Enterprise Guide & Add-In für Microsoft Office – Individuelle Erweiterungsmöglichkeiten mit C#

Sebastian Reimann
viadee Unternehmensberatung GmbH
Anton-Bruchhausen-Straße 8
48147 Münster
sebastian.reimann@viadee.de

Zusammenfassung

Der SAS Enterprise Guide und die Add-Ins für Microsoft Office sind die idealen Tools für PowerUser und Business Analysten. Sie bieten die Möglichkeit, auf SAS Datenbestände zuzugreifen, diese auszuwerten und zu analysieren.
Doch was ist zu tun, wenn die Standard-Funktionen der Anwendungen nicht den eigenen Anforderungen entsprechen oder einfach nicht mehr ausreichen? Wie werden beispielsweise komplexe, eigene Makro-Anwendungen dem Fachanwender zur Verfügung gestellt, ohne dass dieser über fundierte SAS Kenntnisse verfügen muss oder gar SAS Base Programmcode schreiben muss? Wie kann der Anwender individuelle Datensätze in seine Analysen einfließen lassen, ohne diese manuell in SAS Tabellen einzutippen oder aus CSV-Dateien zu importieren? Wie kann der Anwender bei der Erfassung von Daten durch Validierungen und WebServices (Detailauskünfte, Geo-Locations, …) unterstützt werden?
Erfahren Sie im Folgenden, wie mit überschaubarem Aufwand der Enterprise Guide oder die Add-Ins für Microsoft Office um individuelle Funktionen, wie z.B.

- Erfassungsmöglichkeiten für Anwenderdaten
- Validierungen und Erfassungsunterstützungen
- GUI-Assistenten für individuelle SAS-Makroanwendungen
- Import-/Export-Assistenten für Daten von Drittanwendungen

erweitert werden können.

Schlüsselwörter: SAS/Enterprise Guide, Enterprise Guide (EG), Add-In für Microsoft Office, AMO, .NET Framework, C#

1 Allgemeine Vorbemerkungen zum SAS Enterprise Guide und zu den Add-Ins für Microsoft Office

Der SAS Enterprise Guide (EG) stellt eine graphische Benutzerschnittstelle für die SAS Software zur Verfügung, mit der der volle SAS Funktionsumfang in einer komfortablen Microsoft Windows Umgebung genutzt werden kann. Dabei ist es unerheblich, ob die zur Programmausführung genutzte SAS Installation auf dem Windows Client-PC zur Verfügung steht oder ob auf eine gemeinsame SAS Serverinstallation (Windows, UNIX, z/OS) zurückgegriffen wird.

In jedem Fall ist der Enterprise Guide zum Großteil ein Code-Generator, der von SAS optimierten Programmcode erstellt und diesen auf einem verfügbaren SAS Server zur Ausführung bringt. Das auf dem SAS Server erstellte Ergebnis wird im Anschluss an die Ausführung auf dem Client zur Anzeige gebracht. Der Anwender wird durch Assistenten bei der Erstellung des Programmcodes unterstützt, indem die notwendigen Programmeingaben über Formulare und Dialogboxen abgefragt werden.

Alternativ besteht für den erfahrenen SAS Base Programmierer die Möglichkeit, eigenen Base Programmcode in die Enterprise Guide Projekte zu integrieren, um so individuell Einfluss auf den zur Ausführung gebrachten Code zu nehmen und Funktionen auszuführen, die über die Enterprise Guide Standardfunktionen hinausgehen oder von den von SAS bereitgestellten Assistenten nicht unterstützt werden.

Bei den Add-Ins für Microsoft Office (AMO) handelt es sich im weitesten um eine im Funktionsumfang reduzierte Version des Enterprise Guide, die direkt aus den Microsoft Office Produkten (Excel, Word, PowerPoint, Outlook) heraus aufgerufen wird und deren Ergebnisse direkt in die Microsoft Office Produkte zurückgeschrieben werden. Aufgrund der direkten Ergebnisweitergabe an die Office Produkte ist es mit den AMO im Gegensatz zum EG nicht möglich, Zwischenergebnisse auf dem im Hintergrund genutzten SAS Server weiterzuverwenden und in Folgeauswertungen mit einfließen zu lassen. Eine Weiterverarbeitung ist nur innerhalb der genutzten Office-Anwendung möglich.

Technologisch gesehen handelt es sich sowohl beim SAS Enterprise Guide als auch bei den Add-Ins für Microsoft Office um Windows Anwendungen, die auf dem Microsoft .NET Framework basieren. Sie sind dementsprechend nur unter dem Betriebssystem Microsoft Windows lauffähig. Über standardisierte Schnittstellen besteht die Möglichkeit, den Funktionsumfang beider Anwendungen zu erweitern, um so den individuellen Nutzen der Anwendung zu vergrößern.

Im Folgenden wird hauptsächlich die Vorgehensweise zur Erweiterung des SAS Enterprise Guide beschrieben. Da der Enterprise Guide und die Add-Ins für Microsoft Office zum Großteil auf die gleichen .NET Assemblies zurückgreifen und beide Produkte technologisch identisch sind, können die für den Enterprise Guide entwickelten Erweiterungen in der Regel auch aus den Microsoft Office Produkten heraus genutzt werden.

2 Technische Voraussetzungen

Um Erweiterungen für den Enterprise Guide entwickeln zu können, ist eine .NET Entwicklungsumgebung erforderlich. Je nach persönlicher Vorliebe kann eine Visual Basic.NET oder eine C# Entwicklungsumgebung genutzt werden. Weiterhin ist eine lauffähige Installation des SAS Enterprise Guide (oder der Add-Ins für Microsoft Office) notwendig, um die entwickelten Erweiterungen zu verproben.

Um schnell erste Ergebnisse zu entwickeln, können die von Microsoft kostenlos zur Verfügung gestellten Express-Versionen des Visual Studio 2010 genutzt werden[1]. Weiterhin werden von SAS Projektschablonen und Beispiele zur Verfügung gestellt, anhand derer die ersten Schritte zu einer Enterprise Guide Erweiterung nachvollzogen werden können. Auch diese Projektschablonen und Beschreibungen sind im Internet zum Download verfügbar[2].

Die im Folgenden aufgeführten Beispiele zeigen die Erstellung von Enterprise Guide Erweiterungen mit Visual C# 2010 Express unter Nutzung der von SAS zur Verfügung gestellten Projektvorlagen. Um die Beispiele am PC nachvollziehen zu können, muss die ZIP-Datei CS2008SasTemplate.zip aus der URL http://support.sas.com/documentation/onlinedoc/guide/customtasks/samples/VS2008Templates.zip extrahiert und in das Verzeichnis %userprofile%\Eigene Dateien\Visual Studio 2010\Templates\ Project Templates\SAS Custom Tasks kopiert werden. Anschließend stehen die Projektvorlagen in der Anwendung Microsoft Visual C# 2010 Express zur Verfügung.

