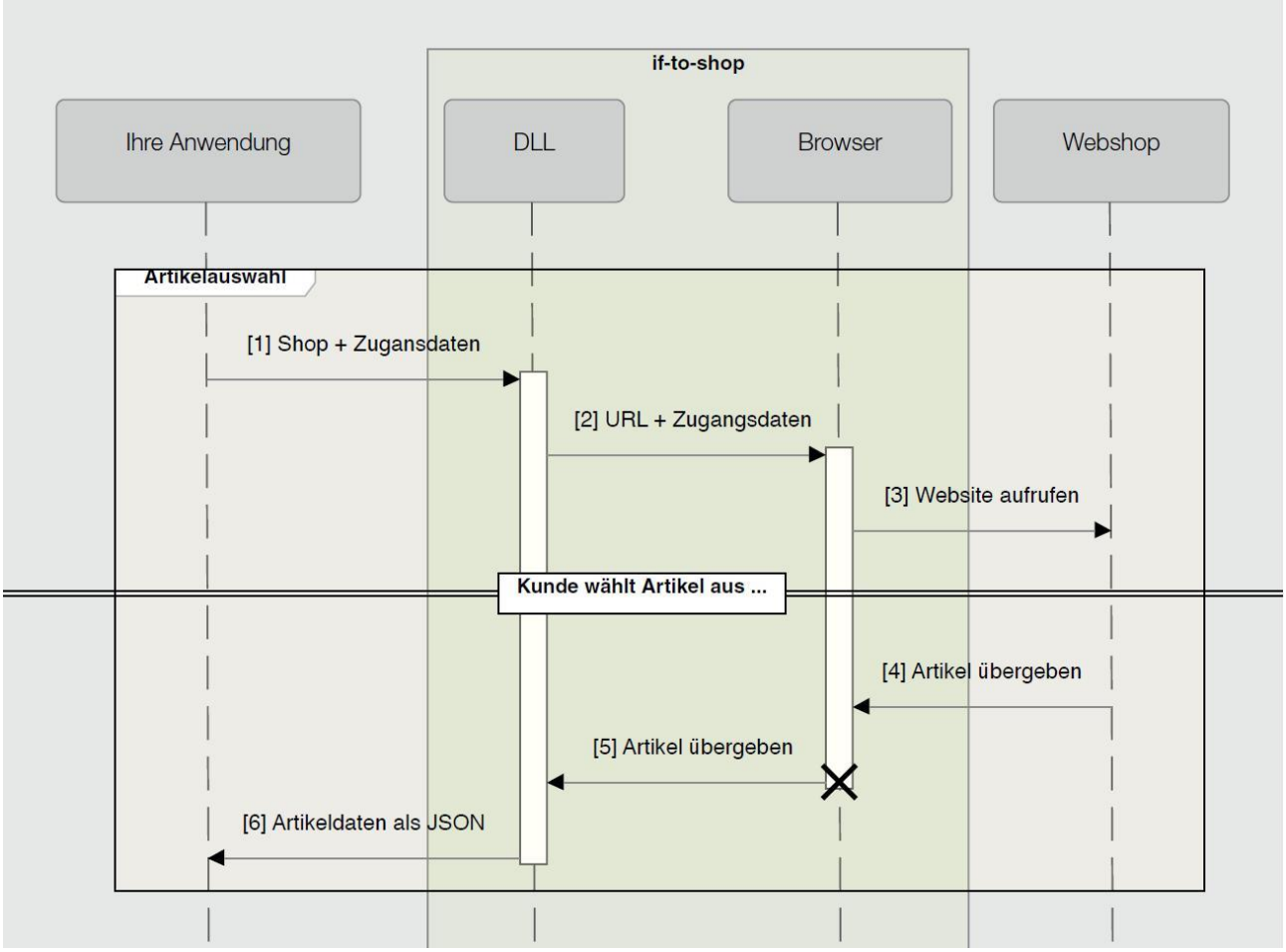
# Inhaltsverzeichnis

# 1. Anbindung von Anwendungen an die if-to-shop-Schnittstelle

Dieses Dokument beschreibt die Anbindung von Software-Anwendungen an die if-to-shop-Schnittstelle. In diesem Dokument wird beschrieben wie der Aufruf der IfToShop.dll implementiert wird und ein Datenaustausch mit der Schnittstelle stattfindet.

Es kann auf Web-Shops unterschiedlicher Lieferanten zugegriffen werden, wo der Anwender Artikel auswählen und in die aufrufende Applikation übernehmen kann. Die ausgewählten Artikel können wieder im Web-Shop aufgerufen und dort angezeigt oder geändert werden. Es ist möglich die Preise von Artikeln anzufragen und zu aktualisieren. Abschließend kann der Anwender die ausgewählten Artikel in den Warenkorb des Web-Shops legen und bestellen, wobei die Bestelldaten an die aufrufende Anwendung zurückgegeben werden.



## 1.1. Initialisierung der Schnittstelle

### Download:

IfToShop stellt alle benötigten Dateien, die für die Implementierung der Schnittstelle benötigt werden, in einer gepackten ZIP-Datei zur Verfügung. Mit den folgenden Links können die ZIP-Dateien, im 32 bzw. 64 Bit-Format, sowie die aktuelle Schnittstellenbeschreibung heruntergeladen werden

https://www.if-to-shop.com/download/ifToShop32.zip

https://www.if-to-shop.com/download/ifToShop64.zip

Senden Sie bitte eine Mail an info@if-to-shop.com um das Passwort zum Entpacken der ZIP-Datei zu erhalten.

### Testzugang:

Sie erhalten ebenfalls Zugangsdaten zu einem Testshop, mit der Sie die Funktionalität der Schnittstelle testen können. Um den Testshop freizuschalten, muss in der Registry unter dem Schlüssel ‚HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\PinnCalc\Corpora‘ der Eintrag ‚WebShopOSG‘ mit dem Wert ‚1‘ eingetragen werden. Mit diesem Eintrag wird der Testanbieter ‚IfToShop‘ in der Providerliste übertragen und die erforderlichen Anbieterdaten heruntergeladen (siehe auch Kapitel 1.2). Beim Endanwender darf der Registry-Eintrag nicht vorhanden sein.

### Implementierung:

Die IfToShop.dll muss als Verweis, inklusive aller Abhängigkeiten, zum Projekt hinzugefügt werden.

**VisualStudio – Nuget-Paketmanager**
Cef.Sharp (Version 79.1.360)
Newtonsoft.Json

**VisualStudio – Nuget-Paketmanager-Konsole**

```
Install-Package CefSharp.WinForms -Version 79.1.360
Install-Package Newtonsoft.Json -Version 13.0.1
```

Wird die Anbindung in C++ realisiert, so muss die IfToShop.dll als Interface implementiert werden. Danach kann eine Instanz erzeugt werden, mit welcher der Datenaustausch mit der IfToShop.dll stattfindet.

Damit die IfToShop.dll immer auf den aktuellsten Stand ist, lässt sich die neueste Version der Dll mit dem Befehl '.../Shop/if-to-Shop.exe -Update' aus dem Internet herunterladen, solange sich die Hauptversions- bzw. die Nebenversionsnummer der IfToShop.dll nicht geändert habt. Sobald sich die Hauptversions- oder die Nebenversionsnummer der IfToShop.dll geändert hat muss die Applikation, welche die Schnittstelle implementiert hat, evtl. angepasst und neu erzeugt werden. In diesem Fall bekommen Sie die benötigten Dateien, und eine erweiterte Schnittstellenbeschreibung, vom Betreiber der IfToShop-Schnittstelle zugesendet.

### Beispiel C#:

```
IIfToShop.IfToShopMethods pIfToShop = null;

pIfToShop = new IIfToShop.IfToShopMethods();
if (!pIfToShop)
{
  // Fehler
}
```

## Beispiel C++:

```
#import "IfToShop.tlb" no_namespace raw_interfaces_only

IIfToShop *pIfToShop;
HRESULT hr;
CLSID clsad;

hr = CoInitialize(NULL);

// reg: Computer\HKEY_CLASSES_ROOT\IIfToShop.IfToShopMethods
hr = CLSIDFromProgID(OLESTR("IIfToShop.IfToShopMethods"), &clsad);
if (!FAILED(hr))
{
  hr = CoCreateInstance(clsad, NULL, CLSCTX_INPROC_SERVER, __uuidof(IIfToShop),
              (LPVOID*)(&pIfToShop));
}
if (FAILED(hr))
{
  // Dll nicht registriert
}
```

## 1.2. Initialisierung der allgemeinen Daten

Folgende allgemeine Angaben müssen an die Schnittstelle übergeben werden:

- Der Pfad, wo das Shop-Verzeichnis aus der ZIP-Datei liegt

- Eine Protokolldatei, in der Angaben zur Analyse und Fehlersuche protokolliert werden.

- Ein Log-Level, welches die Tiefe der Protokollierung angibt. Mögliche Werte sind hierbei: 1=nur Fehler, 2=Fehler und Informationen, 3=Fehler, Informationen und Funktionsaufrufe

- Ein temporärer Pfad, in dem die Grafikdatei bei einem Produkt-Aufruf gespeichert wird.

Mit dem Befehl GetProvider kann eine Liste mit den Lieferantennamen aus der Schnittstelle angefordert werden. Die Liste wird als JSON-Objekt zurückgeliefert und hat das Format:

```
[
  {
    name   : "Eindeutiger Lieferantenname",
    version: "Versionsnummer"
  },
  ...
]
```

Mit Hilfe der Versionsnummer läßt sich ermitteln, ob es neuere Daten (z.B. Adressdaten) für einen Anbieter vorhanden sind. Neue Anbieter werden an der Liste angefügt. Bei Anbietern, dessen Web-Shops nicht mehr aktiv sind, wird die Property 'Active' in der ProviderFlags.json-Datei auf false gesetzt. Die Propertiy läßt sich mit dem Befehl GetProviderFlag abfragen.

