und das Aktualisieren der momentanen Ansicht, um die Änderung sofort für den Benutzer sichtbar zu machen (Listing 3).

Act Views

- Act.UI.ICompanyDetailView
- Act.UI.ICompanyListView
- Act.UI.IContactDetailView
- Act.UI.IContactListView
- Act.UI.IGroupDetailView
- Act.UI.IGroupListView
- Act.UI.ICalendarView
- Act.UI.IOpportunityView
- Act.UI.ITaskView

Einbau eines Zusatzregisters

Um noch ein zusätzliches Tabregister in ACT! einzubauen werden weitere Verweise auf *Act.Framework.ComponentModel.Core* und *Act.UI.Designer.Layout* benötigt, welche Sie im GAC bzw. im Hauptverzeichnis von ACT! 7 finden. Setzen Sie die Eigenschaft *Lokale Kopie* wieder auf *FALSE* (Abbildung 2).

Implementieren Sie das Ereignis *ViewLoaded* des *ActApplication*-Objekts, welches nach jedem *View*-Wechsel aufgerufen wird. Hier kann über den Namen der *View* erfahren werden, welche *View* aktuell dargestellt wird. Eine Liste der möglichen *Views* befindet sich in dem Kasten „Act Views".

Das Beispiel benutzt wieder die Kontakt-Detailansicht und erweitert deren Registeransicht im unteren Bereich um eine weitere *TabPage*. Sobald die richtige *View* anhand des Namens identifiziert wurde, wird geprüft, ob die *TabPage* schon einmal instanziiert wurde. Ist dies nicht der Fall wird sie über das ACT!-SDK hinzugefügt. Merken Sie sich die zurückgegebene *TabPage* in einer Objektvariablen, sodass Sie sicherstellen, diese nur einmal hinzuzufügen. Wenn der Prozess erfolgreich abgeschlossen wurde, wird ein *DataGrid* auf die *TabPage* gesetzt. Dieses soll jeweils Detailinformationen zum aktuell dargestellten Datensatz darstellen (Listing 4). Natürlich lässt sich auch ein mit dem Forms-Designer vorgefertigtes *Windows.Form* in die neue *TabPage* einfügen.

Das *DataGrid* wird mit den Kontaktnamen gefüllt, die der gleichen Firma angehören, der auch der aktuell dargestellte Kontakt angehört. So hat man in diesem Zusatztab die Übersicht über alle weiteren Mitglieder in dieser Firmenhierarchiestufe. Zusätzlich wird das Ereignis

Listing 3

```
Public Sub ButtonClick(ByVal Text As String)
  'Der eingebaute Button wurde geklickt
  Dim FieldDescriptor As Act.Framework.Contacts. ContactFieldDescriptor
  Dim Contact As Act.Framework.Contacts.Contact
  Dim Geburtstag As Date
  Dim Alter As Integer

  'Aktuellen Kontakt holen
  Contact = ActApplication.ApplicationState.CurrentContact

  'FieldDescriptor für das Birthdate-Feld holen
  FieldDescriptor = ActApplication.ActFramework.Contacts.GetFieldDescriptor
                                                ("TBL  CONTACT.BIRTHDATE", True)

  'Prüfen, ob es einen Wert enthält
  If Not FieldDescriptor.GetValue(Contact) Is Nothing Then
    'Wenn ja, dann weiter
    'Zugriff wäre auch über Contact.Fields("TBL  CONTACT.BIRTHDATE", True) möglich
    Geburtstag = FieldDescriptor.GetValue(Contact)

    'Alter errechnen
    Alter = Now.Year - Geburtstag.Year
    If (Now.Month < Geburtstag.Month) Or (Now.Month = Geburtstag.Month And Now.Day
                                                < Geburtstag.Day) Then
      Alter -= 1
    End If

    'Und das Alter in ein weiteres Feld schreiben
    'FieldDescriptor von Benutzerfeld 10 holen
    FieldDescriptor = ActApplication.ActFramework.Contacts.GetFieldDescriptor
                                                ("TBL  CONTACT.USER10", True)

    'Wert schreiben
    'Zugriff wäre auch über Contact.Fields ("TBL  CONTACT.USER10", True) möglich
    FieldDescriptor.SetValue(Contact, Alter.ToString)
    'Kontakt updaten
    Contact.Update()
    'Aktuelle Ansicht aktualisieren
    ActApplication.RefreshView(ActApplication.CurrentViewName)
  End If
End Sub
```

CurrentContactChanged implementiert, da bei jedem neuen Kontakt das *DataGrid* aktualisiert werden muss. Es werden alle Kontakte angezeigt, ausgenommen dem aktuell dargestellten in der Liste (Listing 5; Abbildung 3).

