Kapitel 4
Basisfunktionen

Die richtige Basis ist die halbe Miete. Steigern Sie die Effektivität und Qualität mithilfe von Standards und Wiederverwendbarkeit.

Wenn die grundlegende Struktur der VS-Solution aufgebaut ist, können Sie mit der Anlage der ersten tatsächlichen Anwendungsbestandteile beginnen. Auch dabei ist es ratsam, sich einen gewissen Standard anzueignen und ihn in allen Projekten beizubehalten. Das erleichtert allen Beteiligten die Einarbeitung in Ihre Projekte und ermöglicht einen schnellen und reibungslosen Support, auch bei unbekannten Projekten. Wenn der Code und die Objektstrukturen projektübergreifend identisch sind, muss sich der Entwickler im weitesten Sinne nur noch fachlich in die Anwendung einfinden. Der technische Aufbau ist vertraut, und dank einer logischen Gliederung sind einzelne Objekte im Projekt intuitiv aufzufinden. Auch bei Erweiterungen verhindern Sie durch eine klare und einheitliche Struktur einen zu starken Wildwuchs an unterschiedlichen Entwicklungsansätzen. Wird ein Kollege ausschließlich für eine bestimmte Erweiterung in einem Projekt eingesetzt, weiß er direkt, wo er die einzelnen Bestandteile der Erweiterung ablegen sollte.

Einige Bausteine einer Lösung sind von Anwendung zu Anwendung wiederkehrend. So sollte eine Anwendung grundsätzlich mit einer ausreichenden Loggingfunktionalität ausgestattet sein und über Mehrsprachigkeit verfügen. Auch der Einsatz des Ribbons ist in SharePoint obligatorisch, um ein einheitliches Erscheinungsbild des UI zu wahren. Zusätzlich gibt es für den UI-Bereich immer wieder die gleichen JavaScript-Blöcke, mit denen zum Beispiel Meldungen an den Benutzer ausgegeben oder modale Dialoge gestartet werden. Alle diese Codeabschnitte brauchen Sie nicht in jedem Projekt neu zu entwickeln. Am besten fahren Sie, wenn Sie diese Dinge in eine ausgelagerte Klasse kapseln und sie dadurch in allen Ihren Projekten wiederverwenden können. Das steigert Ihre Effektivität und die Qualität, da Kernfunktionalitäten Ihrer Anwendung bei mehreren Kunden im Einsatz sind. Fehler fallen dadurch schneller auf, können übergreifend korrigiert werden und treten im Folgeprojekt nicht erneut auf.

Als Erstes sollten Sie dazu die Klassen und Methoden in Ihrem Projekt anlegen, die individuelle Projektbestandteile – wie Pfade, Konstanten oder Variablen – haben und nicht komplett gekapselt werden können. Im zweiten Schritt sollten Sie eine Framework-Bibliothek erstellen, in der Sie wiederkehrende Funktionen auslagern und somit die Möglichkeit schaffen, diese auf Ihre unterschiedlichen Projekte zu verteilen und bei Bedarf wiederzuverwenden.

4.1 Additional Page-Header

Additional Page-Header sind eine elegante Möglichkeit, Funktionen anwendungsübergreifend bereitzustellen. In Form von UserControls können Sie z. B. umfangreiche Methoden, die je nach aktuellem Kontext bestimmte Aktionen ausführen, realisieren oder JavaScript-Dateien global bereitstellen. Page-Header sind sehr vielseitig einsetzbar und in den meisten Fällen eine wesentlich bessere Lösung als die fixe Einbindung von Komponenten in die Masterpage oder das immer wiederkehrende und notfalls sogar manuelle Einbinden in alle ApplicationPages oder per WebPart auf den gewünschten Wiki- und WebPart-Seiten.

Der Page-Header wird mit Ihrer Lösung ausgerollt und steht danach als Bestandteil Ihres Features zur Verfügung. Ist das Feature in einem Web oder in einer SiteCollection aktiviert, ist er eingebunden und stellt seine Funktionen automatisch auf jeder Seite des aktuellen *Feature Scopes* zur Verfügung. Wird das Feature deaktiviert, sind auch alle Bestandteile des Page-Headers restlos von Ihrer Seite entfernt. Dies garantiert nicht nur ein sauberes Deployment und Retractment, sondern sorgt auch dafür, dass Ihre Funktionen, anders als bei der Einbindung über eine Masterpage, nur dort verfügbar sind, wo Sie sie auch benötigen. Als ersten Page-Header werden wir nun in TicketPoint 2019 eine JavaScript-Bibliothek bereitstellen, in der alle übergreifenden Skripte bereitgestellt werden.

Öffnen Sie dazu die VS-Solution, navigieren Sie im Solution Explorer zum *UILayer*-Projekt und fügen Sie dort im Anwendungsverzeichnis *TicketPoint2019* unter *CONTROLTEMPLATES* ein UserControl mit dem Namen »JSLib.ascx« hinzu.

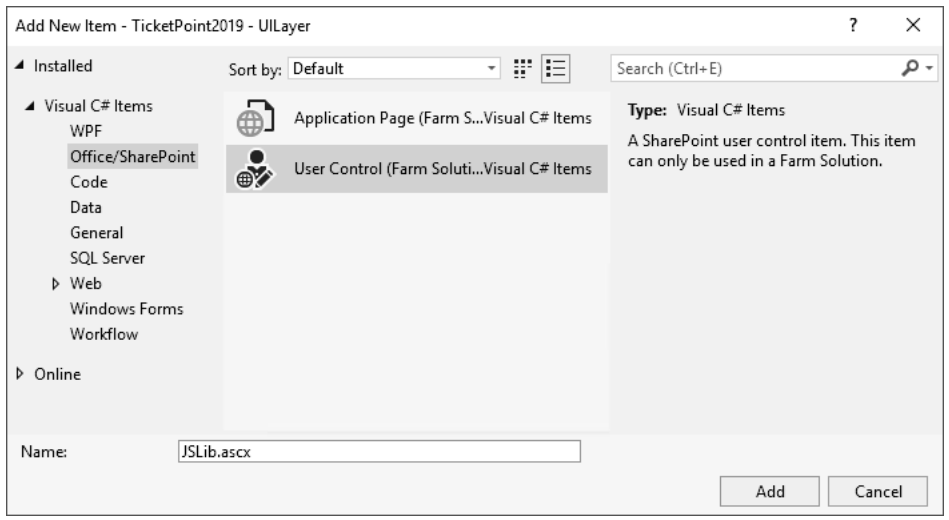

Abbildung 4.1 Hinzufügen von »JSLib.ascx«

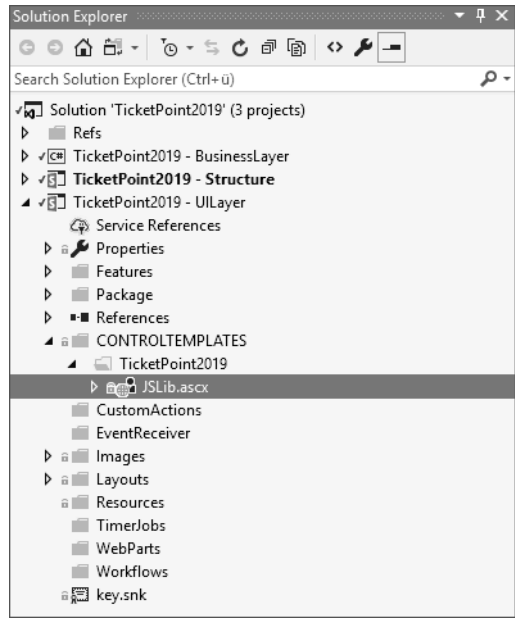

Abbildung 4.2 »JSLib.ascx« in der Projektstruktur

Nachdem Sie das UserControl erstellt haben, das im weiteren Projektverlauf als Ablagebibliothek für JavaScript-Snippets dienen wird, sorgen Sie dafür, dass das Skript automatisch auf allen Seiten eingebunden wird. Hierzu legen Sie nun im Verzeichnis *CustomActions* ein neues, leeres SharePoint-Element mit dem Namen »JSLib« an.

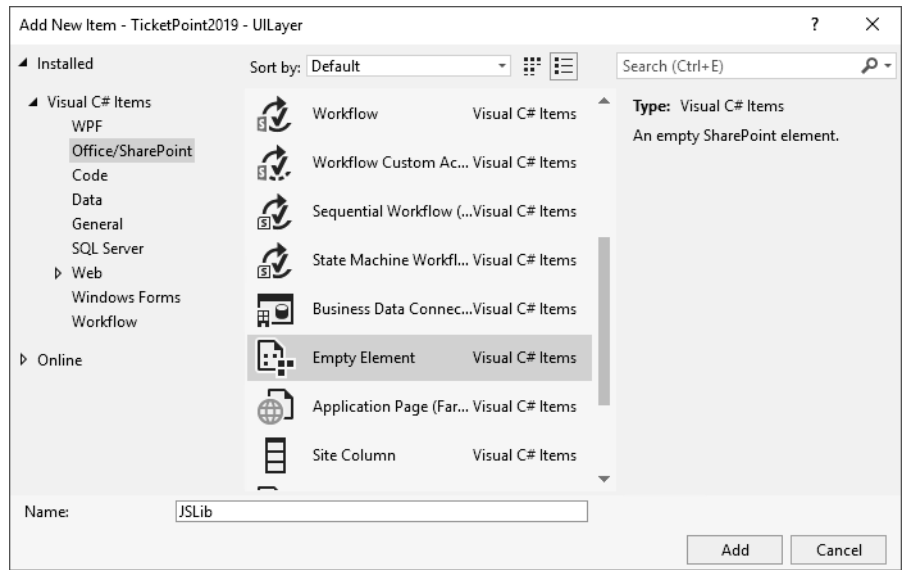

Abbildung 4.3 Hinzufügen von JSLib-CustomAction

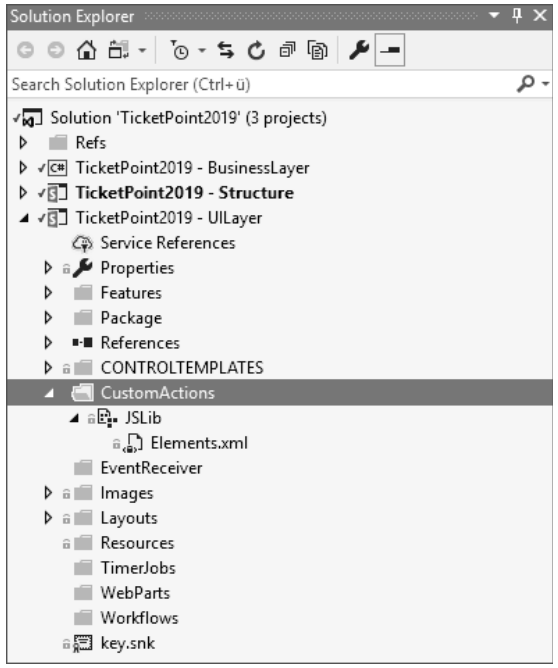

Abbildung 4.4 JSLib-CustomAction in der Projektstruktur

Unterhalb des *JSLib*-Pakets wird automatisch eine Datei *Elements.xml* angelegt. Fügen Sie in dieser Datei nun das XML aus Listing 4.1 ein.

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
    <Control Id="AdditionalPageHead"
            ControlSrc="~/_controltemplates/15/TicketPoint2019/JSLib.ascx"
            Sequence="0" />
</Elements>
```

Listing 4.1 AdditionalPageHeader

Mithilfe dieses XML in den *CustomActions* deklarieren Sie die Datei *JSLib.ascx* in Ihren *CONTROLTEMPLATES* als `AdditionalPageHeader`-Control, das an der nullten Position geladen werden soll. Das heißt, dieses Control wird auf jeder Seite (benutzerdefinierte und Standardseiten) im Web geladen, wenn das Feature aktiv ist.

Wenn Sie mit mehreren Page-Headern arbeiten, können Sie die Position der Controls und somit die Reihenfolge der Einbindung auf der Seite durch die Sequenz beeinflussen. Dies ist beispielsweise bei aufeinander aufbauenden JavaScripts hilfreich. Als letzten Schritt müssen Sie die erstellte CustomAction noch in das gewünschte Fea-

ture aufnehmen. Öffnen Sie dazu das Feature *Fkr.SharePoint.TicketPoint2019.UILayer* und nehmen Sie die *CustomAction* als Featurebestandteil auf.

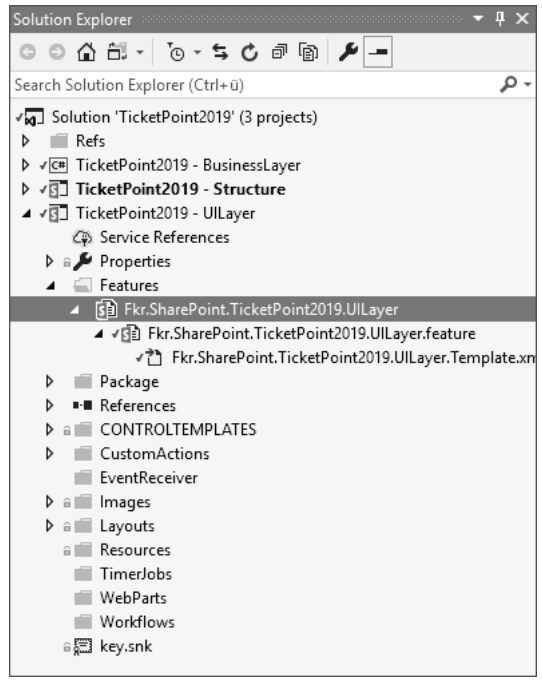

Abbildung 4.5 Feature »UILayer« in der Projektstruktur

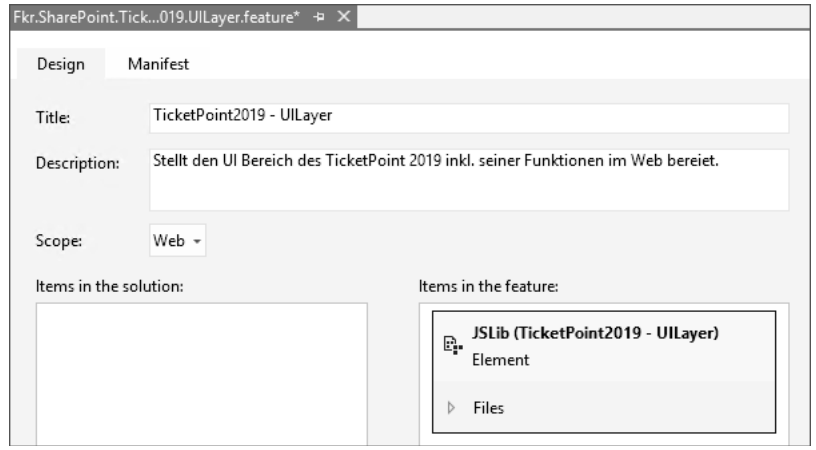

Abbildung 4.6 Feature »UILayer« – JSLib hinzufügen

Deployen Sie die Lösung auf Ihren SharePoint und prüfen Sie so, ob die Bereitstellung fehlerfrei funktioniert.

4.2 Logging

Ein wichtiger Bestandteil einer jeden Anwendung ist ein zuverlässiger Mechanismus zum Loggen von Fehlern und Informationen. In diesem Abschnitt werden wir einen solchen Mechanismus für unsere Anwendung erstellen. Unser Logmechanismus soll anwendungsweit zur Verfügung stehen, Fehler klassifizieren, nachhaltig speichern und eine Methode für eine Just-in-time-Analyse bereitstellen.

Um alle Kriterien zu erfüllen, erstellen wir in unserem *BusinessLogic*-Projekt eine Klasse mit dem Namen *Logging*. In dieser Klasse stellen wir Methoden bereit, die bei ihrem Aufruf die Informationen parallel in das ULS-Log von SharePoint speichern und gleichzeitig direkt im System-Trace ausgeben.

Die Anwendungsbereiche und Loglevel, die innerhalb der Anwendung zur Verfügung stehen sollen, werden wir als Konstanten in unserer Business-Logic anlegen, damit sie jederzeit und ohne große Überlegungen abgerufen werden können. Dazu erstellen wir in unserer Business-Logic eine Klasse mit dem Namen *Constants*. Im Laufe der Entwicklung werden alle notwendigen Konstanten in diese Klasse aufgenommen. Um eine übersichtliche Gliederung der Klasse zu gewährleisten, werden wir themenbezogene Konstanten in entsprechende Unterklassen kapseln, um den Zugriff darauf zu erleichtern. Für unser Log benötigen wir eine Konstante für den Namen unserer Anwendung und die Subklassen `LogCategory` und `LogLevel`. Erstellen Sie nun die entsprechende Klasse und fügen Sie den Code aus Listing 4.2 ein.

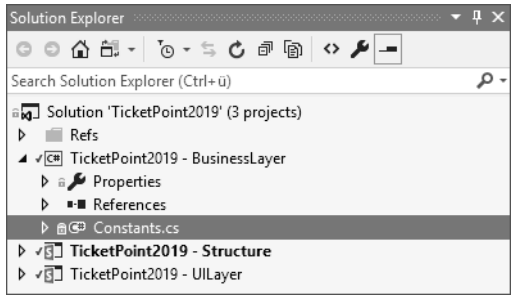

Abbildung 4.7 Klasse »Constants« in der Projektstruktur