Die jeweiligen Adress-Angaben, die Steuernummer sowie die UstId lassen sich mit dem Befehl GetProviderData aus der Schnittstelle abrufen. Mit der Funktion GetProviderLogo und GetProviderDescription werden das Anbieter-Logo bzw. die Anbieter-Beschreibung als jpg-Datei im Grafikpfad zur Verfügung gestellt. Die Adress-Angaben werden im JSON-Format zurückgegeben und entsprechen folgender Form:

```
{
  bezeichnung : 'Kurzname',
  name        : 'Firmenname',
  strasse     : 'Strasse der Firma',
  lkz         : 'Landeskennzeichen',
  plz         : 'Postleitzahl der Firma',
  ort         : 'Firmensitz',
  telefon     : 'Telefonnummer',
  fax         : 'Faxnummer',
  homepage    : 'Homepage',
  email       : 'Mailadresse',
  link        : 'Link zur Registrierungs-Seite der Firma',
  steuernr    : 'Steuernummer der Firma',
  ustid       : 'UmsatzsteuerId der Firma'
}
```

Die Felder für das Login haben beim Lieferanten unterschiedliche Bezeichnungen. Um die Bezeichnungen in der Applikation anzupassen, können die Bezeichnungen mit dem Befehl GetProviderLabels aus der Schnittstelle abgerufen werden. Als Rückgabewert wird ein String geliefert, wobei die einzelnen Labels mit Semikolon getrennt sind. Felder, die im Web-Shop nicht unterstützt werden, enthalten einen Leerstring oder ein null. Das Format der Rückgabe ist:

```
{
  user     : 'Label für Nutzername',
  password : 'Label für Passwort',
  customer : 'Label für Kundennummer',
}
```

Bei den 'GetProvider...'-Befehlen können folgende Fehler auftreten:

- Internetverbindung ist nicht vorhanden: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'Error offline' zurückgeliefert.
- Der Shop ist nicht erreichbar (*): Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.
- Fehlerhafte Zugangsdaten (*): Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.
- Wartungsarbeiten, Störungen etc. (*): Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.
- Der angegebene Anbieter ist nicht bekannt: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'Error existProvider' zurückgeliefert.
- Der Shop ist nicht mehr aktiv: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'Error isProviderActive' zurückgeliefert.
- Sonstige Fehler: Treten sonstige Fehler bei den Befehlen auf, so wird eine entsprechende Fehlermeldung beginnend mit 'Error …' zurückgeliefert.

### Beispiel C#:

```
string ShopDir    = "c:\\Anwendung\\Shop\\";
string LogFile    = "c:\\Anwendung\\Shop\\Log\logFile.log";
string GraphicDir = "c:\\Anwendung\\Shop\\tmp\\";
string LogLevel   = "3";

string Result = pIfToShop.ShopDir(ShopDir);
if (!FAILED(Result))
  Result = pIfToShop.LogFile(LogFile);
if (!FAILED(Result))
  Result = pIfToShop.LogLevel(LogLevel);
if (!FAILED(Result))
  Result = pIfToShop.GraphicDir(GraphicDir);
if (!FAILED(Result))
{
  string ProviderList = pIfToShop.GetProvider();
  if (!FAILED(ProviderList))
  {
    string Provider = JSON(ProviderList).GetFirst("name");
    if (Provider.Length)
    {
      do
      {
        string ProviderData;
        string ProviderLogo;
        string ProviderDesc;
        string ProviderActive = pIfToShop.GetProviderFlag(Provider, "active");
        if (!FAILED(ProviderActive) && ProviderActive == "true")
        {
          ProviderData = pIfToShop.GetProviderData(Provider);
          if (!FAILED(ProviderData))
            ProviderLogo = pIfToShop.GetProviderLogo(Provider);
          if (!FAILED(ProviderLogo))
            ProviderDesc = pIfToShop.GetProviderDescription(Provider);
          if (!FAILED(ProviderDesc))
          {
            string Name    = JSON(ProviderData).Get("name");
            string Strasse = JSON(ProviderData).Get("strasse");
            ...
```

```
          }

          // Labels abfragen
          string ProviderLabels = pIfToShop.GetProviderLabels(Provider);

          string LabelUser    = JSON(ProviderLabels).Get("user");
          string LabelPswd    = JSON(ProviderLabels).Get("password");
          string LabelCusomer = JSON(ProviderLabels).Get("customer");

          Provider = JSON(ProviderList).GetNext("name")
        }
      }
      while(Provider.Length)
    }
  }
}
```

## Beispiel C++:

```
HRESULT hr;
BSTR Result = NULL;
BSTR ProviderList = NULL;
BSTR ProviderLabels = NULL;
std::string ShopDir    = "c:\\Anwendung\\Shop\\";
std::string LogFile    = "c:\\Anwendung\\Shop\\Log\\logFile.log";
std::string GraphicDir = "c:\\Anwendung\\Shop\\tmp\\";
std::string LogLevel   = "3";

hr = pIfToShop->ShopDir(_com_util::ConvertStringToBSTR(ShopDir));
if (!FAILED(hr))
  hr = pIfToShop->LogFile(_com_util::ConvertStringToBSTR(LogFile));
if (!FAILED(hr))
  hr = pIfToShop->LogLevel(_com_util::ConvertStringToBSTR(LogLevel));
if (!FAILED(hr))
  hr = pIfToShop->GraphicDir(_com_util::ConvertStringToBSTR(GraphicDir));
if (!FAILED(hr))
{
  HRESULT hrP = pIfToShop->GetProvider(&ProviderList);
  if (!FAILED(ProviderList))
  {
    std::string Prov = JSON(ProviderList).GetFirst("name");
    if(Prov.length())
    {
      BSTR Provider = _com_util::ConvertStringToBSTR(Prov);
      do
      {
        BSTR ProviderData = NULL;
        BSTR ProviderLogo = NULL;
        BSTR ProviderDesc = NULL;
        BSTR ProviderActive = NULL;
        BSTR ProviderDisorder = NULL;
        hr = pIfToShop->GetProviderFlags(Provider, BSTR("active"), &ProviderActive);
        if (!FAILED(hr) && ProviderActive.find("true") != std::string::npos)
        {
          hr = pIfToShop->GetProviderData(Provider, &ProviderData);
          if (!FAILED(hr) && !FAILED(ProviderData))
            hr = pIfToShop->GetProviderLogo(Provider, &ProviderLogo);
          if (!FAILED(hr) && !FAILED(ProviderLogo))
            hr = pIfToShop->GetProviderDescription(Provider, &ProviderDesc);
          if (!FAILED(hr) && !FAILED(ProviderDesc))
          {
            std::string Name    = JSON(ProviderData).Get("name");
            std::string Strasse = JSON(ProviderData).Get("strasse");
            ...
```

```
          }

          // Labels abfragen
          pIfToShop->GetProviderLabels(Provider, &ProviderLabels);

          std::string LabelUser    = JSON(ProviderLabels).Get("user");
          std::string LabelPswd    = JSON(ProviderLabels).Get("password");
          std::string LabelCusomer = JSON(ProviderLabels).Get("customer");

          Provider = JSON(ProviderList).GetNext("name")
        }
      }
      while(Provider.length())
    }
  }
}
```

### 1.3. Anbindung an einen WebShop

Für den Datenaustausch zwischen einem Web-Shop und der Applikation, muss der Lieferantenname sowie die Zugangsdaten übertragen werden. Der Lieferantenname ist eine eindeutige Kennung für den jeweiligen Web-Shop. Für das Login in einen Web-Shop wird der Nutzername, das Passwort und bei einigen Lieferanten auch die Kundennummer vorausgesetzt. Ist die Kundennummer keine Pflichtangabe, so kann diese leer gelassen werden.

Beim Aufbau einer Verbindung zu einem Web-Shop können folgende Fehler auftreten bzw. Zustände vom Anwender ausgelöst werden:

- Internetverbindung ist nicht vorhanden: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'Error offline' zurückgeliefert.
- Der Shop ist nicht erreichbar (*): Der Dialog muss vom Anwender, z.B. über 'x', geschlossen werden. Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.
- Fehlerhafte Zugangsdaten (*): Der Shop zeigt im Dialog eine entsprechende Meldung an. Nachdem der Anwender den Dialog geschlossen hat, wird ein 'cancel' zurückgeliefert.
- Wartungsarbeiten, Störungen etc. (*): Der Shop zeigt im Dialog eine entsprechende Meldung an. Nachdem der Anwender den Dialog geschlossen hat, wird ein 'cancel' zurückgeliefert.
- Der angegebene Anbieter ist nicht bekannt: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'Error existProvider' zurückgeliefert.
- Der Shop ist nicht mehr aktiv: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'Error isProviderActive' zurückgeliefert.
- Abbruch durch den Anwender über 'x' oder dem Abbruch-Button im Dialog: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'closed' bzw. 'cancel' zurückgeliefert.

Die angegebenen Punkte, die mit (*) markiert wurden, wurden noch nicht von allen Web-Shop-Anbietern umgesetzt.

Beispiel C#:

```
string Provider = "Provider";
string User     = "Username";
string Pswd     = "Password";
string Customer = "Customernumber";

string Result = pIfToShop.Init(Provider, User, Pswd, Customer);
if (FAILED(Result))
{
   if (Result.ToLower().Contains("error offline"))
      // Internet-Verbindung konnte nicht aufgebaut werden
   else if (Result.ToLower().Contains("error existprovider"))
      // unbekannter Provider
   else if (Result.ToLower().Contains("error isprovideractive"))
      // Internet-Verbindung konnte nicht aufgebaut werden
   else if (Result.ToLower().Contains("cancel"))
      // Störung oder Wartung; Shop ist nicht erreichbar; fehlerhafte Zugangsdaten: Abbruch
      // durch Anwender
}
```

Beispiel C++:

```
BSTR Provider = _com_util::ConvertStringToBSTR("Provider");
BSTR User     = _com_util::ConvertStringToBSTR("Username");
BSTR Pswd     = _com_util::ConvertStringToBSTR("Password");
BSTR Customer = _com_util::ConvertStringToBSTR("Customernumber");

BSTR Result = NULL;
HRESULT hrP = pIfToShop->Init(Provider, User, Pswd, Customer, &Result);
if (FAILED(Result))
{
   if (toLower(Result).find("error offline") != std::string::npos(""))
      // Internet-Verbindung konnte nicht aufgebaut werden
   else if (toLower(Result).find("error existprovider"))
      // unbekannter Provider
   else if (toLower(Result).find("error isprovideractive"))
      // Internet-Verbindung konnte nicht aufgebaut werden
   else if (toLower(Result).find("cancel"))
      // Störung oder Wartung; Shop ist nicht erreichbar; fehlerhafte Zugangsdaten: Abbruch
      // durch Anwender
}
```

## 1.4. Aufbau eines Web-Shop-Artikels

Artikel, die an einem Web-Shop übermittelt oder aus einem Web-Shop übernommen werden, werden im JSON-Format übertragen. Der Aufbau entspricht der Form:

```
{
  reference    : 'Eindeutige Referenz auf den Artikel',
  title        : 'Bezeichnung des Artikels',
  shortDescr   : 'Kurztext des Artikels',
  number       : 'Artikelnummer',
  pricePerUnit : 'Preis pro Einheit',
  currency     : 'Währung',
  packagingUnit: 'VE, Zahl, Dezimalzeichen Punkt oder Komma, z.B. 5.34',
  unit         : 'Einheit z.B. m',
  parameterHash: 'Hash-Wert bei einem Variantenartikel',
  gtinNumber   : 'GTIN-Nummer des Artikels',
  longDescr    : 'Beschreibung',
  articleGroup : 'Bezeichnung der Artikelgruppe',
  graphic      : 'URL oder Base64 kodiert',
  serverSiededParameters: 'true|false - Eventuelle Parameter des Artikels werden auf
  dem Server verwaltet und nicht über die Schnittstelle übergeben.',
  parameterized: 'true|false, default: false, Parametrisierter Artikel)
  parameters   : [ weitere Parameter im JSON Format ],
  descriptiveParameters: [ beschreibende Parameter im JSON Format ]
}
```

Ein Artikel muss alle oben beschriebene Properties enthalten, aber nicht alle müssen gefüllt sein. Werte, die keine Pflichtfelder sind, können null oder einen Leerstring enthalten. Pflichtfelder sind durch rote Schrift markiert.
Das JSON-Objekt, welches den Artikel repräsentiert, darf unter keinen Umständen geändert werden, da sonst der Artikel im Web-Shop nicht mehr gefunden wird.