Listing 4

```
Private WithEvents TabPage As System.Windows.Forms.
                                                TabPage
Private WithEvents TabControl As System.Windows.
                                        Forms.TabControl
Private WithEvents DataGrid As System.Windows.Forms.
                                                DataGrid

Private Sub ActApplication_ViewLoaded(ByVal sender
            As Object, ByVal e As Act.UI.ViewEventArgs)
                        Handles ActApplication.ViewLoaded
  'Prüfen welcher View
  If ActApplication.CurrentViewName =
                        "Act.UI.IContactDetailView" Then
    'Prüfen, ob die TabPage schon einmal gesetzt wurde
    If TabPage Is Nothing Then
      'Und beim Kontakt-Detail-View das neue Tab einbauen
      TabPage = AddTab("Simple Plugin")
      If Not TabPage Is Nothing Then
        'DataGrid erzeugen
        DataGrid = New System.Windows.Forms.DataGrid
        DataGrid.Dock = Windows.Forms.DockStyle.Fill
        DataGrid.PreferredColumnWidth = 300
        'Datagrid einhängen
        TabPage.Controls.Add(DataGrid)
        RefreshGrid()
        'Das TabControl merken, um auf einen Resize
        'reagieren zu können
        TabControl = TabPage.Parent
      End If
    End If
  End If
End Sub

Public Function AddTab(ByVal TabCaption As String)
                    As System.Windows.Forms.TabPage
  Dim TabPage As System.Windows.Forms.TabPage

  Try
    'TabPage erzeugen
    TabPage = New System.Windows.Forms.TabPage
    'Caption setzen
    TabPage.Text = TabCaption
    'In die aktuelle Act Ansicht einhängen
    ActApplication.UILayoutDesignerManager.
                        AddTabToCurrentLayout(TabPage)
    'Und zurückgeben
    Return TabPage
  Catch ex As Exception
    'Wenn beim Einhängen ein Fehler auftrat,
    'dann NOTHING zurückgeben
    Return Nothing
  End Try
End Function
```

Wohin mit der kompilierten Plug-in DLL?

Damit ACT! 7 Ihr Plug-in überhaupt bemerkt, muss es in einem bestimmten Ordner liegen. Dies ist der Unterordner *Plugins* im Installationsverzeichnis von ACT! 7. Das Plug-in *muss* in diesem Ordner liegen, weitere Unterordner werden nicht berücksichtigt. Falls Sie also Drittkomponenten bei der Erstellung von Plug-ins benutzen, sollten diese in den GAC kopiert werden, da es sonst schnell zu Versionskonflikten mit anderen Plug-ins führen kann, die die gleichen Komponenten benutzen.

Debuggen einer Plug-in DLL

Gehen Sie wie folgt vor, um eine Plug-in DLL zu debuggen: Stellen Sie in den Pro-