```
namespace Fkr.SharePoint.TicketPoint2019.BusinessLayer
{
    public static class Constants
    {
        public const string ApplicationName = "TicketPoint 2019";
        /// <summary>
        /// Stellt die verfügbaren Log-Kategorien der Anwendung bereit
        /// </summary>
        public static class LogCategory
```



```
        {
            public const string UI = "UI";
            public const string Structure = "Structure";
            public const string BusinessLogic = "BusinessLogic";
            public const string WebPart = "WebPart";
            public const string TimerJob = "TimerJob";
            public const string EventReceiver = "EventReceiver";
        }
        /// <summary>
        /// Stellt die verfügbaren LogLevel der Anwendung bereit
        /// </summary>
        public static class LogLevel
        {
            public const string None = "None";
            public const string High = "High";
            public const string Medium = "Medium";
            public const string Verbose = "Verbose";
            public const string Unexpected = "Unexpected";
            public const string Monitorable = "Monitorable";
        }
    }
}
```

Listing 4.2 Bereitstellung von Konstanten

Nachfolgend erstellen Sie im *BusinessLayer*-Projekt eine Klasse mit dem Namen `Logging` und fügen dem Projekt eine Referenz zur *Microsoft.SharePoint*-Assembly hinzu.

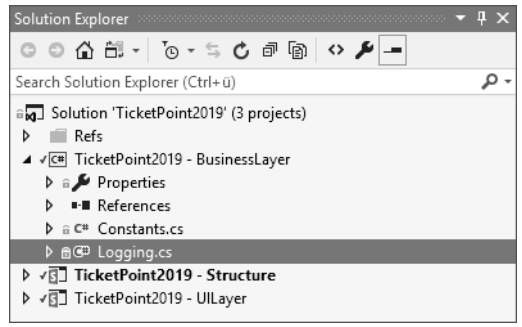

Abbildung 4.8 Klasse »Logging« in der Projektstruktur

Deklarieren Sie die Klasse als statisch und fügen Sie die `using`-Direktiven `Microsoft.SharePoint` und `Microsoft.SharePoint.Administration` ein. Über Objekte aus dem Namespace `Microsoft.SharePoint` werden die Logeinträge mit Benutzerinformationen angereichert, falls diese verfügbar sind. Der Namespace `Microsoft.Share-`

Point.Administration wird benötigt, um auf das ULS-Log von SharePoint zuzugreifen.

```
using Microsoft.SharePoint;
using Microsoft.SharePoint.Administration;
using System;
using System.Runtime.InteropServices;
namespace Fkr.SharePoint.TicketPoint2019.BusinessLayer
{
    public static class Logging
    {
    }
}
```

Listing 4.3 Logging-Klasse

Zu Beginn erstellen wir eine Methode, die Informationen in das ULS-Log von SharePoint schreibt. Fügen Sie dazu der Klasse die Methode aus Listing 4.4 hinzu.

```
private static void Log2Uls(Exception ex,
    string category,
    TraceSeverity traceSeverity)
{
    SPDiagnosticsCategory diagnosticsCategory =
        new SPDiagnosticsCategory(category,
        traceSeverity,
        EventSeverity.Error);
    SPDiagnosticsService.Local.WriteTrace(0,
        diagnosticsCategory,
        traceSeverity,
        ex.Message,
        ex.StackTrace);
}
```

Listing 4.4 ULS-Log

Zu Beginn der Methode wird ein Objekt erzeugt, mit dem die Kategorie, in der der Fehler aufgetreten ist, identifiziert wird. Zur Kategorisierung haben wir unsere Anwendung in die Bereiche UI, Structure, BusinessLayer, WebPart, TimerJob und Event aufgeteilt. Nach der Definition der Kategorie wird der Eintrag im ULS-Log erzeugt.

Als Nächstes benötigen wir eine Methode, mit der wir einen Eintrag im System-Trace erzeugen. Um den Logeintrag mit zusätzlichen Informationen anzureichern, benöti-

gen wir insgesamt drei Methoden. Als Erstes erstellen wir eine Methode, mit deren Hilfe wir die *CorrelationId* der aktuellen Aktion ermitteln.

```
[DllImport("advapi32.dll")]
private static extern uint EventActivityIdControl(uint controlCode,
    ref Guid activityId);
```

Anschließend benötigen wir eine Methode, die den Benutzer ermittelt, in dessen Kontext der Code ausgeführt wird. Falls bei der Ermittlung ein Fehler auftritt oder nicht alle Kontextinformationen verfügbar sind, wird als Fallback ein N/A als String zurückgegeben. Als Ergebnis liefert die Methode den Log-in-Namen des Benutzers.

```
private static string TryGetCurrentUsername()
{
    try
    {
        if (SPContext.Current.Web != null
            && SPContext.Current.Web.CurrentUser != null)
        {
            return SPContext.Current.Web.CurrentUser.LoginName;
        }
    }
    catch { }
    return "N/A";
}
```

Listing 4.5 Aktuellen Benutzer ermitteln

Mit der dritten Methode fassen wir alle Informationen zusammen und schreiben diese in das System-Trace.

```
private static void Log2Trace(Exception ex,
    string category,
    string logLevel)
{
    Guid correlationId = new Guid();
    EventActivityIdControl(1, ref correlationId);
    string username = TryGetCurrentUsername();
    string msg = "";
    if (logLevel == Constants.LogLevel.Monitorable)
    {
        string msgFormat = "FKR: [{0}][{1}] - [{2}]: {3}";
        msg = string.Format(msgFormat,
            Constants.ApplicationName,
```

```
            category,
            username,
            ex.Message);
    }
    else
    {
        string msgFormat = "FKR: [{0}][{1}][{2}] - [{3}]: {4} - {5}";
        msg = string.Format(msgFormat,
            Constants.ApplicationName,
            category,
            logLevel,
            username,
            ex.ToString(),
            (correlationId != null && correlationId != Guid.Empty)
                ? correlationId.ToString()
                : "");
    }
    System.Diagnostics.Trace.WriteLine(msg);
}
```

Listing 4.6 Ins System-Trace loggen

Um Informationen im System-Trace komfortabel finden zu können, stellen wir allen Einträgen ein eindeutiges Kürzel voran. Dies kann entweder ein Firmenkürzel, ein Entwicklerkürzel oder ein frei erfundener Text sein. Wie im Planungsdokument definiert, soll in TicketPoint 2019 jeder Eintrag mit dem Kürzel *FKR* beginnen.

Jetzt, da beide gewünschten Logvarianten verfügbar sind, erstellen wir eine öffentliche Methode, die einen Logeintrag anlegt.

```
public static void Log(Exception ex,
    string category,
    string logLevel)
{
    TraceSeverity traceSeverity;
    switch (logLevel)
    {
        case Constants.LogLevel.None:
            traceSeverity = TraceSeverity.None;
            break;
        case Constants.LogLevel.High:
            traceSeverity = TraceSeverity.High;
            break;
```



```
        case Constants.LogLevel.Medium:
            traceSeverity = TraceSeverity.Medium;
            break;
        case Constants.LogLevel.Verbose:
            traceSeverity = TraceSeverity.Verbose;
            break;
        case Constants.LogLevel.Monitorable:
            traceSeverity = TraceSeverity.Monitorable;
            break;
        default:
            traceSeverity = TraceSeverity.Unexpected;
            break;
    }
    Log2Trace(ex, category, logLevel);
    Log2Uls(ex, category, traceSeverity);
}
```

Listing 4.7 Allgemeine Log-Methode

Die Methode erzeugt als Erstes anhand des übergebenen Loglevels das `traceSeverity`-Objekt für den ULS-Eintrag. Danach werden die beiden Logvarianten aufgerufen.

Um die Entwicklung noch ein wenig zu erleichtern, möchten wir weitere Hilfsmethoden bereitstellen, mit denen die unterschiedlichen Loglevel schneller angesteuert werden können, allerdings ohne diese bei jedem Aufruf aus den Konstanten ermitteln zu müssen.

Dafür erstellen wir zum Abschluss die folgenden Hilfsmethoden, um unsere `Logging`-Klasse abzurunden.

```
public static void LogInfo(Exception ex, string category)
{
    Log(ex, category, Constants.LogLevel.Monitorable);
}
public static void LogInfo(string message,
    string category)
{
    Log(new Exception(message),
        category,
        Constants.LogLevel.Monitorable);
}
public static void LogError(Exception ex,
    string category)
```