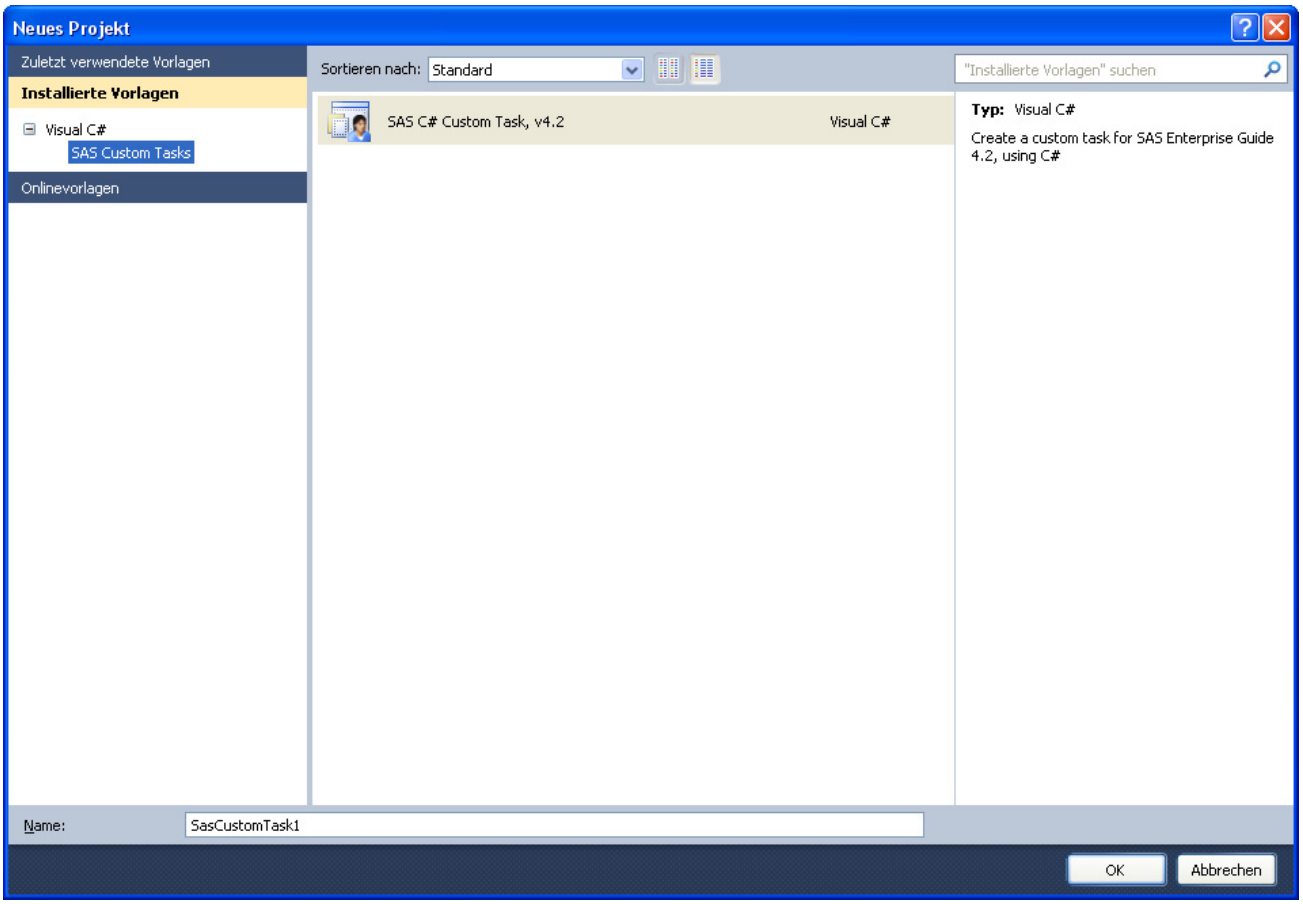

Abbildung 1: Assistent „Neues Projekt“ nach Installation der SAS Projektvorlagen

[1] Download und Nutzungsbedingungen: http://www.microsoft.com/germany/express/
[2] Download der Beispiele und Dokumentation unter http://support.sas.com/documentation/onlinedoc/guide/customtasks/index.htm

3 Die erste Erweiterung

3.1 Auswahl einer zu unterstützenden SAS Funktion

Häufig ist es so, dass der Standard-Funktionsumfang für die alltägliche Nutzung einer Software ausreicht. Trotzdem kommt man über kurz oder lang in eine Situation, in der die Lösung nur mit verhältnismäßig hohem Aufwand erreicht werden kann. In diesen Situationen wünscht man sich eine Spezialfunktion herbei, die für genau einen speziellen Zweck einen Lösungsweg bereitstellt.

Manchmal ist es aber auch so, dass eine Standard-Funktion in gewissen Situationen nicht das gewünschte oder korrekte Ergebnis liefert. In diesem Fall wartet man auf ein Fix oder ein Update, um den Missstand zu beheben. Im Folgenden wird gezeigt, wie auf einfache Weise ein aktueller Bug im Enterprise Guide (bei Nutzung einer Enterprise BI-Server Umgebung) behoben werden kann.

Im Enterprise Guide besteht über einen Assistenten die Möglichkeit, mehrere Eingabetabellen zu einer Ausgabetabelle zu verketten. Hierzu wird die Funktion „Append Table“ aus der Kategorie Data genutzt. Der Funktionsumfang des Assistenten ist trivial. Es können eine beliebige Anzahl an Eingabetabellen gewählt werden. Sämtliche Eingabetabellen werden in einer anzugebenden Ausgabetabelle zusammengeführt. Hierbei kann die Reihenfolge der Verkettung und die Art der Verkettung (View oder Datentabelle) gewählt werden.

Sind SAS Bibliotheken im SAS Metadata-Server definiert, so kann neben der Libref der Bibliothek auch ein beschreibender Bibliotheksname vergeben werden, der nicht den Anforderungen an eine SAS Libref (8 Zeichen) entsprechen muss. Über den von SAS mitgelieferten Assistenten besteht jedoch nicht die Möglichkeit, die Ausgabetabelle in eine solche SAS Library zu speichern, da der vom Assistenten generierte Zielname (Libref + Tabelle) als nicht zulässig interpretiert wird.

Als Ausgabebibliothek wird die in den Metadaten registrierte Bibliothek „Umsatzdaten – BASE“ genutzt, welche intern auf die Libref „UMSATZ“ zurückgreift. Dennoch prüft der Assistent den kompletten Bibliotheksnamen auf Zeichenkonformität für eine SAS Libref und verweigert somit die Nutzung dieser Bibliothek als Ausgabebibliothek.

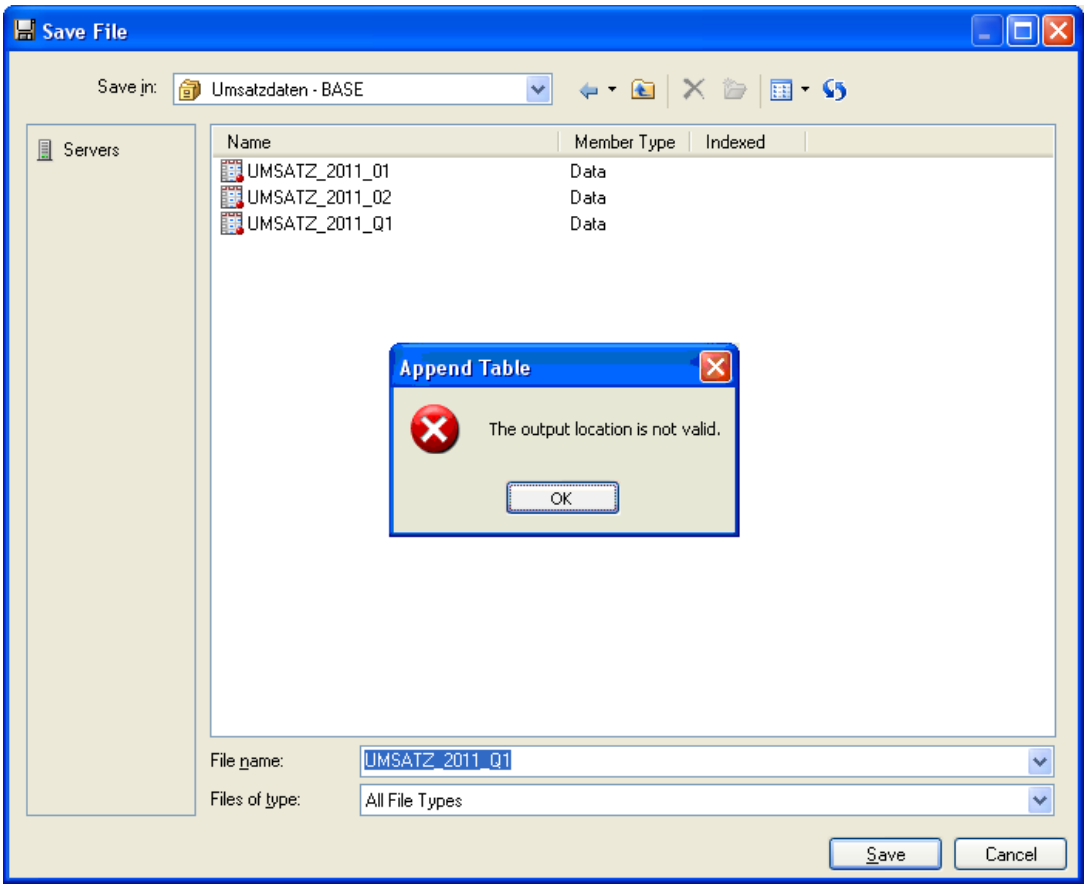

Abbildung 2: Fehlermeldung bei der Wahl der Ausgabebibliothek „Umsatzdaten - BASE“

3.2 Vorbereiten der Entwicklungsumgebung

Nach Anlage eines neuen Entwicklungsprojektes (Projektname AppendTable) im Visual C# 2010 Express wird automatisch eine Projektmappe mit folgendem Inhalt angelegt:

- Klasse AppendTableTask
- Klasse AppendTableTaskSettings
- Klasse AppendTableTaskForm
- Icon task.ico
- Verweise auf die benötigten SAS .NET Assemblies

Bei der Nutzung von Visual C# 2010 Express wird automatisch eine Projektmappe im .NET Framework 4.0 erstellt. Diese Framework-Version ist derzeit nicht kompatibel zu den SAS Enterprise Guide Assemblies. Aus diesem Grund ist in den Projekteigenschaften die .NET Kompatibilitätsstufe auf das .NET Framework 3.5 zu stellen.

