Programmierer-Referenz

List & Label® 20

100% Reporting Power

Die in diesem Handbuch enthaltenen Angaben sind ohne Gewähr und können ohne weitere Mitteilung geändert werden. Die combit GmbH geht hiermit keinerlei Verpflichtungen ein. Die Verfügbarkeit mancher in dieser Anleitung beschriebener Funktionen (bzw. die Vorgehensweise, um darauf zuzugreifen), ist von Version, Release-Stand, eingespielten Servicepacks u.ä. Ihres Systems (z.B. Betriebssystem, Textverarbeitung, Mailprogramm, etc.) sowie seiner Konfiguration abhängig.

Die in diesem Handbuch beschriebene Software wird auf Basis eines Lizenzvertrages geliefert. Der Lizenzvertrag befindet sich bei der Verpackung der CD, bzw. für die Download-Version im Internet unter www.combit.net und wird auch durch das Installationsprogramm angezeigt.

Dieses Handbuch oder Ausschnitte aus diesem Handbuch dürfen ohne schriftliche Genehmigung der combit GmbH nicht kopiert oder in irgendeiner anderen (z.B. digitaler) Form vervielfältigt werden.

Die JPEG-Codierung und -Decodierung wird mit Hilfe der JPEG Library der IJG (Independent JPEG Group) durchgeführt.

Avery and all Avery brands, product names and codes are trademarks of Avery Dennison Corporation.

PDF creation utilizes wPDF3 (c) wpCubed GmbH - www.pdfcontrol.com

DataMatrix and QRCode generation is done using RDataMatrix and QRCode (c) J4L Components

Aztec Barcode creation utilizes free code from Hand Held Inc.

Nicht alle beschriebenen Features sind in allen Editionen verfügbar. Beachten Sie in diesem Zusammenhang die Hinweise zu LL_ERR_LICENSEVIOLATION.

Copyright © combit GmbH 1992-2014; Rev. 20.000
http://www.combit.net
Alle Rechte vorbehalten.

Inhaltsverzeichnis

1. Einführung 9
1.1 Vor der Installation 9
1.1.1 Systemvoraussetzung 9
1.1.2 Lizenzierung 9
1.2 Nach der Installation 9
1.2.1 Startmenü 9
1.2.2 Schneller Designer-Einstieg per Beispiel-Anwendung 10
1.2.3 Programmierbeispiele 11
1.2.4 Dokumentation 11
1.3 Wichtige Konzepte 11
1.3.1 Prinzipieller Aufbau 11
1.3.2 Die verschiedenen Projekttypen 13
1.3.3 Variablen und Felder 13
1.3.4 Verfügbare Sprachen für die Benutzeroberfläche 13
1.4 Einstieg in die Programmierung 14
1.4.1 Überblick 14
1.4.2 Einbindung in .NET 14
1.4.3 Einbindung in Visual Basic 14
1.4.4 Einbindung in Delphi 14
1.4.5 Einbindung in C++Builder 15
1.4.6 Einbindung in C/C++ 15
1.4.7 Einbindung in Java 15
1.4.8 Einbindung in andere Programmiersprachen 15
1.4.9 Hinweise zu Variablen- und Feldnamen 15
1.4.10 Debugging Unterstützung 16

2. Programmierung mit .NET 17
2.1 Einleitung 17
2.1.1 Integration in Visual Studio 17
2.1.2 Komponenten 18
2.2 Erste Schritte 19
2.2.1 List & Label integrieren 19
2.2.2 Komponente lizenzieren 20
2.2.3 Datenquelle anbinden 21
2.2.4 Design 21
2.2.5 Druck 22
2.2.6 Export 23
2.2.7 Wichtige Eigenschaften der Komponente 24
2.2.8 Webreporting 25
2.3 Weitere wichtige Konzepte 26
2.3.1 Datenprovider 26
2.3.2 Variablen, Felder und Datentypen 32
2.3.3 Ereignisse 35

2.3.4 Projekttypen ... 36
2.3.5 Verschiedene Drucker und Kopiendruck ... 37
2.3.6 Designer anpassen und erweitern ... 38
2.3.7 Objekte im Designer ... 39
2.3.8 Berichtscontainer ... 41
2.3.9 Objektmodell (DOM) ... 42
2.3.10 List & Label in WPF Applikationen ... 43
2.3.11 Fehlerhandling mit Exceptions ... 43
2.3.12 Debugging ... 44
2.4 Beispiele ... 45
2.4.1 Einfaches Etikett ... 45
2.4.2 Einfache Liste ... 46
2.4.3 Sammelrechnung ... 47
2.4.4 Karteikarte mit einfachen Platzhaltern drucken ... 47
2.4.5 Unterberichte ... 49
2.4.6 Charts ... 49
2.4.7 Kreuztabellen ... 49
2.4.8 Datenbankunabhängige Inhalte ... 49
2.4.9 Export ... 52
2.4.10 Designer um eigene Funktion erweitern ... 54
2.4.11 Vorschaudateien zusammenfügen und konvertieren ... 55
2.4.12 eMail-Versand ... 56
2.4.13 Projektdateien in Datenbank halten ... 57
2.4.14 Druck im Netzwerk ... 58

3. Programmieren mit der OCX Komponente ... 60
3.1 Einbindung der Komponente ... 60
3.2 Einfache Print- und Design-Methoden ... 60
3.2.1 Funktionsweise ... 60
3.2.2 Verwendung des UserData-Parameters ... 61
3.3 Übergabe von ungebundenen Variablen und Feldern ... 61
3.3.1 Bilder ... 62
3.3.2 Barcodes ... 62
3.4 Auswahl der Sprache ... 62
3.5 Arbeiten mit Ereignissen ... 62
3.6 Anzeigen einer Vorschaudatei ... 63
3.7 Arbeiten mit Vorschau-Dateien ... 63
3.7.1 Öffnen einer Vorschaudatei ... 63
3.7.2 Zusammenführen mehrerer Vorschaudateien ... 63
3.7.3 Debugging ... 64
3.8 Erweiterung des Designers ... 64
3.8.1 Eigene Funktionen dem Formelassistent hinzufügen ... 64
3.8.2 Eigene Objekte dem Designer hinzufügen ... 66
3.9 Das Viewer-OCX-Control ... 67
3.9.1 Übersicht ... 67
3.9.2 Registrierung ... 67

3.9.3 Eigenschaften ... 67
3.9.4 Methoden ... 69
3.9.5 Ereignisse ... 70
3.9.6 Visual C++ Hinweis ... 71
3.9.7 Verpacken in CAB-Files ... 71
3.9.8 Einbau in Ihre Internet-Seite ... 71
3.10 Das Designer-OCX-Control ... 71
3.10.1 Übersicht ... 71
3.10.2 Registrierung ... 72
3.10.3 Eigenschaften ... 72
3.10.4 Methoden ... 73
3.10.5 Verpacken in CAB-Files ... 73
3.10.6 Einbau in Ihre Internet-Seite ... 73

4. Programmieren mit der VCL Komponente ... 74
4.1 Einbindung der Komponente ... 74
4.2 Datenbindung ... 74
4.2.1 Bindung von List & Label an eine Datenquelle ... 74
4.2.2 Arbeiten mit Master-Detail-Datensätzen ... 75
4.2.3 Weitere Möglichkeiten der Datenbindung ... 76
4.3 Einfache Print- und Design-Methoden ... 77
4.3.1 Funktionsweise ... 77
4.3.2 Verwendung des UserData-Parameters ... 78
4.4 Übergabe von ungebundenen Variablen und Feldern ... 78
4.4.1 Bilder ... 78
4.4.2 Barcodes ... 79
4.5 Auswahl der Sprache ... 79
4.6 Arbeiten mit Ereignissen ... 79
4.7 Anzeigen einer Vorschaudatei ... 79
4.8 Arbeiten mit Vorschau-Dateien ... 79
4.8.1 Öffnen einer Vorschaudatei ... 80
4.8.2 Zusammenführen mehrerer Vorschaudateien ... 80
4.8.3 Debugging ... 80
4.9 Erweiterung des Designers ... 80
4.9.1 Eigene Funktionen dem Formelassistent hinzufügen ... 81
4.9.2 Eigene Objekte dem Designer hinzufügen ... 82

5. Programmierung per API ... 84
5.1 Programmierschnittstelle ... 84
5.1.1 Dynamic Link Libraries ... 84
5.1.2 Allgemeines zum Rückgabewert ... 87
5.2 Programmiergrundlagen ... 87
5.2.1 Datenbankunabhängiges Konzept ... 87
5.2.2 Der List & Label Job ... 87
5.2.3 Variablen, Felder und Datentypen ... 88
5.3 Aufruf des Designers ... 93

5.3.1 Grundschema 93
5.3.2 Erläuterung 94
5.4 Der Druckvorgang 96
5.4.1 Die Datenversorgung 96
5.4.2 Echtdatenvorschau oder Druck? 96
5.4.3 Grundlegender Ablauf 96
5.4.4 Erläuterungen 100
5.5 Drucken relationaler Daten 102
5.5.1 Benötigte API-Funktionen 103
5.5.2 Aufruf des Designers 103
5.5.3 Steuerung des Druckvorgangs 105
5.5.4 Umgang mit 1:1-Relationen 111
5.6 Callbacks und Notifications 112
5.6.1 Überblick 113
5.6.2 User-Objekte 113
5.6.3 Definition einer Callback-Routine 114
5.6.4 Datenübergabe an die Callback-Routine 115
5.6.5 Datenübergabe per Nachricht 115
5.6.6 Weitere Hinweise 116
5.7 Fortgeschrittene Programmierung 117
5.7.1 Direkter Druck und Export aus dem Designer 117
5.7.2 Drilldown-Berichte in der Vorschau 120
5.7.3 Unterstützung des Berichtsparameter-Auswahlbereichs in der Vorschau 125
5.7.4 Unterstützung von ausklappbaren Bereichen in der Vorschau 126
5.7.5 Unterstützung von interaktiver Sortierung in der Vorschau 126
5.7.6 Ansteuerung von Chart- und Kreuztabellen-Objekten 127
5.8 Verwendung der DOM-API (ab Professional Edition) 129
5.8.1 Grundlagen 129
5.8.2 Beispiele 133

6. API Referenz 137
6.1 Referenz der Funktionen 137
6.2 Referenz der Callback-Notifications 282
6.3 Verwaltung der Preview-Dateien 308
6.3.1 Überblick 308
6.3.2 Zugriffsfunktionen 308

7. Die Export-Module 334
7.1 Übersicht 334
7.2 Programmierschnittstelle 334
7.2.1 Export-Module global (de)aktivieren 334
7.2.2 Einzelne Export-Module ein- und ausschalten 335
7.2.3 Ausgabemedium festlegen/abfragen 335
7.2.4 Export-spezifische Optionen setzen 337
7.2.5 Export ohne Benutzerinteraktion durchführen 337
7.2.6 Export-Ergebnis abfragen 337

7.3 Programmierreferenz ... 338
7.3.1 Excel Export-Modul ... 338
7.3.2 Grafik Export-Modul ... 345
7.3.3 HTML Export-Modul ... 347
7.3.4 JQM Export-Modul ... 358
7.3.5 MHTML Export-Modul ... 361
7.3.6 PDF Export-Modul ... 362
7.3.7 Powerpoint Export-Modul ... 366
7.3.8 RTF Export-Modul ... 370
7.3.9 SVG Export-Modul ... 375
7.3.10 Text (CSV) Export-Modul ... 381
7.3.11 Text (Layout) Export-Modul ... 383
7.3.12 TTY Export-Modul ... 385
7.3.13 Windows Fax Export-Modul ... 387
7.3.14 Word Export-Modul ... 388
7.3.15 XHTML/CSS Export-Modul ... 392
7.3.16 XML Export-Modul ... 399
7.3.17 XPS Export-Modul ... 404
7.4 Exportdateien digital signieren ... 405
7.4.1 Übersicht ... 405
7.4.2 Signaturvorgang starten ... 406
7.4.3 Programmierschnittstelle ... 406
7.5 Exportdateien per eMail verschicken ... 408
7.5.1 Übersicht ... 408
7.5.2 eMail Parameter per Programm setzen ... 409
7.6 Exportdateien in ZIP-Archiv komprimieren ... 413

8. Sonstiges zur Programmierung ... 415
8.1 Übergabe von NULL-Werten ... 415
8.2 Rundung ... 415
8.3 Geschwindigkeitsoptimierung ... 415
8.4 Projekt-Parameter ... 416
8.4.1 Parametertypen ... 416
8.4.2 Parameterabfrage während des Druckvorgangs ... 417
8.4.3 Vordefinierte Projektparameter ... 417
8.4.4 Automatische Formularspeicherung ... 419
8.5 Webreporting ... 420
8.6 Hinweise zur Verwendung in mehreren Threads (Multithreading) ... 421

9. Fehlercodes ... 422
9.1 Allgemeine Fehlercodes ... 422
9.2 Zusätzliche Fehlercodes der Storage-API ... 426

10. Fehlersuche mit Debwin ... 427

11. Redistribution Ihrer Anwendung ... 428
11.1 Systemvoraussetzung ... 428

11.2 64 Bit Module 428
11.3 Die eigenständige Viewer-Applikation 428
11.3.1 Aufgabe 428
11.3.2 Kommandozeilenoptionen 428
11.3.3 Registrierung 429
11.3.4 Benötigte Dateien 429
11.4 Die List & Label Dateien 429
11.5 Sonstige Konfigurationseinstellungen 431

12. Update-Hinweise für Version 20 432
12.1 Überblick über Änderungen 432
12.1.1 Allgemein/API 432
12.1.2 Neue Features 432
12.2 Umstellung auf List & Label 20 433
12.2.1 Allgemein 433
12.2.2 Umstellung von .NET-Projekten 434
12.2.3 Erweiterungen gegenüber der Vorgängerversion 434
12.2.4 Änderungen gegenüber der Vorgängerversion 434
12.2.5 Umstellung von Projekten mit OCX (z.B. Visual Basic) 434
12.2.6 Umstellung von Projekten mit VCL (z.B. Delphi) 435
12.2.7 Umstellung bei API-Programmierung (z.B. C/C++) 435

13. Hilfe und Support 436

14. Index 437

1. Einführung

Herzlichen Glückwunsch zum Kauf von List & Label. Mit List & Label haben Sie eine leistungsstarke Entwicklerkomponente für Report, Listen-, Etiketten,- Formular-, Chart-, Barcode und Messinstrumente-Druck erworben.

Es handelt sich bei List & Label also nicht um eine eigenständig lauffähige Applikation, sondern um eine Komponente, die nahtlos in Ihr Anwendungsprogramm integriert wird.

Sie werden dabei mit wenigen Zeilen Programm-Code Ihrem Programm Druckfähigkeiten geben, die über ein attraktives Design verfügen und professionellen Ansprüchen genügen.

1.1 Vor der Installation

1.1.1 Systemvoraussetzung

List & Label ist lauffähig unter Windows XP, Windows 2003, Windows Vista, Windows Server 2008, Windows 7 und Windows 8.

Sie benötigen zudem einen beliebigen installierten Windows-Druckertreiber mit einer empfohlenen Auflösung von mindestens 300 dpi, selbst wenn nicht gedruckt wird.

Die Verfügbarkeit mancher beschriebener Funktionen (bzw. die Vorgehensweise, um darauf zuzugreifen), ist von Version, Release-Stand, eingespielten Servicepacks u.ä. Ihres Systems (z.B. Betriebssystem) sowie seiner Konfiguration abhängig. Einige Funktionalitäten stehen ggf. nicht in allen Betriebssystemen zur Verfügung. Die Einschränkungen finden Sie ggfs. an der entsprechenden Stelle erwähnt.

Wenn Sie den Word (DOCX)-Export verwenden möchten, benötigen Sie auf dem Entwicklungs- sowie auf dem Endkundenrechner das .NET Framework 3.5.

1.1.2 Lizenzierung

List & Label gibt es in verschiedenen Editionen, die einen unterschiedlichen Funktions- und Lizenzumfang enthalten. Alle ausführlichen Details hierzu finden Sie unter www.combit.net/reporting/lizenz-faq/.

1.2 Nach der Installation

1.2.1 Startmenü

Nach der Installation von List & Label finden Sie im Windows Startmenü die Programmgruppe combit List & Label 20. Mit Hilfe dieser Programmgruppe gelangen Sie zu allen wichtigen Informationen bezüglich Einbindung, Dokumentationen und Beispielen. Diese Gruppe wird Ausgangspunkt für die folgenden Kapitel sein.

1.2.2 Schneller Designer-Einstieg per Beispiel-Anwendung

Die List & Label Beispielanwendung bietet eine schnelle Möglichkeit sich mit dem Designer und seinen Möglichkeiten vertraut zu machen. Dabei handelt es sich um eine reine Beispielanwendung, die in sich abgeschlossen ist, und diverse Möglichkeiten der Druckausgabe mit List & Label anhand einer festen Datenbank demonstriert.

Sie finden die Beispielanwendung in der Startmenügruppe. Mit Hilfe der Anwendung können Sie sofort den List & Label Designer starten und sich durch eine Vielzahl vorgefertigter Layout-Beispiele einen Überblick über die Funktionalität und Flexibilität verschaffen. Der Designer wird über den Menüpunkt *Design* und Anwählen eines Eintrages - z.B. Rechnung - gestartet. Vor dem eigentlichen Start müssen Sie nur noch eine vorhandene Druckvorlage in dem Dateiauswahldialog auswählen – oder aber einen neuen Dateinamen eingeben. Nun steht Ihnen die volle Funktionalität des List & Label Designers – im Rahmen des Beispielprogrammes – zur Verfügung.

Zusätzlich erlaubt es die List & Label Beispielanwendung, die vorhandenen oder auch neu erstellten Druckvorlagen mit Beispieldaten zu drucken oder aber eines der Exportformate für die Ausgabe zu verwenden. Wählen Sie im Menü *Druck* einen der Einträge. Im darauf folgenden Druckoptionsdialog können Sie das Ausgabeziel bzw. Exportformat wählen.

1.2.3 Programmierbeispiele

Um eine schnelle Einarbeitung in das Konzept von List & Label zu gewährleisten, wird mit der Installation eine Vielzahl von Programmierbeispielen mitgeliefert. Diese finden Sie in der Startmenügruppe unterhalb des Eintrages Beispiele.

Je nach installierter Entwicklungsumgebung finden Sie in den Verzeichnissen viele verschiedene Programmierbeispiele.

Weitere Informationen zu den einzelnen Beispielen sowie Erläuterungen zu den verwendeten Methoden und Komponenten finden Sie im List & Label Startcenter, das direkt nach der Installation gestartet wird oder über die List & Label Programmgruppe erreichbar ist.