```
{
    Log(ex, category, Constants.LogLevel.Unexpected);
}
```

Listing 4.8 Vereinfachte Log-Methoden

Jetzt haben wir einen umfangreichen Loggingmechanismus bereitgestellt, auf den aus der kompletten Anwendung zugegriffen werden kann, um Fehler und sonstige Meldungen umfangreich zu protokollieren.

Deployen Sie die Lösung auf Ihren SharePoint und prüfen Sie so, ob die Bereitstellung fehlerfrei funktioniert.

Begriffe

Die *CorrelationId* ist eine GUID, die als eindeutiges Kennzeichen alle Logeinträge zusammenfasst, die zu einem spezifischen Prozess gehören. Anhand der CorrelationId können Logeinträge eines Prozesses im ULS-Log ermittelt und zusammengefasst werden.

4.3 Mehrsprachigkeit

Um im späteren Lebenszyklus einer Anwendung große Mühen und Kosten zu sparen, sollten Sie eine Anwendung von Beginn an mit Basisfunktionalitäten für eine Mehrsprachigkeit ausstatten. Wenn das Thema Mehrsprachigkeit von Anfang des Projekts an in die Entwicklung einbezogen wird, liegt der Aufwand im Verhältnis zu einem späteren Nachrüsten bei nahezu null.

Um in einer SharePoint-Anwendung die Funktionen für Mehrsprachigkeit umzusetzen, sind lediglich eine Ressourcendatei als Wörterbuch und eine Klasse zum Auslesen des Wörterbuchs notwendig. Da TicketPoint 2019 als geschlossene Anwendung bereitgestellt wird, beschränken wir uns auf das Ausrollen einer einzelnen Ressourcendatei über das *UILayer*-Projekt. Um aus allen Anwendungsbereichen auf das Wörterbuch zugreifen zu können, wird die Übersetzerklasse im *BusinessLayer*-Projekt untergebracht.

Navigieren Sie in der geöffneten TicketPoint-2019-Solution zum *UILayer*-Projekt im Solution Explorer. Starten Sie mit der rechten Maustaste das Kontextmenü für den Mapped Folder RESOURCES und klicken Sie auf ADD und danach auf NEW ITEM. In dem nun gestarteten Dialog wählen Sie als Template die Ressourcendatei aus und vergeben den Namen »TicketPoint 2019.resx«.

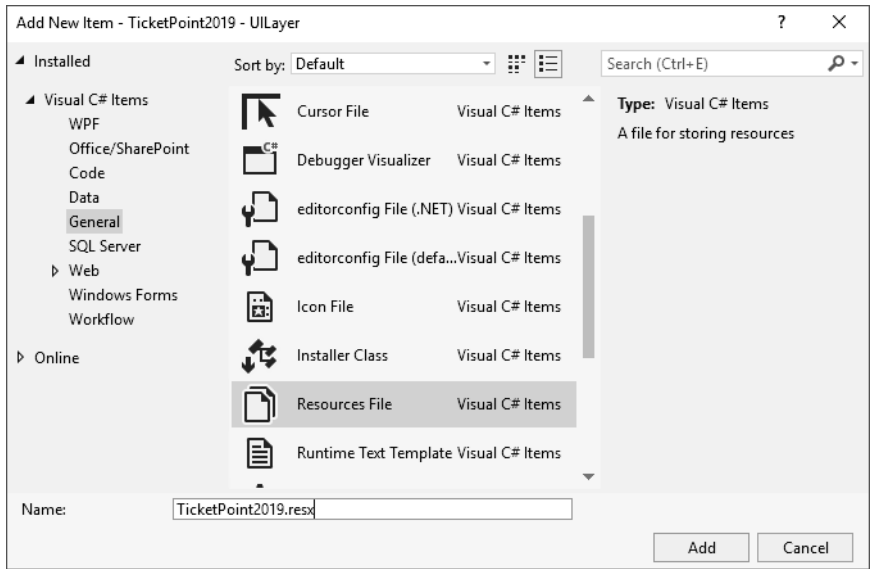

Abbildung 4.9 Add New Item – Resources File

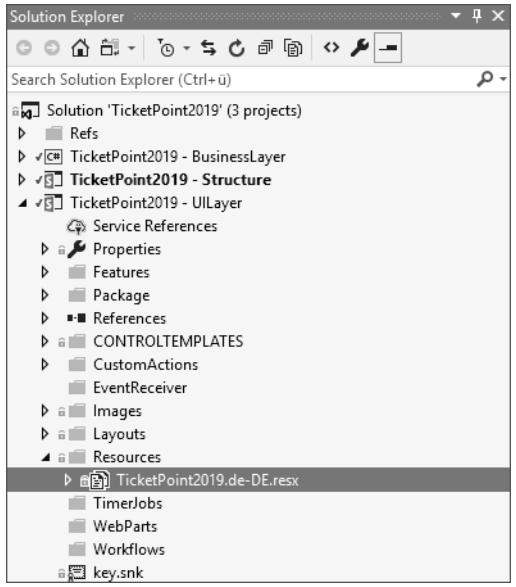

Abbildung 4.10 Ressourcendatei in der Projektstruktur

Wenn Sie die Datei Ihrem Projekt hinzugefügt haben, öffnen Sie mit der rechten Maustaste das Kontextmenü für das *BusinessLayer*-Projekt und klicken dort erst auf ADD und dann auf CLASS... Geben Sie der neuen Klasse den Namen »Localization«.

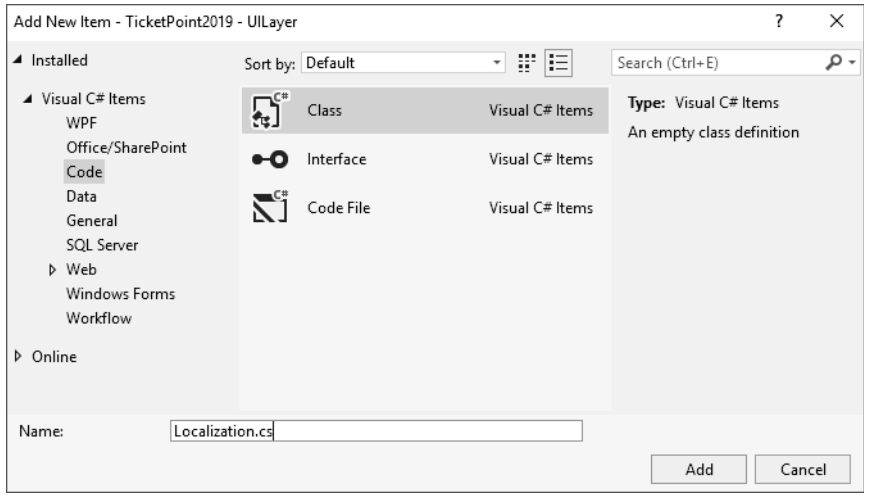

Abbildung 4.11 Add New Item – Localization

Als Erstes fügen Sie der `Localization`-Klasse eine Methode hinzu, die in unserem Wörterbuch nach dem passenden Eintrag sucht und ihn als String zurückgibt.

```
public static string GetString(string key)
{
    try
    {
        string resKey = string.Format("$Resources:TicketPoint2019,{0}", key);
        return SPUtility.GetLocalizedString(resKey
            , "TicketPoint2019"
            , (uint)Thread.CurrentThread.CurrentUICulture.LCID);
    }
    catch (Exception ex)
    {
        Logging.LogError(ex, Constants.LogCategory.BusinessLogic);
        return "ERROR";
    }
}
```

Listing 4.9 Mehrsprachigkeit

Die Methode erstellt den Identifikationsschlüssel unseres gesuchten Begriffs im Wörterbuch, gibt das gewünschte Wörterbuch an und analysiert anhand des aktuell laufenden Threads die benötigte Sprache, in der das Wörterbuch geöffnet werden soll.

Um den Zugriff auf das Wörterbuch im Laufe der Entwicklung zu erleichtern, ist es möglich, die Schlüssel, nach denen im Wörterbuch gesucht werden kann, als Konstanten bereitzustellen. Um die Konstanten unseres Wörterbuchs sauber zu kapseln, erstellen wir die Klasse `Keys` in der `Localization`-Klasse und fügen dort im Laufe der Entstehung unserer Anwendung alle Schlüsselwerte hinzu, die im Wörterbuch angelegt werden.

```
public static class Keys
{
    public const string ApplicationTitle = "ApplicationTitle";
}
```

Um die Schlüssel des Wörterbuchs möglichst übersichtlich zu halten, ist es sinnvoll, einen Sprachmix zu vermeiden und die Schlüssel thematisch zu sortieren. Wir benennen unsere Schlüssel grundsätzlich in englischer Sprache. Die thematische Sortierung können Sie beispielsweise über ein Präfix vor den Schlüsseln realisieren. Mögliche Klassifizierungen sind z. B. Beschriftungen, Schaltflächenbeschriftungen, Meldungen oder Meldungsüberschriften.

Die Umsetzung dieser Regel sieht in TicketPoint 2019 wie in Tabelle 4.1 aus.

Präfix	Beispiel	Beschreibung
`Fld`	`FldTicketnumber`	Spaltennamen
`Ct`	`CtTicket`	Namen von Inhaltstypen
`Lst`	`LstTickets`	Listennamen
`View`	`ViewAllTickets`	Namen von Ansichten
`Txt`	`TxtTicketnumber`	Beschriftungen wie Label etc.
`Val`	`ValPrioHigh`	Wertemengen in Auswahllisten
`Btn`	`BtnSave`	Beschriftung von Schaltflächen
`PageTitle`	`PageTitleAddTicket`	Seitentitel für ApplicationPages
`MsgHeader`	`MsgHeaderSave`	Meldungsüberschriften
`MsgSuccess`	`MsgSuccessSaveTicket`	Erfolgsmeldungen für Aktionen
`MsgError`	`MsgErrorSaveTicket`	Meldungstext im Fehlerfall
`MsgInfo`	`MsgInfoFormsValidation`	Texte für Hinweismeldungen
`MsgReq`	`MsgReqAssignedTo`	Meldungstexte für Feldvalidierungen

Tabelle 4.1 Regeln für die Benennung

Die einzige von diesem Schema abweichende Ausnahme ist der `ApplicationTitle`.

Deployen Sie die Lösung auf Ihren SharePoint und prüfen Sie so, ob die Bereitstellung fehlerfrei funktioniert.

4.4 JavaScript global einbinden

Für die JavaScript-Entwicklung haben wir im Abschnitt für Page-Header die Datei *JSLib.ascx* hinzugefügt. Diese Datei sollten Sie nutzen, um wiederkehrende JavaScript-Funktionen global in der Anwendung bereitzustellen. Um die Arbeit im JavaScript-Umfeld zu erleichtern, bietet es sich an, auf Bibliotheken wie jQuery, AngularJS oder ähnliche zurückzugreifen. Im Fall von TicketPoint 2019 wird jQuery verwendet und über die JSLib auf allen Seiten zur Verfügung gestellt. Erstellen Sie dazu im *UILayer*-Projekt unterhalb von LAYOUTS • TICKETPOINT2019 ein Verzeichnis mit dem Namen *JS*. In diesem Verzeichnis werden zukünftig alle eigenständigen JavaScript-Dateien abgelegt. Laden Sie in das Verzeichnis die aktuellste Version der jQuery-Bibliothek hoch.

jQuery-Download

Die aktuellste jQuery-Version können Sie im Downloadbereich der Herstellerseite (*https://jquery.com*) kostenfrei herunterladen.

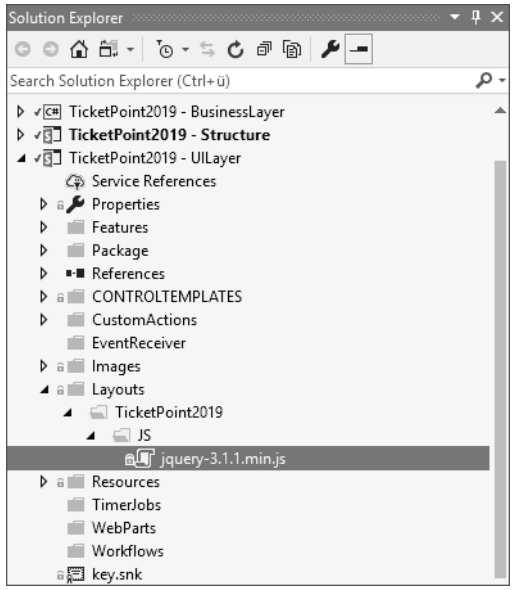

Abbildung 4.12 jQuery einfügen

Öffnen Sie im Anschluss die Datei *JSLib.ascx* und fügen Sie den Codeabschnitt aus Listing 4.10 ein.

```
<script type="text/javascript"
        src="<%=WebUrl %>/_layouts/15/TicketPoint2019/JS/jquery-3.1.1.min.js">
</script>
<script type="text/javascript">
    var $fkr = $.noConflict(true);
</script>
```

Listing 4.10 jQuery einbinden

In der ersten Zeile wird die jQuery-Datei geladen. Mit dem Aufruf im weiteren Skriptblock laden Sie die jQuery-Bibliothek in die Variable `$fkr`. Durch diesen Schritt verhindern Sie Konflikte mit anderen Drittherstellern, sofern diese auf derselben Seite Komponenten nutzen, die mit einer abweichenden jQuery-Version arbeiten.

Um die Einbindung der jQuery-Bibliothek für alle Server dynamisch zu gestalten, legen Sie im *Codebehind* der *JSLib.ascx* eine Property vom Typ String mit dem Namen `WebUrl` an. Diese Property gibt die URL des aktuellen Webs zurück.

```
public string WebUrl
{
    get { return SPContext.Current.Web.Url; }
}
```

Nach diesen Vorbereitungen können Sie Funktionen für die allgemeine Handhabung von SharePoint erstellen. Wiederkehrende Bereiche sind zum einen das Anzeigen von Meldungen im UI, zum anderen das Arbeiten mit modalen Dialogen. Um diese Bereiche zu vereinfachen, ist es sinnvoll, wenn Sie dafür aufzurufende Funktionen bereitstellen.

Für die Anzeige von Meldungen im UI sollten Sie die Statusmeldungen von SharePoint nutzen. Diese können Sie über Funktionen aus der *SP.js*-Bibliothek ansprechen, sie integrieren sich nahtlos in jede Kundenumgebung. Fügen Sie dazu den Codeausschnitt aus Listing 4.11 in den Skriptblock der Datei *JSLib.ascx* ein.

```
var statusId = '';
function addStatusmessage(msgTitle, msgBody, msgColor, reloadPage) {
    $fkr(document).ready(function () {
        SP.SOD.executeFunc('SP.js', 'SP.UI.Status', function () {
            if (msgBody.length > 0) {
                statusId = SP.UI.Status.addStatus(msgTitle, msgBody, true);
                SP.UI.Status.setStatusPriColor(statusId, msgColor);
                if (reloadPage) {
```

```
                setTimeout(doPostback, 1500);
            }
            else {
                setTimeout(removeStatus, 8000);
            }
        }
    });
    document.getElementById('s4-workspace').scrollTop = 0;
});
}
function removeStatus() {
    SP.UI.Status.removeStatus(statusId);
}
function doPostback() {
    SP.UI.ModalDialog.RefreshPage(SP.UI.DialogResult.OK);
}
```

Listing 4.11 Funktionen für Statusmeldungen

Eine Beispielstatusmeldung sehen Sie in Abbildung 4.13.

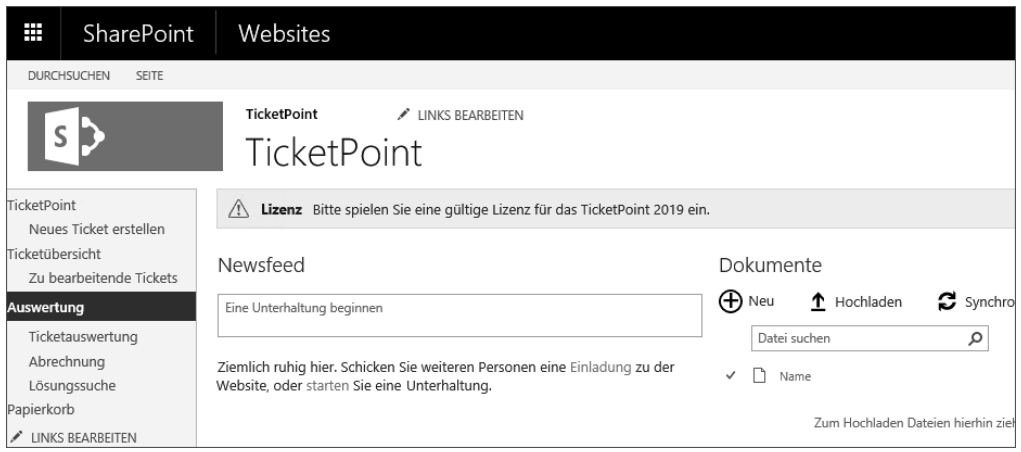

Abbildung 4.13 Beispiel einer Statusmeldung

Der Funktion `addStatusmessage` übergeben wir den Titel der Meldung, den Textinhalt der Meldung sowie die Farbe, in der die Meldung angezeigt werden soll. Der letzte Parameter gibt an, ob die Seite nach dem Anzeigen der Meldung neu geladen werden soll, um Inhalte zu aktualisieren.

Wenn Inhalte aktualisiert werden sollen, wird nach 1,5 Sekunden die Funktion `doPostback` ausgeführt. Diese aktualisiert den Inhalt der aktuellen Seite. Das gibt dem Benutzer ausreichend Zeit, die Meldung wahrzunehmen, und sorgt für eine zeitnahe

Aktualisierung der angezeigten Informationen. Wünscht der Benutzer keine Aktualisierung, wird nach 8 Sekunden die Funktion `removeStatus` aufgerufen. Diese sorgt dafür, dass die aktuell angezeigte Meldung wieder von der Seite entfernt wird, ohne diese neu zu laden.

Das Hinzufügen des Status wird per jQuery erst dann ausgeführt, wenn die aktuelle Seite vollständig geladen ist. Zusätzlich wird über den Aufruf der Funktion `SP.SOD.executeFunc` gewährleistet, dass die notwendigen Bereiche der Datei *SP.js* geladen wurden, bevor zum Anzeigen der Meldung auf diese zugegriffen wird.

Um die Arbeit mit modalen Dialogen zu vereinfachen, sollten Sie zwei Funktionen bereitstellen: eine Funktion zum Aufrufen eines Dialogs ohne Rückgabewerte und eine zweite Funktion, die nach dem Schließen des Dialogs die Rückgabewerte an eine Callback-Funktion übergibt.