Es können vom Web-Shop weitere Parameter geliefert werden, deren Anzahl frei vom Web-Shop vergeben werden. Die weiteren Parameter werden in Form eines Arrays mit JSON-Objekten in der Property 'parameters' übergeben und folgen immer der Form:

```
[
  {
    name  : "Name des Parameters als eindeutige Kennung",
    title : "Bezeichnung des Parameters",
    value : "Wert des Parameters",
    unit  : "Einheit des Parameters"
  },
  ...
]
```

Auch hier müssen alle Properties enthalten sein. Die Property 'unit' kann aber einen Leersting oder null enthalten.
Die weiteren Parameter sind Parameter, die nur bei Variantenartikeln vorkommen können. Diese werden nur vom Web-Shop benutzt und sollten nicht von der aufrufenden Anwendung verwendet werden.

Um in der aufrufenden Applikation die im Web-Shop ausgewählten Parameter anzeigen zu können, können beschreibende Parameter aus dem Web-Shop geliefert werden. Die beschreibenden Parameter sind optional und werden nicht an den Web-Shop zurückgegeben. In der Property 'descriptiveParameters' werden die beschreibenden Parameter in Form eines Arrays mit JSON-Objekten übergeben und folgen immer der folgenden Form:

```
[
  {
    label : "Bezeichnung des Parameters, z.B. Höhe, Länge, Width, Weight",
    value : "Wert des Parameters, z.B. 1.5",
    unit  : "Einheit des Parameters, z.B. qm, lfm"
  },
  ...
]
```

Die beschreibenden Parameter enthalten nähere Angaben zum ausgewählten Artikel. Diese können

von der aufrufenden Anwendung ausgelesen und verwendet werden.

Bei der Verarbeitung von Artikel, die aus dem Web-Shop kommen, unterscheidet man zwischen drei unterschiedlichen Typen:

### Standard-Artikel

Bei einem Standard-Artikel handelt es sich um einen Artikel, bei denen es keine Artikel-Parameter im Web-Shop gibt oder bei denen die unterschiedlichen Artikel-Parameter jeweils eine eigene Artikelnummern besitzen. Die Referenz und die Artikelnummer sind bei Standard-Artikel immer eindeutig. Es können Parameter vorkommen, die aber nur intern vom Web-Shop benutzt werden dürfen. Die Property 'parameterized' enthält den Wert 'false'.

#### Beispiel: Standardartikel mit beschreibenden Parametern

```
{
  "reference"     : "59960800",
  "title"         : "Drücker-Lochteil fest drehbar auf ovaler Rosette F1",
  "shortDescr"    : "Drücker-Lochteil fest",
  "number"        : "59960800",
  "pricePerUnit"  : "11.13",
  "currency"      : "EUR",
  "packagingUnit" : "1.0000",
  "unit"          : "Stück",
  "parameterHash" : "",
  "longDescr"     : "Gekröpft mit festdrehbarer Lagerung. Lochteil mit Rückholfeder und
                     ovaler Abdeckkappe. Links/rechts verwendbar",
  "articleGroup"  : null,
  "graphic"       : "https://media.shop24.com/400177/shop/images/59960800.png",
  "serverSiededParameters":false,
  "parameterized" : false,
  "parameters"    : null,
  "descriptiveParameters":
  [
      {label: 'Material', value: 'ALU'},
      {label: 'Oberfläche', value: 'blank'}
  ]
}
```

#### Beispiel: Artikel mit Verpackungseinheit

```
{
  "reference"    : "0899400617961   12",
  "title"        : "Montagehandschuh Economy",
  "shortDescr"   : "Leichter Handschuh mit atmungsaktiver, schwarzer PU-Beschichtung",
  "number"       : "0899400617961   12",
  "pricePerUnit" : "2.45000",
  "currency"     : "EUR",
  "packagingUnit": "12",
  "unit"         : "St.",
  "parameterHash": "",
  "longDescr"    : "Leichter Handschuh mit atmungsaktiver, schwarzer PU-
                    Beschichtung\nMontagehandschuh Economy",
  "articleGroup" : null,
  "graphic"      : "https://media.shop24.com/source/eshop/media/images/239122.jpg",
  "serverSiededParameters": false,
  "parameterized": false,
  "parameters"   : null,
  "descriptiveParameters": null
}
```

#### Beispiel: Artikel mit Verpackungseinheit und descriptiven Parametern

```
{
  "reference"    : "88439854",
```

```
  "title"        : "2SPA19weiss",
  "shortDescr"   : "Dekorspanplatte 19 mm weiß",
  "number"       : "DEK19W",
  "pricePerUnit" : "8,50",
  "currency"     : "EUR",
  "packagingUnit": "5,80",
  "unit"         : "m²",
  "parameterHash": "",
  "longDescr"    : "Dekorspanplatte 19 mm weiß formaldehydfrei, DIN 123",
  "articleGroup" : "Platten / Spanplatten / UNI / 19 mmv",
  "graphic"      : "https://www.ultimate-shop24.de/graphics/88439854-1",
  "serverSiededParameters": false,
  "parameterized": false,
  "parameters"   : null,
  "descriptiveParameters":
  [
    {"label": "Länge",  value: "2700", unit: "mm"},
    {"label": "Breite", value: "1100", unit: "mm"},
    {"label": "Stärke", value: "19",   unit: "mm"},
    {"label": "Farbe",  value: "weiß", unit: null}

  ]
}
```

### Parametrisierter Artikel mit anwenderseitiger Parameterverwaltung (Variantenartikel)

Bei einem parametrisierten Artikel handelt es sich um einen Artikel, bei denen die unterschiedlichen Artikel-Parameter über die gleiche Referenz und Artikelnummer abgebildet werden. Die Parameter-Auswahl des Nutzers wird als weitere Parameter, in der Property 'parameters', aus dem Web-Shop zurückgeliefert. Die weiteren Parameter werden im Web-Shop für die eindeutige Zuordnung benötigt und dürfen nicht verändert werden. Die Property 'parameterized' enthält den Wert 'true'.

Weil bei den parametrisierten Artikeln die Referenz nicht eindeutig ist, wird über die Parameter eine Hashwert-Berechnung durchgeführt und in die Property 'parameterHash' geschrieben. Der Hashwert wird innerhalt der IfToShop.dll generiert und an den Web-Shop übergeben, von wo er, unverändert, zurückgeliefert wird. Eine Änderung des Hashwerts ist nicht zulässig.

Erfolgt beim Artikel ein Preisupdate oder wurde der Artikel an den Warenkorb im Web-Shop übergeben, so wird der Wert aus der Property 'parameterHash' benutzt, um eine eindeutige Zuordnung zu erhalten (siehe auch Kapitel 1.8 und Kapitel 1.9).

Weil bei parametrisierten Artikel die Artikelnummer und die Referenz nicht eindeutig sind, ist darauf zu achten, dass auch eine generierte Artikelnummer aus dem Präfix des Anbieters sowie der Artikelnummer nicht eindeutig ist. Es muss noch zusätzlich ein Suffix generiert werden, um eine eindeutige Artikelnummer zu erhalten. Dies könnte z.B. durch eine laufende Nummer oder durch das Anhängen des Hashwerts geschehen (siehe Kapitel 1.5).

#### Beispiel: Parametrisierter Artikel mit anwenderseitiger Parameterverwaltung

```
{
  "reference"     : "552430",
  "title"         : "Fenster - 1-flg. DL",
  "shortDescr"    : "Fenster - 1-flg. DL",
  "number"        : "552430",
  "pricePerUnit"  : "620",
  "currency"      : "EUR",
  "packagingUnit" : "1",
  "unit"          : "Stück",
  "parameterHash" : "569781",
  "longDescr"     : "Fenster - 1-flg. DL\nBemaßung: 2700 mm x 1000 mm,\nDekor Innen:
                     Standardfarbe (weiß),\nBeschlag: Standardbeschlag,\nVerglasung: Iso
  "articleGroup"  : null,
```

```
  "graphic"       : "PHN2ZyB3aWR0aD0iMTIwMCIgaGVpZWxpbmctcZWxpbmctc3Ryb2tlIiBz
                     VpZWxpbmctcZWxpbmctc3Ryb2tlIiBz",
  "serverSiededParameters":false,
  "parameterized" : true,
  "parameters"    :
  [
    {"name": "FENSTERSHOP-INTERN", "value": "eyJwcm9kdWt0bmFt"}
  ],
  "descriptiveParameters":
  [
    {"label": "Länge",  value: "2700", unit: "mm"},
    {"label": "Breite", value: "1100", unit: "mm"},
    {"label": "Farbe",  value: "weiß", unit: null}

  ]
}
```


### Parametrisierter Artikel mit serverseitiger Parameterverwaltung

Bei parametrisierten Artikeln mit serverseitiger Parameterverwaltung werden die Parameter für die unterschiedlichen Artikel-Parameter serverseitig gespeichert und nicht von der aufrufenden Applikation übergeben. In diesen Fällen referenziert die Referenz nicht nur den Artikel, sondern auch seine Parameter. Für jede Artikel-Variante wird eine eigene Referenz sowie Artikelnummer generiert. Die Property 'parameters' bleibt bei serverseitiger Parameterverwaltung leer. Die Property 'serverSiededParameters' hat den Wert 'true' und die Property 'parameterized' den Wert 'false'.

Soll ein Artikel mit serverseitiger Parameterverwaltung kopiert werden, muss diese Kopie serverseitig erfolgen, damit beim Ändern der Parameter nicht der Originalartikel überschrieben wird. Zur serverseitigen Herstellung einer Kopie und Rückgabe der neuen Referenz dient der Befehl CopyProduct (siehe Kapitel 1.7).

#### Beispiel: Artikel mit serverseitigen Parameter


```
{
  "reference"     : "1612869861",
  "title"         : "Kunststofffenster",
  "shortDescr"    : "Kunststoffenster Classicline ohne Folierung",
  "number"        : "1612869861",
  "pricePerUnit"  : "202.00",
  "currency"      : "EUR",
  "packagingUnit" : "1",
  "unit"          : "Stück",
   "parameterHash" : "",
  "longDescr"     : "Kunststoffenster Classicline ohne Folierung\nDichtung grau\n
                     Br.=1000mm, Hö.=1000mm\nEntwässerung nach vorne\nDreh-Kipp links
                     Griff Classicline\n",
  "articleGroup"  : "Kunststoffenster",
  "graphic"       : "
                    QVR4XuydB1RUydLH/2bXsOacs65pjZgTGBHMARAzooiYdVGUIJMilnBHDBn15x1z",
  "serverSiededParameters": true,
  "parameterized" : false,
  "parameters"    : null,
  "descriptiveParameters": null
}
```

### 1.5. Auswahl eines Artikels

Mit der Funktion AddProduct wird die Auswahl eines Artikels, durch Aufruf des Web-Shops, gestartet. Der Nutzer kann einen Artikel aus dem Web-Shop auswählen, indem er zu dem Artikel navigiert, den er übernehmen möchte. Durch Betätigen des 'in APP übernehmen'-Buttons übernimmt der Nutzer den Artikel in die Applikation. Die Auswahl kann abgebrochen werden, indem der Nutzer den Abbruch-Button betätigt oder das Fenster über [x] verlässt. Wird das Fenster über Abbruch bzw. über dem [x]-Button verlassen, so wird als Rückgabewert ein 'cancel' bzw. ein 'closed' zurückgegeben. In einem Fehlerfall enthält der Rückgabewert 'Error' mit einer Fehlerbeschreibung bzw. ein 'cancel', falls der Web-Shop nicht erreichbar ist.