1.2.4 Dokumentation

Unterhalb der Menügruppe Dokumentationen finden Sie alle verfügbaren Dokumentationen.

Diese beinhalten die Programmier-Referenz sowie das Designerhandbuch als PDF Dokument. Zusätzlich finden Sie dort auch verschiedene Onlinehilfen, z.B. zum Designer oder den List & Label Komponenten (.NET, VCL, OCX) sowie weitere Informationen zu Redistribution, Webreporting, Debug usw.

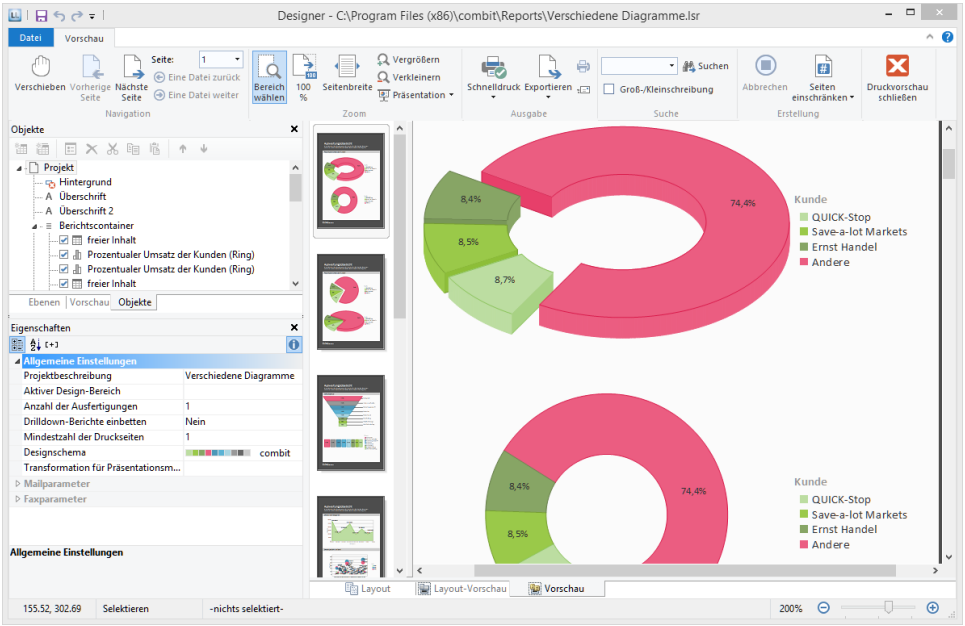
1.3 Wichtige Konzepte

1.3.1 Prinzipieller Aufbau

Bei List & Label handelt es sich nicht um ein eigenständiges Anwendungsprogramm, sondern um eine Komponente, die in Ihr Anwendungsprogramm integriert wird. Mit wenigen Zeilen Programm-Code erweitern Sie damit Ihre Anwendung um Ausgaben und Auswertungen jeglicher Art: Reports, Berichte, Subreports, Listen, Multitabellen, Kreuztabellen, Diagramme, Charts, Messinstrumente, Formulare, Etiketten, Druck, Vorschau, Export und Webreporting.

Berichtsvorlagen im Designer gestalten

Die Designer-Funktionalität zur interaktiven visuellen Gestaltung von Berichten, Druckvorlagen, Auswertungen etc. ist integrierter Bestandteil der List & Label Komponente und wird somit Bestandteil Ihrer Anwendung. Es handelt sich also nicht um eine eigenständige Applikation, sondern der Designer wird von Ihrer Anwendung heraus per Programm-Code gestartet, typischerweise als Reaktion auf einen entsprechenden Menü-Befehl. Der Designer präsentiert sich dann in einem modalen Pop-up Fenster, welches Ihr Anwendungsfenster überlagert.



Diese Designer-Funktionalität können Sie also direkt Ihren Endkunden anbieten, so dass dieser seine eigenen Projekte definieren oder die von Ihnen mitgegebenen Projekte seinen Bedürfnissen anpassen kann.

Drucken oder Exportieren: Erzeugen von Berichten

Um Berichte in der vom Benutzer oder von Ihnen definierten Form auf dem Drucker auszugeben oder in einer Seitenansicht (sog. Druck-Vorschau) darzustellen, werden die auszugebenden Daten an List & Label übergeben. Dies kann je nach Programmiersprache automatisch hinter den Kulissen durch List & Label per Zugriff auf entsprechende Daten-Provider geschehen, oder aber auch explizit durch Ihren Programm-Code, zum Beispiel wenn Ihre Daten gar nicht in einer Datenbank liegen. Auch ein Mix zwischen Datenbank-Daten und Programm-spezifischen Daten ist möglich.

Neben der Ausgabe auf den Drucker oder als Vorschau bietet List & Label weitere Zielformate wie zum Beispiel PDF, HTML, JQM, XML, RTF, XLS, DOCX, TIFF, JPEG, PNG, Bitmap, Text und andere, um das Druckergebnis weiterzuverwenden. Die Ausgabe in diese Zielformate geschieht über spezielle Export-Module. Der eigentliche Export in das jeweilige Format läuft aus Entwicklersicht analog zum sonstigen Druck.

Berichte anzeigen

Die Druck-Vorschau kann zum einen automatisch als Ergebnis des Druckvorgangs durch List & Label angezeigt werden. Zum anderen kann sie aber auch durch den Entwickler über eine separate Komponente in eigene Fenster, Dialoge oder Internet-Seiten eingebettet werden.

1.3.2 Die verschiedenen Projekttypen

List & Label beherrscht verschiedene Projektarten: Etiketten- und Karteikartenprojekte auf der einen Seite und Listenprojekte auf der anderen.

Etiketten und Karteikarten

Diese Projekte bestehen aus einer Anordnung von Objekten, die ein- (Karteikarten) oder mehrfach (zeilen- und spaltenweise, Etiketten) pro Seite ausgedruckt werden.

Listen

Listenprojekte bestehen hingegen einerseits aus Objekten, die einmal pro Seite ausgegeben werden, und aus einem oder mehreren Objekten, welche mit entsprechend den Datensätzen variierenden Inhalten ausgegeben werden. Für solche "Listen- oder Wiederholbereiche" sind die Designerobjekte Tabelle, Kreuztabelle bzw. Berichtscontainer zuständig und stehen daher nur für diesen Projekttyp zur Verfügung.

1.3.3 Variablen und Felder

In List & Label werden zwei Arten von Datenfeldern grundlegend unterschieden: Es gibt Datenfelder, die pro gedruckter Seite (bzw. pro Etikett oder Karteikarte) nur einmal mit Daten gefüllt werden (sprich: von Ihrer Anwendung mit Echtdateninhalt angemeldet werden), dies sind in der List & Label Terminologie "Variablen". Dem gegenüber stehen die Datenfelder, welche mehrfach auf einer Seite mit unterschiedlichen Daten gefüllt werden, zum Beispiel die einzelnen Datenfelder einer Postenliste einer Rechnung. Diese Datenfelder werden in der List & Label Terminologie "Felder" genannt. Diese Felder stehen nur in Tabellenobjekten, Kreuztabellenobjekten und im Berichtscontainer zur Verfügung.

Demzufolge gibt es in Etiketten- und Karteikartenprojekten lediglich Variablen, während in Listenprojekten sowohl Variablen als auch Felder vorkommen können. Für den Druck einer Rechnung würde eine Anwendung typischerweise die Rechnungskopfdaten wie Empfänger-Name und -Adresse und die Belegnummer als Variablen anmelden, hingegen die Postendaten wie Stückzahl, Artikelnummer, Stückpreis etc. als Felder.

1.3.4 Verfügbare Sprachen für die Benutzeroberfläche

Der List & Label Designer ist außer in deutscher auch in weiteren Sprachen erhältlich. Ebenso ist eine Erweiterung auf zusätzliche Sprachen dank des modularen Aufbaus möglich. Bitte erkundigen Sie sich diesbezüglich beim combit Sales Team. Neben dem Designer werden durch die Sprachkits auch die Druck-, Vorschau- und Exportdialoge lokalisiert, sofern es sich nicht um Windows-seitige Standard- Dialoge handelt. Für diese ist die jeweilige Systemsprache entscheidend.

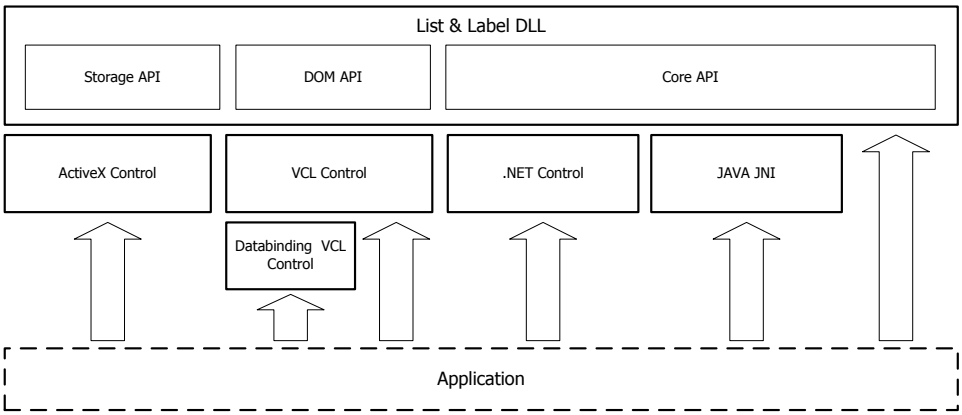
Um ein Sprachkit nach dessen Erwerb (in der Enterprise Edition sind alle verfügbaren Sprachkits bereits im Lieferumfang enthalten) einzubinden, verwenden Sie bei *LlJobOpen()* die entsprechende Sprachkonstante bzw. setzen Sie bei Verwendung einer Komponente die Eigenschaft "Language" auf den gewünschten Wert. An Ihren Kunden liefern

Sie die Sprachdateien (cmll20??.lng, cmls20??.lng) ebenfalls aus. Die Dateien werden dabei von List & Label im gleichen Verzeichnis wie die Haupt-DLL cmll20.dll erwartet.

1.4 Einstieg in die Programmierung

1.4.1 Überblick

Nachfolgende Grafik veranschaulicht schematisch den Aufbau von List & Label aus programmiertechnischer Sicht:



1.4.2 Einbindung in .NET

Lesen Sie weiter in Kapitel "Programmierung mit .NET" und bei Bedarf anschließend weiter ab Kapitel "Die Export-Module". Eine umfangreiche Onlinehilfe beschreibt darüber hinaus die Besonderheiten der Komponente.

1.4.3 Einbindung in Visual Basic

Die Einbindung in Visual Basic erfolgt am einfachsten per OCX/ActiveX Komponente. Lesen Sie hierzu weiter in Kapitel "Programmieren mit der OCX Komponente" und bei Bedarf anschließend weiter ab Kapitel "Die Export-Module".

Wenn Sie auf die Verwendung der OCX Komponente verzichten und lieber direkt per API auf die DLL zugreifen wollen, fügen Sie Ihrem Projekt die Datei cmll20.bas hinzu. Darin enthalten sind die Deklarationen der von Visual Basic verwendbaren DLL-Funktionen. Lesen Sie weiter in Kapitel "Programmierung per API".

1.4.4 Einbindung in Delphi

Die Einbindung in Delphi erfolgt am einfachsten über die VCL Komponente. Lesen Sie hierzu weiter in Kapitel "Programmieren mit der VCL Komponente" und bei Bedarf anschließend weiter ab Kapitel "Die Export-Module".

1.4.5 Einbindung in C++Builder

Die Einbindung in den C++ Builder erfolgt am einfachsten über die VCL Komponente. Lesen Sie hierzu weiter in Kapitel "Programmieren mit der VCL Komponente" und bei Bedarf anschließend weiter ab Kapitel "Die Export-Module".

1.4.6 Einbindung in C/C++

Die Programmierung erfolgt typischerweise über direkte API Aufrufe. Lesen Sie hierzu weiter in Kapitel "Programmierung per API".

1.4.7 Einbindung in Java

Die Integration von List & Label in eine Java Anwendung erfolgt durch Hinzufügen des "combit"-Packages, welches sich jeweils in den mitgelieferten Programmierbeispielen für Java befindet. Die Programmierung erfolgt über direkte API Aufrufe. Lesen Sie hierzu weiter in Kapitel "Programmierung per API". Beachten Sie zusätzlich, dass sich die mitgelieferten Java Native Interface (JNI) Wrapper DLL im List & Label Suchpfad befinden muss. Weitere Informationen hierzu finden Sie im Kapitel "Redistribution Ihrer Anwendung".

1.4.8 Einbindung in andere Programmiersprachen

Für viele Programmiersprachen sind im Lieferumfang von List & Label bereits entsprechende Deklarationsdateien sowie teilweise auch Beispiele enthalten. Diese Dateien finden Sie im entsprechenden Unterverzeichnis Ihrer List & Label-Installation. Folgen Sie Ihrer Programmiersprachendokumentation, um DLLs oder OCX-Controls einzubinden.

Sollte Ihre Programmiersprache nicht enthalten sein, können Sie meist Ihr eigenes Deklarationsfile mit Hilfe der Dokumentation Ihrer Programmiersprache erstellen. Voraussetzung hierfür ist, dass Ihre Programmiersprache den Aufruf von DLL-Funktionen per API unterstützt. Lesen Sie in diesem Fall weiter in Kapitel "Programmierung per API".

Sofern Ihre Programmiersprache die Einbindung von OCX/ActiveX Komponenten unterstützt, können Sie diese meist direkt einbinden. Lesen Sie weiter in Kapitel "Programmieren mit der OCX Komponente".

Im Zweifelsfall können Sie sich auch an unseren Support wenden.

1.4.9 Hinweise zu Variablen- und Feldnamen

Im Variablen-/Feldnamen dürfen nur alphanumerische Zeichen, also 'a'..'z', 'A'..'Z', '0'..'9', '.' und '_' sowie Umlaute verwendet werden. Zahlen dürfen nicht an erster Stelle stehen. Alle nicht-erlaubten Zeichen werden zu einem Unterstrich ('_') umgesetzt. Der Punkt ist ein Hierarchietrenner für die Variablenhierarchie. Darüber können Sie im Designer auch Hierarchien aufbauen. So können Sie z.B. "Person.Adresse.Strasse" und "Person.Adresse.Ort" als Variablen- oder Feldnamen verwenden. Im Designer wird daraus eine hierarchische Ordnerstruktur, d.h. unterhalb von "Person" findet sich ein Ordner "Adresse" mit den Feldern "Ort" und "Strasse".

Die Namen verschiedener Variablen/Felder müssen sich unterscheiden. Auch dürfen Felder und Variablen nicht den gleichen Namen haben.

1.4.10 Debugging Unterstützung

Die Fehlerbeseitigung ist ein wichtiger Teil im Entwicklungsprozess einer Applikation.

List & Label bietet die Möglichkeit, alle Funktionsaufrufe zu protokollieren, um Ihnen die Fehlerbeseitigung in Ihren Programmen zu erleichtern. Dabei werden alle Funktionsaufrufe mit deren Parametern, dem Rückgabewert und eventuellen Fehlermeldungen auf dem Debugging-Output ausgegeben. Diesen können Sie mit dem mitgelieferten Tool Debwin (siehe Kapitel "Fehlersuche mit Debwin") anzeigen lassen.

2. Programmierung mit .NET

2.1 Einleitung

Für die Verwendung von List & Label unter .NET stehen diverse Komponenten zur Verfügung, die die Erstellung von Berichten auf der .NET-Plattform so einfach wie möglich machen. Dieses Tutorial zeigt die wichtigsten Schritte, um schnell und produktiv mit List & Label zu arbeiten.

Die gesamte Programmierschnittstelle ist in der Komponentenhilfe für .NET ausführlich dokumentiert. Diese finden Sie im Ordner "Dokumentation" Ihrer Installation (combit.ListLabel20.chm)

2.1.1 Integration in Visual Studio

Die List & Label-.NET-Komponente wird automatisch in Microsoft Visual Studio eingebunden. Für andere Programmierumgebungen oder bei einer Neuinstallation der Entwicklungsumgebung kann dies auch manuell erfolgen. Die Komponenten liegen als Assembly in den Verzeichnissen "Programmierbare Beispiele und Deklarationen\Microsoft .NET\" sowie "Redistributierbare Dateien\" der List & Label Installation. Die Einbindung geschieht folgendermaßen:

- Menüleiste Extras > Toolboxelemente auswählen
- Reiter .NET Framework Komponenten wählen
- Schaltfläche Durchsuchen... klicken
- combit.ListLabel20.dll auswählen

Nun können die List & Label-Komponenten wie üblich per Drag & Drop aus der Toolbox auf eine Form gezogen werden. Über das Eigenschaftsfenster können die einzelnen Eigenschaften bearbeitet und Ereignisbehandlungen eingefügt werden.

Wenn die Komponenten nicht in Visual Studio ab Version 2010 verfügbar sind, kann dies daran liegen, dass das Client Profile als Zielplattform gewählt wurde. Weitere Hinweise dazu finden sich im Abschnitt "List & Label integrieren".

Um die List & Label .NET Hilfe in den Visual Studio 2010 Help Viewer zu integrieren gehen Sie bitte wie folgt vor:

- Öffnen Sie Visual Studio 2010
- Wählen Sie 'Hilfe > Hilfeeinstellungen verwalten' um den Hilfebibliotheks-Manager zu starten
- Ggf. müssen Sie zunächst einen Ort für den lokalen Inhalt auswählen. Bestätigen Sie diesen Dialog mit 'OK'.
- Wählen Sie den Punkt 'Inhalt von Datenträger installieren'

- Betätigen Sie den Button 'Durchsuchen' und navigieren Sie zum 'Dokumentation\Files'-Unterverzeichnis Ihrer List & Label Installation
- Wählen Sie dort die Datei 'helpcontentsetup.msha' aus und betätigen Sie die Schaltfläche 'Öffnen'
- Zurück im Hilfebibliotheks-Manager betätigen Sie die Schaltfläche 'Weiter'
- Im folgenden Dialog sehen Sie alle verfügbaren Hilfedateien und auch den Eintrag 'combit List & Label 20 - .NET Hilfe'; wählen Sie hier 'Hinzufügen'
- Betätigen Sie nun den Button 'Aktualisieren' um die Hilfe in den Help Viewer zu integrieren
- Bestätigen Sie die digital signierte Hilfe im Dialog 'Sicherheitswarnung' mit 'Ja'
- Nach der Aktualisierung der lokalen Bibliothek betätigen Sie die Schaltfläche 'Fertig stellen' um die Integration der Hilfe abzuschließen. Nun können Sie die List & Label .NET Hilfe verwenden, indem Sie in Visual Studio F1 drücken.