```
function openModal(dialogTitle, dialogUrl) {
    $fkr(document).ready(function () {
        SP.SOD.executeFunc('SP.js', 'SP.UI.ModalDialog', function () {
            var options = {
                title: dialogTitle,
                url: dialogUrl,
                allowMaximize: true,
                showClose: true
            };
            SP.UI.ModalDialog.showModalDialog(options);
        });
    });
}
function openModalWithCallback(dialogTitle, dialogUrl, callbackFunction) {
    $fkr(document).ready(function () {
        SP.SOD.executeFunc('SP.js', 'SP.UI.ModalDialog', function () {
            var options = {
                title: dialogTitle,
                url: dialogUrl,
                allowMaximize: true,
                showClose: true,
                dialogReturnValueCallback: callbackFunction
            };
            SP.UI.ModalDialog.showModalDialog(options);
        });
    });
}
```

Listing 4.12 Funktionen für modale Dialoge

Verfügbare Optionen für den Aufruf modaler Dialoge sehen Sie in Tabelle 4.2.

Option	Beschreibung
`title`	Der Titel des Dialogfensters.
`url`	Die URL der Seite, die im Dialogfenster angezeigt werden soll.
`html`	Das HTML, das im Dialogfenster zur Anzeige gebracht werden soll. Wenn URL und HTML angegeben wurden, wird die URL bevorzugt verwendet.
`x`	Der Abstand der Dialogbox vom linken Seitenrand.
`y`	Der Abstand der Dialogbox vom oberen Seitenrand.
`width`	Definiert die Höhe des Dialogfensters.
`height`	Definiert die Breite des Dialogfensters.
`allowMaximize`	Gibt an, ob die Schaltfläche zum Maximieren des Dialogfensters angezeigt werden soll.
`showMaximized`	Gibt an, ob das Dialogfenster maximiert gestartet werden soll.
`showClose`	Gibt an, ob die Schließen-Schaltfläche für den modalen Dialog angezeigt werden soll.
`autoSize`	Gibt an, ob die Größe des modalen Dialogs automatisch anhand seines Inhalts ermittelt und gesetzt werden soll.
`dialogReturn-ValueCallback`	Ein Pointer auf eine Callback-Funktion, die nach dem Schließen des Dialogs aufgerufen werden soll.
`args`	Ein Objekt, das die Daten enthält, die an den Dialog übergeben werden.

Tabelle 4.2 Optionen für Dialoge

Genau wie beim Aufrufen der Meldungen wird in den Funktionen für die modalen Dialoge per jQuery dafür gesorgt, dass die Seite vollständig geladen ist, bevor der Code ausgeführt wird. Ebenso wird durch den Aufruf der Funktion `SP.SOD.executeFunc` gewährleistet, dass der notwendige Bereich zum Öffnen modaler Dialoge aus der Bibliothek *SP.js* geladen wurde.

Deployen Sie die Lösung auf Ihren SharePoint, und prüfen Sie so, ob die Bereitstellung fehlerfrei funktioniert.

4.5 JS from Codebehind

In vielen Situationen ist es erforderlich, die zuvor erstellten JavaScript-Funktionen aus dem Codebehind heraus aufzurufen. Sie sollten Methoden bereitstellen, die diesen Aufruf erleichtern.

Öffnen Sie dazu das *BusinessLayer*-Projekt und erzeugen Sie ein neues Verzeichnis mit dem Namen *UI*.

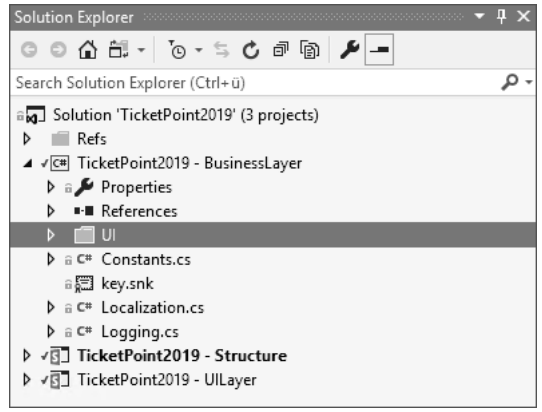

Abbildung 4.14 UI-Verzeichnis erstellen

Danach fügen Sie eine neue Klasse mit dem Namen `Helper.cs` dem Verzeichnis hinzu.

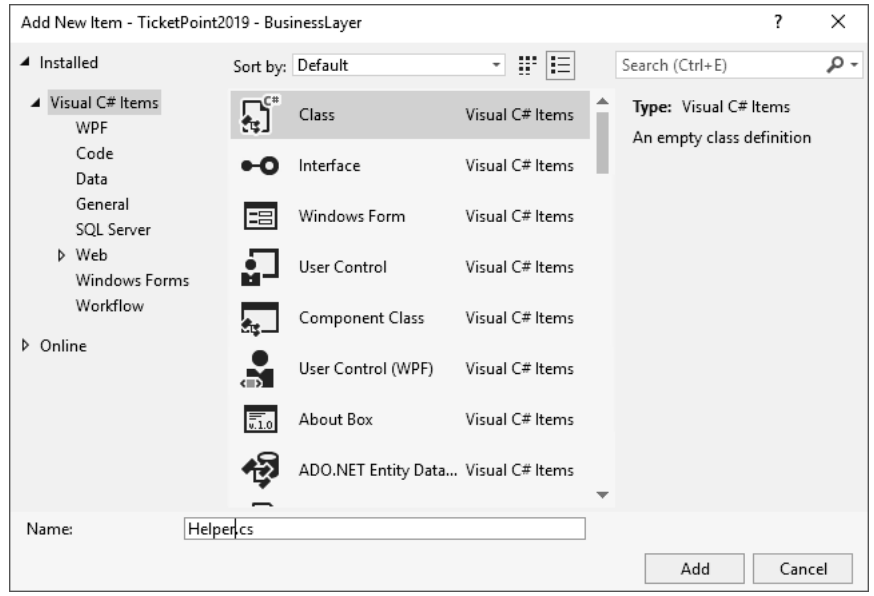

Abbildung 4.15 »UI.Helper.cs« hinzufügen

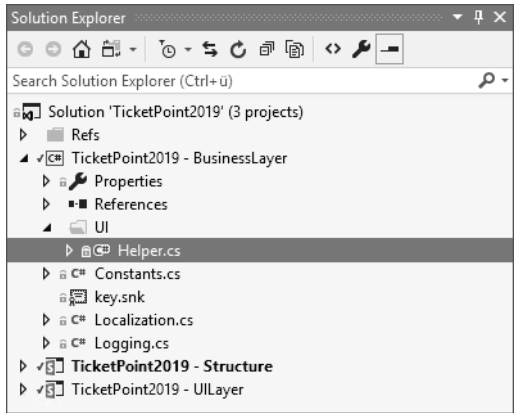

Abbildung 4.16 »UI.Helper.cs« in der Projektstruktur

Fügen Sie nun die notwendigen Methoden zum Aufrufen der JavaScript-Funktionen für Statusmeldungen und zum Öffnen modaler Dialoge ein.

Für den Aufruf von Statusmeldungen fügen Sie die Methode aus Listing 4.13 der Klasse `UI.Helper` hinzu.

```
public static void AddStatusmessageToPage(Page page
    , string messageTitle
    , string messageBody
    , string messageStatus
    , bool refreshPage = false)
{
    string js = @"addStatusmessage(""" + messageTitle + @"""
                    , """ + messageBody + @"""
                    , """ + messageStatus + @"""
                    , " + refreshPage.ToString().ToLower() + ");";
    page.ClientScript.RegisterClientScriptBlock(page.GetType()
        , "msg"
        , js
        , true);
}
```

Listing 4.13 Methode für Statusmeldung

Der Methode übergeben wir das Objekt der aktuellen Seite, den Nachrichtentitel, den Nachrichtentext sowie den Status der Meldung, außerdem die Information, ob die Seite nach Anzeige der Nachricht aktualisiert werden soll. Aus den übergebenen Parametern wird der Funktionsaufruf für das JavaScript als String aufbereitet. Dieser Script-String wird als ClientScriptBlock auf die Seite geschrieben. Durch den registrierten Script-Block wird die zuvor erstellte JavaScript-Funktion aus der *JSLib.ascx* aufgerufen.

Als Nächstes erstellen Sie die Methoden zum Verwalten modaler Dialoge. Fügen Sie dazu die Methoden aus Listing 4.14 in die Klasse `UI.Helper` ein.

```
public static void OpenModalDialog(Page page
    , string dialogTitle
    , string dialogUrl)
{
    string js = @"openModal(""" + dialogTitle + @"""
                    , """ + dialogUrl + @""");";
    page.ClientScript.RegisterClientScriptBlock(page.GetType()
        , "openDlg"
        , js
        , true);
}
public static void OpenModalDialog(Page page
    , string dialogTitle
    , string dialogUrl
    , string callbackFunction)
{
    string js = @"openModalWithCallback(""" + dialogTitle + @"""
                    , """ + dialogUrl + @"""
                    , " + callbackFunction + ");";
    page.ClientScript.RegisterClientScriptBlock(page.GetType()
        , "openDlg"
        , js
        , true);
}
```

Listing 4.14 Methode für modale Dialoge

Der Methode `OpenModalDialog` übergeben wir die aktuelle Seite, den Titel des Dialogs und die zu öffnende URL. Aus den Übergabeparametern wird der Aufruf der JavaScript-Funktion zum Öffnen eines modalen Dialogs als String erstellt. Dieser wird als Skript-Block auf der aktuellen Seite registriert, um die Funktion aus der *JSLib.ascx* aufzurufen. Als Überladung dieser Methode steht eine Variante bereit, der Sie den Namen einer Callback-Funktion im JavaScript übergeben können. Diese JavaScript-Funktion wird aufgerufen, wenn der modale Dialog geschlossen wird, und verarbeitet die zurückgegebenen Informationen.

Selbstverständlich müssen Sie einen modalen Dialog auch einmal schließen. Das Schließen des Dialogs sollten Sie ebenfalls aus dem Codebehind heraus triggern können. Um auch dies zu realisieren, fügen Sie die Methoden aus Listing 4.15 in der Klasse `UI.Helper` hinzu.

```
public static void CloseOrRedirect(Page page, bool commit = false)
{
    string sourceUrl = page.Request.QueryString["Source"];
```