Bei einer erfolgreichen Auswahl wird der ausgewählte Artikel als JSON-Objekt zurückgegeben und kann in der aufrufenden Applikation weiterverarbeitet werden. Zur Erzeugung einer eindeutigen Artikelnummer, lässt sich mit dem GetProviderFlag ein Anbieter-Präfix auslesen. Dieser kann der Bestellnummer vorangestellt werden. Zu beachten ist, dass bei Variantenartikeln zusätzlich zum Präfix ein Suffix an der Artikelnummer gehängt werden muss, weil die Artikelnummer hier nicht eindeutig sein muss (siehe Kapitel 1.4).

Die dazugehörige Grafikdatei kann mit der Funktion GetProductGraphic abgerufen werden. Als Übergabeparameter wird das Grafikformat `"BMP"`, `"PNG"` oder `"JPG"` angegeben, in welches die Grafik konvertiert werden soll. Als Rückgabeparameter wird der Dateiname inkl. der Pfadangabe zurückgegeben. Die Grafikdatei wird im temporären Grafikpfad abgelegt (siehe Kapitel 1.2) .

Die Konvertierung der aus dem Web-Shop erhaltenen Grafik in das übergebene Grafikformat geschieht mit Hilfe des Bildbearbeitungstools 'ImageMagick'. Per Kommandozeile wird ein Konvertierungsbefehl ausgeführt, der die Grafikdatei bzw. die eingebettete Grafik in das gewünschte Ausgabeformat konvertiert. Es besteht die Möglichkeit eine selbst erstellte Kommandozeile an die Schnittstelle zu übergeben, um eigene Formate zu erzeugen oder um eigene Konvertierungsparameter anzugeben. Die angepasste Kommandozeile kann als weiterer Parameter an die Funktion GetProductGraphic übergeben werden. In diesem Fall wird das übergebene Dateiformat ignoriert und stattdessen die Kommandozeile zur Konvertierung herangezogen. Eine Beschreibung der möglichen Kommandozeilenparameter entnehmen Sie bitte der Dokumentation von ImageMagick.

Bei der Auswahl eines Artikels können folgende Fehler auftreten bzw. Zustände vom Anwender ausgelöst werden:

- Internetverbindung ist nicht vorhanden: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'Error offline' zurückgeliefert.
- Der Shop ist nicht erreichbar (*): Der Dialog muss vom Anwender, z.B. über 'x', geschlossen werden. Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.
- Fehlerhafte Zugangsdaten (*): Der Shop zeigt im Dialog eine entsprechende Meldung an. Nachdem der Anwender den Dialog geschlossen hat, wird ein 'cancel' zurückgeliefert.
- Wartungsarbeiten, Störungen etc. (*): Der Shop zeigt im Dialog eine entsprechende Meldung an. Nachdem der Anwender den Dialog geschlossen hat, wird ein 'cancel' zurückgeliefert.
- Abbruch durch den Anwender über 'x' oder dem Abbruch-Button im Dialog: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'closed' bzw. 'cancel' zurückgeliefert.

Die angegebenen Punkte, die mit (*) markiert wurden, wurden noch nicht von allen Web-Shop-Anbietern umgesetzt.

Beispiel C#:

```
string Format      = "BMP";
string CommandLine = "";

string Product = pIfToShop.AddProduct();
if (!FAILED(Product))
{
  string Graphic = pIfToShop.GetProductGraphic(Format, CommandLine);
  if (!FAILED(Graphic))
  {
  // Eindeutige Artikelnummer erzeugen
     string ProviderPrefix = pIfToShop.GetProviderFlag(Provider, "prefix");
  string ItemNumber    = ProviderPrefix + JSON(Product).Get("number");
  string Parameterized = JSON(Product).Get("paramterized");
  if (Parameterized.ToLower().Contains("true")
  {
    // Bei parametrisierten Artikeln um Postfix erweitern
       ItemNumber = ItemNumber + "_" + JSON(Product).Get("parameterHash");
  }
    ...
  }
  else
  {
    if (Graphic.ToLower().Contains("error"))
    {
      // Fehler bei der Konvertierung der Grafik
    }
  }
}
else
{
  if (Product.ToLower().Contains("cancel"))
  {
    // Abbruch über Abbruch-Button
  }
  else if (Product.ToLower().Contains("closed"))
  {
    // Abbruch über x-Button
  }
  else if (Product.ToLower().Contains("error"))
  {
    // Fehler aufgetreten
  }
}
```

Beispiel C++:

```
HRESULT hr;
BSTR Product     = NULL;
BSTR Graphic     = NULL;
BSTR Format      = _com_util::ConvertStringToBSTR("BMP");
BSTR CommandLine = _com_util::ConvertStringToBSTR("");

hr = pIfToShop->AddProduct(&Product);
if (!FAILED(hr) && !FAILED(Product))
{
  std::string Produkt = _com_util::ConvertBSTRToString(Product);
  if(Produkt.length())
  {
```

```
    hrP = pIfToShop->GetProductGraphic(Format, CommandLine, &Graphic);
    if (!FAILED(hr) && !FAILED(Graphic))
    {
    // Eindeutige Artikelnummer erzeugen
       BSTR ProviderPrefix = NULL;
       hr = pIfToShop->GetProviderFlag(Provider, BSTR("prefix"), &ProviderPrefix);
       std::string Prefix        = _com_util::ConvertBSTRToString(ProviderPrefix);
       std::string ItemNumber    = Prefix + JSON(Product1).Get("number");
       std::string Parameterized = JSON(Product1).Get("paramterized");
      if (toLower(Parameterized).find("true") != std::string::npos)
      {
     // Bei parametrisierten Artikeln um Postfix erweitern
      ItemNumber = ItemNumber + "_" + JSON(Product1).Get("parameterHash");
       }
      ...
    }
    else
    {
      std::string Result = _com_util::ConvertBSTRToString(Graphic);
      if (toLower(Result).find("error") != std::string::npos)
      {
        // Fehler bei der Konvertierung der Grafik
      }
    }
  }
}
else
{
  std::string Result = _com_util::ConvertBSTRToString(Product);
  if (toLower(Result).find("cancel") != std::string::npos)
  {
    // Abbruch über Abbruch-Button
  }
  else if (toLower(Result).find("closed“) != std::string::npos)
  {
    // Abbruch über x-Button
  }
  else if (toLower(Result).find("error") != std::string::npos)
  {
    // Fehler aufgetreten
  }
}
```

## 1.6. Anzeige eines zuvor ausgewählten Artikels

Um sich einen zuvor ausgewählten Artikel erneut anzeigen zu lassen oder dessen Parameter (z.B. Länge, Breite etc.) abzuändern, kann der Artikel mit der Funktion EditProduct erneut aufgerufen werden. Auch für einen Artikeltausch lässt sich der Befehl nutzen. Das JSON-Objekt vom zuvor ausgewählten Artikel wird als Parameter übergeben. Falls die Parameter im Web-Shop geändert oder ein anderer Artikel ausgewählt wurde wird der geänderte bzw. neue Artikel als JSON-Objekt zurückgegeben. Die Referenz ändert sich in diesem Fall, es sei denn es handelt sich um einen Variantenartikel bei dem nur Parameter geändert wurden (siehe Kapitel 1.4).

Bei der Anzeige eines zuvor ausgewählten Artikels können folgende Fehler auftreten bzw. Zustände vom Anwender ausgelöst werden:

- Internetverbindung ist nicht vorhanden: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'Error offline' zurückgeliefert.
- Der Shop ist nicht erreichbar (*): Der Dialog muss vom Anwender, z.B. über 'x', geschlossen werden. Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.
- Fehlerhafte Zugangsdaten (*): Der Shop zeigt im Dialog eine entsprechende Meldung an. Nachdem der Anwender den Dialog geschlossen hat, wird ein 'cancel' zurückgeliefert.
- Wartungsarbeiten, Störungen etc. (*): Der Shop zeigt im Dialog eine entsprechende Meldung an. Nachdem der Anwender den Dialog geschlossen hat, wird ein 'cancel' zurückgeliefert.
- Abbruch durch den Anwender über 'x' oder dem Abbruch-Button im Dialog: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'closed' bzw. ein 'cancel' zurückgeliefert.

Die angegebenen Punkte, die mit (*) markiert wurden, wurden noch nicht von allen Web-Shop-Anbietern umgesetzt.

### Beispiel C#:

```
string ProductNew = pIfToShop.EditProduct(ProductOld);
if (!FAILED(ProductNew))
{
  if (ProductNew != ProductOld)
  {
    // Artikel-Parameter wurden geändert oder ein anderer Artikel ausgewählt
  }
}
```

### Beispiel C++:

```
HRESULT hr;
BSTR ProductNew = NULL;
BSTR ProductOld = _com_util::ConvertStringToBSTR(Produkt);

hr = pIfToShop->EditProduct(ProductOld, &ProductNew);
if (!FAILED(hr) && !FAILED(ProductNew))
{
  if (ProductNew != ProductOld)
  {
    // Artikel-Parameter wurden geändert oder ein anderer Artikel ausgewählt
  }
}
```

## 1.7. Kopieren von Artikeln

Es kann vorkommen, dass die Parameter eines Artikels nicht an die Schnittstelle übergeben werden, sondern im Web-Shop selbst, also serverseitig, gespeichert werden. Die Property 'serverSiededParameters' hat in diesem Fall den Wert 'true'. Die Referenz referenziert hier nicht nur den Artikel, sondern auch dessen Parameter.

Falls ein Artikel mit serverseitigen Parametern kopiert und im Web-Shop bearbeitet wird, werden die Änderungen nicht nur an der Kopie, sondern auch am Originalartikel vorgenommen. Um dies zu vermeiden, muss serverseitig eine Kopie vom Artikel mit eigener Referenz erzeugt werden.

Um eine serverseitige Kopie zu erstellen, können die JSON-Objekte von einem oder auch mehreren Artikeln an die Schnittstelle übergeben und die Funktion GetProductCopy ausgeführt werden. Danach können mit der Originalreferenz und dem Originalhashwert die Artikelkopien aus der Schnittstelle geholt werden. Mit dem Befehl InitProductList wird die interne Liste geleert.