Um die List & Label .NET Hilfe wieder aus Visual Studio 2010 Help Viewer zu entfernen, gehen Sie bitte wie oben beschrieben vor und wählen stattdessen den Punkt 'Inhalt entfernen'.

2.1.2 Komponenten

Im Reiter "combit LL20" in der Toolbox finden sich nach der Installation die folgenden Komponenten:

Komponente	Beschreibung
ListLabel	Die wichtigste Komponente. In dieser sind alle zentralen Funktionen wie Druck, Design und Export vereinigt.
DataSource	Eine Komponente, die als Datenquelle direkt an eine ListLabel-Instanz gebunden werden kann. Eine Beschreibung findet sich im Abschnitt "Datenprovider".
DesignerControl	Eine Komponente zur Anzeige des Designers in eigenen Formularen.
ListLabelRTFControl	Eine RTF-Editor-Komponente zur Verwendung in eigenen Formularen.
ListLabelPreviewControl	Eine Vorschau-Komponente, die ebenfalls in eigenen Formularen verwendet werden kann und z.B. den Direktexport nach PDF unterstützt. Um den Druck in ein solches Vorschaucontrol durchzuführen, setzen Sie in der ListLabel-Komponente die Eigenschaft AutoDestination auf LlPrintMode.PreviewControl und wählen das gewünschte Vorschaucontrol für die Eigenschaft "PreviewControl".
ListLabelDocument	Eine Ableitung von PrintDocument. Mit dieser lassen sich

Komponente	Beschreibung
	auch die .NET-eigenen Preview-Klassen zur Anzeige von List & Label Vorschaudateien verwenden.
ListLabelWebViewer	Eine ASP.NET-Komponente zur Anzeige von Vorschaudateien im Internet Explorer und Firefox.

2.2 Erste Schritte

Dieser Abschnitt führt durch die ersten Schritte, die benötigt werden um List & Label in eine bestehende Applikation zu integrieren.

2.2.1 List & Label integrieren

Zunächst muss dem Projekt ein Verweis auf die List & Label Assembly hinzugefügt werden. Diese befindet sich im Verzeichnis "Programmierbare Beispiele und Deklarationen\Microsoft .NET\" der Installation. Über Projekt > Verweis hinzufügen kann die Assembly dem Projekt hinzugefügt werden.

Für das "volle" .NET Framework steht die Assembly combit.ListLabel20.dll zur Verfügung. Wenn als Zielplattform das .NET Client Profile gewünscht ist, muss stattdessen die combit.ListLabel20.ClientProfile.dll referenziert werden.

Das .NET Client Profile ist eine Untermenge des .NET Frameworks. Ab Visual Studio 2010 ist es die Standard-Zielplattform für Winforms-Anwendungen. Wir empfehlen, zunächst in den Eigenschaften des Applikationsprojekts das "volle" Framework als Ziel zu wählen und die combit.ListLabel20.dll Assembly zu referenzieren, um zur Designzeit die Unterstützung durch SmartTags und Eigenschaftseditoren zu haben. Nach der Designphase kann dann die Zielplattform ggf. angepasst und die ClientProfile-Assembly referenziert werden.

Im zweiten Schritt kann dann eine Instanz der Komponente erzeugt werden. Dies erfolgt entweder über die Entwicklungsumgebung direkt, indem die ListLabel-Komponente auf ein Formular gezogen wird. Alternativ kann die Komponente auch dynamisch erzeugt werden:

C#:

```
combit.ListLabel20.ListLabel LL = new combit.ListLabel20.ListLabel();
```

VB.NET:

```
Dim LL As New combit.ListLabel20.ListLabel()
```

In der Regel werden die Namespaces combit.ListLabel20 und combit.ListLabel20.DataProviders für die ganze Datei vorreferenziert, je nach Programmiersprache über "using" (C#) oder "Imports" (VB.NET). Dies spart in der Folge viel Tipparbeit.

C#:

```
using combit.ListLabel20;
using combit.ListLabel20.DataProviders;
```

VB.NET:

```
Imports combit.ListLabel20
Imports combit.ListLabel20.DataProviders
```

Bei der dynamischen Erzeugung sollte die Komponente nach Verwendung über die Dispose-Methode wieder freigegeben werden, damit die nicht-verwalteten Ressourcen möglichst schnell wieder freigegeben werden.

C#:

```
LL.Dispose();
```

VB.NET:

```
LL.Dispose()
```

Aus Performancegründen empfiehlt es sich aber auch bei dynamischer Erzeugung, immer eine Instanz des ListLabel-Objektes global im Speicher zu halten. Diese kann z.B. im Load-Ereignis des Applikationshauptfensters erzeugt und im FormClosed-Ereignis wieder freigegeben werden. Der entscheidende Vorteil dabei ist, dass die List & Label-Module nicht für jede neue Instanz ge- und entladen werden, was bei häufigen Aufrufen oder z.B. auch beim Seriendruck für unerwünschte Verzögerungen sorgen kann.

2.2.2 Komponente lizenzieren

Im Startmenü wird während der Installation der Vollversion (nicht bei der Trial-Version) eine Datei mit persönlichen Lizenz- und Supportinformationen angelegt. Damit List & Label bei Endkunden erfolgreich verwendet werden kann, muss unbedingt der darin enthaltene Lizenzschlüssel an alle Instanzen des ListLabel-Objektes übergeben werden. Das Objekt stellt hierfür die Eigenschaft LicensingInfo zur Verfügung. Ein Beispielaufruf könnte wie folgt aussehen:

C#:

```
LL.LicensingInfo = "A83jHd";
```

VB.NET:

```
LL.LicensingInfo = "A83jHd"
```

wobei der Teil in Anführungszeichen durch den Lizenzcode aus der Textdatei ersetzt werden muss.

Beachten Sie: in der Trialversion muss für den Lizenzcode nichts bzw. ein Leerstring übergeben werden.

2.2.3 Datenquelle anbinden

Für Design und Druck muss List & Label eine Datenquelle bekannt gemacht werden. Einen Überblick über die verfügbaren Datenquellen bietet der Abschnitt "Datenprovider". Natürlich können auch zusätzlich ungebundene, eigene Daten übergeben werden. Ein Beispiel hierfür zeigt der Abschnitt "Datenbankunabhängige Inhalte".

Um die Datenquelle an List & Label anzubinden stellt die Komponente die Eigenschaft DataSource zur Verfügung. Auch hier kann die Anbindung entweder interaktiv in der Entwicklungsumgebung über das Eigenschaftenfenster oder die SmartTags der Komponente vorgenommen werden oder alternativ auf Code-Ebene erfolgen:

C#:

```
LL.DataSource = CreateDataSet();
```

VB.NET:

```
LL.DataSource = CreateDataSet()
```

Die CreateDataSet()-Routine steht hierbei für eine Routine aus Ihrem Programm, die das gewünschte DataSet für den Bericht bereitstellt.

2.2.4 Design

Der Designer wird über die Methode *Design()* aufgerufen und wird dann in einem modalen Pop-up Fenster dargestellt, welches Ihr Anwendungsfenster überlagert. Zuvor muss immer eine Datenquelle zugewiesen werden – diese ist die Basis für die im Designer verfügbaren Daten. Daher gibt es auch keine alleinstehende Design-Anwendung; die Daten werden immer direkt aus der Applikation zur Verfügung gestellt, List & Label selbst greift niemals direkt auf Daten zu.

Der vollständige Aufruf – in diesem Beispiel mit einem DataSet als Datenquelle – wäre:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();
LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()
LL.Design()
LL.Dispose()
```

Standardmäßig wird hierbei ein Dateiauswahldialog für den Anwender angezeigt, in dem er entweder einen neuen Namen für die Berichtsdatei vergeben und so einen neuen Bericht erzeugen kann oder eine bestehende Datei zur Bearbeitung auswählen kann. Natürlich kann dies auch unterdrückt werden – der Abschnitt "Wichtige Eigenschaften der Komponente" beschreibt dies.

Die Verwendung des Designers selbst ist in der zugehörigen Onlinehilfe bzw. im Designerhandbuch detailliert erklärt. Das Ergebnis des Designprozesses sind in der Regel vier Dateien, die durch den Designer angelegt wurden. Die Dateiendungen können über die FileExtensions-Eigenschaft der ListLabel-Komponente frei bestimmt werden. Die folgende Tabelle beschreibt die Dateien für den Standardfall.

Datei	Inhalt
<Berichtsname>.lst	Die eigentliche Projektdatei. Diese enthält Informationen über die Formatierung der zu druckenden Daten, nicht aber die Daten selbst.
<Berichtsname>.lsv	Eine JPEG-Datei mit einer Miniaturdarstellung des Projektes für die Anzeige im Dateiauswahldialog.
<Berichtsname>.lsp	Datei mit benutzerspezifischen Drucker- und Exporteinstellungen. Diese Datei sollte nicht weitergegeben werden, wenn der Designrechner nicht mit dem Druckrechner identisch ist, da dann der darin angegebene Drucker meist nicht existiert.
<Berichtsname>.~lst	Wird ab dem zweiten Speichern im Designer angelegt und enthält eine Sicherung der Projektdatei.

Die wichtigste Datei ist dabei natürlich die Projektdatei. Die anderen Dateien werden von List & Label automatisch zur Laufzeit der Anwendung erstellt.

Zur Druckzeit wird dann aus der Kombination von Projektdatei und Datenquelle der eigentliche Bericht erstellt. In der Praxis ist es häufig auch gewünscht, die Projektdateien in einer zentralen Datenbank zu halten. Wie dies gemacht wird, beschreibt der Abschnitt "Projektdateien in Datenbank halten".

2.2.5 Druck

Der Druck wird über die Methode *Print()* aufgerufen. Zuvor muss im Designer eine Projektdatei für die Datenstruktur der gewählten Datenquelle erstellt werden. Am einfachsten bindet man die Komponente zur Druck- und Designzeit an die gleiche Datenquelle. Dann zeigt auch die Vorschau im Designer die richtigen Daten an und der Anwender kann sich ein gutes Bild vom Ergebnis zur Laufzeit machen. Ein vollständiger Aufruf des Drucks wäre:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()
LL.Print()
LL.Dispose()
```

Auch hier erscheinen in der Standard-Einstellung zunächst ein Dateiauswahl-, dann ein Druckoptionsdialog. Der Abschnitt "Wichtige Eigenschaften der Komponente" beschreibt, wie diese umgangen bzw. vorausgefüllt werden können, wenn dies erwünscht ist.

2.2.6 Export

Unter Export wird die Ausgabe auf eines der unterstützten Ausgabeformate wie PDF, HTML, RTF, XLS usw. verstanden. Der Start eines Exports ist codeseitig identisch mit dem eines Drucks, im Druckoptionsdialog kann der Anwender neben den "normalen" Ausgabeformaten Drucker, Datei und Vorschau auch ein beliebiges Exportformat wählen. Soll ein Format als Standardwert vorgewählt werden, kann dies vor Druckstart wie folgt erfolgen:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();
LL.ExportOptions.Add(LlExportOption.ExportTarget, "PDF");
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()
LL.ExportOptions.Add(LlExportOption.ExportTarget, "PDF")
LL.Print()
LL.Dispose()
```

Die verfügbaren Exportziele listet die folgende Tabelle auf:

Exportziel	Wert für ExportTarget
Drucker	PRN

Exportziel	Wert für ExportTarget
Vorschau	PRV
Adobe PDF Format	PDF
XHTML/CSS Format	XHTML
Multi-Mime HTML Format	MHTML
Microsoft Excel Format	XLS
Microsoft Word Format	DOCX
Rich Text Format (RTF)	RTF
Microsoft XPS Format	XPS
Multi-TIFF-Grafik	PICTURE_MULTITIFF
TIFF-Grafik	PICTURE_TIFF
PNG-Grafik	PICTURE_PNG
JPEG-Grafik	PICTURE_JPEG
Bitmap-Grafik	PICTURE_BMP
Metafile-Grafik (EMF)	PICTURE_EMF
Datei	FILE
HTML Format	HTML
HTML jQuery Mobile Format	JQM
Nadeldrucker (TTY)	TTY
SVG Format	SVG
Text (CSV) Format	TXT
Text (Layout) Format	TXT_LAYOUT
XML Format	XML

Auch die weiteren Optionen (z.B. Schriftarteinbettung, Verschlüsselung etc.) lassen sich direkt aus dem Code mit Standardwerten vorbelegen. Dies erfolgt wie im Beispiel oben ebenfalls über die ExportOptions-Klasse, die *LlExportOption*-Enumeration beinhaltet für alle unterstützten Optionen eigene Werte.

Am häufigsten werden diese benötigt, um einen "stillen" Export durchzuführen. Hierfür kann bequemer auch die Export()-Methode der Komponente verwendet werden. Beachten Sie das Export-Beispiel für C# und VB.NET.

2.2.7 Wichtige Eigenschaften der Komponente

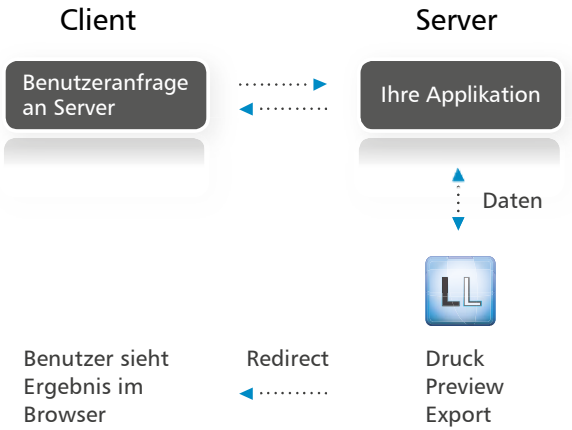
Das Verhalten von Druck, Design und Export kann durch einige Eigenschaften der Komponente beeinflusst werden. Die Wichtigsten sind in der folgenden Tabelle zusammengestellt:

Eigenschaft	Funktion
AutoProjectFile	Name der zu verwendenden Projektdatei. Dies ist der Vorgabename für den Anwender, wenn diesem ein Datei- Auswahldialog zur Verfügung gestellt wird. Ansonsten ist dies der Name des zu verwendenden Projekts (Voreinstellung: leer).
AutoDestination	Ausgabeformat. Wenn gewünscht, kann dem Anwender über diese Eigenschaft ein Format vorgegeben werden, z.B. Druck nur auf Drucker oder Vorschau erlaubt (Voreinstellung: LIPrintMode.Export). Wenn eine Auswahl von Exportformaten erlaubt werden soll, kann dies über das Setzen von LIOptionString.ExportsAllowed erfolgen. Dies wird im Abschnitt "Einschränkung von Exportformaten" gezeigt.
AutoFileAlsoNew	Bestimmt, ob der Benutzer für das Design auch einen noch nicht vorhandenen Dateinamen angeben darf, um ein neues Projekt zu erzeugen (Voreinstellung: true).
AutoProjectType	Legt den Projekttypen fest. Die verschiedenen Projekttypen sind im Abschnitt "Projekttypen" erklärt (Voreinstellung: LIProject.List).
AutoShowPrintOptions	Bestimmt, ob der Druckoptionsdialog angezeigt oder unterdrückt wird (Voreinstellung: true, Anzeigen).
AutoShowSelectFile	Bestimmt, ob der Dateiauswahldialog angezeigt oder unterdrückt wird (Voreinstellung: true, Anzeigen).
AutoMasterMode	Dient gemeinsam mit der DataMember-Eigenschaft dazu, bei 1:n-verknüpften Datenstrukturen die Haupttabelle als Variablen zu übergeben. Ein Beispiel findet sich im Abschnitt "Variablen, Felder und Datentypen".

2.2.8 Webreporting

Der Druck innerhalb einer Webapplikation ist im Grunde nichts anderes als ein Export z.B. auf das PDF-Format, bei dem alle Dialoge unterdrückt werden. Wie dies grundsätzlich funktioniert ist im Abschnitt "Export ohne Benutzerinteraktion" beschrieben. Nachdem der Bericht auf diese Weise erstellt wurde, kann der Browser des Anwenders über die üblichen Mechanismen auf die erstellte Datei geleitet werden. Alternativ kann die Datei auch direkt per eMail an den Anwender gesendet werden, wenn die Erstellung z.B. zeitgesteuert erfolgen soll (s. Abschnitt "eMail-Versand").

Folgende Grafik veranschaulicht das Funktionsprinzip:



In der Regel werden die Projektdateien innerhalb einer Client-Applikation erstellt und dann zusammen mit der Webapplikation veröffentlicht. Für das Design im Browser stehen Ihnen für alle gängigen Browser Designer-Controls zur Verfügung. Dies demonstriert das mitgelieferte Webreporting-Beispiel für C# und VB.NET.

Auch auf dem Server muss zumindest ein Druckertreiber installiert sein, damit der Druck erfolgreich durchgeführt werden kann. List & Label arbeitet eng mit dem Windows-GDI zusammen und benötigt daher für alle Operationen einen Drucker-Gerätekontext. Auch der Microsoft XPS Druckertreiber (ab .NET Framework 3.5 auf allen Systemen automatisch installiert) eignet sich als Referenzdrucker.

2.3 Weitere wichtige Konzepte

2.3.1 Datenprovider

Die Datenversorgung in List & Label erfolgt über Datenprovider. Dies sind Klassen, die das Interface IDataProvider aus dem Namespace combit.ListLabel20.DataProviders implementieren. In diesem Namespace ist bereits eine Vielzahl von Klassen enthalten, die als Datenprovider fungieren können. Eine ausführliche Klassenreferenz ist in der .NET Komponentenhilfe enthalten.

Für augenscheinlich nicht direkt unterstützte Dateninhalte findet sich meist trotzdem ein passender Provider. Businessdaten aus Anwendungen können in der Regel über den Objektdatenprovider übergeben werden, liegen die Daten in kommaseparierter Form vor kann der Datenprovider aus dem "Dataprovider"-Beispiel verwendet werden. Viele andere Datenquellen unterstützen die Serialisierung nach XML, so dass dann der XmlDataProvider verwendet werden kann. Wenn nur einige wenige zusätzliche Informationen übergeben werden sollen, so ist auch dies direkt möglich – ein Beispiel zeigt der Abschnitt "Datenbankunabhängige Inhalte".

Ist List & Label an einen solchen Datenprovider gebunden, werden die folgenden Features automatisch unterstützt, sofern die zugrunde liegende Datenquelle dies ermöglicht:

- Echtdatenvorschau im Designer
- Berichtscontainer und relationale Datenstrukturen
- Sortierungen
- Drilldown

Die folgende Übersicht listet die wichtigsten Klassen und die von Ihnen unterstützten Datenquellen auf.

AdoDataProvider

Ermöglicht den Zugriff auf Daten der folgenden ADO.NET Elemente:

- DataView
- DataTable
- DataViewManager
- DataSet

Der Provider kann implizit zugewiesen werden, indem die DataSource-Eigenschaft auf eine Instanz einer der unterstützten Klassen gesetzt wird. Natürlich kann der Provider aber auch explizit zugewiesen werden.

Der Provider unterstützt automatisch einstufige Sortierungen, nach jedem Feld kann auf- oder absteigend sortiert werden.

Beispiel:

C#:

```
ListLabel LL = new ListLabel();
AdoDataProvider provider = new AdoDataProvider(CreateDataSet());
LL.DataSource = provider;
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
Dim provider As New AdoDataProvider(CreateDataSet())
LL.DataSource = provider
LL.Print()
LL.Dispose()
```

DataProviderCollection

Ermöglicht die Kombination mehrerer anderer Datenprovider in einer Datenquelle. Der Provider kann zum Beispiel Daten aus mehreren DataSet-Klassen kombinieren oder unterstützt die Mischung aus XML und eigenen Objektdaten.

Der Provider unterstützt die Sortierungen, die die in der Collection enthaltenen Provider unterstützen.

Beispiel:

C#:

```
DataSet ds1 = CreateDataSet();
DataSet ds2 = CreateOtherDataSet();

// Daten von ds1 und ds2 in einer Datenquelle kombinieren
DataProviderCollection providerCollection = new DataProviderCollection();
providerCollection.Add(new AdoDataProvider(ds1));
providerCollection.Add(new AdoDataProvider(ds2));
ListLabel LL = new ListLabel();
LL.DataSource = providerCollection;
LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim ds1 As DataSet = CreateDataSet()
Dim ds2 As DataSet = CreateOtherDataSet()

' Daten von ds1 und ds2 in einer Datenquelle kombinieren
Dim providerCollection As New DataProviderCollection()
providerCollection.Add(New AdoDataProvider(ds1))
providerCollection.Add(New AdoDataProvider(ds2))
Dim LL As New ListLabel()
LL.DataSource = providerCollection
LL.Design()
LL.Dispose()
```

DataSource

Dieser Datenprovider nimmt eine Sonderstellung ein, da er als Komponente direkt aus der Toolbox eingefügt werden kann. Die Komponente verfügt über einige wenige Eigenschaften, die auch über die SmartTags zur Verfügung stehen. Die wichtigste Eigenschaft ist "ConnectionProperties". Über den dahinterliegenden Editor kann direkt aus der Entwicklungsumgebung eine Verbindungszeichenfolge (Connection String) erstellt werden, die den Zugriff auf folgende Datenquellen ermöglicht:

- Microsoft Access
- ODBC-Datenquellen (z.B. Excel-Daten)
- Microsoft SQL-Server (auch dateibasiert)
- Oracle-Datenbanken

Einmal konfiguriert steht die Datenquelle im Auswahlfenster für die DataSource der List-Label-Komponente zur Verfügung und kann so direkt zugewiesen werden. Über den Link "Berichtsdesigner öffnen" in den SmartTags der ListLabel-Komponente kann auch der

Designer direkt aus der Entwicklungsumgebung geöffnet werden, so dass keine einzige Codezeile mehr notwendig ist, um auf Daten einer DataSource zuzugreifen.

Der Provider unterstützt automatisch einstufige Sortierungen, nach jedem Feld kann auf- oder absteigend sortiert werden.

DbCommandSetDataProvider

Erlaubt es, mehrere IDbCommand Implementierungen in einer Datenquelle zu kombinieren. Der Provider kann z.B. dazu verwendet werden, um auf mehrere SQL-Tabellen zuzugreifen und Relationen zwischen diesen zu definieren. Eine andere Möglichkeit ist es Daten aus bspw. Microsoft SQL- und Oracle-Datenbanken in einer Datenquelle zu kombinieren.

Der Provider unterstützt automatisch einstufige Sortierungen, nach jedem Feld kann auf- oder absteigend sortiert werden.

ObjectDataProvider

Erlaubt den Zugriff auf Objektstrukturen. Der Provider kann mit folgenden Typen/Schnittstellen zusammenarbeiten:

- IEnumerable (setzt jedoch mindestens einen Datensatz voraus)
- IEnumerable<T>
- IListSource

Die Eigenschaftsnamen und -typen können über die ITypedList Schnittstelle beeinflusst werden. Wenn nur der Name geändert werden soll ist es meist einfacher, das DisplayNameAttribute zu verwenden. Einzelne Member können über das Browsable(False) Attribut unterdrückt werden.

Der Provider kann leere Aufzählungen durchlaufen solange diese stark typisiert sind. Ansonsten wird mindestens ein Element in der Aufzählung vorausgesetzt. Dieses erste Element bestimmt den Typ der für das weitere Durchlaufen verwendet wird.

Der Provider unterstützt Sortierung automatisch sobald die Datenquelle die IBindingList Schnittstelle implementiert.

Dieser Datenprovider unterstützt auch die Bindung an LINQ Abfrageresultate, da diese IEnumerable<T> sind.

Bei Verwendung von EntityCollection<T>-Objekten als Datenquelle prüft der ObjectDataProvider zunächst mit Hilfe der IsLoaded-Eigenschaft den Zustand der Unterrelation und ruft gegebenenfalls dynamisch *Load()* auf. Damit werden die Daten bereitgestellt wenn sie benötigt werden. Beispiel:

C#:

```
class Car
{
    public string Brand { get; set; }
    public string Model { get; set; }
}
```

```
List<Car> cars = new List<Car>();
cars.Add(new Car { Brand = "VW", Model = "Passat"});
cars.Add(new Car { Brand = "Porsche", Model = "Cayenne"});
ListLabel LL = new ListLabel();
LL.DataSource = new ObjectDataProvider(cars);
LL.Design();
LL.Dispose();
```

VB.NET:

```
Public Class Car
    Dim _brand As String
    Dim _model As String

    Public Property Brand() As String
    Get
        Return _brand
    End Get
    Set(ByVal value As String)
        _brand = value
    End Set
    End Property
    Public Property Model() As String
    Get
        Return _model
    End Get
    Set(ByVal value As String)
        _model = value
    End Set
    End Property
End Class

Dim LL As New ListLabel
Dim Cars As New List(Of Car)()
Dim Car As New Car
Car.Model = "Passat"
Car.Brand = "VW"
Cars.Add(Car)
Car = New Car
Car.Model = "Cayenne"
Car.Brand = "Porsche"
Cars.Add(Car)
LL.DataSource = New ObjectDataProvider(Cars)
LL.AutoProjectType = LlProject.List
LL.Design()
LL.Dispose()
```

OleDbConnectionDataProvider

Ermöglicht das Binden an eine OleDbConnection (z.B. Access Datenbankdatei). Der Provider unterstützt automatisch einstufige Sortierungen, nach jedem Feld kann auf- oder absteigend sortiert werden.

Beispiel:

C#:

```
OleDbConnection conn = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=" + DatabasePath);
OleDbConnectionDataProvider provider = new OleDbConnectionDataProvider(conn);
ListLabel LL = new ListLabel();
LL.DataSource = provider;
LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim conn As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
DatabasePath)
Dim provider As New OleDbConnectionDataProvider(conn)
Dim LL As New ListLabel()
LL.DataSource = provider
LL.Design()
LL.Dispose()
```

OracleConnectionDataProvider

Ermöglicht das Binden an eine OracleConnection. Dieser Provider ist nicht verfügbar, wenn die Zielplattform das .NET Client Profile ist. Der Provider unterstützt automatisch einstufige Sortierungen, nach jedem Feld kann auf- oder absteigend sortiert werden.

SqlConnectionDataProvider

Ermöglicht das Binden an eine SqlConnection. Der Provider unterstützt automatisch einstufige Sortierungen, nach jedem Feld kann auf- oder absteigend sortiert werden.

Beispiel:

C#:

```
SqlConnection conn = new
SqlConnection(Properties.Settings.Default.ConnectionString);
SqlConnectionDataProvider provider = new SqlConnectionDataProvider(conn);
ListLabel LL = new ListLabel();
LL.DataSource = provider;
LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim conn As New SqlConnection(Properties.Settings.Default.ConnectionString)
Dim provider As New SqlConnectionDataProvider(conn)
Dim LL As New ListLabel()
LL.DataSource = provider
LL.Design()
LL.Dispose()
```

XmlDataProvider

Ermöglicht den einfachen Zugriff auf XML-Daten. Es werden keine Schemainformationen aus XML-/XSD-Dateien verwendet und keine Constraints/Randbedingungen behandelt. Der Haupteinsatzzweck dieser Klasse ist der schnelle und einfache Zugriff auf verschachtelte XML-Daten. Der Provider unterstützt keine Sortierungen.

Beispiel:

C#:

```
XmlDataProvider provider = new XmlDataProvider(@"c:\users\public\data.xml");
ListLabel LL = new ListLabel();
LL.DataSource = provider;
LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim provider As New XmlDataProvider("c:\users\public\data.xml")
Dim LL As New ListLabel()
LL.DataSource = provider
LL.Design()
LL.Dispose()
```

2.3.2 Variablen, Felder und Datentypen

Variablen und Felder sind die dynamischen Textblöcke für Berichte und enthalten den dynamischen Teil der Daten. Variablen ändern sich typischerweise einmal pro Seite oder Bericht – ein Beispiel sind die Kopfdaten einer Rechnung mit Rechnungsnummer und Adressat. Felder hingegen ändern sich in der Regel für jeden Datensatz, typische Vertreter sind also z.B. die Postendaten einer Rechnung.

Im Designer werden Variablen stets außerhalb, Felder nur innerhalb des Berichtscontainers (des "Tabellenbereichs") angeboten und können auch nur dort verwendet werden. Die Trennung dient vor allem dazu, dem Endanwender das Leben leichter zu machen – wenn er ein Feld in den "Außenbereich" platzieren würde, wäre das Ergebnis je nach Druckreihenfolge entweder der Inhalt des Ersten oder Letzten Datensatzes.

Für beide Baustein-Typen gilt, dass sie hierarchisch angeordnet werden können – im Designer macht sich dies durch eine Ordnerstruktur bemerkbar. Die Datenbank-

Tabellennamen werden von der Datenbindung automatisch berücksichtigt, so dass alle Daten der "Bestelldaten"-Tabelle in einem Ordner "Bestelldaten" dargestellt werden.

Eigene Daten können ebenfalls hierarchisch angeordnet werden, indem ein Punkt als Hierarchietrenner verwendet wird (also z.B. "Zusatzdaten.Benutzername"). Wie eigene Daten hinzugefügt werden können und wie die Anmeldeinformationen der Datenbindung beeinflusst werden können zeigt der Abschnitt "Datenbankunabhängige Inhalte".

Variablen und Felder bei Datenbindung

Wenn in einer 1:n hierarchisch verknüpften Datenstruktur wie z.B. "Rechnungskopf" und "Rechnungsposten" die Bestelldaten-Tabelle als Variablen, die Bestellposten hingegen als Felder angemeldet werden sollen, kann dies über die Eigenschaften DataMember und AutoMasterMode der Komponente erreicht werden:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Bestelldaten als Variablen
LL.DataMember = "Rechnungskopf";
LL.AutoMasterMode = LlAutoMasterMode.AsVariables;

LL.Design();
LL.Dispose();
```

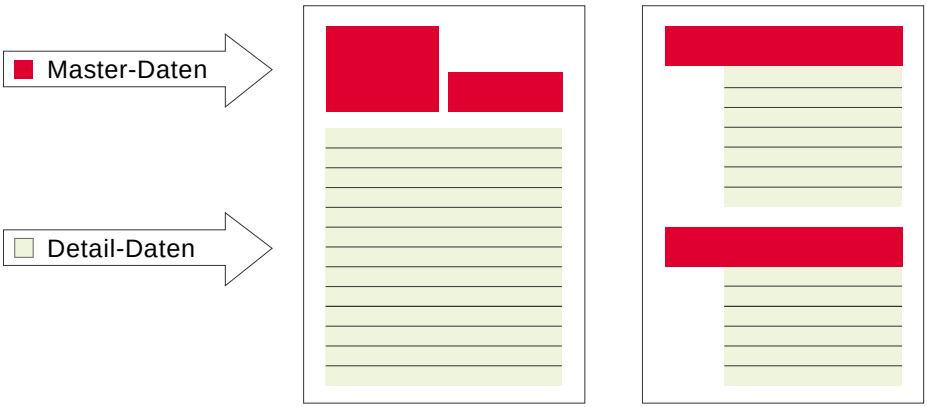
VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Bestelldaten als Variablen
LL.DataMember = "Rechnungskopf"
LL.AutoMasterMode = LlAutoMasterMode.AsVariables

LL.Design()
LL.Dispose()
```

Zur Druckzeit wird in diesem Falle automatisch ein Seriendruck generiert, wenn also z.B. ein Rechnungsformular designed wurde, wird für jeden Datensatz aus der Rechnungskopf-Tabelle eine eigene Rechnung mit eigener Seitennummerierung, Summierung etc. erzeugt.



Auswirkung der Option AutoMasterMode. Links: "AsVariables", rechts "AsFields"

Datentypen

Variablen und Felder werden typisiert übergeben, d.h. je nach Inhalt in der Datenbank als Text, Zahl etc. Dies besorgt die Datenbindung in der Regel automatisch, eine explizite Übergabe des Typs ist nur dann notwendig, wenn zusätzlich eigene Daten übergeben werden. Auch dann wird meist schon der passende Datentyp vorgewählt (z.B. bei Übergabe von DateTime-Objekten).

Die folgende Tabelle zeigt die wichtigsten Datentypen.

Datentyp	Verwendung
LlFieldType.Text	Text.
LlFieldType.RTF	RTF-formatierter Text. Dieser Feldtyp kann im Designer direkt in einem RTF-Feld bzw. RTF-Objekt verwendet werden.
LlFieldType.Numeric LlFieldType.Numeric_Integer	Zahl. Die Datenbindung unterscheidet automatisch zwischen Gleitkommazahlen und Integer-Werten.
LlFieldType.Boolean	Logische Werte.
LlFieldType.Date	Datums- und Zeitwerte (DateTime).
LlFieldType.Drawing	Grafik. In der Regel wird der Dateiname angegeben, für Bitmaps und EMF-Dateien ist auch eine direkte Übergabe des Speicherhandles möglich. Die Datenbindung untersucht automatisch Byte[]-Felder auf Ihren Inhalt und meldet diese als Grafik an, wenn sich ein passendes Format findet.
LlFieldType.Barcode	Barcode. Barcodes können am einfachsten als Instanzen der LlBarcode-Klasse direkt in den Add-Methoden der Variables- und Fields-Eigenschaft

Datentyp	Verwendung
	übergeben werden.
LlFieldType.HTML	HTML. Der Inhalt der Variablen ist ein gültiger HTML-Stream, ein Dateiname oder eine URL.

2.3.3 Ereignisse

Die folgende Tabelle zeigt einige wichtige Ereignisse der ListLabel-Komponente. Eine vollständige Referenz findet sich in der Komponentenhilfe für .NET.

Event	Verwendung
AutoDefineField/ AutoDefineVariable	Diese Ereignisse werden für jedes Feld bzw. jede Variable vor der Anmeldung bei List & Label aufgerufen. Über die Event-Argumente kann der Feldname und Inhalt angepasst werden oder die Anmeldung komplett unterdrückt werden. Beispiele hierfür im Abschnitt "Datenbankunabhängige Inhalte".
AutoDefineNewPage	Dieses Ereignis wird zu Beginn jeder Seite aufgerufen. Wenn die Anwendung seitenspezifische Zusatzdaten benötigt, die nicht aus der Datenquelle selbst stammen, können diese in diesem Event per LL.Variables.Add() hinzugefügt werden. Beispiele hierfür im Abschnitt "Datenbankunabhängige Inhalte".
AutoDefineNewLine	Dieses Ereignis wird für jede Zeile aufgerufen. Wenn die Anwendung zeilenspezifische Zusatzdaten benötigt, die nicht aus der Datenquelle selbst stammen, können diese in diesem Event per LL.Fields.Add() hinzugefügt werden. Beispiele hierfür im Abschnitt "Datenbankunabhängige Inhalte".
DrawObject DrawPage DrawTableLine DrawTableField	Diese Ereignisse werden jeweils einmal vor und einmal nach dem Druck des zugehörigen Elements aufgerufen, also z.B. für jede Tabellenzelle (DrawTableField). Die Ereignisargumente enthalten ein Graphics-Objekt und das Rechteck der Ausgabe, so dass die Anwendung eigene Informationen zusätzlich selber ausgeben kann. Es kann sich hierbei um eine besondere Schattierung, einen "Demo"-Schriftzug oder eine komplette eigene Ausgabe handeln.
VariableHelpText	Dient zur Anzeige eines Hilfetexts für Variablen und Felder für den Endanwender im Designer.

2.3.4 Projekttypen

Je nach Berichtstyp stehen drei verschiedene Modi des Designers zur Verfügung. Welcher Modus verwendet wird, hängt vom Wert der Eigenschaft AutoProjectType ab.

Listen

Dies ist der Standard und entspricht dem Wert LlProject.List für die AutoProjectType Eigenschaft.

Typische Anwendungsfälle sind Rechnungen, Adresslisten, Auswertungen mit Charts und Kreuztabellen, mehrspaltige Listen, kurz alle Berichtstypen für die ein tabellarisches Element benötigt wird. Nur in diesem Modus steht der Berichtscontainer zur Verfügung (s. Abschnitt "Berichtscontainer").

Etiketten

Dieser Projekttyp entspricht dem Wert LlProject.Label für die AutoProjectType Eigenschaft.

Er wird für die Ausgabe von Etiketten verwendet. Da es hier keine tabellarischen Bereiche und auch keinen Berichtscontainer gibt, stehen lediglich Variablen, nicht aber Felder zur Verfügung (vgl. Abschnitt "Variablen, Felder und Datentypen").

Wenn die verwendete Datenquelle mehrere Tabellen enthält, z.B. Produkte, Kunden etc., kann über die Eigenschaft DataMember der Komponente die Quelltabelle für den Etikettendruck ausgewählt werden:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Produkte als Quelldaten
LL.DataMember = "Produkte";

// Etikett als Projekttyp wählen
LL.AutoProjectType = LlProject.Label;

LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Produkte als Quelldaten
LL.DataMember = "Produkte"

' Etikett als Projekttyp wählen
LL.AutoProjectType = LlProject.Label

LL.Design()
```

```
LL.Dispose()
```

Karteikarten

Dieser Projekttyp entspricht dem Wert LlProject.Card für die AutoProjectType Eigenschaft.