```
    string redirectUrl = (!string.IsNullOrWhiteSpace(sourceUrl))
        ? sourceUrl
        : (SPContext.Current.List != null)
            ? SPContext.Current.List.DefaultViewUrl
            : SPContext.Current.Web.Url;
    CloseOrRedirect(page, redirectUrl, commit);
}
public static void CloseOrRedirect(Page page
    , string redirectUrl
    , bool commit = false)
{
    string js = "";
    if (SPContext.Current.IsPopUI)
    {
        if (commit)
            js = @"setTimeout(function() {
                        window.frameElement.commitPopup();
                    }, 0);";
        else
            js = @"setTimeout(function() {
                        window.frameElement.cancelPopUp();
                    }, 0);";
    }
    else
    {
        js = "STSNavigate('" + redirectUrl + "');";
    }
    page.ClientScript.RegisterClientScriptBlock(page.GetType()
        , "close"
        , js
        , true);
}
```

Listing 4.15 Methode zum Schließen von Formularen

Die erste Überladung der Methode `CloseOrRedirect` nimmt als Parameter die aktuelle Seite an. Als zweiten, optionalen Parameter übergeben wir eine `commit`-Information. Dieser Parameter gibt an, ob der Dialog mit einer Erfolgsmeldung oder einem normalen Abschluss beendet werden soll. In der Methode wird nach der URL gesucht, zu der navigiert werden soll. Als Erstes wird versucht, den `Source`-Parameter der aktuellen Seite zu ermitteln. Sollte dieser nicht verfügbar sein, wird die URL der aktuellen Liste gesucht. Kann auch sie nicht gefunden werden, wird auf die URL des aktuellen Webs zurückgegriffen. Nachdem die entsprechende Redirect-URL ermittelt wurde,

wird die zweite Überladung der Methode aufgerufen. Diese prüft über die Property `IsPopUI` des aktuellen SharePoint-Kontexts, ob die aktuelle Seite als modaler Dialog geöffnet ist. Wenn die Seite als modaler Dialog geöffnet wurde, wird das JavaScript zum Schließen des Dialogs als String erzeugt. Hierbei wird anhand des Parameters `commit` entschieden, ob der Dialog mit oder ohne Erfolgsmeldung geschlossen wird. Sollte es sich nicht um einen modalen Dialog handeln, wird ein Skript zum Redirect auf die übergebene URL als String erstellt.

Zum Abschluss wird das erzeugte Skript als Skript-Block auf die aktuelle Seite geschrieben.

Deployen Sie die Lösung auf Ihren SharePoint und prüfen Sie so, ob die Bereitstellung fehlerfrei funktioniert.

4.6 Projekttemplate erstellen

Wenn Sie alle Vorbereitungen für die unterschiedlichen Projekttypen *BusinessLayer*, *Structure* und *UILayer* abgeschlossen haben, sollten Sie für jedes der Projekte ein Visual-Studio-Template erstellen. Durch die Erstellung von Projekttemplates können Sie jederzeit mit wenig Aufwand weitere Projekte aufbauen, bei denen Sie sich die bisher investierte Vorbereitungszeit sparen.

Für die Erstellung eines Projekttemplates klicken Sie in Visual Studio unter File auf Export Template...

Wählen Sie im ersten Dialog des Wizards das gewünschte Projekt aus. In diesem Beispiel arbeiten wir mit dem *Structure*-Projekt.

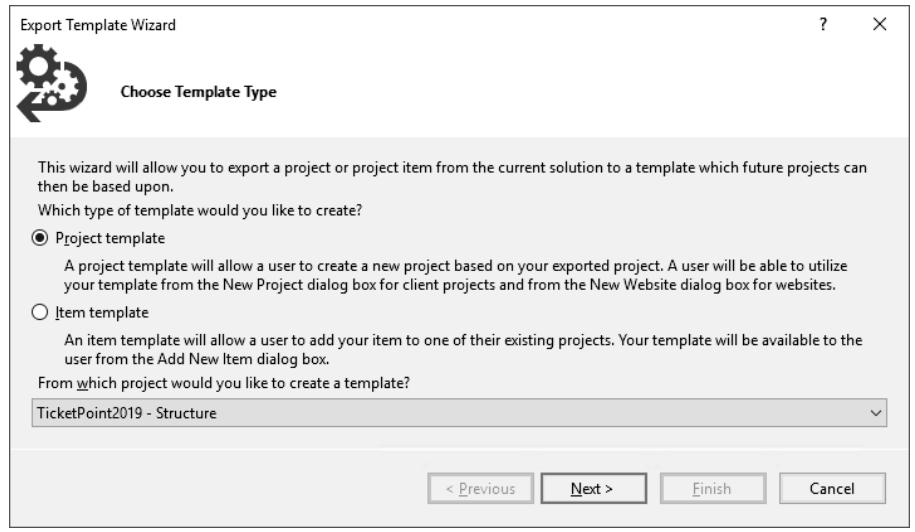

Abbildung 4.17 Wizard – Projektauswahl

Im zweiten Abschnitt des Wizards geben Sie die Informationen zum Template an und wählen das Icon und ein Vorschaubild aus.

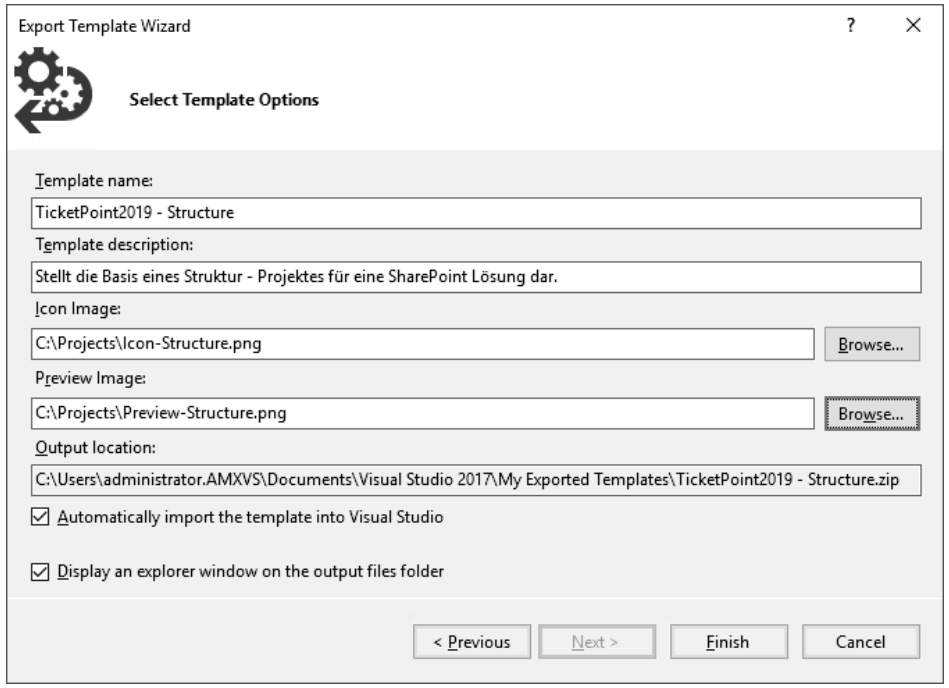

Abbildung 4.18 Wizard – Optionen

Für die unterschiedlichen Templates sollten Sie die Angaben aus Tabelle 4.3 übernehmen.

Projekt	Option	Wert
Structure	TEMPLATE NAME	SharePoint 2019 – Structure
Structure	TEMPLATE DESCRIPTION	erstellt die Basis eines Struktur-Projekts für eine SharePoint-Lösung
UILayer	TEMPLATE NAME	SharePoint 2019 – UILayer
UILayer	TEMPLATE DESCRIPTION	erstellt die Basis eines UILayer-Projekts für eine SharePoint-Lösung
BusinessLayer	TEMPLATE NAME	SharePoint 2019 – BusinessLayer
BusinessLayer	TEMPLATE DESCRIPTION	erstellt die Basis des BusinessLayer-Projekts für eine SharePoint-Lösung

Tabelle 4.3 Templatedefinition

Wählen Sie für jedes Template ein passendes Icon und ein treffendes Vorschaubild.

Wenn Sie die Angaben in diesem Schritt des Wizards getätigt haben, klicken Sie auf FINISH, um den Export abzuschließen.

Führen Sie den Export für alle drei Projekte Ihrer Lösung durch. Nach Abschluss des kompletten Exports stehen Ihnen die bisherigen Vorbereitungen als Templates für neue Projekte in Visual Studio zur Verfügung.

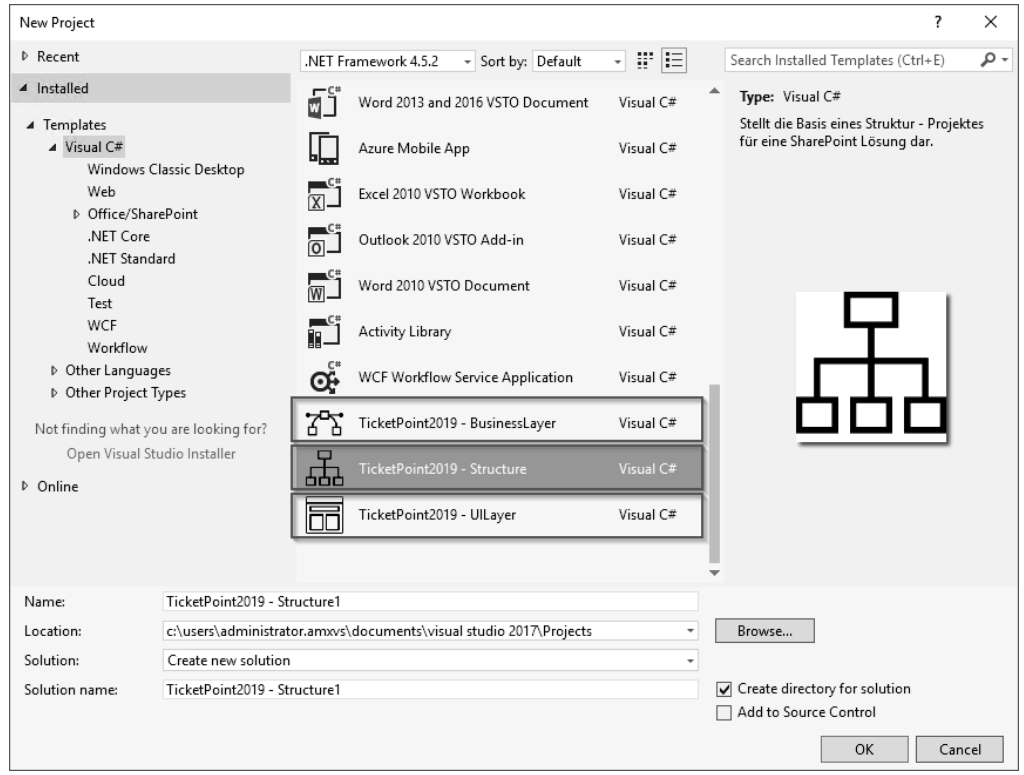

Abbildung 4.19 Projekttemplates

Kapitel 11
Umsetzung EventReceiver

Erfahren Sie, welche Möglichkeiten Sie haben, auf unterschiedliche Events in SharePoint zu reagieren und sich diese in Ihrer Anwendung zunutze zu machen.

In diesem Kapitel setzen wir die EventReceiver von TicketPoint 2019 um. Zunächst ein paar allgemeine Hinweise zur Entwicklung von EventReceivern.

Wie Sie bereits im Planungsteil erfahren haben, gibt es synchrone EventReceiver, die ausgelöst werden, bevor die Aktion, z. B. das Ändern eines ListItems, vollständig durchgeführt wurde, und asynchrone EventReceiver, die ausgelöst werden, nachdem die Aktion vollständig ausgeführt wurde, also z. B. nachdem die Daten, die am ListItem geändert wurden, in der Inhaltsdatenbank gespeichert wurden. Der Zugriff auf die Werte des ListItems unterscheidet sich dabei. Bei den asynchronen Events können Sie auf das `SPListItem`-Objekt über die `ListItem`-Property des `SPItemEventProperties`-Objekts zugreifen. Mit diesem Objekt können Sie verfahren wie mit jedem anderen `SPListItem`-Objekt, das Sie z. B. direkt aus einer Liste ausgelesen haben. Wenn Sie sich jedoch in einem synchronen Event befinden, können Sie dieses `ListItem`-Objekt nicht verwenden, um die aktuellen Werte auszulesen. Im `ItemAdding`-Event ist die `ListItem`-Property *null*. Im `ItemUpdating`-Event können Sie diese Property dazu verwenden, auf die Werte zuzugreifen, die vor dem Speichern in dem Item hinterlegt waren. Wenn Sie die aktuellen Werte bzw. die Werte, die gespeichert werden sollen, benötigen, müssen Sie über die `AfterProperties`-Property des `SPItemEventProperties`-Objekts darauf zugreifen. Dabei müssen Sie jedoch beachten, dass sich die Werte aus den Properties des `SPListItem` von den Werten aus den `AfterProperties` unterscheiden. Ein Beispiel, wie Sie dabei mit einer Lookup-Spalte verfahren können, finden Sie in diesem Kapitel, wenn wir die Methode `StatusChanged` beschreiben.

Änderungen am ListItem können Sie im synchronen EventReceiver direkt in den `AfterProperties` vornehmen. Hierzu ist kein Update des Objekts nötig. Wenn Sie Änderungen im asynchronen Event am ListItem vornehmen wollen, müssen Sie dies über das `SPListItem`-Objekt durchführen. Hier ist dann ein Update notwendig, damit die Änderungen übernommen werden.

Beachten Sie dabei, dass Sie durch das Update wiederum die EventReceiver triggern, die dann wieder das Update auslösen etc. Um dies zu verhindern, gibt es im Kontext eines EventReceivers die Property `EventFiringEnabled`, mit der Sie durch das Setzen auf `false` die Events abschalten, bis Sie durch das Setzen der Property auf `true` diese wieder aktivieren.

Beispiel:

```
this.EventFiringEnabled = false;
//Code, der ein Update auslöst
this.EventFiringEnabled = true;
```

Hinweis

Setzen Sie diese Möglichkeit sparsam ein und prüfen Sie genau, ob dadurch nicht unerwünschte Nebeneffekte auftreten. Durch das Abschalten der Events werden alle Events, die im aktuellen Thread ausgeführt werden, für die Laufzeit des Threads deaktiviert. Das heißt, dass auch Standard-SharePoint-EventReceiver nicht ausgeführt werden.

Diese Probleme werden meist erst im späteren Projektverlauf erkannt. Wenn Sie z. B. eine Taxonomy-Spalte verwenden, werden durch die Standard-SharePoint-EventReceiver versteckte Spaltenwerte gesetzt, die für die Indexierung der Inhalte durch die SharePoint-Suche zwingend erforderlich sind. Das Nichtsetzen dieser Spalte hat zur Folge, dass die ListItems nicht vollständig indiziert werden.

Der Zugriff auf Objekte aus dem SharePoint-Kontext ist in einem EventReceiver nicht möglich, so wird z. B. durch `SPContext.Current.Web` eine NullReferenceException ausgelöst. Um Zugriff auf die Objekte zu erhalten, verwenden Sie die `SPItemEventProperties`. Dieses Objekt enthält z. B. auch das aktuelle `SPWeb`-Objekt.

Bevor wir mit der eigentlichen Programmierung des EventReceivers beginnen, noch ein Hinweis zum Debuggen von EventReceivern: Genau wie beim Debuggen von Formularen hängen Sie sich an den w3wp-Prozess.

11.1 Benutzerbenachrichtigungen

Kommen wir nun zum EventReceiver, der zum Versenden der Benutzerbenachrichtigungen verwendet wird. Zuerst erstellen Sie den EventReceiver im *UILayer*-Projekt im Ordner *EventReceiver*.

Klicken Sie dazu im Kontextmenü auf Add New Item und wählen Sie im Anschluss unter Office/SharePoint EventReceiver aus. Geben Sie als Namen »TicketEventReceiver« ein.

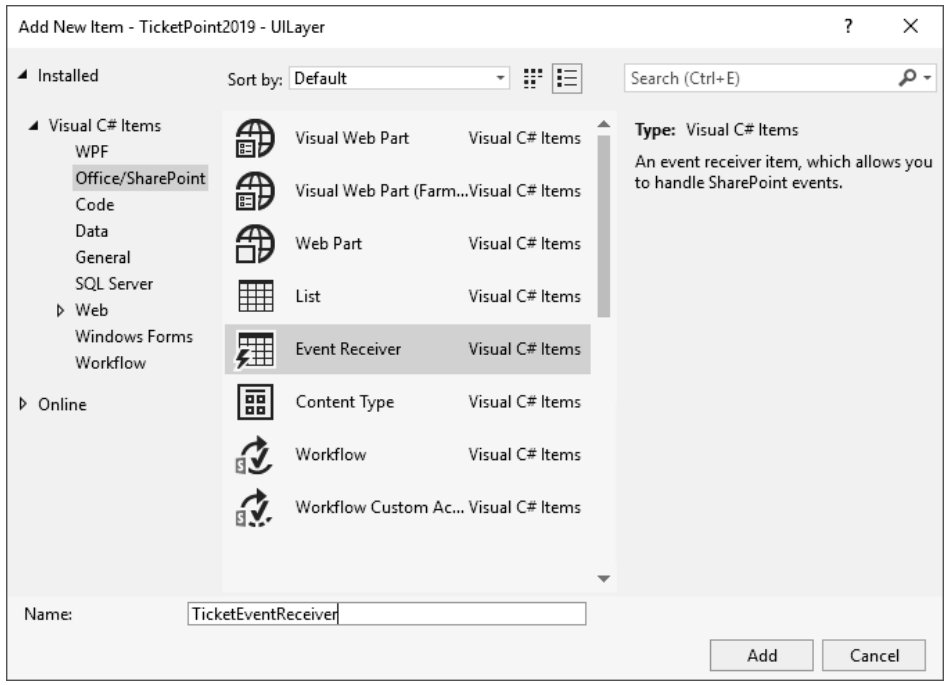

Abbildung 11.1 »Add New Item«-EventReceiver

Im nächsten Schritt des Wizards wählen Sie List Item Events als Typ aus, da Sie einen EventReceiver für die Ticketliste erstellen wollen. Als Eventquelle wählen Sie Custom List aus. Das hat zur Folge, dass der EventReceiver mit dem automatisch generierten XML an jede benutzerdefinierte Liste gehängt wird. Das passen wir später an, um die Events auf die Ticketliste zu beschränken.

Als zu behandelnde Events wählen Sie An item is being updated und An item was added aus. Wir verwenden hier das `ItemAdded`-Event, da im `ItemAdding`-Event noch kein ListItem erstellt wurde und somit die `ListItemId` nicht feststeht. Diese benötigen wir jedoch für den Platzhalter `TicketLink`.

Das `ItemUpdating`-Event verwenden wir, um die Möglichkeit zu haben, über die `AfterProperties` zu prüfen, ob sich der Ticketstatus geändert hat. Das ist erforderlich, da wir nur bei einer Änderung des Ticketstatus eine E-Mail versenden wollen.

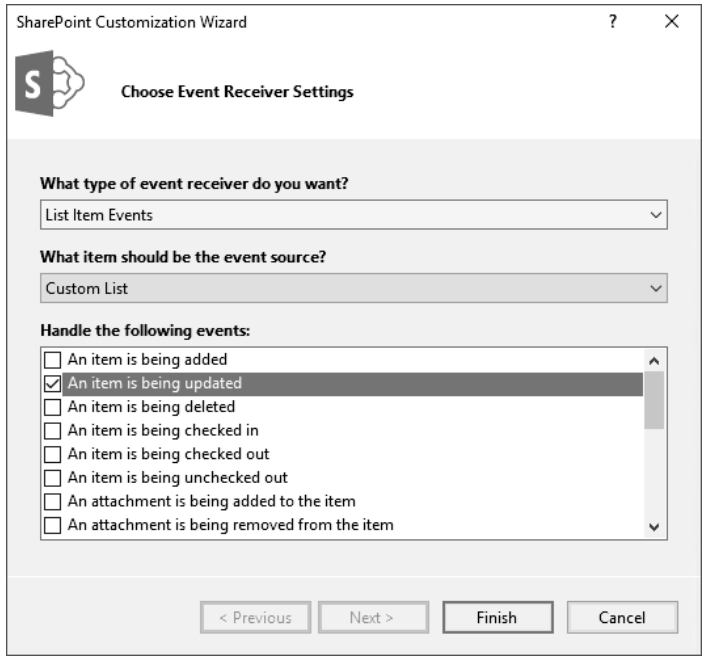

Abbildung 11.2 »ItemUpdating«-Event auswählen

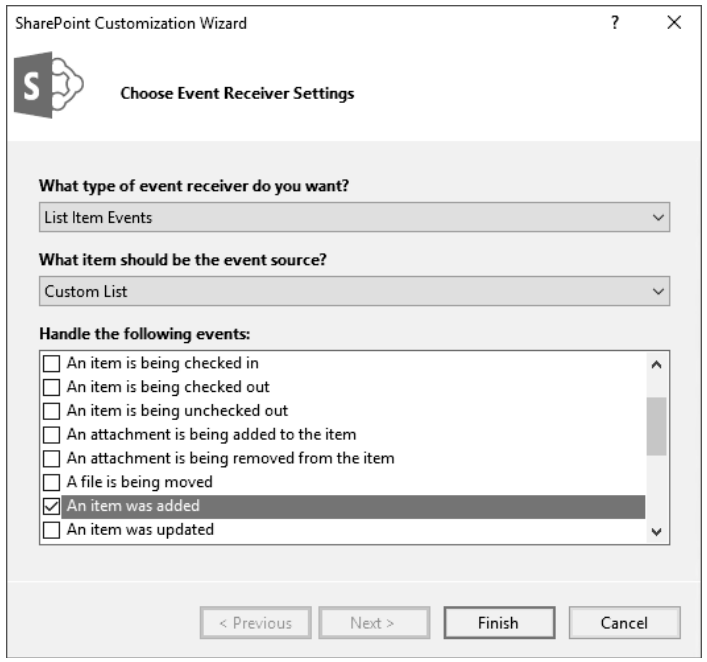

Abbildung 11.3 »ItemAdded«-Event auswählen

Nach dem Klick auf den FINISH-Button wird dann eine Klasse mit zwei Methoden erstellt, die auf die beiden ausgewählten Events reagiert. Zusätzlich wird automatisch eine XML-Datei generiert, die die Events automatisch an die im Wizard ausgewählte Eventquelle hängt.

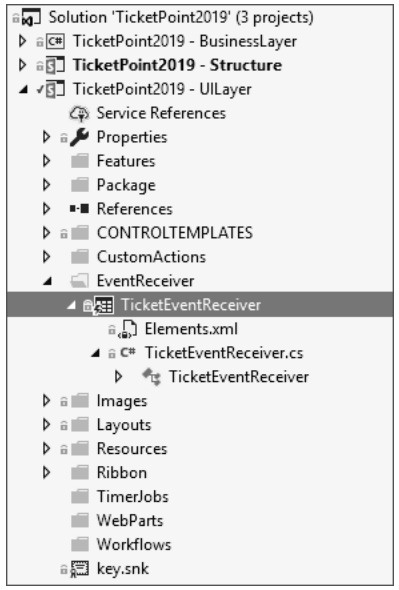

Abbildung 11.4 »TicketEventReceiver« in der Projektstruktur

Das XML zum Hinzufügen der EventReceiver an die Quellen sehen Sie in Listing 11.1:

```
<Receivers ListUrl="Lists/tickets">
    <Receiver>
        <Name>TicketEventReceiverItemUpdating</Name>
        <Type>ItemUpdating</Type>
         <Assembly>$SharePoint.Project.AssemblyFullName$</Assembly>
            <Class>
            Fkr.SharePoint.TicketPoint2019.UILayer.TicketEventReceiver
        </Class>
        <SequenceNumber>10000</SequenceNumber>
    </Receiver>
        <Receiver>
        <Name>TicketEventReceiverItemAdded</Name>
            <Type>ItemAdded</Type>
        <Assembly>$SharePoint.Project.AssemblyFullName$</Assembly>
        <Class>
            Fkr.SharePoint.TicketPoint2019.UILayer.TicketEventReceiver
        </Class>
```

```
    <SequenceNumber>10000</SequenceNumber>
    </Receiver>
</Receivers>
```

Listing 11.1 EventReceiver anhängen

Schauen wir uns den XML-Code einmal genauer an:

```
<Receivers ListUrl="Lists/tickets">
```

Dies ist die einzige Stelle, an der Sie eine Anpassung durchführen müssen. Im automatisch generierten XML stand hier vorher `ListTemplateId="100"`, was zur Folge hat, dass der EventReceiver an alle Listen vom Typ *Benutzerdefinierte Liste* gehängt wird. Da Sie den EventReceiver ausschließlich an der Ticketliste benötigen, schränken Sie die Listen über das `ListUrl`-Attribut auf die Ticketliste ein.

```
<Name>TicketEventReceiverItemUpdating</Name>
```

Sie können einen beliebigen Namen eintragen. Dieser Name ist später in der Oberfläche nicht sichtbar. Sie sollten aber grundsätzlich einen beschreibenden Namen verwenden, um bei mehreren EventReceivern den Überblick zu behalten.

```
<Type>ItemUpdating</Type>
```

Bei `Type` tragen Sie den Typ des Events ein. Der Typname ist identisch mit dem Methodennamen in der Eventklasse.

```
<Assembly>$SharePoint.Project.AssemblyFullName$</Assembly>
<Class>
    Fkr.SharePoint.TicketPoint2019.UILayer.TicketEventReceiver
</Class>
```

Über `Assembly` und `Class` definieren Sie die Assembly und die Klasse, in der sich die Methoden für die Eventbehandlung befinden.

```
<SequenceNumber>10000</SequenceNumber>
```

Mit der `SequenceNumber` können Sie die Eventreihenfolge, wenn mehrere Events an einer Liste hängen, beeinflussen. Je größer die `SequenceNumber`, desto später wird das Event ausgeführt.

Nun fügen Sie den EventReceiver zum Feature im *UILayer* hinzu. Wenn Sie diesen Schritt auslassen, werden die EventReceiver nicht bei der Ticketliste registriert und somit nicht ausgeführt.

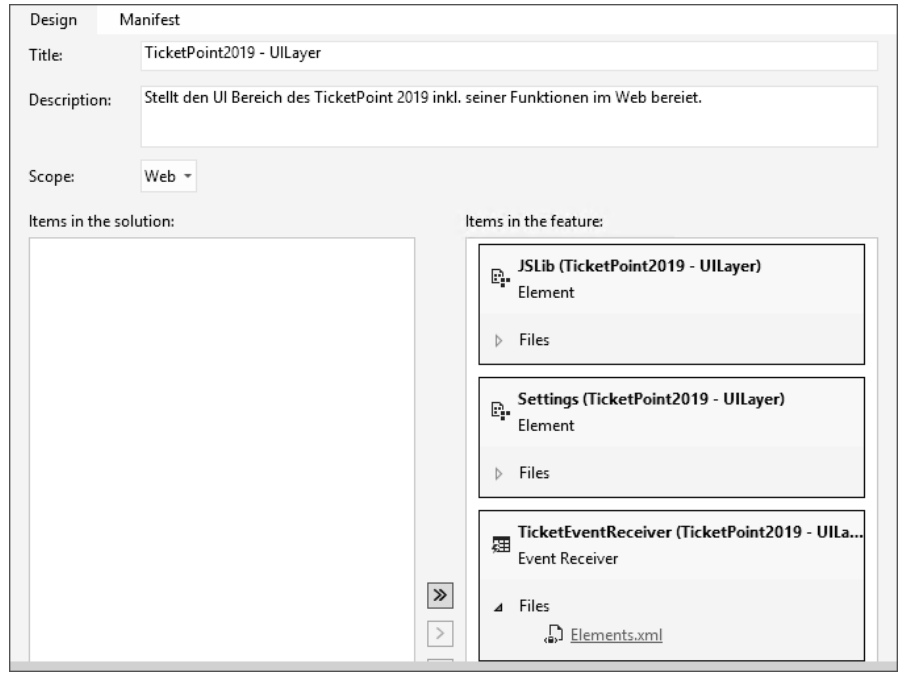

Abbildung 11.5 Feature mit »TicketEventReceiver«

Um zu prüfen, ob die EventReceiver korrekt hinzugefügt wurden, können Sie z. B. den SharePoint Manager verwenden. Navigieren Sie dazu im SharePoint Manager zur Ticketliste und öffnen Sie den Punkt EventReceivers.

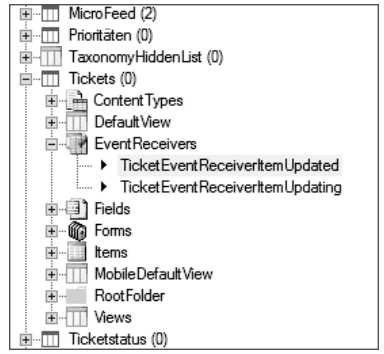

Abbildung 11.6 SharePoint Manager – Events der Ticketliste

Wie im Planungsteil dieses Buchs bereits angekündigt, schauen wir uns die Möglichkeit, EventReceiver über Code anzuhängen, ebenfalls an. Öffnen Sie dazu die `List.cs`-Klassendatei im *Structure*-Projekt und ergänzen Sie die Methode aus Listing 11.2:

```
public static void EnsureEventReceiver(
    SPList list,
    IEnumerable<SPEventReceiverType> eventReceiverTypes,
    Type type,
    SPEventReceiverSynchronization synchronization,
    int sequenceNumber,
    string className)
{
    foreach (SPEventReceiverType eventReceiverType in eventReceiverTypes)
    {
        string name = string.Concat(className, eventReceiverType.ToString());
         if (list.EventReceivers.Cast<SPEventReceiverDefinition>().All(
            i => i.Name != name))
        {
            SPEventReceiverDefinition eventReceiverDefinition =
                list.EventReceivers.Add();
                eventReceiverDefinition.Name = name;
                eventReceiverDefinition.Type = eventReceiverType;
                eventReceiverDefinition.Assembly = type.Assembly.FullName;
                eventReceiverDefinition.Class = type.FullName;
                eventReceiverDefinition.SequenceNumber = sequenceNumber;
                eventReceiverDefinition.Synchronization = synchronization;
                eventReceiverDefinition.Update();
        }
    }
}
```

Listing 11.2 Methode zum sicheren Anhängen eines EventReceivers

Die Methode prüft für jeden der übergebenen EventReceiver-Typen, ob er bereits der Liste zugeordnet wurde, und fügt ihn hinzu, wenn er noch nicht existiert. Dieser Schritt verhindert, dass ein EventReceiver mehrfach der Liste hinzugefügt wird. Die einzelnen Properties sind die gleichen wie bei der Definition über XML.

Ersetzen Sie die `ItemAdded`-Methode im EventReceiver durch die Methode, die Sie in Listing 11.3 finden.

```
public override void ItemAdded(SPItemEventProperties properties)
{
    try
    {
         CheckAndSendMail(properties);
    }
    catch (Exception ex)
```



```
    {
        Logging.LogError(ex, Constants.LogCategory.EventReceiver);
    }
    base.ItemAdded(properties);
}
```

Listing 11.3 »ItemAdded-Event«

Im `ItemAdding`-Event wird die Methode `CheckAndSendMail` aufgerufen, in der sich die komplette Logik des EventReceivers befindet. Im Anschluss wird lediglich noch die `base`-Methode aufgerufen.

Ersetzen Sie die `ItemUpdating`-Methode durch die Methode aus Listing 11.4:

```
public override void ItemUpdating(SPItemEventProperties properties)
{
    try
    {
        //Status geändert?
        if (StatusChanged(properties))
        {
             CheckAndSendMail(properties);
        }
    }
    catch (Exception ex)
    {
        Logging.LogError(ex, Constants.LogCategory.EventReceiver);
    }
    base.ItemUpdating(properties);
}
```

Listing 11.4 »ItemUpdating-Event«

Auch hier wird die `CheckAndSendMail`-Methode und im Anschluss die `base`-Methode des Events aufgerufen. Der Unterschied ist an dieser Stelle, dass vorher mit der `StatusChanged`-Methode geprüft wird, ob sich der Ticketstatus geändert hat. Eine Anforderung an das Versenden der Benachrichtigungen ist, dass nur dann eine E-Mail gesendet werden soll, wenn sich der Status geändert hat. Schauen wir uns einmal die `StatusChanged`-Methode genauer an.