In der Regel werden die eingerichteten Verweise auf die benötigten SAS .NET Assemblies nicht auf die korrekten Pfade eingerichtet. Von daher sind zunächst die Verweise zu löschen und durch korrekte Verweise auf die Dateien

- %ProgramFiles%\SAS\EnterpriseGuide\(Version)\SAS.Shared.AddIns.dll
- %ProgramFiles%\SAS\EnterpriseGuide\(Version)\SAS.Tasks.Toolkit.dll

zu ersetzen. Erst durch diese Änderung wird das C#-Projekt kompilierbar. Für die DLL-Verweise kann die Eigenschaft „Lokale Kopie“ auf false gestellt werden, da es im Rahmen der Kompilierung nicht notwendig ist, die SAS-DLLs in das Ausgabeverzeichnis zu kopieren.

3.3 Klasse AppendTableTask

Die automatisch aus der Projektvorlage erzeugte Task-Klasse stellt die Basis für eine Enterprise Guide Erweiterung dar. Diese Klasse erweitert die von SAS zur Verfügung gestellte Basisklasse SAS.Tasks.Toolkit.SasTask. Über diese Basisklasse werden die für SAS Custom Tasks notwendigen SAS Interfaces implementiert.

Über die Annotation ClassId muss für die Klasse eine eindeutige Klassen-ID festgelegt werden. Hierzu kann jeder GUID-Generator, wie z.B. der Web-Generator http://www.guidgen.com genutzt werden. Weiterhin wird über die Annotation IconLocation der Name der Icon-Datei angegeben, welches für die Symbolisierung des Tasks im Enterprise Guide (Task List und Process Flow) genutzt wird. Der IconPfad wird inklusive Namespace und Dateiname (z.B. AppendTable.viadee.ico) angegeben. Wichtig für die Icon-Datei ist, dass diese während des Buildvorgangs als eingebettete Ressource behandelt wird und in das Assembly integriert wird.

Weitere mögliche Annotationen werden in der folgenden Tabelle aufgelistet. Diese können zur Beschreibung des Tasks angegeben werden. Wird eine Annotation nicht genutzt, wird ein Standardwert verwendet.

Tabelle 1: Annotationen für SAS Custom Tasks

Annotation	**Mögliche Werte**
ApplicationSupported	Clientanwendungen, die diesen Task unterstützen: • ApplicationName.EGuide • ApplicationName.SASAddIn • ApplicationName.All
ClassId	GUID zur eindeutigen Identifizierung der Klasse
CreatesDataDirectly	Wahrheitswert, ob Daten direkt generiert oder aus Eingabedaten abgeleitet werden
IconLocation	Zeichenkette mit dem Pfad zum Icon für diesen Task
InputRequired	Art der Eingabedaten: • InputResourceType.None • InputResourceType.Data • InputResourceType.File • InputResourceType.Olap • InputResourceType.InformationMap
RequiresActualDatasource	Wahrheitswert, ob eine Datenquelle benötigt wird
SASMacroDependency	Zeichenkette mit für den Task notwendigen SAS

Annotation	Mögliche Werte
	Makros
SASMetadataRequired	Wahrheitswert, ob Metatdaten benötigt werden
SASTaskCategory	Zeichenkette mit der Kategorie für den Task
SASTaskThreadSafe	Wahrheitswert, ob der Task threadsafe ist
SupportsObsOption	Wahrheitswert, ob die System-Option obs unterstützt wird
TaskDeprecated	Kennzeichnung eines Tasks als abgelaufen. Weitere Infos über benannte String-Parameter AsOfVersion und ReplacementTask
TaskMigratorClass	Zeichenkette mit Assemblynamen und Typnamen der Migrator-Klasse
Version	Zeichenkette mit der Version des Tasks

Die Task-Klasse selbst gliedert sich in die Bereiche

- Private members
- Initialization
- Overrides

Als privates Member des Tasks wird ein Verweis auf die Settings-Klasse gespeichert. Diese Klasse dient dazu, Benutzereinstellungen zu verwalten und zu speichern, um diese im Rahmen der SAS Code-Generierung weiter verwenden zu können. Siehe hierzu Kapitel 3.4

Im Initialisierungsbereich sind die Methoden angeordnet, die während der Initialisierungsphase des Tasks aufgerufen werden. Während dieser Phase müssen folgende Informationen festgelegt werden:

Tabelle 2: Während der Initialisierung festzulegende Attribute

Attribut	Datentyp	Beschreibung
AllowModifyCustomCode	Boolean	Wahrheitswert, ob die Modifikation des generierten Codes unterstützt wird
GeneratesReportOutput	Boolean	Wahrheitswert, ob ReportOutput generiert wird
GeneratesSasCode	Boolean	Wahrheitswert, ob SAS Code generiert wird
ProcsUsed	String	Liste der SAS Prozeduren, die genutzt warden
ProductsRequired	String	Liste der erforderlichen SAS Produkte
ProductsOptional	String	Liste der optionalen SAS Produkte
StandardCategory	Boolean	Wahrheitswert, ob es sich bei der Task-Kategorie um eine SAS Standardkategorie handelt
TaskCategory	String	Kategoriename
TaskDescription	String	Task-Beschreibung

Attribut	Datentyp	Beschreibung
TaskName	String	Task-Bezeichnung
TaskType	ShowType	Art des SAS Task: ShowType.Code ShowType.Interactive ShowType.Wizard

Im Overrides-Bereich werden in der Basisklasse definierte Methoden überschrieben und neu implementiert. Zu den wichtigsten neu zu implementierenden Methoden zählen

- Public override bool Initialize()
- Public override String GetXmlState()
- Public override void RestoreStateFromXml(String xmlState)
- Public override String GetSasCode()

In der Initialisierungsmethode wird der Verweis auf die Settings-Klasse des Task hergestellt. Über den Rückgabeparameter kann die Initialisierung bei Bedarf abgebrochen werden, falls bestimmte, für den Task notwendige Rahmenparameter nicht verfügbar sind.

Die Methoden GetXmlState() und RestoreStateFromXml(String xmlState) werden von SAS aufgerufen, wenn die im XML String gespeicherten Parameter ausgelesen oder gespeichert werden. Diese Methode muss so implementiert sein, dass alle notwendigen Parameter aus dem XML String ausgelesen werden können und dass alle erforderlichen Parameter beim Verlassen des Tasks im XML String gespeichert werden.

Die Methode GetSasCode() wird vom Enterprise Guide aufgerufen, sobald der Task zur Ausführung gebracht werden soll und der für die Ausführung benötigte SAS Code abgerufen werden muss. Vor dem Aufruf der Methode wird immer zunächst die Initialisierungsmethode und die RestoreStateFromXml(String xmlState) Methode aufgerufen, um auf alle Task Parameter zugreifen zu können.

Werden von dem Task Ausgabetabellen erzeugt, so müssen weiterhin folgende Eigenschaften der Basisklasse überschrieben werden:

- Public override int OutputDataCount {get;}
- Public override List<ISASTaskDataDescriptor> OutputDataDescriptorList {get;}

Über diese beiden öffentlichen Eigenschaften wird festgelegt, wie viele Ausgabedateien erzeugt werden und wie diese benannt und typisiert sind. Werden diese beiden Eigenschaften nicht überschrieben, kann ein Task zwar Ausgabedateien erstellen, diese werden jedoch im Prozessfluss im Enterprise Guide nicht weiter dokumentiert und können von nachfolgenden Tasks nicht problemlos als Eingabedateien genutzt werden. Weiterhin werden im Prozessfluss nicht die notwendigen Abhängigkeiten der Tasks untereinander aufgebaut.