Karteikartenprojekte sind ein Spezialfall von Etikettenprojekten mit genau einem seitenfüllenden Etikett. Typische Anwendungsfälle sind der Karteikartendruck (z.B. alle Kundeninformationen auf einen Blick) oder Serienbriefe. Die Ansteuerung erfolgt genau analog zum Abschnitt "Etiketten", es gelten die gleichen Hinweise und Einschränkungen.

2.3.5 Verschiedene Drucker und Kopiendruck

List & Label bietet eine komfortable Unterstützung für die Aufteilung eines Berichts auf verschiedene Drucker oder die Ausgabe von Kopien mit "Kopie"-Vermerk. Das Beste daran – es handelt sich um reine Designer-Features, die automatisch von List & Label unterstützt werden.

Layout-Bereiche

Die Layout-Bereiche dienen zur Aufteilung des Projekts auf mehrere Seitenbereiche mit unterschiedlichen Eigenschaften. Typische Anwendungsfälle sind z.B. unterschiedliche Drucker für erste Seite, Folgeseiten und letzte Seite. Weitere Anwendungsmöglichkeiten sind Mischung von Hoch- und Querformat innerhalb eines Berichts.

Demonstriert wird die Verwendung z.B. in der List & Label Beispielanwendung (im Startmenü direkt auf der Basisebene) unter Design > Erweiterte Beispiele > Mischung von Hoch- und Querformat.

Ausfertigungen und Kopien

Sowohl Ausfertigungen als auch Kopien dienen zur Ausgabe mehrerer Exemplare eines Berichts. Die Kopien sind dabei "echte" Hardwarekopien, d.h. hier wird der Drucker angewiesen, mehrere Exemplare der Ausgabe zu erstellen. Naturgemäß sind diese Exemplare untereinander alle identisch und werden mit den gleichen Druckereinstellungen erzeugt.

Wenn die Ausgaben unterschiedliche Eigenschaften haben sollen (z.B. Original aus Schacht 1, Kopie aus Schacht 2) oder ein "Kopie"-Wasserzeichen ausgegeben werden soll, kann dies über die Ausfertigungssteuerung erreicht werden. Hierfür wird im Designer die Eigenschaft "Anzahl der Ausfertigungen" auf einen Wert größer eins gesetzt. Dann steht für die Layout-Bereiche die Funktion "IssueIndex" zur Verfügung, so dass z.B. ein Bereich mit der Bedingung "IssueIndex()==1" (Original) und ein weiterer mit der Bedingung "IssueIndex()==2" (Kopie) erstellt werden kann.

Die Objekte im Designer erhalten eine neue Eigenschaft "Darstellungsbedingung für Ausfertigungsdruck", mit der auf ganz ähnliche Weise der Wasserzeichendruck realisiert werden kann.

Demonstriert wird die Verwendung z.B. in der List & Label Beispielanwendung (im Startmenü direkt auf der Basisebene) unter Design > Rechnung > Rechnung mit Ausfertigungsdruck.

2.3.6 Designer anpassen und erweitern

Der Designer ist für die Anwendung keine "Black Box", sondern kann in vielen Bereichen beeinflusst werden. Neben dem Sperren von Funktionen und Menüpunkten können eigene Elemente hinzugefügt werden, die Berechnungen, Aktionen oder Ausgaben in die Funktionslogik verlegen.

Menüpunkte, Objekte und Funktionen sperren

Dreh- und Angelpunkt für die Designereinschränkung ist die DesignerWorkspace-Eigenschaft des ListLabel-Objekts. Diese bietet die in der folgenden Tabelle aufgelisteten Eigenschaften für die Designereinschränkung.

Eigenschaft	Funktion
ProhibitedActions	Diese Eigenschaft dient dazu, einzelne Menüpunkte aus dem Designer zu entfernen.
ProhibitedFunctions	Diese Eigenschaft dient dazu, einzelne Funktionen aus dem Designer zu entfernen.
ReadOnlyObjects	Diese Eigenschaft dient dazu, Objekte im Designer gegen Bearbeitung zu sperren. Die Objekte sind weiterhin sichtbar, können aber innerhalb des Designers nicht bearbeitet oder gelöscht werden.

Das folgende Beispiel zeigt, wie der Designer so angepasst werden kann, dass kein neues Projekt mehr angelegt werden kann. Zudem wird die Funktion "ProjectPath$" entfernt und das Objekt "Demo" gegen Bearbeitung gesperrt.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Designer einschränken
LL.DesignerWorkspace.ProhibitedActions.Add(LlDesignerAction.FileNew);
LL.DesignerWorkspace.ProhibitedFunctions.Add("ProjectPath$");
LL.DesignerWorkspace.ReadOnlyObjects.Add("Demo");

LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
```

```
LL.DataSource = CreateDataSet()

' Designer einschränken
LL.DesignerWorkspace.ProhibitedActions.Add(LlDesignerAction.FileNew)
LL.DesignerWorkspace.ProhibitedFunctions.Add("ProjectPath$")
LL.DesignerWorkspace.ReadOnlyObjects.Add("Demo")

LL.Design()
LL.Dispose()
```

Designer erweitern

Der Designer kann um eigene Funktionen, Objekte und Aktionen erweitert werden.

Eigene Funktionen können dafür verwendet werden, komplexere Berechnungen in die Applikation zu verlegen bzw. Funktionalitäten nachzurüsten, die im Standardumfang des Designers nicht vorhanden sind.

Ein Beispiel für das Hinzufügen einer eigenen Funktion findet sich im Abschnitt "Designer um eigene Funktion erweitern".

Beispiele für eigene Objekte oder Aktionen sowie eine weitere eigene Funktion zeigt das "Designer Erweiterungs-Beispiel", das unter "Sonstiges" im .NET-Beispielbereich im Startmenü zu finden ist.

2.3.7 Objekte im Designer

Einige Objekte im Designer dienen nur der grafischen Gestaltung (z.B. Linie, Rechteck, Ellipse). Die meisten anderen Objekte interagieren aber mit den zur Verfügung gestellten Daten. Hierfür stehen z.T. eigene Datentypen zur Verfügung oder es gibt Konvertierungsfunktionen, die die Umwandlung von Inhalten erlauben, so dass diese im jeweiligen Objekt verwendet werden können. Die folgende Aufstellung gibt einen Überblick über die am häufigsten verwendeten Objekte, die zugehörigen Datentypen und Designerfunktionen zur Umwandlung von Inhalten.

Die Hinweise für die Einzelobjekte gelten in gleicher oder ähnlicher Weise auch für (Bild-, Barcode-, usw.) Spalten in Tabellenelementen.

Text

Ein Textobjekt besteht aus mehreren Absätzen. Jeder dieser Absätze hat einen eigenen Inhalt. Dies kann entweder direkt eine Variable sein oder alternativ eine Formel, die mehrere Dateninhalte kombiniert. Für die Darstellung einzelner Variablen ist in der Regel keine spezielle Konvertierung notwendig. Sollen mehrere Variablen unterschiedlichen Typs (s. Abschnitt "Datentypen") innerhalb einer Formel kombiniert werden, müssen die einzelnen Bestandteile auf den gleichen Datentypen (z.B. Zeichenkette) konvertiert werden. Ein Beispiel für die Kombination von Zahlen und Zeichenketten wäre:

```
"Gesamtsumme: "+Str$(Sum(Artikel.Preis),0,2)
```

Die folgende Tabelle listet einige der Konvertierungsfunktionen auf, die in diesem Zusammenhang häufiger gebraucht werden.

Von / In	Datum	Zahl	Zeichnung	Barcode	Text
Datum	-	DateToJulian()	-	-	Date$()
Zahl	JulianToDate()	-	-	Barcode(Str$())	FStr$() Str$()
Zeichnung	-	-	-	-	Drawing$()
Barcode	-	Val(Barcode$())		-	Barcode$()
Text	Date()	Val()	Drawing()	Barcode()	-

Bild

Der Inhalt eines Bildobjekts wird über die Eigenschaftsliste bestimmt. Die Eigenschaft **Datenquelle** bietet die drei Werte **Dateiname**, **Formel** und **Variable** an.

- Die Einstellung **Dateiname** dient zur Verwendung einer festen Datei wie z.B. eines Firmenlogos. Wenn die Datei nicht mitgeliefert werden soll, kann sie auch direkt in den Bericht eingebettet werden, der Dateiauswahldialog bietet eine entsprechende Option.
- Über **Formel** kann der Inhalt über eine Zeichenkette festgelegt werden, die einen Pfad enthält. Die dafür benötigte Funktion ist "Drawing".
- Über **Variable** können schon als Bild übergebene Inhalte dargestellt werden (s. Abschnitt "Datentypen").

Barcode

Der Inhalt eines Barcodeobjekts wird über einen Dialog bestimmt. Dieser bietet als Datenquelle die drei Optionen **Text**, **Formel** und **Variable** an.

- Die Einstellung **Text** dient zur Verwendung eines festen Texts/Inhalts im Barcode. Zusätzlich zum Inhalt können der Typ und – z.B. bei 2D-Barcodes – weitere Eigenschaften zur Fehlerkorrektur oder Codierung eingestellt werden.
- Über **Formel** kann der Inhalt über eine Zeichenkette festgelegt werden, die den Barcodeinhalt enthält. Die dafür benötigte Funktion ist "Barcode".
- Über **Variable** können schon als Barcode übergebene Inhalte dargestellt werden (s. Abschnitt "Datentypen").

RTF-Text

Der Inhalt eines RTF-Textobjekts wird über einen Dialog bestimmt. Dieser bietet unter **Quelle** die Optionen **(freier Text)** oder eine Auswahl evtl. übergebener RTF-Variablen (s.u.)

- Die Einstellung **(freier Text)** dient zur Verwendung eines festen Texts/Inhalts im RTF-Objekt. Innerhalb des Objekts kann an jeder Stelle ein Dateninhalt verwendet werden (z.B. für personalisierte Serienbriefe), indem in der Toolleiste auf das Formelsymbol geklickt wird.
- Über die Auswahl einer Variablen können schon als RTF übergebene Inhalte dargestellt werden (s. Abschnitt "Datentypen").

HTML

Der Inhalt eines HTML-Objekts wird über einen Dialog bestimmt. Dieser bietet als Datenquelle die drei Optionen **Dateiname**, **URL** und **Formel** an.

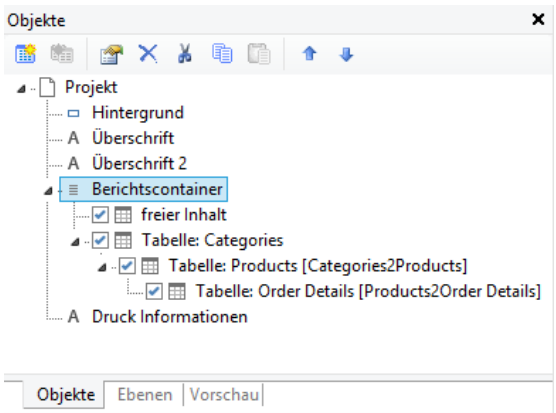
- Die Einstellung **Dateiname** dient zur Verwendung einer festen HTML-Datei.
- Über **URL** kann eine URL angegeben werden, von der der HTML-Inhalt heruntergeladen werden soll.
- Über **Formel** können schon als HTML-Stream übergebene Inhalte dargestellt werden (s. Abschnitt "Datentypen").

Siehe dazu auch die Bemerkung für "HTML-formatierter Text" im Abschnitt "Variablen, Felder und Datentypen".

2.3.8 Berichtscontainer

Der Berichtscontainer ist das zentrale Element für Listen-Projekte. Er erlaubt die Darstellung von tabellarischen Daten (auch mehrspaltig oder verschachtelt), Statistiken und Charts sowie Kreuztabellen. Die Daten können auch mehrfach in unterschiedlicher Form ausgegeben werden – z.B. zunächst eine grafische Auswertung der Verkäufe über die Jahre und anschließend eine detaillierte tabellarische Aufstellung.

Die Inhalte des Containers werden unterhalb des Berichtscontainerobjekts im Toolfenster "Objekte" innerhalb des Designers eingeblendet. Über dieses Fenster können neue Inhalte hinzugefügt oder bestehende bearbeitet werden. Das Fenster dient als eine Art "Drehbuch" für den Bericht, da darin exakt der Ablauf der einzelnen Berichtselemente zu sehen ist.



Damit der Berichtscontainer zur Verfügung steht, muss ein Datenprovider (vgl. Abschnitt "Datenprovider") als Datenquelle verwendet werden. Prinzipiell ist es auch möglich, den gesamten Druck über die low-level API-Funktionen des LICore-Objekts selbst durchzuführen, dies ist aber nicht die empfohlene Vorgehensweise, da dann viele Features (Designervorschau, Drilldown, Berichtscontainer,...) eigens unterstützt werden müssen. Viel sinnvoller ist es, im Zweifel einen eigenen Datenprovider zu schreiben. Hinweise dazu im Abschnitt "Datenbankunabhängige Inhalte".

Alle mitgelieferten Listen-Beispiele für das .NET Framework verwenden den Berichtscontainer und liefern so Anschauungsmaterial für die verschiedenen Einsatzzwecke.

Eine detaillierte Beschreibung für die Verwendung dieses Elements findet sich im Designerhandbuch im Abschnitt "Berichtscontainer einfügen".

2.3.9 Objektmodell (DOM)

Während der Designer eine sehr komfortable und mächtige Oberfläche zur Bearbeitung der Projektdateien bietet, kann es oft auch gewünscht sein, Objekt- oder Berichtseigenschaften direkt per Code zu bestimmen. So kann die Anwendung z.B. dem Anwender einen vorgeschalteten Dialog zur Datenvorauswahl anbieten und den Designer bereits mit einem so vorbereiteten Projekt öffnen. Ein Beispiel hierfür zeigt das Beispiel "Einfaches DOM-Beispiel", das unter "Sonstiges" im .NET-Beispielbereich im Startmenü zu finden ist.

Der Zugriff auf das Objektmodell ist erst ab der Professional Edition möglich.

Die folgende Tabelle listet die wichtigsten Klassen und Eigenschaften aus dem Namespace combit.ListLabel20.Dom auf.

Klasse	Funktion
ProjectList ProjectLabel ProjectCard	Die eigentlichen Projektklassen. Diese stellen das Wurzelelement des Projektes dar. Schlüsselmethoden sind Open, Save und Close.
<Projekt>.Objects	Eine Auflistung der Objekte innerhalb des Projekts. Die Objekte sind von ObjectBase abgeleitet und verfügen jeweils über eigene Eigenschaften und ggf. Auflistungen (z.B. Textabsätze).
<Projekt>.Regions	Eine Auflistung der Layout-Bereiche des Projekts. Hierüber kann z.B. eine seitenabhängige Druckersteuerung realisiert werden. Weitere Informationen finden sich im Abschnitt "Layout-Bereiche".
ObjectText	Repräsentiert ein Textobjekt. Schlüsseleigenschaft ist Paragraphs, der eigentliche Inhalt des Texts.
ObjectReportContainer	Repräsentiert einen Berichtscontainer. Schlüsseleigenschaft ist SubItems, der eigentliche Inhalt des Berichtscontainers.

Klasse	Funktion
SubItemTable	Repräsentiert eine Tabelle innerhalb des Berichtscontainers. Diese besteht aus verschiedenen Zeilenbereichen (Lines-Eigenschaft), die wiederum verschiedene Spalten (Columns-Eigenschaft einer Zeile) haben.

Innerhalb der einzelnen Klassen findet sich über die IntelliSense-Unterstützung recht einfach die gesuchte Eigenschaft. Eine vollständige Referenz über alle Klassen liefert die Komponentenhilfe für .NET.

2.3.10 List & Label in WPF Applikationen

Da List & Label selbst eine nicht-visuelle Komponente ist, kann es in WPF Applikationen genau wie in WinForms Applikationen genutzt werden. Der Designer selbst ist kein WPF Fenster, was aber die Funktionalität nicht beeinflusst. Als Ersatz für das WinForms PreviewControl liefern wir einen WPF Viewer mit, der zur Anzeige der Vorschaudateien genutzt werden kann.

Bitte beachten Sie, dass der WPF-Viewer für die Anzeige der Dokumente zwingend auf den installierten XPS Document Writer angewiesen ist. Wenn dieser Treiber auf dem Zielsystem nicht vorhanden ist, können keine Dokumente angezeigt werden.

2.3.11 Fehlerhandling mit Exceptions

List & Label definiert eine Reihe eigener Exceptions, die alle von der gemeinsamen Basisklasse ListLabelException abgeleitet sind und so zentral abgefangen werden können. Wenn die Applikation ein eigenes Exception-Handling vornehmen soll, können Aufrufe an List & Label in einen Exceptionhandler gekapselt werden:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

try
{
    LL.Design();
}
catch (ListLabelException ex)
{
    MessageBox.Show(ex.Message);
}
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()
```

```
Try
        LL.Design()
Catch ex As ListLabelException
        MessageBox.Show(ex.Message)
End Try
LL.Dispose()
```

Die Message-Eigenschaft der Exception-Klasse enthält einen Fehlertext, der – wenn ein entsprechendes Sprachkit vorhanden ist – in der Regel auch lokalisiert ist und so direkt dem Anwender angezeigt werden kann.

Eine vollständige Referenz über alle Exception-Klassen liefert die Komponentenhilfe für .NET.

2.3.12 Debugging

Probleme die auf dem Entwicklerrechner auftreten können meist leicht gefunden werden – hier kann direkt mit den üblichen Features der Entwicklungsumgebung gearbeitet werden und ein Problem so recht schnell eingegrenzt werden. Der erste Schritt besteht darin, eventuell auftretende Exceptions abzufangen und deren Ursache zu überprüfen (vgl. Abschnitt "Fehlerhandling mit Exceptions").

Als Entwicklungskomponente wird List & Label aber natürlich unter einer Vielzahl verschiedener Konstellationen bei den Endanwendern ausgeführt. Um Probleme dort möglichst einfach zu finden, steht ein eigenes Debug-Tool zur Verfügung, das bei selten oder nur auf bestimmten Systemen auftretenden Problemen eine Protokollierungsfunktion bietet, mit deren Hilfe Probleme auch auf Systemen ohne Debugger untersucht werden können.