Listing 5

```
Private Sub ActApplication_CurrentContactChanged
    (ByVal sender As Object, ByVal e As System.EventArgs)
          Handles ActApplication.CurrentContactChanged
  'Kontakt wurde gewechselt, DataGrid aktualisieren
  RefreshGrid()
End Sub

Private Sub RefreshGrid()
  Dim DataTable As New DataTable("Firmenmitglieder")
  Dim DataRow As DataRow
  Dim Contact As Act.Framework.Contacts.Contact
  Dim ContactOfCompany As Act.Framework.
                                        Contacts.Contact
  Dim Company As Act.Framework.Companies.Company

  DataTable.Columns.Add("Kontakt",
                                    GetType(System.String))

  'Aktuellen Kontakt holen
  Contact = ActApplication.ApplicationState.
                                        CurrentContact
  'Aktuell verlinkte Firma des aktuellen Kontakts holen
  Company = ActApplication.ActFramework.Contacts.
                                GetLinkedCompany(Contact)

  'Gab es überhaupt eine verlinkte Firma
  If Not Company Is Nothing Then
    'Alle Kontakte der Firma einlesen
    For Each ContactOfCompany In Company.
                                        GetContacts(Nothing)
      'Alle Kontakte, bis auf den aktuellen darstellen
      If Not Contact.ID.Equals(ContactOfCompany.ID) Then
        'Neue Zeile erzeugen und einfügen
        DataRow = DataTable.NewRow
        DataRow(0) = nz(ContactOfCompany.FullName, "")
        DataTable.Rows.Add(DataRow)
      End If
    Next
  End If

  'Und in das DataGrid rein
  DataGrid.DataSource = DataTable.DefaultView
End Sub
```

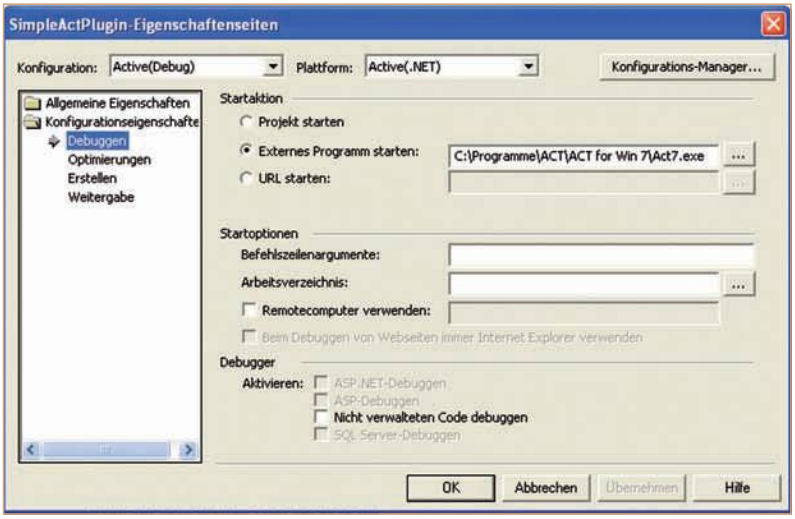

Abb. 4: Debug-Einstellungen im Projektfenster 1

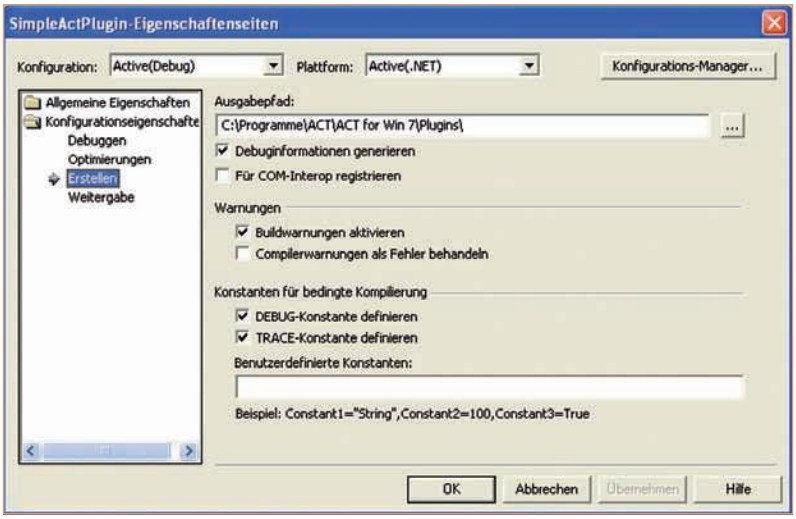

Abb. 5: Debug-Einstellungen im Projektfenster 2

Fünf Fragen an die Entwickler

Sage-Mitarbeiter Ryan Zuk, zuständig für Press & Analyst Relations bei Sage in den USA, beantwortete die folgenden Fragen auf Bitten des Autors stellvertretend für die ACT!-Entwickler.