Um zu überprüfen, ob der Artikel serverseitige Parameter besitzt, muss die Property 'serverSiededParameters' ausgewertet werden.

Bei der Kopieren eines Artikels können folgende Fehler auftreten:

- Internetverbindung ist nicht vorhanden: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'Error offline' zurückgeliefert.
- Der Shop ist nicht erreichbar (*): Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.
- Fehlerhafte Zugangsdaten (*): Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.
- Wartungsarbeiten, Störungen etc. (*): Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.

Die angegebenen Punkte, die mit (*) markiert wurden, wurden noch nicht von allen Web-Shop-Anbietern umgesetzt.

Beispiel C#:

```
string Provider = "Provider";
string Result;

pIfToShop.InitProductList();

string UseCopyFkt1 = JSON(Product1).Get("serverSiededParameters");
string UseCopyFkt2 = JSON(Product2).Get("serverSiededParameters");
if (UseCopyFkt1.ToLower().Contains("true"))
  Result = pIfToShop.AddProduct2List(Product1);
if (!FAILED(Result) && UseCopyFkt2.ToLower().Contains("true"))
  Result = pIfToShop.AddProduct2List(Product2);
if (!FAILED(Result))
{
  Result = pIfToShop.CopyProduct();
  if (!FAILED(Result))
  {
    if (UseCopyFkt1.ToLower().Contains("true"))
    {
      string Reference1 = JSON(Product1).Get("reference");
      string Product1New = pIfToShop.GetProductCopy(Reference1);
      if (!FAILED(Product1New))
      {
        ...
      }
```

```
    }
    if (UseCopyFkt2.ToLower().Contains("true"))
    {
      string Reference2 = JSON(Product2).Get("reference");
      string Product2New = pIfToShop.GetProductCopy(Reference2);
      if (!FAILED(Product2New))
      {
        ...
      }
    }
  }
}
```

## Beispiel C++:

```
HRESULT hr;
BSTR Result = NULL;

BSTR Product1 = _com_util::ConvertStringToBSTR(Produkt1);
BSTR Product2 = _com_util::ConvertStringToBSTR(Produkt2);

hr = pIfToShop->InitProductList();
if (!FAILED(hr))
{
  string UseCopyFkt1 = JSON(Product1).Get("serverSiededParameters");
  string UseCopyFkt2 = JSON(Product2).Get("serverSiededParameters");
  if (toLower(UseCopyFkt1).find("true") != std::string::npos)
    hr = pIfToShop->AddProduct2List(Product1, &Result);
  if (!FAILED(hr) && !FAILED(Result)
  {
    if (toLower(UseCopyFkt2).find("true") != std::string::npos)
      hr = pIfToShop->AddProduct2List(Product2, &Result);
    if (!FAILED(hr) && !FAILED(Result))
    {
      hr = pIfToShop->CopyProduct(&Result);
      if (!FAILED(hr) && !FAILED(Result))
      {
        BSTR ProductNew1, ProductNew2;
        if (toLower(UseCopyFkt1).find("true") != std::string::npos)
        {
          BSTR Reference1 = JSON(Product1).Get("reference").asBSTR();
          hr = pIfToShop->GetProductCopy(Reference1, &ProductNew1);
          if (!FAILED(hr) && !FAILED(ProductNew1))
          {
            ...
          }
        }
        if (toLower(UseCopyFkt2).find("true") != std::string::npos)
        {
          BSTR Reference2 = JSON(Product2).Get("reference").asBSTR();
          hr = pIfToShop->GetProductCopy(Reference2, &ProductNew2);
          if (!FAILED(hr) && !FAILED(ProductNew2))
          {
            ...
          }
        }
      }
    }
  }
}
```

## 1.8. Preisabfrage von Artikeln

Um bei bestehenden Artikeln den aktuellen Preis aus dem Web-Shop abzufragen, kann die Funktion PriceUpdate benutzt werden. Es können die JSON-Objekte von einem oder auch mehreren Artikel an die Schnittstelle übergeben werden. Mit der Funktion priceUpdate werden die aktuellen Preise abgefragt und die JSON-Objekte aktualisiert. Die aktualisierten Artikel können mit der Referenz und dem Hashwert aus der Schnittstelle abgerufen werden. Mit dem Befehl InitProductList wird die interne Liste geleert.

Wurde ein Artikel nicht mehr im Web-Shop gefunden, so wird bei der Funktion GetProduct ein Leerstring zurückgegeben. Dieser Artikel ist nicht mehr im Web-Shop vorhanden.

Bei der Preisabfrage eines Artikels können folgende Fehler auftreten:

- Internetverbindung ist nicht vorhanden: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'Error offline' zurückgeliefert.
- Der Shop ist nicht erreichbar (*): Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.
- Fehlerhafte Zugangsdaten (*): Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.
- Wartungsarbeiten, Störungen etc. (*): Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.

Die angegebenen Punkte, die mit (*) markiert wurden, wurden noch nicht von allen Web-Shop-Anbietern umgesetzt.

Beispiel C#:

```
pIfToShop.InitProductList();
string Result = pIfToShop.AddProduct2List(Product1);
if (!FAILED(Result))
  Result = pIfToShop.AddProduct2List(Product2);
if (!FAILED(Result))
  Result = pIfToShop.AddProduct2List(Product3);
if (!FAILED(Result))
{
  Result = pIfToShop->PriceUpdate();
  if (!FAILED(Result))
  {
    double Preis1, Preis2, Preis3:
    string Reference1 = JSON(Product1).Get("reference");
    string Reference2 = JSON(Product2).Get("reference");
    string Reference3 = JSON(Product3).Get("reference");
    string Hashvalue1 = JSON(Product1).Get("hashvalue");
    string Hashvalue2 = JSON(Product2).Get("hashvalue");
    string Hashvalue3 = JSON(Product3).Get("hashvalue");

    Product1 = pIfToShop.GetProduct(Reference1, Hashvalue1);
    if (!FAILED(Product1))
      Preis1 = JSON(Product1).Get("price").AsDouble();

    Product2 = pIfToShop.GetProduct(Reference2, Hashvalue2);
    if (!FAILED(Product2))
      Preis2 = JSON(Product2).Get("price").AsDouble();

    Product3 = pIfToShop.GetProduct(Reference3, Hashvalue3);
    if (!FAILED(Product3))
      Preis3 = JSON(Product3).Get("price").AsDouble();
  }
}
```

### Beispiel C++:

```
HRESULT hr;
BSTR Result = NULL;

BSTR Product1 = _com_util::ConvertStringToBSTR(Product1);
BSTR Product2 = _com_util::ConvertStringToBSTR(Product2);
BSTR Product3 = _com_util::ConvertStringToBSTR(Product3);

hr = pIfToShop->InitProductList();
if (!FAILED(hr))
  hr = pIfToShop->AddProduct2List(Product1, &Result);
if (!FAILED(hr) && !FAILED(Result))
  hr = pIfToShop->AddProduct2List(Product2, &Result);
if (!FAILED(hr) && !FAILED(Result))
  hr = pIfToShop->AddProduct2List(Product3, &Result);
if (!FAILED(hr) && !FAILED(Result))
{
  hr = pIfToShop->PriceUpdate(&Result);
  if (!FAILED(hr) && !FAILED(Result))
  {
    double Preis1, Preis2, Preis3;
    BSTR Reference1 = JSON(Product1).Get("reference").asBSTR();
    BSTR Reference2 = JSON(Product2).Get("reference").asBSTR();
    BSTR Reference3 = JSON(Product3).Get("reference").asBSTR();
    BSTR Hashvalue1 = JSON(Product1).Get("hashvalue").asBSTR();
    BSTR Hashvalue2 = JSON(Product2).Get("hashvalue").asBSTR();
    BSTR Hashvalue3 = JSON(Product3).Get("hashvalue").asBSTR();

    hr = pIfToShop->GetProduct(Reference1, Hashvalue1, &Product1);
    if (!FAILED(hr) && !FAILED(Product1))
      Preis1 = JSON(Product1).Get("price").AsDouble();

    hr = pIfToShop->GetProduct(Reference2, Hashvalue2, &Product2);
    if (!FAILED(hr) && !FAILED(Product2))
      Preis2 = JSON(Product2).Get("price").AsDouble();

    hr = pIfToShop->GetProduct(Reference3, Hashvalue3, &Product3);
    if (!FAILED(hr) && !FAILED(Product3))
      Preis3 = JSON(Product3).Get("price").AsDouble();
  }
}
```

## 1.9. Übergabe von Artikeln an den Warenkorb

Mit der Funktion Add2CartList lassen sich JSON-Objekte von einem oder auch mehreren Artikeln an die Schnittstelle übergeben. Der Aufbau der JSON-Objekte entspricht dem Aufbau der zuvor mit AddProduct heruntergeladenen Artikel (siehe Kapitel 1.4). Außerdem werden die Mengen übermittelt, die bestellt werden sollen. Die Funktion CartUpdate übermittelt die Artikel an den Warenkorb im Shop und öffnet diesen. Im Warenkorb können nachträglich die Mengen geändert oder Artikel gelöscht werden. Es können auch weitere Artikel in den Warenkorb gelegt werden. Die weiteren Artikel werden aber nicht an die Schnittstelle zurück übertragen, sondern werden separat bestellt.

Mit dem Befehl InitCartList wird die interne Liste geleert.

Nachdem der Anwender den Warenkorb fertig bearbeitet und die Bestellung abgeschlossen hat, wird ein JSON-Objekt mit den Bestelldaten aller Artikel aus der Schnittstelle zurückgegeben. Der Aufbau ist folgender:

```
[
 {
  reference        : 'Eindeutige Referenz auf den Artikel',
  parameterHash    : 'Hash-Wert bei einem Variantenartikel',
  commissionNumber : 'Kommissionsnummer bzw. Kommsissiontext',
  deliveryDate     : 'Lieferdatum',
  valid            : true|false (Artikelverfügbarkeit),
  count            : 'Anzahl im Warenkorb',
  price            : 'Nettopreis für Artikel*Anzahl',
  currency         : 'Währung'
 },
 ...
]
```

Bei der Property 'parameterHash' handelt es sich um eine Hashwert-Berechnung über die Artikel-Parameter. Die Property ist notwendig, falls ein Web-Shop bei Varianten-Artikeln keine eindeutige Referenz zurückgibt. In diesem Fall lässt sich mit der Referenz und dem Hashwert eine eindeutige Zuordnung herstellen (siehe Kapitel 1.4).

Bei einigen Anbietern besteht die Möglichkeit bei der Artikel-Übergabe eine Kommissionsnummer bzw. Kommissionstext mit an den Web-Shop zu übertragen. Die Kommissionsnummer wird im Warenkorb angezeigt und wird später in der Lieferanten-Rechnung mit aufgeführt (siehe Kapitel 1.14).

Eine weitere Eigenschaft, die bei einigen Anbietern im Warenkorb mit angezeigt werden, ist das Lieferdatum. Das Lieferdatum wird für jeden Artikel im Warenkorb angezeigt und kann dort geändert werden.