Wird durch den Task eine Benutzerschnittstelle bereitgestellt, so muss folgende Methode der Basisklasse überschrieben werden:

- Public override ShowResult Show(IWin32Window owner)

Über diese Methode wird die Benutzerschnittstelle angezeigt und das DialogResult der Benutzerschnittstelle überwacht. Je nach Rückgabewert der Benutzerschnittstelle können die weiteren Aktivitäten des Prozessflusses gesteuert werden. Hierzu stehen die ShowResult Rückgabewerte „Cancelled“, „RunNow“ und „RunLater“ zur Verfügung. So kann die Programmausführung im Prozessfluss aus der Benutzerschnittstelle heraus gesteuert werden.

Der vollständige Programmcode der Klasse AppendTableTask ist diesem Dokument im Anhang A beigefügt.

3.4 Klasse AppendTableTaskSettings

Wie bereits in Kapitel 3.3 erwähnt, werden die durch den Benutzer individuell für einen Task vorgegebenen Konfigurationsparameter in einem XML String gespeichert. Um aus der Benutzerschnittstelle einfach auf die Daten zurückgreifen zu können, wird beim Öffnen des Task der XML String eingelesen und die darin enthaltenen Werte in den Membervariablen der Settings Klasse gespeichert.

Für das aktuelle Beispiel müssen folgende Attribute als Benutzereingaben gespeichert werden:

Tabelle 3: Individuelle, zu speichernde Task-Attribute

Attribut	**Datentyp**	**Beschreibung**
genView	Boolean	Wahrheitswert, ob eine View generiert werden soll
genTable	Boolean	Wahrheitswert, ob eine Tabelle generiert werden soll
outputData	ISASTaskData	Name und Bibliothek der Ausgabetabelle (bzw. View), die vom Task generiert wird

Über den Konstruktor der Settings Klasse werden die Attribute mit Default-Werten belegt. Die Ausgabetabelle wird standardmäßig in der Work-Bibliothek erstellt und mit einem zufälligen Tabellennamen versehen. Um das ISASTaskData Objekt erstellen zu können, wird ein Verweis auf den Task Consumer (in diesem Fall ein Verweis auf den Workspace Server der Enterprise Guide Instanz) benötigt. Dieser stellt eine Methode bereit, um aus einem Servernamen, einem Bibilotheksnamen und einem Tabellennamen ein ISASTaskData Objekt zu erstellen.

Weiterhin sind in der Settings Klasse die Methoden ToXml() und FromXml(String xml) hinterlegt. Diese erzeugen den vom Task zu speichernden XML String oder lesen diesen wieder ein. Die jeweiligen Methoden werden (wie in Kapitel 3.3 beschrieben) automatisch beim Öffnen und Speichern des Tasks bzw. direkt vor der Ausführung des Tasks aufgerufen, um die korrekten Werte für die SAS Code-Generierung verfügbar zu haben.

Es wird ein XML Dokument ViadeeAppendTableTask erzeugt bzw. eingelesen. Dieses enthält die Attribute genView und genTable sowie einen Child Knoten outputData mit den Attributen Library, Member und Server, über die sich die Ausgabetabelle genau spezifizieren lässt. Es ist zwingend darauf zu achten, dass beide Methoden zueinander kompatibel sind, d.h. dass der erzeugte XML String alle zu speichernden Werte enthält und dass alle Werte aus dem XML String auch wieder eingelesen werden. Weiterhin ist zu beachten, dass bei einem Versionsupdate des Tasks die Methoden so angepasst werden müssen, dass auch Prozessflüsse mit älteren Versionen des Tasks weiterhin korrekt verarbeitet werden können und ggf. um sinnvolle Standardwerte für neue Attribute erweitert werden.

Als letztes enthält diese Klasse die Generierungsmethode für das SAS Programm des Tasks. Hierzu werden die Eingabetabellen des Tasks ausgelesen und daraus das SAS Programm generiert.

Wird eine andere SAS Funktion durch einen Assistenten unterstützt, so sind die in der Settings Klasse und im XML String zu speichernden Attribute sowie das generierte SAS Programm entsprechend den Erfordernissen anzupassen. Die hier angegebenen Attribute beziehen sich ausschließlich auf das aufgeführte Beispiel, welches für den Einstieg bewusst klein gehalten wird.

Der vollständige Programmcode der Klasse AppendTableTaskSettings ist diesem Dokument im Anhang B beigefügt.

3.5 Klasse AppendTableTaskForm

Die für den zu entwickelnden Task wichtigste Klasse ist das Formularelement, über welches die Einstellungen durch den Benutzer vorgenommen werden. Bei diesem Formularelement handelt es sich im Grunde um eine konventionelle Microsoft .NET Windows Form (System.Windows.Form), welche um einige SAS Spezifika erweitert wurde. Aus diesem Grund ist diese Klasse nicht von System.Windows.Forms.Form, sondern von der Klasse SAS.Tasks.Toolkit.Controls.TaskForm abgeleitet.

Die folgende Abbildung zeigt die gewünschte Benutzerschnittstelle mit den zu unterstützenden Funktionen:

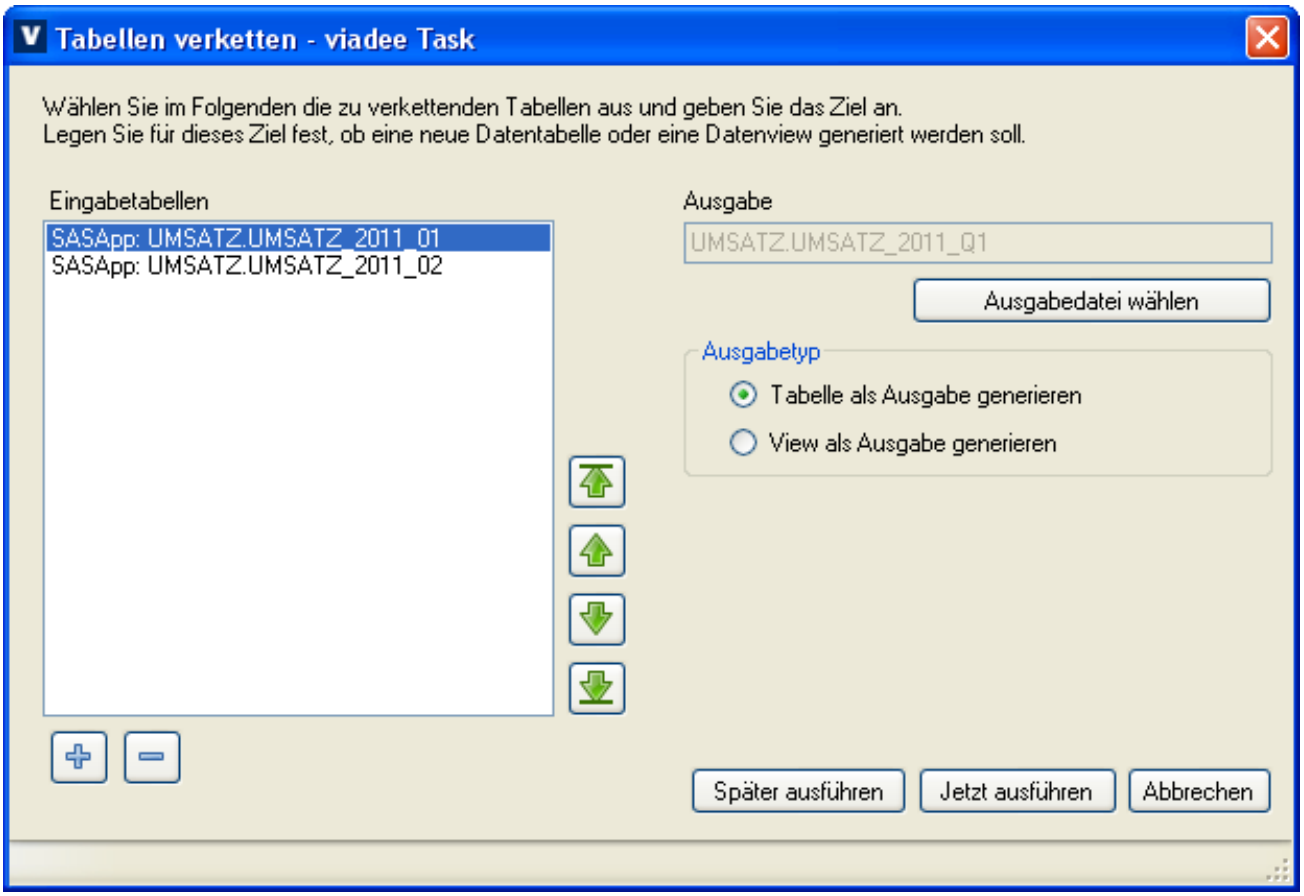

Abbildung 3: Zu implementierende Benutzerschnittstelle „Tabellen verketten“

Die Benutzerschnittstelle besteht auf der linken Seite aus einer Listbox, in der die Eingabetabellen aufgelistet sind. Diese können über die Buttons an der rechten Listbox-Seite in ihrer Reihenfolge verändert werden. Weiterhin können über die +/- Buttons weitere Tabellen hinzugefügt, bzw. vorhandene Tabellen entfernt werden. Beim Entfernen ist zu beachten, dass die letzte Tabelle nicht entfernt werden kann.

Auf der rechten Seite wird die Ausgabetabelle angegeben. Der Tabellenname (inkl. der SAS Library) wird in einer Textbox ausgegeben. Der Wert ist jedoch gesperrt, so dass der Speicherort nur über den Button auf dem Server gewählt werden kann. So wird vermieden, dass unzulässige Werte als Ausgabetabelle angegeben werden können.

Über die Radio Buttons kann gewählt werden, ob als Ergebnis des Tasks eine neue Zieltabelle oder eine View auf die Eingabetabellen generiert werden soll. Über die Aktionsbuttons am unteren Rand kann gewählt werden, ob die vorgenommenen Änderungen verworfen werden sollen oder ob die Änderungen gespeichert werden sollen. Beim Speichern besteht weiterhin die Auswahl, ob der Task nach dem Speichern direkt ausgeführt wird oder ob der Task nur gespeichert und geschlossen wird und die Ausführung des Tasks separat über den Enterprise Guide gestartet werden muss.

Um diesen Funktionsumfang implementieren zu können, wird für den Konstruktor der Klasse ein Verweis auf den Task Consumer (in diesem Fall ein Verweis auf den Workspace Server der Enterprise Guide Instanz) und ein Verweis auf die Settings Klasse benötigt. Während der Initialisierungsphase der Klasse werden die Komponenten (Buttons, Labels und sonstige Form Elemente) initialisiert und die Aufrufparameter in lokalen Klassenvariablen gespeichert.

Als nächstes wird das Load-Ereignis der Klasse überschrieben. Hierbei wird zunächst das Load-Ereignis der Basisklasse aufgerufen. Anschließend werden die Formularelemente mit den in der Settings-Klasse gespeicherten Werten initialisiert, so dass der Benutzer beim Öffnen des Formulars die korrekten Werte angezeigt bekommt.

Eine Besonderheit stellen die Eingabetabellen dar, da diese nicht in der Settings Klasse gespeichert werden, sondern über die Kollektion InputData des Task Consumers abgerufen werden müssen. Diese ist notwendig, damit der Enterprise Guide z.B. über das Kontext-Menü eines Tasks die Möglichkeit bietet, die Eingabetabellen zu verändern. Dieses Ereignis wird nicht direkt an den Task weitergeleitet, so dass in diesem Fall die Settings Klasse nicht aktualisiert werden könnte. Sämtliche Eingabetabellen werden beim Laden des Formulars in eine Listenstruktur (BindingList) überführt, da diese sehr einfach an eine Listbox gebunden werden kann und Änderungen an der Liste automatisch an das Formularfeld weitergegeben werden.

Um beim Verlassen des Formulars zwischen einem Abbruch und einem Speichern-Ereignis unterscheiden zu können, muss auch die OnClosing Methode der Basisklasse überschrieben werden. Nur wenn das Ergebnis (DialogResult) des Formulars den Wert OK (== Run Now) oder NO (== Run Later) enthält, dürfen die veränderten Werte der Formularfelder an die Settings-Klasse übergeben werden, da nur in diesem Fall die Änderungen gespeichert werden dürfen. In allen anderen Fällen wird das Formular geschlossen, ohne die Änderungen zu speichern. In einer weiteren Ausbaustufe wäre noch eine Abfrage denkbar, über die der Anwender bei einem Abbruch mit nicht gespeicherten Änderungen auf diese Änderungen hingewiesen wird und er ggf. den Vorgang dann noch stoppen kann. Zum Abschluss der überschriebenen Methode wird noch das OnClosing Ereignis der Basisklasse aufgerufen, um das Formular korrekt zu schließen.

Weitere zu implementierende Ereignisse sind die Button Ereignisse des Formulars. Folgende Ereignisse werden vom Formular ausgelöst:

Tabelle hinzufügen (btnAdd_Click)

Zum Hinzufügen einer Tabelle wird ein Navigationsfenster benötigt, über welches der Anwender die Möglichkeit hat, aus allen verfügbaren Bibliotheken die gewünschten Tabellen der Liste der zu verkettenden Tabellen hinzuzufügen. Der Task Consumer bietet hier eine Methode, die ein derartiges Navigationsfenster bereitstellt und die Auswahl des Benutzers als Rückgabewert zur Verfügung stellt. Es wird die Methode

showInputDataSelector des Task Consumers genutzt. Über die Aufrufparameter der Methode kann das Auswahlfenster soweit angepasst werden, dass Tabellen nur von einem vorgegebenen SAS Server abgerufen werden können und dass nur eine oder mehrere Tabellen gewählt werden können. Weiterhin kann die vorselektierte Bibliothek und die vorselektierte Tabelle angegeben werden. Als Ergebnis der Methode wird ein Array an Input Datenquellen übergeben, welches in einem privaten Attribut gespeichert wird und als Liste an das Listboxfeld gebunden wird, so dass die gewählten Tabellen automatisch angezeigt werden.

Tabelle löschen (btnDelete_Click)

Beim Löschen einer Tabelle aus der Liste wird über Standard Listenoperationen das Array-Element an der gewählten Stelle aus der Liste entfernt und anschließend die Anzeige aktualisiert. Das letzte Listenelement kann nicht gelöscht werden.

Tabelle ganz nach oben verschieben (btnMoveTop_Click)

Dieses Ereignis entfernt das gewählte Listenelement aus der Liste und fügt es an der ersten Listenposition wieder ein. Diese Operation ist beim ersten Listenelement nicht möglich.

Tabelle eine Position nach oben verschieben (btnMoveUp_Click)

Dieses Ereignis entfernt das gewählte Listenelement aus der Liste und fügt es an der nächsten kleineren Listenposition wieder ein. Diese Operation ist beim ersten Listenelement nicht möglich.

Tabelle eine Position nach unten verschieben (btnMoveDown_Click)

Dieses Ereignis entfernt das gewählte Listenelement aus der Liste und fügt es an der nächsten größeren Listenposition wieder ein. Diese Operation ist beim letzen Listenelement nicht möglich.

Tabelle ganz nach unten verschieben (btnMoveBottom_Click)

Dieses Ereignis entfernt das gewählte Listenelement aus der Liste und fügt es an der letzten Listenposition wieder ein. Diese Operation ist beim letzten Listenelement nicht möglich.

Ausgabetabelle wählen (btnSelectAusgabe_Click)

Zum Wählen der Ausgabetabelle wird, analog zum Hinzufügen weiterer Eingabetabellen, das vom Task Consumer bereitgestellte Auswahlfenster genutzt. In diesem Fall wird die Methode ShowOutputDataSelector genutzt. Auch hier kann gewählt werden,

ob die Ausgabetabelle auf einem fest vorgegebenen SAS Server oder einem beliebigen SAS Server erstellt werden kann. Weiterhin kann die für die Navigation gewählte Ausgangsbibliothek angegeben werden. Das Ergebnis der Wahl wird in einem privaten Klassenattribut gespeichert und zur Visualisierung in dem Textboxfeld angezeigt.