Wann und warum fiel die Entscheidung für das .NET Framework? Standen Alternativen zur Diskussion?
Die Entscheidung fiel bereits im Jahr 2000 – der Grund war der große Fortschritt, den Microsoft mit .NET als Anwendungsplattform gemacht hatte, und dass .NET von einem Produktivitätsstandpunkt aus sehr attraktiv gegenüber den damaligen Alternativen wirkte. Die Alternativen waren Java, Delphi und C++.

Wie hat das .NET Framework die Entwicklung positiv beeinflusst? Was waren die größten Hindernisse, die überwunden werden mussten?
Das .NET-Framework half uns maßgeblich, die Entwicklungszeit zu reduzieren. Ein Hindernis war die am Anfang hohe Lernkurve – angesichts der Tiefe und Detailfülle des .NET Framework.

Wurden bei der Umsetzung Schwachstellen im .NET Framework deutlich oder gibt es Dinge, die Microsoft hätte anders oder besser lösen können?
Da fällt uns nur das *NGen-Tool* aus dem SDK ein, das uns bezüglich seiner Performance-Gewinne etwas enttäuscht hatte.

Wieso erfolgte die Entscheidung für C# und welche Eigenschaften der Sprache sprachen für die Entscheidung?
C# ist eine überlegene OO-Sprache, die von Grund auf für eine hohe Produktivität konzipiert wurde. Microsoft investiert sehr viel in seine Weiterentwicklung, sodass sie gegenüber VB.NET und C++ immer attraktiver wird.

Wie sieht die Zukunft des Projekts aus? Gibt es bereits Überlegungen das .NET Framework 2.0 zu nutzen?
Wir evaluieren .NET 2.0 und VS 2005 für künftige Versionen.

jekteigenschaften unter KONFIGURATIONSEIGENSCHAFTEN | DEBUGGEN | STARTAKTION | EXTERNES PROGRAMM STARTEN den Pfad von ACT! 7 ein. Bei einer Standardinstallation sollte dies *C:\Programme\ACT\ACT for Win 7\Act7.exe* sein (Abbildung 4). Als Ausgabepfad unter KONFIGURATIONSEIGENSCHAFTEN|ERSTELLEN|AUSGABEPFAD stellen Sie den Ordner für die Plug-ins ein (Abbildung 5). Bei einem erneuten Debug-Start der DLL wird die DLL im richtigen Ordner erstellt und im Anschluss ACT! 7 mit einer debugfähigen DLL gestartet.

Fazit

Die Entwicklung von Plug-ins mit ACT! 7 ist im Gegensatz zu den Vorversionen nicht einfacher geworden. Hat man aber erst einmal seine Erfahrungen mit dem SDK gemacht, stellt man fest, dass man mit diesem viele Dinge realisieren kann, die mit dem SDK der Vorversionen nicht möglich waren. Bleibt die Hoffnung, dass der Hersteller die Dokumentation des SDK noch nachbessert und die Beispiele zahlreicher werden. In den USA ist im September 2005 bereits ACT! 8 (seit Dezember 2005 Version 8.1) erschienen [3] – in Deutschland voraussichtlich im März 2006. Allerdings erinnert diese Version mehr an ein größeres Update der 7er Version. Das SDK bleibt bis auf Kleinigkeiten kompatibel zum 7er SDK (nach Registrierung auf www.act.com herunterladbar), sodass hier nicht noch einmal neuer Lernaufwand entsteht. Des Weiteren gibt es eine englische *Online Discussion Group*, die auch zum Thema SDK einige Informationen beinhaltet [4]. Für alle Zugänge auf die Online-Ressourcen müssen Sie sich erst auf den Webseiten ein Benutzerkonto anlegen.

Mark Boland arbeitet seit mehreren Jahren als selbstständiger Software-Entwickler auf Basis von .NET, VB6 und VBA. Im Zusammenhang mit dieser Tätigkeit entwickelte er eine Reihe von Add-ons und Plug-ins für das CRM-System ACT!

Links & Literatur

[1] www.crm-service.com/download/act704sdk.zip
[2] www.sage.de/public2/com/kontakt/kontakt.asp?Thema=ACT&Download=j
[3] www.act.com/products/trial/?campaign=HP_QL3
[4] webx.act.com