Um zu prüfen, ob der Anbieter die Übergabe der Kommissionsnummer oder des Lieferdatums unterstützt, kann die Funktion GetProviderFlag aufgerufen werden.

Bei der Übergabe von Artikeln an den Warenkorb können folgende Fehler auftreten bzw. Zustände vom Anwender ausgelöst werden:

- Internetverbindung ist nicht vorhanden: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'Error offline' zurückgeliefert.
- Der Shop ist nicht erreichbar (*): Der Dialog muss vom Anwender, z.B. über 'x', geschlossen werden. Die aufrufende Anwendung bekommt, aus der Schnittstelle, ein 'cancel' zurückgeliefert.
- Fehlerhafte Zugangsdaten (*): Der Shop zeigt im Dialog eine entsprechende Meldung an. Nachdem der Anwender den Dialog geschlossen hat, wird ein 'cancel' zurückgeliefert.

- Wartungsarbeiten, Störungen etc. (*): Der Shop zeigt im Dialog eine entsprechende Meldung an. Nachdem der Anwender den Dialog geschlossen hat, wird ein 'cancel' zurückgeliefert.
- Abbruch durch den Anwender über 'x' oder dem Abbruch-Button im Dialog: Die aufrufende Anwendung bekommt aus der Schnittstelle ein 'closed' bzw. 'cancel' zurückgeliefert.

Die angegebenen Punkte, die mit (*) markiert wurden, wurden noch nicht von allen Web-Shop-Anbietern umgesetzt.

### Beispiel C#:

```
string CommNo1 = "", CommNo2 = "";
string DeliveryDate1 = "", DeliveryDate1 = "";
string Provider = "Provider";

string CommSupport = pIfToShop.GetProviderFlag(Provider, "commissionNumber");
if (!FAILED(CommSupport) && CommSupport == "true")
{
  CommNo1 = KommissionsNr1;
  CommNo2 = KommissionsNr2;
}

string DeliverySupport = pIfToShop.GetProviderFlag(Provider, "deliveryDate");
if (!FAILED(DeliverySupport) && DeliverySupport == "true")
{
  DeliveryDate1 = Lieferdatum1;
  DeliveryDate2 = Lieferdatum2;
}

pIfToShop.InitCartList();
string Result = pIfToShop.Add2CartList(Product1, Count1, CommNo1, DeliveryDate1);
if (!FAILED(Result))
  Result = pIfToShop.Add2CartList(Product2, Count2, CommNo2, DeliveryDate2);
if (!FAILED(Result))
{
  string CartResult = pIfToShop.CartUpdate();
  if (!FAILED(CartResult))
  {
    string Referenz1    = JSON(CartResult).Get("reference", 1);
    string Referenz2    = JSON(CartResult).Get("reference", 2);
    string Hash1        = JSON(CartResult).Get("parameterHash", 1);
    string Hash2        = JSON(CartResult).Get("parameterHash", 2);
    string KommNr1      = JSON(CartResult).Get("commissionNumber", 1);
    string KommNr2      = JSON(CartResult).Get("commissionNumber", 2);
    string Lieferdatum1 = JSON(CartResult).Get("deliveryDate", 1);
    string Lieferdatum2 = JSON(CartResult).Get("deliveryDate", 2);
    double Preis1       = JSON(CartResult).Get("price", 1).AsDouble();
    double Preis2       = JSON(CartResult).Get("price", 2).AsDouble();
    double Menge1       = JSON(CartResult).Get("count", 1).AsDouble();
    double Menge2       = JSON(CartResult).Get("count", 2).AsDouble();
    bool Valid1         = JSON(CartResult).Get("valid", 1).AsBool();
    bool Valid2         = JSON(CartResult).Get("valid", 2).AsBool();
  }
}
```

Beispiel C++:

```
HRESULT hr;
BSTR Result = NULL, CartResult = NULL;
BSTR CommNo1 = NULL, CommNo2 = NULL;
BSTR DeliveryDate1 = NULL, DeliveryDate1 = NULL;
BSTR CommSupport = NULL, DeliverySupport = NULL;
BSTR Provider = _com_util::ConvertStringToBSTR("Provider");

BSTR Product1 = _com_util::ConvertStringToBSTR(Produkt1);
BSTR Product2 = _com_util::ConvertStringToBSTR(Produkt2);
BSTR Count1   = _com_util::ConvertStringToBSTR(Menge1);
BSTR Count2   = _com_util::ConvertStringToBSTR(Menge2);

hr = pIfToShop->GetProviderFlag(Provider, BSTR("commissionNumber"), &CommSupport);
if (!FAILED(hr) && CommSupport.find("true") != std::string::npos))
{
  CommNo1 = _com_util::ConvertStringToBSTR(KommissionsNr1);
  CommNo2 = _com_util::ConvertStringToBSTR(KommissionsNr2);
}

hr = pIfToShop->GetProviderFlag(Provider, BSTR("deliveryDate"), &DeliverySupport);
if (!FAILED(hr) && DeliverySupport.find("true") != std::string::npos))
{
  DeliveryDate1 = _com_util::ConvertStringToBSTR(Lieferdatum1);
  DeliveryDate2 = _com_util::ConvertStringToBSTR(Lieferdatum2);
}

hr = pIfToShop->InitCartList();
if (!FAILED(hr))
  hr = pIfToShop->Add2CartList(Product1, Count1, CommNo1, DeliveryDate1, &Result);
if (!FAILED(hr) && !FAILED(Result))
  hr = pIfToShop->Add2CartList(Product2, Count2, CommNo2, DeliveryDate2, &Result);
if (!FAILED(hr) && !FAILED(Result))
{
  hr = pIfToShop->CartUpdate(&CartResult);
  if (!FAILED(hr) && !FAILED(CartResult))
  {
    std::string Referenz1    = JSON(CartResult).Get("reference", 1);
    std::string Referenz2    = JSON(CartResult).Get("reference", 2);
    std::string Hash1        = JSON(CartResult).Get("parameterHash", 1);
    std::string Hash2        = JSON(CartResult).Get("parameterHash", 2);
    std::string KommNr1      = JSON(CartResult).Get("commissionNumber", 1);
    std::string KommNr2      = JSON(CartResult).Get("commissionNumber", 2);
    std::string Lieferdatum1 = JSON(CartResult).Get("deliveryDate", 1);
    std::string Lieferdatum2 = JSON(CartResult).Get("deliveryDate", 2);
    double Preis1            = JSON(CartResult).Get("price", 1).AsDouble();
    double Preis2            = JSON(CartResult).Get("price", 2).AsDouble();
    double Menge1            = JSON(CartResult).Get("count", 1).AsDouble();
    double Menge2            = JSON(CartResult).Get("count", 2).AsDouble();
    bool Valid1              = JSON(CartResult).Get("valid", 1).AsBool();
    bool Valid2              = JSON(CartResult).Get("valid", 2).AsBool();
 }
}
```

## 1.10. Öffnen der Registrierungsseite eines Lieferanten

Hat ein Nutzer für einen Web-Shop noch keine gültigen Zugangsdaten, so kann mit der Funktion OpenRegistrationPage die Registrierungsseite des Anbieters geöffnet werden. Dort kann sich der Nutzer anmelden und Zugangsdaten erhalten.

Beispiel C#:

```
string Provider = "Provider";

// Öffnen der Registrierungs-Seite
string Result = pIfToShop->OpenRegistrationPage(Provider);
if (FAILED(hr) || Result.ToLower().Contains("error"))
{
   // Fehler aufgetreten
}
```

Beispiel C++:

```
HRESULT hr;
BSTR Result = NULL;
BSTR Provider = _com_util::ConvertStringToBSTR("Provider");

// Öffnen der Registrierungs-Seite
hr = pIfToShop->OpenRegistrationPage(Provider, &Result);
std::string Result = _com_util::ConvertBSTRToString(Result);
if (FAILED(hr) || toLower(Result).find("error") != std::string::npos)
{
   // Fehler aufgetreten
}
```

## 1.11. Anzeigen der Datenschutzrichtlinien und der Lizenzbestimmungen

Eine Applikation, welche die IfToShop-Schnittstelle implementiert, sollte eine Möglichkeit zur Verfügung stellen, um die Datenschutzrichtlinien von IfToShop sowie die Lizenzbestimmungen anzuzeigen.

Mit dem Befehl OpenPrivacyPolicy wird die Seite mit den Datenschutzrichtlinien aufgerufen.

Beispiel C#:

```
// Öffnen der Datenschutzrichtlinien
string Result = pIfToShop->OpenPrivacyPolicy();
if (FAILED(hr) || Result.ToLower().Contains("error"))
{
   // Fehler aufgetreten
}
```

Beispiel C++:

```
HRESULT hr;
BSTR Result = NULL;

// Öffnen der Datenschutzrichtlinien
hr = pIfToShop->OpenPrivacyPolicy(&Result);
std::string Result = _com_util::ConvertBSTRToString(Result);
if (FAILED(hr) || toLower(Result).find("error") != std::string::npos)
{
   // Fehler aufgetreten
}
```

Der Befehl OpenLicenseTerms öffnet die Seite mit den Lizenzbestimmungen.

Beispiel C#:

```
// Öffnen der Lizenzbestimmungen
string Result = pIfToShop->OpenLicenseTerms();
if (FAILED(hr) || Result.ToLower().Contains("error"))
{
   // Fehler aufgetreten
}
```

Beispiel C++:

```
HRESULT hr;
BSTR Result = NULL;

// Öffnen der Lizenzbestimmungen
hr = pIfToShop->OpenLicenseTerms(&Result);
std::string Result = _com_util::ConvertBSTRToString(Result);
if (FAILED(hr) || toLower(Result).find("error") != std::string::npos)
{
   // Fehler aufgetreten
}
```

## 1.12. Erzeugen einer Kontakt-Mail an einen neuen Lieferanten

Sollte dem Nutzer ein Lieferant fehlen, für die er eine Web-Shop-Anbindung wünscht, so kann über den Befehl SendMissingProviderMail eine E-Mail erzeugt werden, die nach der Erzeugung mit dem Standard-Mail-Client geöffnet wird und an den Lieferanten versendet werden kann. Zusätzlich wird die E-Mail an den Betreiber der IfToShop-Schnittstelle geschickt, damit dieser weitere Schritte einleiten kann. Der Hinweis, dass der IfToShop-Betreiber eine Kopie der E-Mail erhält, sollte, aus datenschutztechnischen Gründen, immer mit angegeben werden, wenn die Funktionalität zum Versenden einer Kontakt-Mail in die Applikation implementiert wird.

Als Parameter muss der Ansprechpartner des Empfängers im Format `"Anrede;Vor- und Nachname; E-Mail-Adresse"` angegeben werden. Als Anrede sind die Angaben 'Herr' oder 'Frau' zulässig. Ist das Geschlecht des Ansprechpartners nicht bekannt oder handelt es sich beim Ansprechpartner um eine Person mit diversem Geschlecht, kann die Anrede leer gelassen werden. Die Anrede wird in der E-Mail folgendermaßen umgesetzt: Anrede Herr = Sehr geehrter Herr Vor- und Nachname, Anrede Frau = Sehr geehrte Frau Vor- und Nachname, Anrede Leer = Guten Tag Vor- und Nachname.