Natürlich kann die Logging-Funktion auch auf dem Entwicklerrechner genutzt werden und bietet auch dort die Möglichkeit, sämtliche Aufrufe und Rückgabewerte schnell auf einen Blick zu prüfen.

Protokolldatei anfertigen

Tritt ein Problem nur auf einem Kundensystem auf, sollte auf diesem zunächst eine Protokolldatei erstellt werden. Hierzu dient das Tool Debwin3, welches im "Tools"-Verzeichnis der List & Label-Installation installiert wird.

Debwin3 muss vor der Applikation gestartet werden. Über Logging > Force Debug Mode wird die Protokollierung erzwungen. Wenn anschließend die Applikation gestartet wird, werden sämtliche Aufrufe an die Komponente mit ihren Rückgabewerten sowie einige Zusatzinformationen zu Modulversionen, Betriebssystem etc. protokolliert.

Jede unter .NET geworfene Exception entspricht im Protokoll einem negativen Rückgabewert einer Funktion. Im Protokoll finden sich meist weitere hilfreiche Informationen, eine typische Ausgabe könnte wie folgt aussehen.

```
CMLL20  : 12:30:02.082 00000df0/02 3  llSelectFileDlgTitleEx(2,0X001310BE,
'(NULL)',0x00008002,0X0344F7C8,260,00000000)
CMLL20  : 12:30:03.672 00000df0/02 4  =-99 (Der Benutzer hat den Vorgang
abgebrochen.) -> '*.lst'
```

Man sieht zunächst die DLL, die die Ausgabe erzeugt hat, eine Timestamp für die Ausgabe, die Thread-ID des ausgebenden Threads, eine fortlaufende Nummer sowie den eigentlichen Aufruf mit allen Parametern. In der Folgezeile erfolgt dann die Rückgabe eines Fehlercodes (-99) und eine Erklärung dafür – in diesem Falle hat der Benutzer den Dateiauswahldialog mit "Abbrechen" beendet.

Soll die Anwendung ohne Hilfe von Debwin3 Debug-Protokolle erstellen, kann dies z.B. über die Konfigurationsdatei der Anwendung erreicht werden. Eine Protokollierung kann darin wie folgt erzwungen werden:

```
<configuration>
 <appSettings>
  <add key="ListLabel DebugLogFilePath"
    value="c:\users\public\debug.log "/>
  <add key="ListLabel EnableDebug" value="1"/>
 </appSettings>
</configuration>
```

2.4 Beispiele

Die Beispiele in diesem Abschnitt zeigen, wie einige typische Aufgabenstellungen gelöst werden. Der Code kann als Kopiervorlage für eigene Erweiterungen dienen.

Aus Übersichtlichkeitsgründen wird auf die übliche Fehlerbehandlung verzichtet. Alle Exceptions werden somit direkt in der Entwicklungsumgebung aufgefangen. Für "echte" Applikationen empfehlen wir ein Exception-Handling wie im Abschnitt "Fehlerhandling mit Exceptions" beschrieben.

2.4.1 Einfaches Etikett

Um ein Etikett auszugeben, wird der Projekttyp LlProject.Label benötigt. Wenn eine Datenquelle mit mehreren Tabellen wie z.B. ein DataSet angebunden wird, kann über die Eigenschaft DataMember ausgewählt werden, welche Daten im Etikett bereitgestellt werden sollen.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Produkte als Quelldaten
LL.DataMember = "Produkte";

// Etikett als Projekttyp wählen
LL.AutoProjectType = LlProject.Label;

// Designer aufrufen
LL.Design();

// Drucken
```

```
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Produkte als Quelldaten
LL.DataMember = "Produkte"

' Etikett als Projekttyp wählen
LL.AutoProjectType = LlProject.Label

' Designer aufrufen
LL.Design()

'Drucken
LL.Print()
LL.Dispose()
```

2.4.2 Einfache Liste

Der Druck und das Design von einfachen Listen ist der "Standardfall" und kann schon mit wenigen Codezeilen gestartet werden:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Designer aufrufen
LL.Design();

// Drucken
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Designer aufrufen
LL.Design()

'Drucken
LL.Print()
LL.Dispose()
```

2.4.3 Sammelrechnung

Eine Sammelrechnung ist ein impliziter Seriendruck. Die Kopf- oder Elterndaten enthalten für jeden Beleg einen Datensatz, der 1:n mit den Detail- oder Kinddaten verknüpft ist. Um einen solchen Beleg zu designen und zu drucken, muss List & Label die Elterntabelle über die DataMember-Eigenschaft bekannt gegeben werden. Zudem muss die AutoMasterMode-Eigenschaft auf AsVariables gesetzt werden, wie im folgenden Beispiel gezeigt:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Bestelldaten als Variablen
LL.DataMember = "Rechungskopf";
LL.AutoMasterMode = LlAutoMasterMode.AsVariables;

// Designer aufrufen
LL.Design();

// Drucken
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Bestelldaten als Variablen
LL.DataMember = "Rechnungskopf"
LL.AutoMasterMode = LlAutoMasterMode.AsVariables

' Designer aufrufen
LL.Design()

'Drucken
LL.Print()
LL.Dispose();
```

2.4.4 Karteikarte mit einfachen Platzhaltern drucken

Der Druck eines ganzseitigen Projektes, das einfach nur an verschiedenen Stellen durch die Applikation bestimmte Platzhalter enthält ist am einfachsten über das Binden an ein passendes Objekt zu bewerkstelligen:

C#:

```
public class DataSource
{
```

```
    public string Text1 { get; set; }
    public double Number1 { get; set; }
    ...
}

// Datenquelle vorbereiten
object dataSource = new DataSource { Text1 = "Test", Number1 = 1.234 };
ListLabel LL = new ListLabel();
LL.DataSource = new ObjectDataProvider(dataSource);
LL.AutoProjectType = LlProject.Card;

// Designer aufrufen
LL.Design();

// Drucken
LL.Print();
LL.Dispose();
```

VB.NET:

```
Public Class DataSource
        Dim _text1 As String
        Dim _number As Double

        Public Property Text1() As String
                Get
                        Return _text1
                End Get
                Set(ByVal value As String)
                        _text1 = value
                End Set
        End Property
        Public Property Number1() As Double
                Get
                        Return _number
                End Get
                Set(ByVal value As Double)
                        _number = value
                End Set
        End Property
End Class

' Datenquelle vorbereiten
Dim dataSource As Object = New DataSource()
dataSource.Text1 = "Test"
dataSource.Number1 = 1.234
Dim LL As New ListLabel()
LL.DataSource = New ObjectDataProvider(dataSource)
LL.AutoProjectType = LlProject.Card

' Designer aufrufen
```

```
LL.Design()

' Drucken
LL.Print()
LL.Dispose()
```

2.4.5 Unterberichte

Die Strukturierung von Berichten mit Hilfe des Berichtscontainers ist ein reines Designerfeature. Insofern unterscheidet sich die Ansteuerung nicht von der für "normale" Listen, wie im Abschnitt "Einfache Liste" gezeigt.

Für die Ausgabe von Untertabellen wird vorausgesetzt, dass Eltern- und Kinddaten durch eine Relation miteinander verknüpft sind. Dann kann im Berichtscontainer zunächst ein Tabellenelement für die Elterntabelle gestaltet werden. Im nächsten Schritt kann über die Funktionsleiste im Objektfenster ein Unterelement mit den Kinddaten eingefügt werden.

Zur Druckzeit wird dann für jeden Datensatz der Elterntabelle automatisch der passende Kind-Unterbericht eingefügt. Demonstriert wird die Verwendung z.B. in der List & Label Beispielanwendung (im Startmenü direkt auf der Basisebene) unter Design > Erweiterte Beispiele > Unterberichte und Relationen.

2.4.6 Charts

Auch die Diagrammfunktion wird durch den Berichtscontainer automatisch unterstützt. Die Ansteuerung erfolgt also auch hier wie im Abschnitt "Einfache Liste" gezeigt.

Die List & Label Beispielanwendung (im Startmenü direkt auf der Basisebene) enthält unter Design > Erweiterte Beispiele eine Vielzahl von verschiedenen Chart-Beispielen.

2.4.7 Kreuztabellen

Wenig überraschend werden Kreuztabellen analog den Hinweisen im Abschnitt "Einfache Liste" angesteuert.

Die List & Label Beispielanwendung (im Startmenü direkt auf der Basisebene) enthält unter Design > Erweiterte Beispiele eine Vielzahl von verschiedenen Kreuztabellen-Beispielen.

2.4.8 Datenbankunabhängige Inhalte

Nicht immer liegen alle Daten in einer Datenbank oder einem DataSet vor. So kann es erwünscht sein, zusätzlich zu den Daten aus der Datenquelle weitere Daten wie z.B. den Benutzernamen innerhalb der Applikation, den Projektnamen oder ähnliche Informationen auszugeben. In anderen Fällen scheint auf den ersten Blick kein passender Datenprovider in Sicht zu sein. Diese Fälle werden in den folgenden Abschnitten betrachtet.

Zusätzliche Inhalte übergeben

Wenn nur einige wenige Variablen oder Felder zusätzlich zu den Daten der Datenbindung hinzugefügt werden sollen, gibt es zwei Möglichkeiten:

- Wenn die Daten über die Laufzeit des Berichts konstant sind, können sie einfach vor dem Design- oder Druckaufruf per LL.Variables.Add hinzugefügt werden.
- Wenn die Daten sich von Seite zu Seite oder sogar Zeile zu Zeile ändern, können die Informationen innerhalb des AutoDefineNewPage oder AutoDefineNewLine-Ereignisses per LL.Fields.Add übergeben werden.

Das folgende Beispiel zeigt beide Ansätze:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Zusätzliche Datenfelder anmelden
LL.Variables.Add("Zusatzdaten.Benutzername", GetCurrentUserName());
LL.Variables.Add("Zusatzdaten.Projektname", GetCurrentProjectName());
…

// Ereignisbehandlung für eigene Felder hinzufügen
LL.AutoDefineNewLine += new AutoDefineNewLineHandler(LL_AutoDefineNewLine);

// Designer aufrufen
LL.Design();

// Drucken
LL.Print();
LL.Dispose();

…

void LL_AutoDefineNewLine(object sender, AutoDefineNewLineEventArgs e)
{
    // ggf. zum nächsten Datensatz wechseln, wenn dies notwendig ist
    // GetCurrentFieldValue ist eine Funktion Ihrer Applikation, die
    // den Inhalt des Datenfeldes liefert.
    LL.Fields.Add("Zusatzdaten.Zusatzfeld", GetCurrentFieldValue());
}
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Zusätzliche Datenfelder anmelden
LL.Variables.Add("Zusatzdaten.Benutzername", GetCurrentUserName())
LL.Variables.Add("Zusatzdaten.Projektname", GetCurrentProjectName())
…

' Designer aufrufen
LL.Design()

' Drucken
```

```
LL.Print()
LL.Dispose()

…

Sub LL_AutoDefineNewLine(sender As Object, e As AutoDefineNewLineEventArgs)_
Handles LL.AutoDefineNewLine
    ' ggf. zum nächsten Datensatz wechseln, wenn dies notwendig ist
    ' GetCurrentFieldValue ist eine Funktion Ihrer Applikation, die
    ' den Inhalt des Datenfeldes liefert.
    LL.Fields.Add("Zusatzdaten.Zusatzfeld", GetCurrentFieldValue())
End Sub
```

Daten aus der Datenbindung unterdrücken

Einzelne, nicht benötigte Felder oder Variablen (z.B. ID-Felder die im Druck nicht benötigt werden) können mit Hilfe der AutoDefineField- und AutoDefineVariable-Ereignisse unterdrückt werden.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Ereignisbehandlung für Feldunterdrückung hinzufügen
LL.AutoDefineField += new AutoDefineElementHandler(LL_AutoDefineField);

// Designer aufrufen
LL.Design();

// Drucken
LL.Print();
LL.Dispose();

…

void LL_AutoDefineField(object sender, AutoDefineElementEventArgs e)
{
    if (e.Name.EndsWith("ID"))
        e.Suppress = true;
}
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Designer aufrufen
LL.Design()

' Drucken
```

```
LL.Print()
LL.Dispose()

…

Sub LL_AutoDefineField(sender As Object, e As AutoDefineElementEventArgs)_
Handles LL.AutoDefineNewField
        If e.Name.EndsWith("ID") Then
                e.Suppress = True
        End If
End Sub
```

Vollständig eigene Datenstrukturen/-inhalte

Für augenscheinlich nicht direkt unterstützte Dateninhalte findet sich meist trotzdem ein passender Provider. Businessdaten aus Anwendungen können in der Regel über den Objektdatenprovider übergeben werden, liegen die Daten in kommaseparierter Form vor kann der Datenprovider aus dem "Dataprovider"-Beispiel verwendet werden. Viele andere Datenquellen unterstützen die Serialisierung nach XML, so dass dann der XmlDataProvider verwendet werden kann.

Natürlich kann aber auch eine eigene Klasse verwendet werden, die die DataProvider-Schnittstelle unterstützt. Ein guter Startpunkt hierfür ist das DataProvider-Beispiel, das einen einfachen CSV-Datenprovider demonstriert. Oft kann auch eine der vorhandenen Klassen als Basisklasse verwendet werden. Wenn z.B. eine Datenquelle angebunden werden soll, die die IDbConnection-Schnittstelle implementiert, kann von DbConnectionDataProvider geerbt werden. Hier muss dann lediglich die Init-Methode überschrieben werden, in der die verfügbaren Tabellen und Relationen bereitgestellt werden müssen. In der Komponentenhilfe für .NET findet sich ein Beispiel, wie dies z.B. für SQL-Serverdaten im SqlConnectionDataProvider gemacht wird. Die meisten Datenbanksysteme stellen ähnliche Mechanismen zur Verfügung.

Unter http://lldataproviders.codeplex.com finden sich Open Source-Implementierungen für eine Vielzahl weiterer Datenquellen wie MySQL, PostgreSQL, SQLite, DB2 und Oracle. Auch diese können ein guter Ausgangspunkt für eigene Provider sein.

2.4.9 Export

Die Exportformate lassen sich per Code vollständig "fernsteuern", so dass keine Benutzeraktion mehr notwendig ist. Zudem kann die Auswahl der Formate so eingeschränkt werden, wie es für den jeweiligen Bericht notwendig oder gewünscht ist.

Export ohne Benutzerinteraktion

Die Ansteuerung erfolgt über die ExportConfiguration-Klasse der ListLabel-Komponente.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Ziel und Pfad (hier: PDF) und Projektdatei angeben
```

```
ExportConfiguration exportConfig = new ExportConfiguration(LlExportTarget.Pdf,
"<Zieldateiname mit Pfad>", "<Projektdateiname mit Pfad>");

// Ergebnis anzeigen
exportConfig.ShowResult = true;

// Export starten
LL.Export(exportConfig);
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Ziel und Pfad (hier: PDF) und Projektdatei angeben
Dim exportConfiguration As New ExportConfiguration(LlExportTarget.Pdf,
"<Zieldateiname mit Pfad>", "<Projektdateiname mit Pfad>")

' Ergebnis anzeigen
exportConfiguration.ShowResult = True
' Export starten
LL.Export(exportConfiguration)
LL.Dispose()
```

Viele weitere Optionen lassen sich über die ExportOptions Auflistung der ListLabel Komponente setzen.

Einschränkung von Exportformaten

Wenn dem Endanwender nur einzelne Exportformate erlaubt sein sollen, kann die Liste der Formate auf genau diese eingeschränkt werden. Dies ist über das Setzen der Option LlOptionString.Exports_Allowed möglich. Eine Liste der verfügbaren Formate findet sich im Abschnitt "Export".

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// nur PDF und Vorschau erlauben
LL.Core.LlSetOptionString(LlOptionString.Exports_Allowed, "PDF;PRV");

// Drucken
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()
```

```
' nur PDF und Vorschau erlauben
LL.Core.LlSetOptionString(LlOptionString.Exports_Allowed, "PDF;PRV")

' Drucken
LL.Print()
LL.Dispose()
```

2.4.10 Designer um eigene Funktion erweitern

Das folgende Beispiel zeigt, wie eine Funktion hinzugefügt werden kann, die es ermöglicht, den Wert eines Registrierungsschlüssels innerhalb eines Berichts abzufragen. Das Ergebnis der Funktion könnte dann z.B. in Darstellungsbedingungen für Objekte verwendet werden. Natürlich können die Eigenschaften der DesignerFunction-Klasse alternativ auch direkt im Eigenschaften-Fenster der Entwicklungsumgebung angelegt werden.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Funktion initialisieren
DesignerFunction RegQuery = new DesignerFunction();
RegQuery.FunctionName = "RegQuery";
RegQuery.GroupName = "Registrierung";
RegQuery.MinimalParameters = 1;
RegQuery.MaximumParameters = 1;
RegQuery.ResultType = LlParamType.String;
RegQuery.EvaluateFunction += new
EvaluateFunctionHandler(RegQuery_EvaluateFunction);

// Funktion hinzufügen
LL.DesignerFunctions.Add(RegQuery);

LL.Design();
LL.Dispose();

…

void RegQuery_EvaluateFunction(object sender, EvaluateFunctionEventArgs e)
{
    // Registrierungsschlüssel auslesen
    RegistryKey key = Registry.CurrentUser.OpenSubKey(@"Software\combit\");
    e.ResultValue = key.GetValue(e.Parameter1.ToString()).ToString();
}
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()
```

```
' Funktion initialisieren
Dim RegQuery As New DesignerFunction()
RegQuery.FunctionName = "RegQuery"
RegQuery.GroupName = "Registrierung"
RegQuery.MinimalParameters = 1
RegQuery.MaximumParameters = 1
RegQuery.ResultType = LlParamType.String

' Funktion hinzufügen
LL.DesignerFunctions.Add(RegQuery)

LL.Design()
LL.Dispose()

…

Sub RegQuery_EvaluateFunction(sender As Object,
      e As EvaluateFunctionEventArgs) Handles RegQuery.EvaluateFunction
    ' Registrierungsschlüssel auslesen
    Dim key As RegistryKey
    RegistryKey = Registry.CurrentUser.OpenSubKey("Software\combit\")
    e.ResultValue = key.GetValue(e.Parameter1.ToString()).ToString()
End Sub
```

2.4.11 Vorschaudateien zusammenfügen und konvertieren

Das Vorschauformat kann als Ausgangsformat verwendet werden, wenn z.B. mehrere Berichte zu einem zusammengefügt werden sollen oder neben einem direkten Ausdruck auch eine Archivierung als PDF gewünscht wird. Das folgende Beispiel zeigt einige Möglichkeiten der PreviewFile-Klasse.