```
private bool StatusChanged(SPItemEventProperties properties)
{
    string currentValue = GetValue(properties,
        Constants.FieldInternalName.Ticketstatus) as string;
    string beforeValue = properties.ListItem[
```

```
        Constants.FieldInternalName.Ticketstatus] as string;
    return new SPFieldLookupValue(currentValue).LookupId
        != new SPFieldLookupValue(beforeValue).LookupId;
}
```

Listing 11.5 StatusChanged-Methode

Der aktuelle Wert – genauer genommen der zukünftige, da das Update noch nicht vollständig abgeschlossen ist – wird mithilfe der `GetValue`-Methode aus den `AfterProperties` der `SPItemEventProperties` ausgelesen. Den vorherigen Wert der Ticketstatusspalte erhalten wir durch Zugriff auf die `ListItem`-Property. In ihr befindet sich das `SPListItem`-Objekt mit dem Stand der Werte, bevor der Update-Prozess gestartet wurde. Diese beiden Werte können nicht über String-Operationen verglichen werden, da sich im vorherigen Wert ein `LookupValue`-String befindet, in den `AfterProperties` allerdings nur die ID des ausgewählten Elements.

Beispiel:

```
currentValue = "3"
beforeValue = "3;#abgeschlossen"
```

Durch das Erzeugen von `SPFieldLookupValue`-Objekten können diese beiden Werte einfach verglichen werden.

Die `GetValue`-Methode liefert den Wert der Spalte je nach Eventtyp aus den `AfterProperties` oder aus dem `ListItem`.

```
private object GetValue(SPItemEventProperties prop, string internalName)
{
    switch (prop.EventType)
    {
        case SPEventReceiverType.ItemAdding:
                return prop.AfterProperties[internalName];
        case SPEventReceiverType.ItemAdded:
                return prop.ListItem[internalName];
        case SPEventReceiverType.ItemUpdating:
                return prop.AfterProperties[internalName];
        case SPEventReceiverType.ItemUpdated:
                return prop.ListItem[internalName];
        default:
                throw new Exception("Der Eventtyp wird nicht unterstützt.");
    }
}
```

Listing 11.6 Wertermittlung

Um zu prüfen, ob für den jeweiligen Ticketstatus eine E-Mail gesendet werden soll, ergänzen Sie die Methode aus Listing 11.7. Die Methode prüft für den übergebenen Ticketstatus im `Settings`-Objekt, ob die E-Mail-Benachrichtigung aktiv ist oder nicht.

```
private bool CheckStatus(int statusId, Settings settings)
{
    if (statusId == settings.StatusNew && settings.StatusMailNew)
        return true;
    else if (statusId == settings.StatusAssumed &&
            settings.StatusMailAssumed)
        return true;
    else if (statusId == settings.StatusCanceled &&
            settings.StatusMailCanceled)
        return true;
    else if (statusId == settings.StatusFinished &&
            settings.StatusMailFinished)
        return true;
    else
        return false;
}
```

Listing 11.7 Statusprüfung

Die Überprüfung, ob für einen bestimmten Supporter eine E-Mail gesendet werden soll, lagern wir in eine zusätzliche Methode aus:

```
private bool CheckSupporter(SPUser supporter)
{
    bool result = true;
    SPList list = Globals.GetList(supporter.ParentWeb,
        Constants.ListUrl.ConfigSupporter);
    SPQuery query = new SPQuery();
    query.Query = "<Where><Eq>" +
        $"<FieldRef Name = '{Constants.FieldInternalName.Supporter}'" +
        "LookupId = 'True'/>"+
        $"<Value Type = 'User'>{supporter.ID}</Value>"+
        "</Eq></Where>";
    query.ViewFields = $"<FieldRef Name =
        '{Constants.FieldInternalName.InfoMailActive}' />";
    query.ViewFieldsOnly = true;
    SPListItemCollection items = list.GetItems(query);
    if (items.Count == 1)
    {
        bool? mailActive =
```

```
            items[0][Constants.FieldInternalName.InfoMailActive] as bool?;
        if (mailActive.HasValue)
            result = mailActive.Value;
    }
    return result;
}
```

Listing 11.8 Prüfung, ob der aktuelle Benutzer »Supporter« ist

Die Methode fragt die Supporterkonfigurationsliste ab. Durch die Option `LookupId='True'` erfolgt die Suche über die ID des Benutzers. Um die Datenmengen möglichst gering zu halten, schränken Sie die Spalten der Ergebnismenge durch das Setzen der `ViewFields` und der Option `ViewFieldsOnly` ein. Wenn für den Supporter ein Eintrag gefunden wurde, wird die Spalte *Informations-E-mail aktiv* ausgewertet, ansonsten wird direkt `true` zurückgeliefert. Das bedeutet, wenn für einen Supporter keine Konfiguration hinterlegt wurde, wird eine E-Mail gesendet.

Denselben Schritt führen Sie mit der Methode `CheckCustomer` für die Kunden durch:

```
private bool CheckCustomer(SPListItem customer)
{
    bool result = true;
    SPList list = Globals.GetList(customer.Web,
        Constants.ListUrl.ConfigCustomer);
    SPQuery query = new SPQuery();
    query.Query = "<Where><Eq>" +
        $"<FieldRef Name = '{Constants.FieldInternalName.Customer}'" +
        $"LookupId = 'True'/>" +
        $"<Value Type = 'Lookup'>{customer.ID}</Value>" +
        "</Eq></Where>";
    query.ViewFields = "<FieldRef Name = " +
        $"'{Constants.FieldInternalName.InfoMailActive}' />";
    query.ViewFieldsOnly = true;
    SPListItemCollection items = list.GetItems(query);
    if (items.Count == 1)
    {
        bool? mailActive =
            items[0][Constants.FieldInternalName.InfoMailActive] as bool?;
        if (mailActive.HasValue)
            result = mailActive.Value;
    }
    return result;
}
```

Listing 11.9 Prüfung, ob der aktuelle Benutzer »Kunde« ist

Die Methode fragt die Kundenkonfigurationsliste ab. Durch die Option `LookupId='True'` erfolgt die Suche über die `ListItemId` des Kunden. Wenn für den Kunden ein Eintrag gefunden wurde, wird die Spalte *Informations-E-Mail aktiv* ausgewertet, ansonsten wird `true` zurückgeliefert. Das bedeutet, wenn für einen Kunden keine Konfiguration hinterlegt wurde, wird eine E-Mail gesendet.

Bei neu erstellten Tickets sollen alle Supporter per E-Mail benachrichtigt werden. Alle Supporter befinden sich in der Gruppe *Supporter*. Wir greifen auf diese Gruppe zu und prüfen je Benutzer, ob in der Konfigurationsliste die E-Mail-Benachrichtigungen für den jeweiligen Supporter abgeschaltet wurden. Nur wenn für den Supporter die E-Mail-Benachrichtigungen aktiv sind und eine E-Mail-Adresse hinterlegt wurde, wird diese E-Mail-Adresse zur Ergebnismenge hinzugefügt.

```
private string GetSupporterMailAddresses(SPWeb web)
{
    SPGroup group = web.SiteGroups[Localization.GetString(
        Localization.Keys.TxtSupporter)];
    string result = "";
    foreach (SPUser user in group.Users)
    {
        if (CheckSupporter(user)
                && !string.IsNullOrWhiteSpace(user.Email))
        {
            if (!string.IsNullOrWhiteSpace(result))
                result += ",";
            result += user.Email;
        }
    }
    return result;
}
```

Listing 11.10 Ermittlung der E-Mail-Adresse des Supporters

Sie benötigen noch eine Methode, die aus dem Lookup-Wert für den Kunden das Kunden-Item aus der Kundenliste ausliest. Ergänzen Sie die Methode aus Listing 11.11.

```
private SPListItem GetCustomer(SPWeb web, string lookupValue)
{
    if (!string.IsNullOrWhiteSpace(lookupValue))
    {
        SPList customerList = Globals.GetList(web,
            Constants.ListUrl.Customer);
            SPFieldLookupValue lv = new SPFieldLookupValue(lookupValue);
        return customerList.GetItemById(lv.LookupId);
```

```
    }
    else
        return null;
}
```

Listing 11.11 Ermittlung des Kunden

Kommen wir nun zur Methode `CheckAndSendMail`.

```
private void CheckAndSendMail(SPItemEventProperties properties)
{
    Settings settings = new Settings(properties.Web);
    string address = "";
    string subject = "";
    string body = "";
    bool sendMail = false;
    //Status
    int statusId = 0;
    object statusValue = GetValue(properties,
        Constants.FieldInternalName.Ticketstatus);
    if (statusValue != null)
    {
        SPFieldLookupValue lv = new
            SPFieldLookupValue(statusValue.ToString());
        statusId = lv.LookupId;
    }
    //Prüfen, ob für den Status E-Mails aktiv sind
    if (CheckStatus(statusId, settings))
    {
        //Supporter werden bei neuen Tickets informiert,
        //Kunde bei allen anderen Ticketstatusänderungen
        if (statusId == settings.StatusNew)
        {
            subject = settings.SupporterMailSubject;
            body = settings.SupporterMailBody;
            address = GetSupporterMailAddresses(properties.Web);
            sendMail = !string.IsNullOrWhiteSpace(address);
        }
        else
        {
            subject = settings.CustomerMailSubject;
            body = settings.CustomerMailBody;
            SPListItem customer = GetCustomer(properties.Web,
                GetValue(properties,
```



```
                    Constants.FieldInternalName.Customer).ToString());
            sendMail = CheckCustomer(customer);
            address = customer[Constants.FieldInternalName.MailAddress]
                as string;
        }
        if (sendMail)
        {
            subject = Email.ReplacePlaceholder(subject, properties);
            body = Email.ReplacePlaceholder(body, properties);
            Email.SendMail(properties.Web, address, subject, body);
        }
    }
}
```

Listing 11.12 Versenden der E-Mail

In der Methode wird mithilfe der vorher erstellten Methoden geprüft, ob die Benutzerbenachrichtigungen aktiv sind. Dann wird geprüft, ob für den aktuellen Status E-Mails gesendet werden sollen. Im Anschluss liest die Methode die konfigurierten Texte aus und prüft, ob für den jeweiligen Kunden bzw. Supporter diese E-Mail gesendet werden soll. Zum Schluss ersetzt sie, wenn die E-Mail gesendet werden soll, die Platzhalter und sendet die E-Mail.

11.2 E-Mail-Versand

Für den E-Mail-Versand und das Ersetzen der Platzhalter benötigen wir noch weitere Funktionen. Für diese Methoden erstellen Sie die Klasse `Email` im *BusinessLayer*-Projekt. Wir berücksichtigen bei der Entwicklung direkt, dass wir diese Klasse auch später bei den TimerJobs für den E-Mail-Versand nutzen möchten.

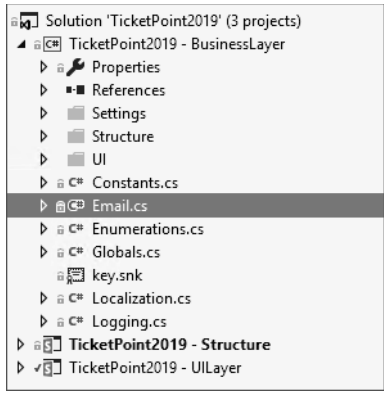

Abbildung 11.7 »Email«-Klasse

Zuerst erstellen Sie eine Enumeration, um alle möglichen Platzhalter festzulegen, die dann automatisch ersetzt werden sollen. Wie Sie sehen, erstellen wir auch direkt die Platzhalter für die TimerJobs, z. B. `CountPendingTickets`.

```
public enum Placeholder
{
    [Description("Ticketnummer")]
    TicketNumber,
    [Description("Betreff")]
    Subject,
    [Description("Problembeschreibung")]
    Problemdescription,
    [Description("Ansprechpartner")]
    ContactPerson,
    [Description("Kundenname")]
    Customer,
    [Description("E-Mail-Adresse des Kunden")]
    CustomerEmail,
    [Description("Priorität")]
    Priority,
    [Description("Ticketstatus")]
    Ticketstatus,
    [Description("Erstellt am")]
    Created,
    [Description("Zuletzt geändert am")]
    Modified,
    [Description("Link zum Ticket")]
    TicketLink,
    [Description("Aktuelles Datum")]
    CurrentDate,
    [Description("Anzahl der anstehenden Tickets")]
    CountPendingTickets,
    [Description("Anzahl der bearbeiteten Tickets der letzten Woche")]
    CountProcessedTickets,
    [Description("Anzahl der abgeschlossenen Tickets der letzten Woche")]
    CountClosedTickets,
    [Description("Anzahl der erstellten Tickets der letzten Woche")]
    CountCreatedTickets
}
```

Listing 11.13 Platzhalterdefinition

Um die Platzhalter in einem Text zu ersetzen, fügen Sie die Methode aus Listing 11.14 ein.


```
public static string ReplacePlaceholder(string text, object data,
    Dictionary<string, string> values = null)
{
    Regex regex = new Regex(@"\[\[[a-z A-Z]*]]");
    MatchCollection matchCol = regex.Matches(text);
    foreach (Match match in matchCol)
    {
       string placeholder = match.Value.Replace("[[", "").Replace("]]", "");
            text = text.Replace(match.Value,
            GetPlaceholderValue(placeholder, data, values));
    }
    return text;
}
```

Listing 11.14 Hilfsmethode zum Austauschen der Platzhalter im Text

Die Methode erwartet den Text, in dem die Platzhalter ersetzt werden sollen, ein Datenobjekt und ein Dictionary mit zusätzlichen Daten. Das Datenobjekt ist allgemein gehalten, damit wir diese Methode aus dem EventReceiver mit `SPListItemEventProperties` und aus dem TimerJob mit dem `SPListItem`-Objekt aufrufen können. Die folgenden Methoden werden ebenfalls, soweit möglich, mit dem allgemeinen Objekt arbeiten. Das Dictionary wird später zusätzliche Daten enthalten, die nicht im ListItem enthalten sind, zum Beispiel die Anzahl aller offenen Tickets. Um alle Platzhalter zu ersetzen, suchen wir sie über einen Regex-Ausdruck und tauschen sie im Anschluss aus. Mit dem Regex-Ausdruck `\[\[[a-z A-Z]*]]` suchen wir in dem übergebenen Text die Platzhalter in eckigen Klammern. Erstellen Sie die Methode `GetPlaceholderValue`, die den Text für den Platzhalter ermittelt.