Der vollständige Programmcode der Klasse AppendTableTaskForm ist diesem Dokument im Anhang C beigefügt.

3.6 Installation im Enterprise Guide

Nachdem das Visual C# 2010 Express Projekt entwickelt wurde, kann dieses über die Menüfunktion Debuggen – Projektmappe erstellen kompiliert werden. Sollten während der Kompilierung Fehler oder Hinweise festgestellt werden, so werden diese im Fenster „Fehlerliste“ angezeigt und müssen behoben werden, bevor die Erweiterung im Enterprise Guide genutzt werden kann. Im Idealfall werden 0 Fehler, 0 Warnungen und 0 Informationen ausgegeben.

Für einen ersten Test kann im Enterprise Guide ein Verweis auf die neu kompilierte DLL-Datei eingerichtet werden. Hierzu ist im Menü Tools – Add-Ins – Add-In Manager in der User-Registry ein Verweis auf die neu erzeugte DLL herzustellen.

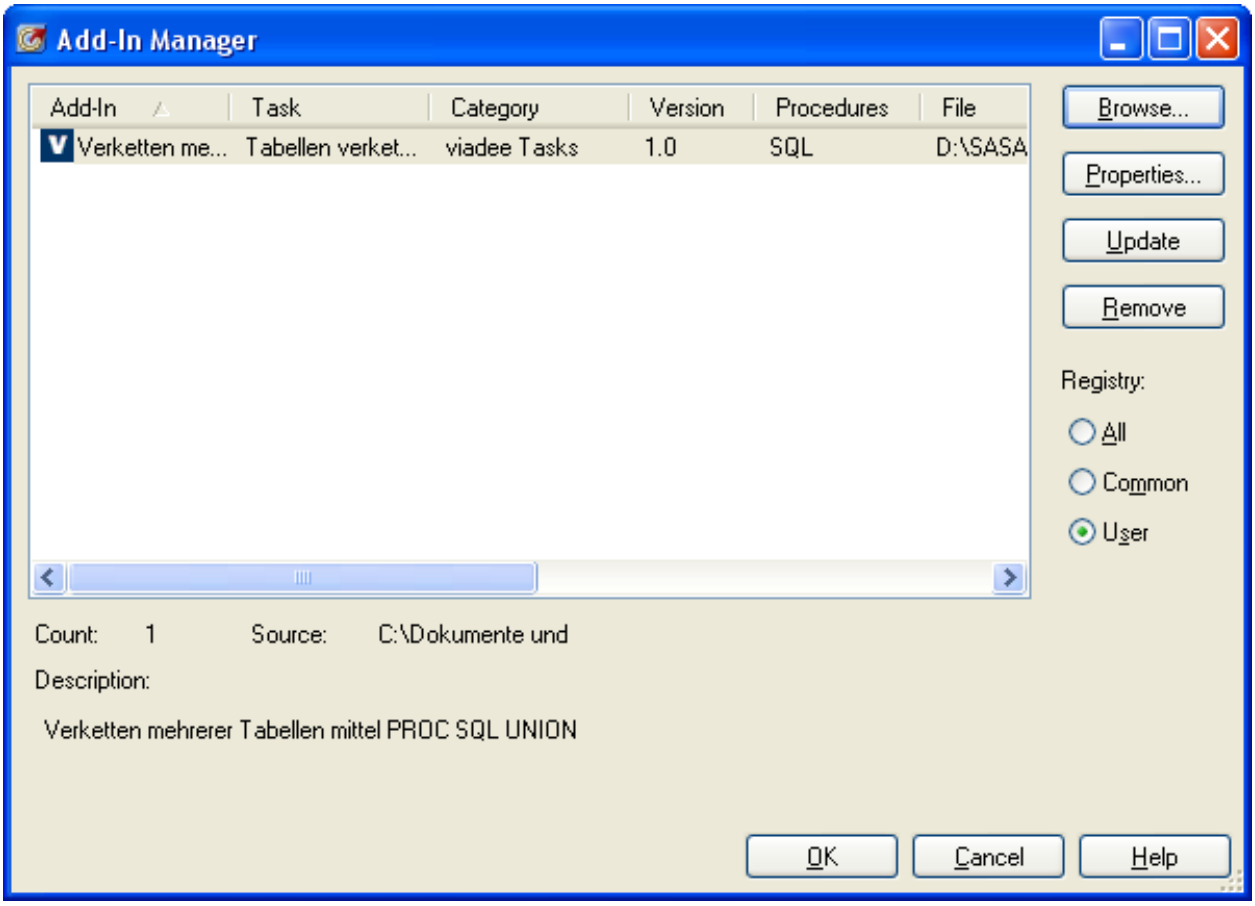

Abbildung 4: Add-In Registry im Enterprise Guide

Anschließend wird der neu erstellte Task in der Taskliste des Enterprise Guide angezeigt und kann im Projekt verwendet werden.

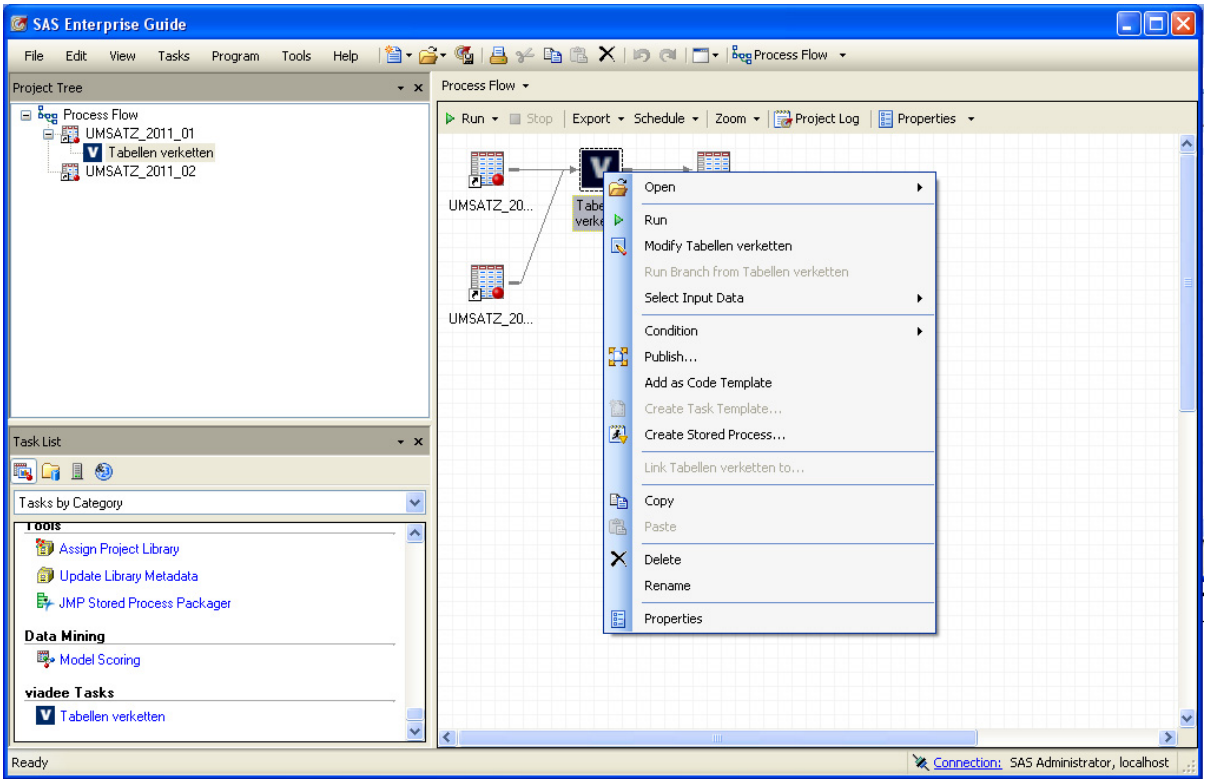

Abbildung 5: Darstellung des neuen Tasks im Prozessfluss im Enterprise Guide

Über das Kontextmenü kann der neue Task modifiziert werden. In diesem Fall wird das Benutzerformular angezeigt und kann verändert werden. Alternativ kann die Eingabedatenquelle über das Kontextmenü festgelegt werden und der Task ausgeführt werden. Nach der Ausführung wird die Ausgabetabelle automatisch erstellt und geöffnet. Im SAS Log wird der generierte SAS Code angezeigt, so dass die Funktionsweise des ersten Custom Tasks leicht nachvollzogen werden kann.