C#:

```
// Vorschaudateien öffnen, Deckblatt mit Schreibzugriff
PreviewFile cover = new PreviewFile(@"<Pfad>\deckblatt.ll", false);
PreviewFile report = new PreviewFile(@"<Pfad>\report.ll", true);

// Bericht an Deckblatt anhängen
cover.Append(report);

// Gesamtbericht drucken
cover.Print();

// Bericht als PDF konvertieren
cover.ConvertTo(@"<Pfad>\report.pdf");

// Vorschaudateien freigeben
report.Dispose();
cover.Dispose();
```

VB.NET:

```
' Vorschaudateien öffnen, Deckblatt mit Schreibzugriff
Dim cover As New PreviewFile("<Pfad>\deckblatt.ll", False)
Dim report As New PreviewFile("<Pfad>\report.ll", True)

' Bericht an Deckblatt anhängen
cover.Append(report)

' Gesamtbericht drucken
cover.Print()

' Bericht als PDF konvertieren
cover.ConvertTo("<Pfad>\report.pdf")

' Vorschaudateien freigeben
report.Dispose()
cover.Dispose()
```

2.4.12 eMail-Versand

Der eMail-Versand kann ebenfalls über die Liste der Exportoptionen angesteuert werden (vgl. Abschnitt "Export ohne Benutzerinteraktion"), wenn Export und Versand in einem Arbeitsgang erfolgen sollen. Ein Beispiel hierfür zeigt das Export-Beispiel, das unter "Sonstiges" im .NET-Beispielbereich im Startmenü zu finden ist.

Unabhängig von einem vorherigen Export ist es aber über die MailJob-Klasse auch möglich, beliebige Dateien per eMail zu versenden. Dies ist insbesondere dann interessant, wenn aus einer Vorschaudatei als Quelle z.B. eine PDF-Datei generiert wird (vgl. Abschnitt "Vorschaudateien zusammenfügen und konvertieren") und diese versendet werden soll.

C#:

```
// Mailjob instanzieren
MailJob mailJob = new MailJob();

// Optionen setzen
mailJob.AttachmentList.Add(@"<Pfad>\report.pdf");
mailJob.To = "info@combit.net";
mailJob.Subject = "Hier kommt der Report";
mailJob.Body = "Bitte sehen Sie sich das Attachment an.";
mailJob.Provider = "XMAPI";
mailJob.ShowDialog = true;

// eMail versenden
mailJob.Send();
mailJob.Dispose();
```

VB.NET:

```
' Mailjob instanzieren
Dim mailJob As New MailJob()

' Optionen setzen
mailJob.AttachmentList.Add("<Pfad>\report.pdf")
mailJob.To = "info@combit.net"
mailJob.Subject = "Hier kommt der Report"
mailJob.Body = "Bitte sehen Sie sich das Attachment an."
mailJob.Provider = "XMAPI"
mailJob.ShowDialog = True

' eMail versenden
mailJob.Send()
mailJob.Dispose()
```

2.4.13 Projektdateien in Datenbank halten

Projektdateien können auch direkt in Datenbanken gespeichert werden. Neben der Möglichkeit, diese direkt aus der Datenbank zu entpacken und im lokalen Dateisystem zu speichern kann diese Arbeit auch auf List & Label abgewälzt werden. Die Print und Design-Methoden haben Überladungen, die die direkte Angabe eines Streams erlauben.

Bei Verwendung dieser Überladungen sind einige wichtige Verhaltensänderungen zu beachten. Hintergrund für diese ist das Fehlen eines lokalen Dateikontextes und damit verbunden die fehlende Möglichkeit, neue Dateien anzulegen:

- Im Designer ist es nicht möglich, ein neues Projekt anzulegen
- Die Menüpunkte **Datei** > **Speichern unter** und **Datei** > **Öffnen** sind nicht verfügbar
- Die Projektbaustein-Funktionalität ist deaktiviert
- Drilldown ist nicht verfügbar
- Die Designerfunktion "ProjectPath$" ist nicht verfügbar

Für den Designfall kann es natürlich passieren, dass der übergebene Stream modifiziert wird. In diesem Fall müssen Sie nach Ende des Designs den aktualisierten Stream in die Datenbank schreiben.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();
byte[] report = GetReportFromDatabase();
MemoryStream memStream = new MemoryStream(report);

LL.Print(LlProject.List, memStream);
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()
Dim report As Byte() = GetReportFromDatabase()
Dim memStream As New MemoryStream(report)

LL.Print(LlProject.List, memStream)
LL.Dispose()
```

2.4.14 Druck im Netzwerk

Beim Druck im Netzwerk gilt es, zwei Dinge zu beachten:

- Vorschaudateien werden in der Regel mit dem Namen der Projektdatei und der Endung "LL" im gleichen Verzeichnis wie die Projektdatei angelegt. Wenn also zwei Nutzer die gleiche Datei auf die Vorschau drucken wollen, wird der zweite Anwender eine Fehlermeldung erhalten. Dies kann durch die Verwendung von *LlPreviewSetTempPath()* verhindert werden (s. Beispiel unten).
- Ähnliches gilt für die Druckereinstellungsdateien. Auch diese werden – mit der aktuell gewählten Endung – im Verzeichnis der Projektdatei gesucht bzw. angelegt. Hier sollte *LlSetPrinterDefaultsDir()* verwendet werden.Dies ist auch wichtig, wenn die Drucker, die den Anwendern zur Verfügung stehen, von Arbeitsplatz zu Arbeitsplatz wechseln. Dies ist ein Grund, warum Sie die Druckereinstellungsdateien nicht an den Kunden redistributieren sollten.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// lokalen Temporärpfad setzen
LL.Core.LlPreviewSetTempPath(Path.GetTempPath());

// Druckoptionen sollten in benutzerspezifischem Unterverzeichnis
// abgelegt werden, damit Änderungen dauerhaft übernommen werden
LL.Core.LlSetPrinterDefaultsDir(<Pfad>);

LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' lokalen Temporärpfad setzen
LL.Core.LlPreviewSetTempPath(Path.GetTempPath())

' Druckoptionen sollten in benutzerspezifischem Unterverzeichnis
' abgelegt werden, damit Änderungen dauerhaft übernommen werden
```

```
LL.Core.LlSetPrinterDefaultsDir(<Pfad>)

LL.Print()
LL.Dispose()
```

Eine Alternative stellt die Verwendung der Stream-Überladungen der Print- und Designmethoden dar. Diese sorgen z.B. "automatisch" für die Speicherung der Druckoptionen im übergebenen Stream. Hinweise und ein Beispiel finden sich im Abschnitt "Projektdateien in Datenbank halten".

3. Programmieren mit der OCX Komponente

Parallel zu der VCL-Komponente und dem .NET Assembly erhalten Sie mit List & Label die OCX-Komponente(n) für die Integration in Ihre IDE. Das folgende Kapitel bezieht sich ausschließlich auf die Arbeit mit dieser Komponente. Wenn Sie nicht mit der OCX-Komponente arbeiten, können Sie diesen Abschnitt überspringen.

3.1 Einbindung der Komponente

Die Einbindung erfolgt mit Hilfe der Datei cmll20o.ocx, die das Control beinhaltet. Diese Datei finden Sie im Verzeichnis "Redistributierbare Dateien". Weitere Informationen zur Installation des OCX in Ihre IDE finden Sie in der Onlinehilfe Ihrer Entwicklungsumgebung.

Im Anschluss an die Einbindung des Controls sollte automatisch ein Symbol im Komponentenbereich der Werkzeugleiste bzw. Toolbox Ihrer IDE angelegt sein.

Sie können nun beginnen, List & Label durch die angebotenen Eigenschaften individuell an Ihre Bedürfnisse anzupassen und die benötigte Programmlogik zu implementieren. Hierzu bieten sich zwei unterschiedliche Ansätze an:

- Die einfachen Print- und Design-Methoden
- Eine eigene, iterative Druckschleife

Die erstgenannte Möglichkeit wird nachfolgend vorgestellt. Der iterative Ansatz entspricht weitgehend der direkten Verwendung der DLL und ist somit durch die allgemeine Beschreibung der List & Label API abgedeckt.

3.2 Einfache Print- und Design-Methoden

3.2.1 Funktionsweise

Die Print- und Design-Methoden des OCX-Controls implementieren eine standardisierte Druckschleife, die für den Großteil der einfacheren Anwendungen (Anwendungen, die nur mit einer Tabelle arbeiten) direkt verwendbar ist. Der Druck von mehreren Tabellen erfolgt über eine eigene Druckschleife (siehe Kapitel Drucken relationaler Daten). Die Daten werden bei diesem Ansatz innerhalb der Ereignisse *CmndDefineVariables* und *CmndDefineFields* an List & Label übergeben. Auf diese Weise können beliebige Datenquellen individuell angebunden werden. Die Ereignisargumente erlauben den Zugriff auf nützliche Informationen wie die übergebenen Benutzerdaten, den *Design*-Mode und so weiter. Über die Eigenschaft *pbLastRecord* wird der Druckschleife mitgeteilt, dass der letzte Datensatz erreicht wurde. Solange dies nicht der Fall ist, wird das jeweilige Ereignis wiederholt aufgerufen, um die Daten abzufragen.

Die Design-Methode stellt den Designer in einem modalen Pop-up Fenster dar, welches Ihr Anwendungsfenster überlagert.

Zusätzliche Optionen lassen sich im Ereignis *CmndSetPrintOptions* festlegen. Intern wird dieses Ereignis nach dem Aufruf von *LlPrintWithBoxStart()* aber vor dem eigentlichen Druck ausgelöst.

Eine sehr einfache Verwendung der Methode Print sieht wie folgt aus:

```
Private Sub ListLabel1_CmndDefineVariables(ByVal nUserData As Long, ByVal bDummy
As Long, pnProgressInPerc As Long, pbLastRecord As Long)

 Dim i As Integer

 For i = 0 To Recordset.Fields.Count - 1

   Select Recordset.Fields(i).Type
    Case 3, 4, 6, 7: para = LL_NUMERIC: content$ = Recordset.Fields(i)
    Case 8: para = LL_DATE_MS: a! = CDate(Recordset.Fields(i)): content$ = a!:
    Case 1: para = LL_BOOLEAN:  content$ = Recordset.Fields(i)
    Case Else: para = LL_TEXT: content$ = Recordset.Fields(i)
   End Select

   nRet = LL.LlDefineVariableExt(Recordset.Fields(i).Name, content$, para)
 Next i

 If bDummy = 0 Then
   pnProgressInPerc = Form1.Data1.Recordset.PercentPosition
   Recordset.MoveNext
 End If
End Sub
```

3.2.2 Verwendung des UserData-Parameters

Die Methoden *Print* und *Design* erlauben die Übergabe eines Parameters *UserData* vom Typ integer. Mithilfe dieses Parameters können Sie in den Ereignis verschiedene Daten für List & Label bereitstellen. So wäre es z.B. möglich in den Events anhand des Parameters sowohl Daten für den Rechnungsdruck als auch für eine Kundenliste bereit zu stellen.

3.3 Übergabe von ungebundenen Variablen und Feldern

Die Übergabe von Variablen und Feldern entspricht dem regulären Prinzip von List & Label. Für die Anmeldung stehen drei "API-Varianten" zur Verfügung.

API	Beschreibung
LlDefineVariable	Definiert eine Variable vom Typ LL_TEXT und deren Inhalt.
LlDefineVariableExt	Wie oben und zusätzlich kann der List & Label Datentyp mit übergeben werden.
LlDefineVariableExtHandle	Wie oben, wobei der Inhalt nun ein Handle sein muss.

Ein Beispiel für die Anmeldung einer Variablen vom Typ Text sieht folgendermaßen aus:

```
LL.LlDefineVariableExt("MeineVariable", "Inhalt", LL_TEXT)
```

Sie finden die Konstanten für die List & Label Datentypen in der Unit cmll20.bas (VB) in Ihrem List & Label Installationsverzeichnis wieder.

3.3.1 Bilder

Um Bilder zu übergeben, die als Datei auf dem System vorhanden sind, verwenden Sie

```
LL.LlDefineVariableExt ("Picture", <Dateipfad>, LL_DRAWING)
```

Die Übergabe von Grafiken erfolgt mittels der "API-Variante" *LlDefineVariableExtHandle()*. Weitere Informationen zu dieser Funktion finden Sie im Kapitel "API Referenz".

3.3.2 Barcodes

Die Übergabe von Barcodes wird durch die Verwendung der Konstante *LL_BARCODE...* erreicht. Ein Beispiel für die Anmeldung einer Barcode-Variable vom Typ EAN13 sieht wie folgt aus:

```
LL.LlDefineVariableExt('EAN13', '123456789012',LL_BARCODE_EAN13);
```

3.4 Auswahl der Sprache

Der List & Label Designer liegt in zahlreichen Sprachen vor. Die Komponente unterstützt Sie somit intensiv bei der Realisierung von mehrsprachigen Desktop-Applikationen. Es gibt zwei Möglichkeiten, List & Label die zu verwendende Sprache mitzuteilen.

- Weisen Sie der Eigenschaft Language die entsprechende Sprache zu:

```
LL.Language = CMBTLANG_GERMAN
```

- Stellen Sie die Sprache direkt an der Komponente ein

Hinweis: Bitte beachten Sie, dass Sie zur Anzeige der jeweiligen Sprache das entsprechende Language Kit benötigen. Welche Kits wir derzeit anbieten und was diese kosten, erfahren Sie in unserem Online-Shop unter www.combit.net.

3.5 Arbeiten mit Ereignissen

List & Label bietet eine Vielzahl von Callbacks an, die bei der List & Label OCX-Komponente als Events implementiert wurden.

Eine Umfangreiche Beschreibung der zur Verfügung stehenden Events finden Sie in der mitgelieferten Onlinehilfe zur OCX-Komponente.

3.6 Anzeigen einer Vorschaudatei

Zur Anzeige einer Vorschau existiert eine separate Komponente. Um diese Komponente nutzen zu können, müssen Sie die Datei cmll20v.ocx in Ihre IDE einbinden. Es bietet spezialisierte Möglichkeiten zur Verwendung des List & Label Vorschauformates. Es ist z.B. möglich aus dem Control heraus einen PDF-Export zu starten. Zusätzlich kann das Control an die eigenen Bedürfnisse angepasst werden, indem Toolbar-Buttons über Eigenschaften des Controls aus- bzw. eingeblendet werden können. Es ist auch möglich auf die Click-Ereignisse der Buttons zu reagieren und evtl. eigene Behandlungsroutinen zu hinterlegen.

3.7 Arbeiten mit Vorschau-Dateien

Die List & Label Storage-API erlaubt den Zugriff auf die LL-Vorschaudateien. Sie können allgemeine Informationen oder die einzelnen Seiten abfragen, mehrere Dateien zusammenfügen und Benutzerdaten abspeichern. Zur Verwendung der Storage-API müssen Sie die Unit cmls20.bas in Ihr Projekt aufnehmen oder die Funktionen über das OCX-Control aufrufen.

3.7.1 Öffnen einer Vorschaudatei

Sie können die Vorschaudatei mit Hilfe der Funktion *LlStgsysStorageOpen()* öffnen. Über eine ganze Reihe von weiteren Funktionen stehen nun allgemeine Informationen über die Datei zur Verfügung.

Visual Basic:

```
Dim hStgOrg As Long

hStgOrg = LL.LlStgsysStorageOpen("C:\Test.ll", "", False, True)
```

3.7.2 Zusammenführen mehrerer Vorschaudateien

Sie können mehrere Vorschaudateien zusammenführen. Hierzu müssen Sie zunächst die Zieldatei öffnen. Da ein schreibender Zugriff notwendig ist, müssen Sie für den zweiten Parameter ReadOnly false übergeben. Anschließend können Sie mit Hilfe der Funktion *LlStgSysAppend()* die Dateien zusammenführen.

Visual Basic:

```
Dim hStgOrg As Long
Dim hStgAppend As Long

hStgOrg = LL.LlStgsysStorageOpen("C:\Test1.ll", "", False, True)
hStgAppend = LL.LlStgsysStorageOpen("C:\Test2.ll", "", False, True)

LL.LlStgsysAppend hStgOrg, hStgAppend
```

```
LL.LlStgsysStorageClose hStgOrg
LL.LlStgsysStorageClose hStgAppend
```

3.7.3 Debugging

Das Debugging der OCX-Komponente können Sie aktivieren, indem Sie die Eigenschaft *DebugMode* im Quellcode auf "1" setzen, z.B.:

```
LL.LlSetDebug = 1
```

Weitere Informationen über das Debugtool Debwin finden Sie in Kapitel "Fehlersuche mit Debwin".

3.8 Erweiterung des Designers

List & Label bietet vielfältige Möglichkeiten, den Designer zu erweitern. Hierzu zählen unter anderem die verschiedenen Ereignisse der Komponente die beispielsweise einen Eingriff in das Menü und die angebotenen Funktionen erlauben. Doch die Möglichkeiten gehen noch weiter…

3.8.1 Eigene Funktionen dem Formelassistent hinzufügen

Eine der wichtigsten und mächtigsten Möglichkeiten des Designers sind der Formelassistent und die dort angebotenen Funktionen. Mit Hilfe einer speziellen List & Label Komponente für das OCX ist es zudem möglich, ganz individuelle Funktionen in den Designer einzubinden. Um das Control verwenden zu können, müssen Sie die Datei cmll20fx.ocx in Ihre IDE einbinden.

Zum Hinzufügen einer neuen Funktion fügen Sie zur Designzeit diese Komponente auf einem Formular ein. Im Eigenschaftsfenster dieser Komponente können Sie nun die notwendigen Parameter einstellen.

Eigenschaft	Beschreibung
Name	Der eindeutige Name der Designerfunktion
Description	Eine zusätzliche Beschreibung der Funktion für den Formelassistenten
GroupName	Die Gruppe in der die Funktion im Formelassistenten angezeigt wird
Visible	Gibt an, ob die Funktion im Assistenten angezeigt wird oder nicht
MinimumParameters	Die minimale Anzahl von Parametern. Gültig sind Werte zwischen 0 und 4.