Die Parameter des Absenders sollten den Ansprechpartner, die Anschrift und die Kontaktdaten enthalten. Der Parameter für den Ansprechpartner sollte Vor- und Nachname enthalten. Für die Anschrift und die Kontaktdaten können mehrere Zeilen angegeben werden, die durch ein Semikolon getrennt sind.

### Beispiel C#:

```
string EmpfängerAnsprechpartner = "Herr;Max Mustermann;Mustermann@web.de";
string AbsenderAnsprechpartner  = "Erika Musterfrau";
string AbsenderAnschrift        = "Musterfirma GmbH;Musterstraße 1;12345 Musterort";
string AbsenderKontakt          = "Telefon: 0123/456;E-Mail: Musterfirma@web.de";

// Erzeugen der Email und aufrufen des Standard-Mail-Clients
string Result = pIfToShop->SendMissingProviderMail(EmpfängerAnsprechpartner,
                    AbsenderAnsprechpartner, AbsenderAnschrift, AbsenderKontakt);
if (FAILED(hr) || Result.ToLower().Contains("error"))
{
   // Fehler aufgetreten
}
```

### Beispiel C++:

```
HRESULT hr;
BSTR Result = NULL;
BSTR EmpfängerAnsprechpartner = _com_util::ConvertStringToBSTR
                                   ("Herr;Max Mustermann;Mustermann@web.de");
BSTR AbsenderAnsprechpartner  = _com_util::ConvertStringToBSTR
                                   ("Erika Musterfrau");
BSTR AbsenderAnschrift        = _com_util::ConvertStringToBSTR
                                   ("Musterfirma GmbH;Musterstraße 1;12345 Musterort");
BSTR AbsenderKontakt          = _com_util::ConvertStringToBSTR
                                   ("Telefon: 0123/456;E-Mail: Musterfirma@web.de");

// Erzeugen der Email und aufrufen des Standard-Mail-Clients
hr = pIfToShop->SendMissingProviderMail(EmpfängerAnsprechpartner,
                    AbsenderAnsprechpartner, AbsenderAnschrift, AbsenderKontakt, &Result);
std::string Result = _com_util::ConvertBSTRToString(Result);
if (FAILED(hr) || toLower(Result).find("error") != std::string::npos)
{
   // Fehler aufgetreten
}
```

## 1.13. Autoupdate der IfToShop.dll

Um die IfToShop.dll beim Anwender immer aktuell zu halten, lässt sich die IfToShop.dll automatisch, im laufenden Betrieb, aus dem Internet herunterladen. Hierzu muss in der aufrufenden Anwendung der Befehl '…/Shop/if-to-shop.exe -Update' aufgerufen werden. Die if-to-shop.exe lädt die neuste IfToShop.dll herunter, solange die Haupt- bzw. die Nebenversionsnummer mit denen der if-to-shop.exe übereinstimmen. Zu beachten ist, dass das Herunterladen der IfToShop.dll vor der Initialisierung erfolgen muss.

Bei Änderungen an der Schnittstelle, die eine Überarbeitung der aufrufenden Anwendung erfordert, werden Ihnen die neusten Dateien sowie die neusten Schnittstellenbeschreibung, vom Betreiber der If-To-Shop-Schnittstelle, zugesendet.

### Beispiel C#:

```
Process P = new Process();
P.StartInfo.FileName = "c:\\MeineAnwendung\\Shop\\if-to-Shop.exe";
P.StartInfo.Arguments = "-Update";
P.Start();
```

### Beispiel C++:

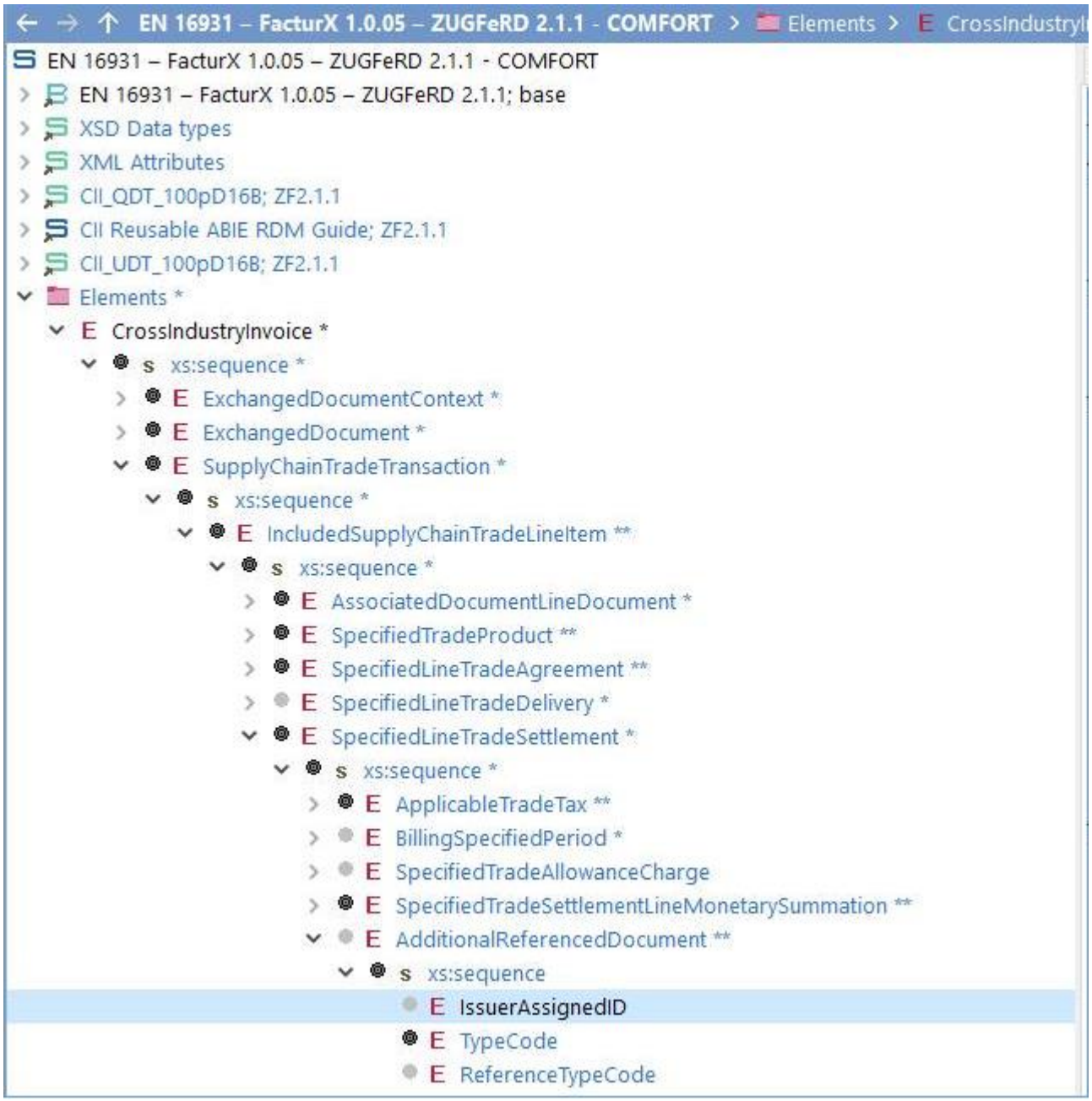
```
char App[1024];
sprintf(App, "%sshop\\if-to-Shop.exe -Update", HomeDir.c_str());

PROCESS_INFORMATION pi;
STARTUPINFO StartupInfo;

memset(&StartupInfo,0,sizeof(StartupInfo));
StartupInfo.cb = sizeof(StartupInfo);
StartupInfo.dwFlags = STARTF_USESHOWWINDOW;
StartupInfo.wShowWindow = SW_HIDE|SW_MINIMIZE;
bool ret = CreateProcessA(
                        NULL,
                        App,
                        NULL,
                        NULL,
                        TRUE,0,
                        (LPVOID)NULL,
                        NULL,
                        &StartupInfo,
                        &pi
                      );
if (pi.hProcess)
{
   DWORD ExitCode = WaitForSingleObject(pi.hProcess, INFINITE);
}
CloseHandle (pi.hProcess);
```

## 1.14. Übertragung der Kommissionsnummer in einer ZUGFeRD-Rechnung

Bei einigen WebShop-Anbietern besteht die Möglichkeit für jeden übertragenen Artikel an den WebShop-Warenkorb eine Kommissionsnummer bzw. Kommissionstext mit anzugeben (siehe Kapitel 1.9).

Diese Kommission wird im Warenkorb mit angezeigt und bei der Rechnungserstellung in der Rechnung mit eingetragen. Falls die Rechnung im ZUGFeRD-Format 2.0 oder höher übertragen wird, wird die Kommission in der XML-Datei der ZUGFeRD-Rechnung mit angegeben.

Die Kommission steht auf Positionsebene der Rechnung im Block '*IncludedSupplyChain-TradeLineItem*' unter '*AdditionalReferencedDocument*'. Dort wird im Tag '*IssuerAssignedID*' die Kommission angegeben. Zur Kennzeichnung, dass es sich um die Kommission handelt, wird im Tag '*ReferenceTypeCode*' der Code 'AAJ' für Lieferauftragsnummer eingetragen.



# 2. Anhang

## 2.1. Liste der Funktionen

***HRESULT** **AddProduct(**BSTR** &Product)***

Funktion:

Die Funktion AddProduct startet einen Web-Shop-Aufruf, um einen Artikel aus dem Web-Shop zu übernehmen. Der übernommene Artikel wird als JSON-Objekt zurückgegeben.

Parameter:

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Product: Rückgabe des ausgewählten Artikels als JSON-Objekt

***HRESULT** **AddProduct2List(**BSTR** Product, **BSTR** &Result)***

Funktion:

Mit der Funktion AddProduct2List lassen sich ein oder mehrere Artikel an die Schnittstelle übergeben, um diese mit der Funktion CopyProduct serverseitig zu kopieren oder mit der Funktion PriceUpdate die aktuellen Preise aus dem Web-Shop zu holen.

Parameter:

Product: Artikel, welcher in der Schnittstelle bearbeitet werden soll

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Result: Fehlercode, falls die Übergabe an die Schnittstelle nicht erfolgreich war

***HRESULT** **Add2CartList(**BSTR** Product, **double** Count, **BSTR** CommissionNo, **BSTR** Delivery, **BSTR** &Result)***

Funktion:

Mit der Funktion Add2CartList lassen sich ein oder mehrere Artikel an die Schnittstelle übergeben, um diese mit der Funktion cartUpdate an den Warenkorb im Web-Shop zu übermitteln.