4 Ausblick

Das vorherige Beispiel hat gezeigt, wie leicht Assistentenfunktionen des Enterprise Guide mit Microsoft Visual C# 2010 Express entwickelt werden können. Der hier dargestellte Task sollte jedoch hauptsächlich als Anregung dienen, um zu zeigen, was alles mit Custom Tasks im Enterprise Guide möglich ist.

Neben der Generierung von SAS Code für Standard SAS Funktionen besteht natürlich auch die Möglichkeit, Eingabeassistenten für eigene, komplexe SAS Makros oder SAS Prozessflüsse zu entwickeln. So könnten beispielsweise die Aufrufparameter für ein statistisches Analysemakro über einen Enterprise Guide Assistenten definiert werden. So könnten direkt bei der Parameterdefinition wichtige Gültigkeitsprüfungen und Vali-

dierungen vorgenommen werden, um so den Aufruf des Makros nur mit gültigen Aufrufparametern zu ermöglichen.

Neben der Code-Generierung für eine SAS Verarbeitung können Custom Task auch genutzt werden, um in der SAS Programmiersprache fehlende Funktionen durch .NET Funktionen zu ergänzen.

Eine derartige Funktionalität wäre die automatische Datenergänzung einer Eingabetabelle durch Werte, die über einen Webservice im Internet oder im Intranet abgerufen werden. Hierzu wird ausgehend von dem Task Consumer (z.B. Workspace-Server des Enterprise Guide Prozesses) eine OLE DB Connection erstellt, über den auf eine Datentabelle auf dem SAS Server zugegriffen wird. Diese Datentabelle kann über eine .NET DataTable durchlaufen und verarbeitet werden. Das Ergebnis der Verarbeitung kann anschließend auf den SAS Server zurückgespielt werden und dann mit SAS Mitteln weiterverarbeitet werden. Hierzu könnte beispielsweise die GetSasCode() Methode des Tasks genutzt werden.

```
Server = new SAS.Tasks.Toolkit.SasServer(Consumer.AssignedServer);
OleDbConnection connection= Server.GetOleDbConnection();
String sql = "SELECT * FROM KUNDEN.KUNDENTABELLE ORDER BY ID";
connection.Open();
OleDbDataAdapter adapter = new OleDbDataAdapter(sql, connection);
DataTable table = new DataTable();
adapter.Fill(table);
connection.Close();
StringBuilder sb = new StringBuilder();
WebService.NameService ns = new WebService.NameService();
sb.AppendLine("DATA Work.KUNDENTABELLE; ");
sb.AppendLIne("  SET KUNDEN.KUNDENTABELLE; STOP; ");
sb.AppendLine("RUN;");
foreach (DataRow row in table.Rows) {
  WebService.AdressInfo a = ns.SucheAdresse(Convert.ToInt32(row["ID"]));
  if (a != null) {
    sb.AppendLine("PROC SQL NOPRINT;");
    sb.AppendLine(string.Format("INSERT INTO WORK.KUNDENTABELLE
          (NAME, STRASSE, PLZ, ORT, TELEFON, TELEFAX, ID) VALUES
          ('{0}', '{1}', '{2}', '{3}', '{4}', '{5}', {6});",
          a.Name, a.Strasse, a.PLZ, a.Ort, a.Telefon, a.Telefax, row["ID"]));
    sb.AppendLine("QUIT;");
  }
}
sb.AppendLine("DATA KUNDEN.KUNDENTABELLE; SET WORK.KUNDENTABELLE; RUN;");
return sb.ToString();
```

In diesem Fall würde der gesamte Task ohne jegliche Benutzerinteraktion auskommen. Es würden alle Sätze aus der Eingangstabelle ausgelesen, in einer temporären Zwischentabelle würden für alle Sätze die aktualisierten Daten zwischengespeichert und nach Abschluss der Datenaktualisierung über den WebService würde die Quelltabelle durch die aktuellen Daten ersetzt werden. Erweitern könnte man dieses Beispiel natürlich noch um ein Benutzerformular, über welches der Aktualisierungs-Service, die Ak-

tualisierungsvariable oder die Quelltabelle auswählen lässt. Genauso könnten die zu aktualisierenden Sätze im Benutzer-Frontend angezeigt und auswählbar gemacht werden bzw. nur bestimmte Sätze aktualisiert werden. Eine direkte Aktualisierung ist auch über einen OleDbCommand möglich, über den ein SQL direkt an den SAS Server zur Verarbeitung geschickt werden kann. Alternativ können über das SasServer-Objekt bzw. den Task Consumer SAS Programme direkt ausgeführt werden, ohne hierzu auf die Ausführung des Enterprise Guide Prozessflusses angewiesen zu sein. So können Datenänderungen bzw. –aktualisierungen direkt aus dem Benutzer-Frontend initiiert und gespeichert werden.

Analysiert man die unterschiedlichsten Möglichkeiten, die sich durch die Erweiterbarkeit des Enterprise Guide und der Add-Ins für Microsoft Office mittels des .NET Frameworks ergeben, so erkennt man schnell, dass hier eine sehr mächtige Toolbox zur Verfügung steht, die nur darauf wartet in vollem Umfang genutzt zu werden.

Literatur

[1] SAS Enterprise Guide – Creating Custom Add-In Tasks for SAS Enterprise Guide, http://support.sas.com/documentation/onlinedoc/guide/customtasks/index.htm, Abrufdatum: 17.02.2011

Anhang A: Listing Klasse AppendTableTask.cs

```
using System;
using System.Text;
using System.Collections;
using System.Collections.Generic;
using SAS.Shared.AddIns;
using SAS.Tasks.Toolkit;

namespace AppendTable {
  [ClassId("25804bac-08f5-4663-be25-556c0d4cd52a")]
  [IconLocation("AppendTable.viadee.ico")]
  [Version(1.0)]
  [InputRequired(InputResourceType.Data)]
  public class AppendTableTask : SAS.Tasks.Toolkit.SasTask {
    #region Private Members
      private AppendTableTaskSettings settings;
    #endregion
    #region Initialization
      public AppendTableTask() {
        InitializeComponent();
      }
      private void InitializeComponent() {
        this.GeneratesReportOutput = false;
        this.ProcsUsed = "SQL";
        this.ProductsRequired = "BASE";
        this.TaskCategory = "viadee Tasks";
        this.TaskDescription = "Verketten mehrerer Tabellen mittels
                               PROC SQL UNION";
```

```
        this.TaskName = "Tabellen verketten";
      }
    #endregion
    #region Overrides
      public override bool Initialize() {
        settings = new AppendTableTaskSettings(this.Consumer);
        return true;
      }
      public override int OutputDataCount {
        get { return 1; }
      }
      public override List<ISASTaskDataDescriptor>
                              OutputDataDescriptorList {
        get {
          List<ISASTaskDataDescriptor> outputData =
                      new List<ISASTaskDataDescriptor>();
          outputData.Add( SASTaskDataDescriptor.CreateLibrefData
                      Descriptor(Consumer.AssignedServer,
                                 settings.outputData.Library,
                                 settings.outputData.Member,
                                 settings.outputData.Library + "." +
                                 settings.outputData.Member));
          return outputData;
        }
      }
      public override string GetXmlState() {
        return settings.ToXml();
      }
      public override void RestoreStateFromXml(string xmlState) {
        settings = new AppendTableTaskSettings(this.Consumer);
        settings.FromXml(xmlState);
      }
      public override ShowResult Show(
                System.Windows.Forms.IWin32Window Owner) {
        AppendTableTaskForm dlg =
                       new AppendTableTaskForm(this.Consumer,
                                               settings);
        System.Windows.Forms.DialogResult dlgResult =
                                  dlg.ShowDialog(Owner);
        if ((dlgResult == System.Windows.Forms.DialogResult.OK) ||
            (dlgResult == System.Windows.Forms.DialogResult.No)) {
          settings = dlg.Settings;
          if (dlgResult == System.Windows.Forms.DialogResult.OK) {
            return ShowResult.RunNow;
          } else {
            return ShowResult.RunLater;
          }
        } else {
          return ShowResult.Canceled;
        }
      }
      public override string GetSasCode() {
        return settings.GetSasProgram();
      }
    #endregion
  }
}
```