```
private static string GetPlaceholderValue(string placeholder, object data,
    Dictionary<string, string> values)
{
    string result = "";
    return result;
}
```

Listing 11.15 Hilfsmethode zum Ermitteln der Platzhalterwerte

Um die Platzhalter zu ersetzen, die aus den Daten des ListItems ausgelesen werden, fügen Sie die Codezeilen aus Listing 11.16 ein.

```
if (placeholder == Placeholder.ContactPerson.ToString())
        result =
 GetFieldValue(data, Constants.FieldInternalName.ContactPerson);
```

```
else if (placeholder == Placeholder.Created.ToString())
    result = GetFieldValue(data, Constants.FieldInternalName.Created);
else if (placeholder == Placeholder.Customer.ToString())
    result = GetFieldValue(data, Constants.FieldInternalName.Customer);
else if (placeholder == Placeholder.Priority.ToString())
    result = GetFieldValue(data, Constants.FieldInternalName.Priority);
else if (placeholder == Placeholder.Problemdescription.ToString())
    result = GetFieldValue(data,
        Constants.FieldInternalName.Problemdescription);
else if (placeholder == Placeholder.Subject.ToString())
    result = GetFieldValue(data, Constants.FieldInternalName.Subject);
else if (placeholder == Placeholder.TicketNumber.ToString())
    result = GetFieldValue(data, Constants.FieldInternalName.Ticketnumber);
else if (placeholder == Placeholder.Ticketstatus.ToString())
    result = GetFieldValue(data, Constants.FieldInternalName.Ticketstatus);
else if (placeholder == Placeholder.Modified.ToString())
    result = GetFieldValue(data, Constants.FieldInternalName.Modified);
```

Listing 11.16 Wertermittlung je Platzhalter

Für diese Platzhalter wird die `GetFieldValue`-Methode aufgerufen, die den Wert aus dem Datenobjekt ausliest und zurückliefert. Diese Methode werden Sie im Anschluss erstellen. Fügen Sie den Code aus Listing 11.17 ein.

```
else if (placeholder == Placeholder.CountClosedTickets.ToString())
    result = values != null ?
        values[Placeholder.CountClosedTickets.ToString()] : "";
else if (placeholder == Placeholder.CountCreatedTickets.ToString())
    result = values != null ?
        values[Placeholder.CountCreatedTickets.ToString()] : "";
else if (placeholder == Placeholder.CountPendingTickets.ToString())
    result = values != null ?
        values[Placeholder.CountPendingTickets.ToString()] : "";
else if (placeholder == Placeholder.CountProcessedTickets.ToString())
    result = values != null ?
        values[Placeholder.CountProcessedTickets.ToString()] : "";
```

Listing 11.17 Wertermittlung je Platzhalter

Die Platzhalter für die zusätzlichen Daten aus dem Dictionary werden hier ersetzt. Wenn die Daten vorhanden sind, werden sie über den Key ausgelesen.

Um den Platzhalter für das aktuelle Datum zu ersetzen, ergänzen Sie den Code aus Listing 11.18.


```
else if (placeholder == Placeholder.CurrentDate.ToString())
    result = DateTime.Now.ToString("d", CultureInfo.GetCultureInfo(1031));
```

Listing 11.18 Wertermittlung je Platzhalter

Um den Platzhalter für den Link auf das aktuelle Ticket zu ersetzen, fügen Sie den Code aus Listing 11.19 ein.

```
else if (placeholder == Placeholder.TicketLink.ToString())
    result = $"<a href='{GetTicketUrl(data)}'>"
        + $"{GetFieldValue(data,
        Constants.FieldInternalName.Ticketnumber)}</a>";
```

Listing 11.19 Wertermittlung je Platzhalter

Zum Schluss fügen Sie den Code aus Listing 11.20 ein, um die E-Mail-Adresse des Kunden einfügen zu können.

```
else if (placeholder == Placeholder.CustomerEmail.ToString())
{
    SPList customerList = Globals.GetList(GetWeb(data),
        Constants.ListUrl.Customer);
    SPListItem customer = GetListItem(customerList, data,
        Constants.FieldInternalName.Customer);
    result = GetFieldValue(customer,
        Constants.FieldInternalName.MailAddress);
}
```

Listing 11.20 Wertermittlung je Platzhalter

Mit der Methode `GetListItem` wird auf das Kunden-ListItem zugegriffen, um im Anschluss den Wert der E-Mail-Spalte auszulesen. Die `GetListItem`-Methode werden wir in diesem Kapitel noch implementieren.

Nun erstellen Sie die Methode `GetFieldValue`. Diese Methode liefert den Wert aus dem `ListItem` oder den `EventProperties`. Wenn es sich um ein `SPListItem` handelt, wird die Überladung, die die Daten aus dem `ListItem` ausliest, aufgerufen. Handelt es sich um `SPItemEventProperties`, wird zusätzlich der Eventtyp geprüft. Beim `Added`-Event befinden sich die Daten im `ListItem`. Beim `Updating`-Event müssen die Daten aus den `AfterProperties` ausgelesen werden, weshalb wir die Überladung für die `SPItemEventProperties` aufrufen.

```
private static string GetFieldValue(object data, string internalName)
{
    if (data is SPListItem)
        return GetFieldValue(data as SPListItem, internalName);
```

```
    else if (data is SPItemEventProperties)
    {
        SPItemEventProperties itemEventProps =
            data as SPItemEventProperties;
        if(itemEventProps.EventType == SPEventReceiverType.ItemUpdating)
            return GetFieldValue(itemEventProps, internalName);
        else
            return GetFieldValue(itemEventProps.ListItem, internalName);
    }
    else
    {
        Logging.LogError(new Exception("Es werden nur Daten vom Typ SPListItem
 und SPItemEventProperties unterstützt."),
        Constants.LogCategory.BusinessLogic);
        return "";
    }
```

Listing 11.21 Ermittlung von Feldwerten aus EventProperties

Fügen Sie die Überladung für die `SPListItem`s ein. Die Methode greift auf die Spalte zu und verwendet die `GetFieldValueAsText`-Methode, um sich anhand des Werts direkt den für die Ausgabe formatierten Text zu holen.

```
private static string GetFieldValue(SPListItem item, string internalName)
{
    return item.Fields.GetFieldByInternalName(internalName).
        GetFieldValueAsText(item[internalName]);
}
```

Listing 11.22 Ermittlung von Feldwerten aus einem ListItem

Die Überladung für die `SPListItemEventProperties` ist etwas komplexer. Hier müssen Sie je nach Spaltentyp die Daten anders für die Ausgabe aufbereiten. Dazu holen Sie sich zuerst das `Field`-Objekt aus dem ListItem.

```
public static string GetFieldValue(SPItemEventProperties
    ticketEventProperties, string internalName)
{
    SPField field = ticketEventProperties.ListItem.Fields.
        GetFieldByInternalName(internalName);
    string result = "";

    return result;
}
```

Listing 11.23 Ermittlung von Feldwerten aus ItemEventProperties

Da die Werte für `Created` und `Modified` nicht in den `AfterProperties` enthalten sind, müssen wir uns diese Werte anderweitig beschaffen. Den Zeitpunkt der Erstellung können wir aus dem `ListItem` auslesen, da er sich nicht ändert. Für das Änderungsdatum nutzen wir das aktuelle Datum. Der Updateprozess ist zwar noch nicht vollständig abgeschlossen und der Wert könnte um wenige Millisekunden abweichen, für die Ausgabe in einer Informations-E-Mail spielt das aber keine Rolle.

```
if (internalName == Constants.FieldInternalName.Created)
    result = GetFieldValue(ticketEventProperties.ListItem, internalName);
else if (internalName == Constants.FieldInternalName.Modified)
    result = field.GetFieldValueAsText(DateTime.Now);
```

Listing 11.24 Ermittlung von Feldwerten aus ItemEventProperties

Werte vom Typ `Text` oder `Multiline-Text` können wir mit der Standardmethode formatiert auslesen.

```
else if (field.Type == SPFieldType.Text || field.Type == SPFieldType.Note)
    result = field.GetFieldValueAsText(
        ticketEventProperties.AfterProperties[internalName]);
```

Listing 11.25 Ermittlung von Feldwerten aus ItemEventProperties

Bei Spalten vom Typ *Benutzer* stehen wir vor der Herausforderung, dass wir in den `AfterProperties` nur die ID des Benutzers erhalten. Um Zugriff auf das Benutzerobjekt mit dem ermittelten Anzeigenamen zu bekommen, greifen wir über das `SPWeb`-Objekt auf die `SiteUsers`-Property und dann auf die `GetByID`-Methode zu. Die `SiteUsers`-Property enthält alle Benutzer der SiteCollection.

> **Hinweis**
>
> Beachten Sie, dass derselbe Benutzer in einer anderen SiteCollection nicht unbedingt dieselbe ID hat. Im Fall von TicketPoint ist dies hier nicht relevant.

```
else if (field.Type == SPFieldType.User)
{
    string temp =
        ticketEventProperties.AfterProperties[internalName].ToString();
    SPUser user =
        ticketEventProperties.Web.SiteUsers.GetByID(int.Parse(temp));
    result = user.Name;
}
```

Listing 11.26 Ermittlung von Feldwerten aus ItemEventProperties

Zuletzt fehlt Ihnen noch die Möglichkeit, Lookup-Werte zu ersetzen. Über das `SPFieldLookup`-Objekt lesen wir die Properties `LookupWebId` und `LookupList` aus, um das Web und die Liste zu erzeugen, damit wir im Anschluss das ListItem auslesen können, um dann mit unserer `GetFieldValue`-Methode den formatierten Wert auszulesen.

```
else if (field.Type == SPFieldType.Lookup)
{
    SPFieldLookup lookupField = field as SPFieldLookup;
    string temp =
        ticketEventProperties.AfterProperties[internalName].ToString();
    using (SPWeb lookupWeb =
        ticketEventProperties.Web.Site.OpenWeb(lookupField.LookupWebId))
    {
        SPList list = lookupWeb.Lists[new Guid(lookupField.LookupList)];
        SPListItem item = list.GetItemById(int.Parse(temp));
        result = GetFieldValue(item, lookupField.LookupField);
    }
}
```

Listing 11.27 Ermittlung von Feldwerten aus ItemEventProperties

Um das `SPWeb`-Objekt aus dem Datenobjekt auszulesen, erstellen Sie die Methode aus Listing 11.28, die, je nachdem, um welchen Objekttyp es sich handelt, auf das `SPWeb`-Objekt zugreift und dieses zurückliefert.

```
private static SPWeb GetWeb(object data)
{
    if (data is SPListItem)
         return (data as SPListItem).Web;
    else if (data is SPItemEventProperties)
            return (data as SPItemEventProperties).Web;
    else
    {
         Logging.LogError(new Exception("Es werden nur Daten vom Typ
                SPListItem und SPItemEventProperties unterstützt."),
            Constants.LogCategory.BusinessLogic);
            return null;
    }
}
```

Listing 11.28 Ermittlung des aktuellen Webs

Die Methode aus Listing 11.29 liefert aus dem Datenobjekt das ListItem einer Lookup-Spalte, wenn die Lookup-Spalte gefüllt ist. Auch hier wird wieder zwischen dem `SPListItem`- und dem `SPListItemEventProperties`-Objekt unterschieden.


```
private static SPListItem GetListItem(SPList list,
    object data, string lookupFieldName)
{
    SPListItem result = null;
    if (data is SPListItem)
    {
        SPListItem listItem = data as SPListItem;
        if (listItem[lookupFieldName] != null)
        {
            SPFieldLookupValue lv = new
                SPFieldLookupValue(listItem[lookupFieldName].ToString());
            result = list.GetItemById(lv.LookupId);
        }
    }
    else if (data is SPItemEventProperties)
    {
        SPItemEventProperties eventProperties =
            data as SPItemEventProperties;
        if (eventProperties.AfterProperties[lookupFieldName] != null)
        {
            SPFieldLookupValue lv = new SPFieldLookupValue(
                eventProperties.AfterProperties[lookupFieldName].ToString());
            result = list.GetItemById(lv.LookupId);
        }
    }
    else
    {
        Logging.LogError(new Exception("Es werden nur Daten vom Typ SPListItem
                                        und SPItemEventProperties unterstützt."),
        Constants.LogCategory.BusinessLogic);
    }
    return result;
}
```

Listing 11.29 Ermittlung des aktuellen ListItems

Fügen Sie die `GetTicketUrl`-Methode ein, um den Link auf das Ticket zu generieren:

```
public static string GetTicketUrl(object data)
{
    SPListItem ticket = null;
    if (data is SPListItem)
        ticket = data as SPListItem;
    else if (data is SPItemEventProperties)
```

```
        ticket = (data as SPItemEventProperties).ListItem;
    return ticket.Web.Site.MakeFullUrl(
        ticket.ParentList.DefaultDisplayFormUrl) + "?ID=" + ticket.ID;
}
```

Listing 11.30 Ermittlung der Ticket-URL

Da Sie nun alle Methoden zum Ersetzen der Platzhalter erstellt haben, kommen wir zum eigentlichen Versand der E-Mail. Dazu erstellen Sie eine Methode, die die Einstellungen für ausgehende E-Mails, die in der CENTRAL ADMINISTRATION vorgenommen wurden, ausliest und dann die E-Mail versendet:

```
public static void SendMail(SPWeb web, string address,
    string subject, string body)
{
    MailMessage message = new MailMessage();
    message.IsBodyHtml = true;
        message.From =
        new MailAddress(web.Site.WebApplication.OutboundMailSenderAddress);
    SPOutboundMailServiceInstance smtpServer =
        web.Site.WebApplication.OutboundMailServiceInstance;
    if (smtpServer != null)
    {
        SmtpClient smtp = new SmtpClient(smtpServer.Server.Address);
            message.To.Add(address);
            message.Subject = subject;
            message.Body = body;
            smtp.Send(message);
    }
    else
    {
            throw new Exception("Die E-Mail-Einstellungen in SharePoint
            sind nicht konfiguriert.");
    }
}
```

Listing 11.31 E-Mail senden

Damit haben Sie nun alle Methoden erstellt, um die erforderlichen E-Mails versenden zu können. Deployen Sie die Solution und aktivieren Sie das Strukturfeature. Starten Sie *smtp4dev* (siehe Abschnitt 2.4, »smtp4dev«) und überprüfen Sie die Funktion der EventReceiver durch Erstellen und Ändern von Tickets.