Parameter:

Product: Artikel, der in den Warenkorb gelegt werden soll

Count: Bestellmenge

CommissionNo: Kommissionsnummer, die im warenkorb mit angezeigt werden soll

Delivery: Lieferdatum, an welchem der Artikel geliefert werden soll

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Result: Fehlercode, falls die Übergabe an die Schnittstelle nicht erfolgreich war

***HRESULT** **CartUpdate(**BSTR** &OrderData)***

Funktion:

Die Funktion CartUpdate übermittelt die mit Add2CartList übergebenen Artikel an den Warenkorb im Web-Shop. Nach dem Abschluß der Bestellung, werden die Bestelldaten als JSON-Objekt zurückgegeben.

Parameter:

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

OrderData: Rückgabe der Bestelldaten im JSON-Format

***HRESULT CopyProduct(BSTR &Result)***

Funktion:

Die Funktion CopyProduct kopiert serverseitig die mit AddProduct2List übergebenen Artikel im Web-Shop. Mit der Funktion GetProductCopy können die kopierten Artikel aus der Schnittstelle abgerufen werden.

Parameter:

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Result: Fehlercode, falls der Aufruf der Funktion fehlerhaft war

***HRESULT EditProduct(BSTR Product, BSTR &Product2)***

Funktion:

Die Funktion EditProduct übergibt einen zuvor ausgewählten Artikel als JSON-Objekt an den Web-Shop. Der übergebene Artikel wird im Web-Shop angezeigt. Dort kann er geändert oder ein anderer Artikel ausgewählt werden.

Parameter:

Product: Artikel, welcher im Web-Shop angezeigt werden soll

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Product2: Rückgabe des geänderten oder neu ausgewählten Artikels als JSON-Objekt. Enthält das JSON-Objekt von Product, falls keine Änderungen vorgenommen wurden

***HRESULT GetProduct(BSTR Reference, BSTR Hashvalue, BSTR &Product)***

Funktion:

Mit der Funktion GetProduct können Artikel aus der Schnittstelle abgerufen werden, deren Preise mit der Funktion PriceUpdate aktualisiert wurden.

Parameter:

Reference: Referenz des Original-Artikels

Hashvalue: Hashwert des Original-Artikels

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Product: Rückgabe des aktualisierten Artikels als JSON-Objekt.

***HRESULT GetProductCopy(BSTR Reference, BSTR &Product)***

Funktion:

Mit der Funktion GetProductCopy lassen sich die mit CopyProduct erzeugten serverseitigen Artikel-Kopien aus der Schnittstelle abrufen.

Parameter:

Reference: Referenz des Original-Artikels

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Product: Rückgabe des kopierten Artikels als JSON-Objekt.

***HRESULT GetProductGraphic(BSTR Format, BSTR CommandLine, BSTR &Graphic)***

Funktion:

Die Funktion GetProductGraphic konvertiert die vom Web-Shop übertragene Grafik-Datei bzw. eingebettete Grafik in ein gewünschtes Grafikformat. Optional kann die Grafik auch mit einer Kommandozeile erzeugt werden.

Parameter:

Format: Konvertierungsformat der Artikel-Grafik ('BMP', 'JPG', 'PNG')

Commandline: Optionale Kommandozeile bei abweichender Konvertierung oder Leerstring

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Graphic: Pfad und Dateiname der konvertierten Grafik

***BSTR GetProvider()***

Funktion:

Mit der Funktion GetProvider lassen sich alle unterstützen Lieferanten sowie deren Versionsnummern ermitteln.

Parameter:

Rückgabewerte:

BSTR: Rückgabe der Lieferantennamen inkl. Versionsnummer als JSON-Objekt.

***HRESULT GetProviderData(BSTR Provider, BSTR &Provider)***

Funktion:

Mit der Funktion GetProviderData lassen sich die Daten eines Lieferanten aus der Schnittstelle ermitteln.

Parameter:

Provider: Eindeutiger Lieferantenname

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Provider: Rückgabe der Lieferantendaten als JSON-Objekt. Fehlercode, falls Lieferantendaten nicht ermittelt werden konnten.

***HRESULT GetProviderDescription(BSTR Provider, BSTR &Graphic)***

Funktion:

Mit der Funktion GetProviderDescription lässt sich die Beschreibungs-Grafik des Lieferanten abrufen. Die Grafik wird im temporären Grafikpfad zur Verfügung gestellt.

Parameter:

Provider: Eindeutiger Lieferantenname

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Graphic: Pfad und Dateiname der abgerufenen Grafik. Fehlercode, falls Grafik nicht ermittelt werden konnte.

***HRESULT GetProviderLogo(BSTR Provider, BSTR &Graphic)***

Funktion:

Mit der Funktion GetProviderLogo lässt sich das Logo des Lieferanten abrufen. Die Grafik wird im

temporären Grafikpfad zur Verfügung gestellt.

Parameter:

Provider: Eindeutiger Lieferantenname

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Graphic: Pfad und Dateiname der abgerufenen Grafik. Fehlercode, falls Grafik nicht ermittelt werden konnte.

***HRESULT GetProviderLabels(BSTR Provider, BSTR &Labels)***

Funktion:

Mit der Funktion GetProviderLabels können die Bezeichnungen der Login-Felder aus der Schnittstelle abgerufen werden. Die Rückgabe ist ein JSON-Objekt. Properties mit Leerstring bzw. null werden vom Lieferanten nicht unterstützt.

Parameter:

Provider: Eindeutiger Lieferantenname

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Labels: Labels der Login-Felder als JSON-Objekt. Fehlercode, falls Labels nicht ermittelt werden konnten.

***HRESULT GetProviderFlag(BSTR Provider, BSTR FlagName, BSTR &Result)***

Funktion:

Mit der Funktion GetProviderFlag lassen sich Angaben zum jeweiligen Lieferanten ermitteln. So kann z.B. mit der Abfrage GetProviderFlag(Provider, “commission“, &Result) ermittelt werden, ob der Lieferant die Kommissionsnummern-Übergabe unterstützt.

Parameter:

Provider: Eindeutiger Lieferantenname

FlagName: Flag, welches abgefragt werden soll. Mögliche FlagNamen zur Zeit sind:

'active' : Angabe, ob der Anbieter noch aktiv ist

'demo' : Angabe, ob es sich um einen Demo-Webshop handelt

'prefix' : Präfix für die Artikelnummern-Generierung

'gtinNumber' : Angabe, ob der Webshop die GTIN-Abfrage unterstützt

'commissionNumber' : Angabe, ob der Webshop die Übergabe einer Kommissionsnummer unterstützt

'deliveryDate' : Angabe, ob der Webshop die Übergabe des Lieferdatums unterstützt

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Result: Wert des abgefragten Flags (z.B. 'true', 'false')

***HRESULT GraphicDir(BSTR Path)***

Funktion:

Die Funktion GraphicDir setzt den Pfad, in dem konvertierte oder abgerufene Dateien gespeichert werden.

Parameter:

Path: Pfadangabe zur Speicherung von Grafikdateien

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

***HRESULT Init(BSTR Provider, BSTR User, BSTR Pswd, BSTR Customer, BSTR &Result)***

Funktion:

Die Funktion Init startet den Anmeldevorgang an den Web-Shop des Lieferanten.

Parameter:

Provider: Eindeutiger Lieferantenname

User: Nutzername für Login

Pswd: Passwort für Login

Customer: Kundenummer für Login

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Result: Fehlercode, falls init nicht fehlerfrei ausgeführt wurde

***HRESULT InitCartList()***

Funktion:

Die Funktion InitCartList initialisiert die Warenkorb-Liste in der Schnittstelle und löscht alle Artikel, die sich noch in der Liste befinden.

Parameter:

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

***HRESULT InitProductList()***

Funktion:

Die Funktion InitProductList initialisiert die Artikel-Liste in der Schnittstelle und löscht alle Artikel, die sich noch in der Liste befinden.

Parameter:

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

***HRESULT LogFile(BSTR LogFile)***

Funktion:

Mit der Funktion LogFile wird der Dateiname übergeben, in dem Protokolleinträge eingetragen werden sollen.

Parameter:

LogFile: Datei inkl. Pfadangabe, in dem Protokolleinträge eingetragen werden

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

***HRESULT logLevel(BSTR LogLevel)***

Funktion:

Mit der Funktion LogLevel kann die Protokollierungstiefe angegeben werden.

Parameter:

LogLevel: Angabe, welche die Tiefe der Protokollierung angibt

1 = nur Fehler

2 = Fehler und Informationen

3 = Fehler, Informationen und Funktionsaufrufe

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

***HRESULT OpenPrivacyPolicy(BSTR &Result )***

Funktion:

Mit der Funktion OpenPrivacyPolicy wird die Datenschutzrichtlinien von Chromium angezeigt.

Parameter:

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Result: Fehlercode, falls Aufruf der Funktion nicht erfolgreich war

***HRESULT OpenRegistrationPage(BSTR Name, BSTR &Result )***

Funktion:

Mit der Funktion OpenRegistrationPage wird die Registrierungsseite des Lieferanten aufgerufen.

Parameter:

Name: Name des Lieferanten

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Result: Fehlercode, falls Aufruf der Funktion nicht erfolgreich war

***HRESULT PriceUpdate(BSTR &Result)***

Funktion:

Die Funktion PriceUpdate startet die Preisabfrage für alle Artikel, die mit der Funktion AddProduct2List an die Schnittstelle übergeben wurden.

Parameter:

Rückgabewerte:

HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)

Result: Fehlercode, falls Preis-Update nicht erfolgreich war

***HRESULT SendMissingProviderMail(BSTR EmpfängerAnsprechpartner, BSTR AbsenderAnsprechpartner, BSTR AbsenderAnschrift,BSTR AbsenderKontakt, BSTR &Result)***

Funktion:

Mit der Funktion SendMissingProviderMail kann eine E-Mail erzeugt werden, die mit dem Standard-E-Mail-Programm geöffnet wird.

Parameter:

EmpfängerAnsprechpartner: Ansprechpartner des Empfängers

Format: 'Anrede;Vor-und Nachname; Email-Adresse'

AbsenderAnsprechpartner: Ansprechpartner des Absenders

Format: 'Vor-und Nachname'

AbsenderAnschrift: Anschrift des Absenders

```
  Format: 'Anschrift 1;Anschrift 2;Anschrift 3;Anschrift 4;Anschrift 5'
AbsenderKontakt: Kontaktdaten des Absenders
  Format: 'Kontakt 1;Kontakt 2;Kontakt 3;Kontakt 4;Kontakt 5'
Rückgabewerte:
HRESULT: Return-Wert, ob die Funktion erfolgreich ausgeführt wurde (0 = ok; < 0 = failed)
Result: Fehlercode, falls Aufruf der Funktion nicht erfolgreich war
```

## 2.3. Änderungshistorie

| |
|---|
| **Änderungen zu Version 1.1:**<br>• Kapitel 'Initialisierung der Schnittstelle' erweitert um die Beschreibung des Testzugangs.<br>• Download-Links im Kapitel 'Initialisierung der Schnittstelle' ersetzt. |
| |
| |
| |
| |
| |
| |
| |
| |
| |