Anhang B: Listing Klasse AppendTableTaskSettings.cs

```
using System;
using System.Text;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Xml;
using SAS.Shared.AddIns;
using AppendTable.Helper;

namespace AppendTable {
  public class AppendTableTaskSettings {
    private ISASTaskConsumer3 Consumer;
    public AppendTableTaskSettings(ISASTaskConsumer3 consumer) {
      genView = false;
      genTable = true;
      this.Consumer = consumer;
      Random r = new Random();
      this.outputData = Consumer.LibrefData(Consumer.AssignedServer,
                                            "WORK",
                        "VIADEE_APPEND_" + r.Next().ToString());
    }
    public bool genView { get; set; }
    public bool genTable { get; set; }
    public ISASTaskData outputData { get; set; }
    public string ToXml() {
      XmlDocument doc = new XmlDocument();
      XmlElement el = doc.CreateElement("ViadeeAppendTableTask");
      el.SetAttribute("genView", genView.ToString());
      el.SetAttribute("genTable", genTable.ToString());
      doc.AppendChild(el);
      XmlElement output = doc.CreateElement("outputData");
      output.SetAttribute("Library", outputData.Library);
      output.SetAttribute("Member", outputData.Member);
      output.SetAttribute("Server", outputData.Server);
      el.AppendChild(output);
      return doc.OuterXml;
    }
    public void FromXml(string xml) {
      XmlDocument doc = new XmlDocument();
      try {
        doc.LoadXml(xml);
        XmlElement el = doc["ViadeeAppendTableTask"];
        genView = Convert.ToBoolean(
                  el.GetAttribute("genView").ToString());
        genTable = Convert.ToBoolean(
                  el.GetAttribute("genTable").ToString());
        XmlElement output = el["outputData"];
        this.outputData = this.Consumer.LibrefData(
                      output.GetAttribute("Server").ToString(),
                      output.GetAttribute("Library").ToString(),
                      output.GetAttribute("Member").ToString());
      } catch (XmlException) { }
    }
    public string GetSasProgram() {
      int counter = 0;
      StringBuilder sb = new StringBuilder();
      sb.AppendLine("PROC SQL NOPRINT;");
      if (this.genView) {
        sb.Append("  CREATE VIEW ");
      } else {
```

```
        sb.Append("  CREATE TABLE ");
      }
      sb.AppendLine(outputData.Library + "." +
                    outputData.Member + " AS ");
      foreach (ISASTaskData data in Consumer.InputData) {
        if (counter > 0) {
          sb.AppendLine("         OUTER UNION CORR       ");
        }
        sb.Append("     SELECT * FROM ");
        sb.Append(data.Library);
        sb.Append(".");
        sb.AppendLine(data.Member);
        counter++;
      }
      sb.AppendLine("; QUIT;");
      return sb.ToString();
    }
  }
}
```

Anhang C: Listing Klasse AppendTableTaskForm.cs

```
using System;
using System.ComponentModel;
using System.Collections;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using SAS.Shared.AddIns;
using SAS.Tasks.Toolkit.Controls;
using AppendTable.Helper;

namespace AppendTable {
  public partial class AppendTableTaskForm :
                        SAS.Tasks.Toolkit.Controls.TaskForm {
    public AppendTableTaskSettings Settings { get; set; }
    private ISASTaskData outputData { get; set; }
    private BindingList<InputTable> inputData { get; set; }
    public AppendTableTaskForm(
          SAS.Shared.AddIns.ISASTaskConsumer3 consumer,
          AppendTableTaskSettings settings) {
      InitializeComponent();
      this.Consumer = consumer;
      this.Settings = settings;
    }
    protected override void OnLoad(EventArgs e) {
      base.OnLoad(e);
      this.rbView.Checked    = Settings.genView;
      this.rbTabelle.Checked = Settings.genTable;
      this.outputData        = Settings.outputData;
      this.tbAusgabe.Text    = this.outputData.Library + "." +
                               this.outputData.Member;
      this.inputData         = new BindingList<InputTable>();
      foreach (ISASTaskData data in Consumer.InputData) {
        inputData.Add(new InputTable(data));
      }
      this.lbEingabe.DataSource = this.inputData;
    }
    protected override void OnClosing(CancelEventArgs e) {
```

```
  if ((this.DialogResult == DialogResult.OK) ||
      (this.DialogResult == DialogResult.No)) {
    Settings.genTable = this.rbTabelle.Checked;
    Settings.genView = this.rbView.Checked;
    Settings.outputData = this.outputData;
    Consumer.ClearInputData();
    foreach (InputTable data in this.inputData) {
      Consumer.AddInputData(data.data);
    }
  }
  base.OnClosing(e);
}
private void btnSelectAusgabe_Click(object sender,
                                    EventArgs e) {
  string cookie = string.Empty;
  ISASTaskDataName outputData =
    this.Consumer.ShowOutputDataSelector(this.Owner,
                            ServerAccessMode.OneServer,
                            this.outputData.Server,
                            this.outputData.Library,
                            this.outputData.Member,
                            ref cookie);
  if ((outputData != null) &&
      ((this.outputData.Server != outputData.Server) ||
       (this.outputData.Library != outputData.Library) ||
       (this.outputData.Member != outputData.Member))) {
    this.outputData = this.Consumer.LibrefData(
                           outputData.Server,
                           outputData.Library,
                           outputData.Member);
    this.tbAusgabe.Text = this.outputData.Library + "." +
                          this.outputData.Member;
  }
}
private void btnAdd_Click(object sender, EventArgs e) {
  string cookie = string.Empty;
  ISASTaskData[] dataArray =
     this.Consumer.ShowInputDataSelector(this.Owner,
                          DataSelectionMode.MultipleInput,
                          ServerAccessMode.OneServer,
                          this.Consumer.ActiveData.Server,
                          this.Consumer.ActiveData.Library,
                          this.Consumer.ActiveData.Member,
                          ref cookie);
  if (dataArray != null) {
    if (dataArray.Length > 0) {
      for (int i = 0; i < dataArray.Length; i++) {
        inputData.Add(new InputTable(dataArray[i]));
      }
    }
  }
}
private void btnDelete_Click(object sender, EventArgs e) {
  int selectedIndex = this.lbEingabe.SelectedIndex;
  if ((selectedIndex >= 0) && (selectedIndex<inputData.Count)) {
    if (inputData.Count > 1) {
      inputData.RemoveAt(selectedIndex);
      this.lbEingabe.SelectedIndex = 0;
    }
  }
}
private void btnMoveTop_Click(object sender, EventArgs e) {
  int selectedIndex = this.lbEingabe.SelectedIndex;
```

```
      if ((selectedIndex >= 0) && (selectedIndex<inputData.Count)) {
        InputTable table = inputData[selectedIndex];
        inputData.RemoveAt(selectedIndex);
        inputData.Insert(0, table);
        this.lbEingabe.SelectedIndex = 0;
      }
    }
    private void btnMoveBottom_Click(object sender, EventArgs e) {
      int selectedIndex = this.lbEingabe.SelectedIndex;
      if ((selectedIndex >= 0) && (selectedIndex<inputData.Count)) {
        InputTable table = inputData[selectedIndex];
        inputData.RemoveAt(selectedIndex);
        inputData.Add(table);
        this.lbEingabe.SelectedIndex = inputData.Count - 1;
      }
    }
    private void btnMoveUp_Click(object sender, EventArgs e) {
      int selectedIndex = this.lbEingabe.SelectedIndex;
      if ((selectedIndex > 0) && (selectedIndex<inputData.Count)) {
        InputTable table = inputData[selectedIndex];
        inputData.RemoveAt(selectedIndex);
        inputData.Insert(selectedIndex - 1, table);
        this.lbEingabe.SelectedIndex = selectedIndex - 1;
      }
    }
    private void btnMoveDown_Click(object sender, EventArgs e) {
      int selectedIndex = this.lbEingabe.SelectedIndex;
      if ((selectedIndex >= 0) &&
          (selectedIndex < inputData.Count - 1)) {
        InputTable table = inputData[selectedIndex];
        inputData.RemoveAt(selectedIndex);
        inputData.Insert(selectedIndex + 1, table);
        this.lbEingabe.SelectedIndex = selectedIndex + 1;
      }
    }
  }
}
```