MaximumParameters	Die maximale Anzahl von Parametern. Gültig sind auch hier Werte zwischen 0 und 4. Der Wert muss gleich oder größer der minimalen Anzahl sein. Eine größere Anzahl ergibt optionale Parameter.
ResultType	Der Datentyp des Rückgabewerts

Mit Hilfe der Eigenschaften können Sie die neue Designerfunktion individuell einstellen. Die Parameter für die Designerfunktionen werden im Quellcode festgelegt. Dies kann wie folgt aussehen:

Visual Basic:

```
Private Sub InitializeDesFunction()
        Dim param1 As DesignerFunctionsParameter
        Dim param2 As DesignerFunctionsParameter

        Set param1 = DesFunc_Add.Parameter1
        param1.Description = "First Value"
        param1.Type = LlParamType.ParamType_Double

        Set param2 = DesFunc_Add.Parameter2
        param2.Description = "Second Value"
        param2.Type = LlParamType.ParamType_Double

        DesFunc_Add.ParentComponent = ListLabel1
End Sub
```

Um die Funktion schließlich zum Leben zu erwecken, müssen Sie das Ereignis DesFunc_Add_EvaluateFunction behandeln. Über die Ereignisargumente erhalten Sie Zugriff auf die vom Benutzer eingegebenen Parameter. Um beispielsweise die Summe der beiden Parameter zurück zu liefern, verwenden Sie folgende Zeilen:

Visual Basic:

```
Private Sub DesFunc_Add_EvaluateFunction(ResultValue As Variant,
                ResultType As CMLL20FXLibCtl.LlParamType,
                DecimalPositions As Long,
                ByVal Parameters As Long, ByVal Parameter1 As Variant,
                ByVal Parameter2 As Variant, ByVal Parameter3 As Variant,
                ByVal Parameter4 As Variant)

        ResultValue = CDbl(Parameter1) + CDbl(Parameter2)
        ResultType = ParamType_Double

End Sub
```

Zwei weitere Ereignisse erlauben Ihnen optional eine weitergehende Anpassung der Funktion. Über DesFunc_Add_CheckFunctionSyntax können Sie eine Syntaxprüfung vornehmen. Hier können Sie die Datentypen der Parameter überprüfen und beispielswei-

se sicherstellen, dass die Parameter in einem bestimmten Bereich liegen. Über DesFunc_Add_ParameterAutoComplete ist es möglich, verschiedene Vorschlagswerte für das AutoComplete-Feature des Formelassistenten vorzugeben.

3.8.2 Eigene Objekte dem Designer hinzufügen

Analog zur Erweiterung der Designer-Funktionen können Sie auch eigene Objektarten definieren und an List & Label anmelden. Die neuen Objekte stehen dem Benutzer anschließend in gewohnter Weise über die Symbolleiste links und das Menü zur Verfügung.

Zum Hinzufügen steht Ihnen auch hier eine spezielle Komponente zur Verfügung. Um diese nutzen zu können, müssen Sie die Datei cmll20ox.ocx in Ihre IDE einbinden.

Nach dem Hinzufügen dieser Komponente zu Ihrem Formular können Sie die Eigenschaften des neuen Objekts im Eigenschaftsfenster der Komponente festlegen.

Die Tabelle zeigt eine Übersicht:

Eigenschaft	Beschreibung
Name	Der eindeutige Name des Objekts
Description	Diese Beschreibung wird im Designer angezeigt. Sie darf Leerzeichen enthalten, sollte jedoch nicht länger als 30 Zeichen lang sein.
Icon	Das Icon des Objekts, das im Designer in der Symbolleiste und im Menü angezeigt wird. Es sollte sich um ein 16x16 Pixel großes Icon mit 16 Farben handeln.

Die Komponente bietet Ihnen drei Ereignisse an. Zunächst wird bei der Anlage eines neuen Objekts durch den Benutzer das Ereignis DesObj_Picture_CreateDesignerObject ausgelöst. Falls gewünscht können Sie dem Benutzer hier einen Einstiegsdialog anzeigen. Dies kann beispielsweise ein Assistent sein, der dem Benutzer die Konfiguration des neuen Objekts vereinfacht. Bietet sich die Verwendung im konkreten Fall nicht an, verzichten Sie einfach auf die Behandlung des Ereignisses.

Das Ereignis DesObj_Picture_EditDesignerObject wird ausgelöst, wenn der Benutzer doppelt auf das neu eingefügte Objekt klickt oder aber den Eintrag "Eigenschaften" aus dem Kontextmenü wählt.

Nachdem der Benutzer das Objekt editiert hat, werden Sie von List & Label aufgefordert, das Objekt darzustellen. Es wird hierzu das Ereignis DesObj_Picture_DrawDesignerObject ausgelöst. Sie können nun mit den bekannten Methoden im Arbeitsbereich zeichnen. Hierbei ist selbstverständlich auch der Zugriff auf die hinterlegten Objekteigenschaften möglich beziehungsweise sinnvoll.

3.9 Das Viewer-OCX-Control

3.9.1 Übersicht

Das Control cmll20v.ocx kann dazu verwendet werden, List & Label-Preview-Dateien anzusehen und zu drucken.

Eingefügt werden kann es z.B.

- in eigenen Projekten
- auf einer Internet-Seite

Beim Druck werden die Preview-Dateien so gedruckt, dass sie optimal auf die Drucker-Seite eingepasst werden. Dabei werden die Eigenschaften wie "physikalische Seite" und das Breiten-Höhen-Verhältnis beachtet, um ein möglichst genaues Abbild des Originals auch auf anderen Druckern zu erzeugen.

Wenn statt eines Dateinamens eine Internet-URL angegeben wird, lädt das Control diese temporär auf die Festplatte (in den Internet-Cache) und zeigt sie dann an (sofern eine registrierte URLMON.DLL auf dem System vorhanden ist, s.u.).

Bitte beachten Sie, dass ein Browser benötigt wird, der ActiveX unterstützt, z.B. der Internetexplorer.

3.9.2 Registrierung

Das OCX-Control können Sie auf dem üblichen Weg mit "REGSVR32 CMLL20V.OCX" registrieren oder über Ihre Entwicklungsumgebung anmelden. Vor der Registrierung müssen die abhängigen Module registriert worden sein.

Oft wird die Registrierung auch über Ihr SETUP-Programm vorgenommen.

3.9.3 Eigenschaften

AsyncDownload [in, out] BOOL: Gibt an, ob ein eventueller (Internet-) Download asynchron durchgeführt wird oder nicht. Ein asynchroner Download hat den Vorteil, dass das Programm die Seite mit dem OCX-Control schon anzeigen kann, allerdings muss dann darauf geachtet werden, dass direkte Befehle an das Control (GotoFirst etc.) nicht sofort nach der URL-Zuweisung gesendet werden können (siehe Event LoadFinished). Die Einstellung wirkt sich nicht auf lokale Dateien aus. Voreinstellung: TRUE

Enabled [in, out] BOOL: Gibt an, ob das Control enabled oder disabled ist. Dies wirkt sich auf die Benutzerschnittstelle aus, die dann keine Aktionen zulässt. Voreinstellung: TRUE

BackColor [in, out] OLE_COLOR: Hintergrundfarbe. Das ist die Farbe, die

- den gesamten Hintergrund einnimmt, wenn das Control im Design-State ist oder die Preview-Datei nicht gefunden wird
- den Hintergrund einnimmt, der außerhalb des Papiers angezeigt wird

Voreinstellung: COLOR_BTNFACE [Systemfarbe: Dialoghintergrund]

FileURL [in, out] BSTR: Diese Eigenschaft ist der Name der anzuzeigenden Preview-Datei. Dies kann entweder ein Dateiname oder eine URL sein. Voreinstellung: <leer>

Pages [out] LONG: Die Gesamtanzahl der in der Preview-Datei enthaltenen Seiten.

CurrentPage [in, out] LONG: Hierüber kann die anzuzeigende Seite gesetzt oder abgefragt werden. Voreinstellung: 1

ToolbarEnabled [in, out] BOOL: Gibt an, ob die Toolbar angezeigt werden soll. Die Toolbar ist nicht unbedingt notwendig, da die gesamte Funktionalität extern aufgerufen werden kann (siehe LLVIEW20.EXE und das Menü). Sie können also leicht eine eigene Toolbar hinzufügen. Voreinstellung: TRUE

ToolbarButtons [out] LPDISPATCH: Gibt ein ToolbarButtons Objekt zurück, das den Status der einzelnen Toolbarbuttons lesen und setzen kann. Das Objekt stellt folgende Methoden bereit:

- GetButtonState([in] nButtonID) LONG Bei Übergabe einer TLB_ Konstante wird der Status des Buttons zurückgegeben.

Wert	Bedeutung	Konstante
-1	Versteckt	TLBS_PRV_HIDE
0	Default	TLBS_PRV_DEFAULT
1	Aktiviert	TLBS_PRV_ENABLED
2	Deaktiviert	TLBS_PRV_DISABLED

Visual Basic:

```
Dim oTlb as ToolbarButtons
Set oTlb = LlViewCtrl1.ToolbarButtons
MsgBox oTlb.GetButtonState(TLB_PRV_FILEEXIT)
```

- SetButtonState([in] nButtonID, [in] nButtonState) Setzt den Status des angegebenen Buttons. Die zulässigen Werte sind analog zu oben.

Visual Basic:

```
Dim oTlb as ToolbarButtons
Set oTlb = LlViewCtrl1.ToolbarButtons
oTlb.SetButtonState TLB_PRV_FILEEXIT, TLBS_PRV_HIDE
```

Die entsprechenden IDs finden Sie in der Datei MENUID.TXT in Ihrer List & Label Installation.

ShowThumbnails[in, out] BOOL: Gibt an, ob die Seitenvorschau im Control angezeigt wird. Voreinstellung: TRUE

SaveAsFilePath [in, out] BSTR: Hierüber kann ein Pfad angegeben werden, der im Dateiauswahl-Dialog als Voreinstellung angezeigt werden soll. Bei Verwendung der Methode SaveAs wird der vom Benutzer verwendete Dateiname zurückgegeben.

CanClose[out] BOOL: Gibt an, ob das Control beendet werden darf. Liefert die Eigenschaft den Wert FALSE zurück, darf das Control noch nicht beendet werden.

Version [out] LONG: Gibt die Versionsnummer des OCX-Controls zurück (MAKELONG(lo,hi)).

3.9.4 Methoden

GotoFirst: Springt zur ersten Seite

GotoPrev: Eine Seite zurück

GotoNext: Eine Seite vorwärts

GotoLast: Springt zur letzten Seite

ZoomTimes2: Erhöht den Zoom-Faktor auf das doppelte

ZoomRevert: Geht auf die letzte Zoom-Einstellung

ZoomReset: Setzt den Zoom auf 1-fach ("einpassen") zurück

SetZoom ([in] long nPercentage) : Setzt den Zoom-Faktor anhand des übergebenen Wertes. (1-30)

PrintPage ([in] long nPage, [in] long hDC): Druckt die angegebene Seite (mit Druckerdialog, wenn hDC = 0)

PrintCurrentPage ([in] long hDC): Druckt die momentan angezeigte Seite (mit Druckerdialog, wenn hDC=0)

PrintAllPages ([in] BOOL bShowPrintOptions): Druckt alle Seiten im Projekt (mit Druckerdialog, wenn bShowPrintOptions = TRUE)

SendTo: Startet das "Senden"

SaveAs: Startet das "Speichern Als"

RefreshToolbar: Aktualisiert die Toolbar

GetOptionString([in] BSTR sOption) BSTR: Gibt die Einstellungen des Mailversandes zurück. Die verfügbaren Optionen finden Sie im Kapitel "Exportdateien per eMail verschicken"

SetOptionString([in] BSTR sOption, [in] BSTR sValue) BSTR: Hiermit werden die Einstellungen des Mailversandes gesetzt. Die verfügbaren Optionen finden Sie im Kapitel "Exportdateien per eMail verschicken". Zusätzlich unterstützt das Control die folgenden Optionen:

Print.NoProgressDlg: Kann verwendet werden, um den Fortschrittsdialog beim Druck zu unterdrücken.

Wert	Bedeutung
0	Beim Druck wird eine Fortschrittsanzeige eingeblendet.
1	Die Fortschrittsanzeige wird unterdrückt.
Default	0

3.9.5 Ereignisse

BtnPress

Syntax:

```
BtnPress(short nID): BOOL
```

Aufgabe:

Gibt an, dass ein Button der Toolbar gedrückt wurde.

Parameter:

nID: Die entsprechenden IDs finden Sie in der Datei MENUID.TXT in Ihrer List & Label Installation.

Rückgabewert:

TRUE, wenn Sie den Button ignoriert haben wollen bzw. die zugehörige Aktion selbst durchgeführt haben, ansonsten (default) FALSE.

PageChanged

Syntax:

```
PageChanged
```

Aufgabe:

Teilt Ihnen mit, dass eine neue Seite angezeigt wird.

Parameter:

-

Rückgabewert:

-

LoadFinished

Syntax:

```
LoadFinished(BOOL bSuccessful)
```

Aufgabe:

Weist darauf hin, dass die Datei vollständig auf der lokalen Platte gespeichert wurde. Wichtig ist dies beim asynchronen Download.

Parameter:

bSuccessful: TRUE, wenn die Seite erfolgreich heruntergeladen wurde, FALSE: bei fehlerhaftem Download. Ein erfolgreicher Download ist aber noch kein Indiz dafür, dass auch die Datei korrekt ist - das kann über eine nachfolgende Abfrage der Property 'Pages' überprüft werden: ein Wert größer 0 zeigt an, dass die Datei korrekt ist.

Rückgabewert:

-

3.9.6 Visual C++ Hinweis

In Visual C++ (zumindest 5.0) erhält man u.U. eine "Assertion Failed"-Meldung in occcont.cpp. Diese Assertion erscheint nur für den Debug-Build und hat weiter keine Auswirkungen auf Ihre Applikation. Vermutlich ist dies ein Nebeneffekt der ATL-Bibliothek von Microsoft.

3.9.7 Verpacken in CAB-Files

Ein CAB File befindet sich im Lieferumfang von List & Label.

3.9.8 Einbau in Ihre Internet-Seite

Das Control kann wie erwähnt auch auf einer Internet-Seite platziert werden. Die Eigenschaften können dann über <PARAM>-Tags verwaltet werden, Sie können aber auch über Skript-Sprachen auf das Control zugreifen, z.B. können Sie es mit FrontPage einfügen und über VBScript und andere Controls dessen Eigenschaften verändern. Als Beispiel können Sie die mitgelieferte Datei l20vdemo.htm betrachten.

3.10 Das Designer-OCX-Control

3.10.1 Übersicht

Mit dem Control cmll20id.ocx steht Ihnen der Designer als OCX-Control zur Verfügung. Bitte beachten Sie, dass Sie mit diesem Designer Control keine Designer Erweiterungen (wie z.B. eigene Funktionen) anwenden können. Aufgrund seiner Implementation werden derzeit nur Einzeldatei-Projekte unterstützt. Außerdem wird, um Bandbreite zu sparen, nur ein Datensatz pro Tabelle übertragen. Unter anderem (z.B. keine Ribbon/Menüband Unterstützung) bestehen folgende Einschränkungen für das DesignerControl:

- keine Unterstützung für Bausteine
- keine Unterstützung für Drilldown

- Bilder, PDF-Dokumente etc. müssen in der Projektdatei eingebettet sein (automatisch durch die UI erzwungen)
- keine Echtdatenvorschau im Designer

3.10.2 Registrierung

Das OCX-Control können Sie auf dem üblichen Weg mit "REGSVR32 CMLL20ID.OCX" registrieren oder über Ihre Entwicklungsumgebung anmelden. Vor der Registrierung müssen die abhängigen Module registriert worden sein.

Oft wird die Registrierung auch über Ihr SETUP-Programm vorgenommen.

Sollten Sie mit dem .NET-Framework für ASP.NET entwickeln, steht Ihnen ein separates Control (DesignerControl) im Namespace combit.ListLabel20.Web zur Verfügung.

3.10.3 Eigenschaften

DataStructureUrl [in, out] BSTR: Diese Eigenschaft ist die URL einer Datei welche die Informationen über angemeldete Variablen/Felder enthält. Diese Datei wird mit Hilfe der Funktion *LlJobStateSave* erzeugt. Diese Eigenschaft muss angegeben werden. Voreinstellung: <leer>

ProjectFileUrl [in, out] BSTR: Diese Eigenschaft ist die URL der Projektdatei. Dieser Wert muss angegeben werden. Voreinstellung: <leer>

ProjectType [in, out] LONG: Diese Eigenschaft bestimmt den Projekttyp.

Wert	Bedeutung
1	LL_PROJECT_LABEL
2	LL_PROJECT_LIST
3	LL_PROJECT_CARD
Voreinstellung	2

UploadProjectFileUrl [in, out] BSTR: Diese Eigenschaft ist die URL die zum Upload der Projektdatei verwendet wird. Wird diese Eigenschaft nicht gesetzt, erfolgt der Upload der Projektdatei über ***ProjectFileUrl.*** Voreinstellung: <leer>

UploadOnSave [in, out] BOOL: Diese Eigenschaft gibt an, ob beim Speichern im Designer sofort ein Upload stattfinden soll. Voreinstellung: TRUE

UploadType [in, out] LONG: Diese Eigenschaft gibt an, welche Upload-Methode verwendet wird. Bei HTTP POST wird als ***UploadProjectFileUrl*** eine HTML-Seite erwartet, die den Upload entgegen nehmen kann.

Wert	Bedeutung
1	HTTP POST
2	HTTP PUT
Voreinstellung	1

UploadRequiresAuthentication [in, out] BOOL: Diese Eigenschaft gibt an, ob beim Upload eine Eingabe von Benutzername und Passwort stattfinden soll. Die Daten werden dabei lediglich für diese Session abgefragt. Im Normalfall macht dies nur bei HTTP PUT einen Sinn. Voreinstellung: FALSE

3.10.4 Methoden

Save: Speichert die aktuelle Projektdatei.

3.10.5 Verpacken in CAB-Files

Ein CAB File befindet sich online jeweils getrennt nach Lizenztyp unter:

http://www.combit.net/de/products/ll/l20ddemo/ent/cmll20id.cab

und

http://www.combit.net/de/products/ll/l20ddemo/stdpro/cmll20id.cab

3.10.6 Einbau in Ihre Internet-Seite

Das Control kann wie erwähnt auch auf einer Internet-Seite platziert werden. Die Eigenschaften können dann über <PARAM>-Tags verwaltet werden, Sie können aber auch über Skript-Sprachen auf das Control zugreifen.

```
<object id="InplaceDesigner" classid="CLSID:438BF0A4-E7B0-4119-ABAC-933505CC6759"
codebase="http://www.combit.net/de/products/ll/l20ddemo/stdpro/cmll20id.cab">
…
  <param name="UploadType" value="2" />
…
</object>
```

4. Programmieren mit der VCL Komponente

Parallel zu der OCX-Komponente und dem .NET Assembly erhalten Sie mit List & Label die VCL-Komponente(n) für die Integration in die IDE von Embarcadero. Das folgende Kapitel bezieht sich ausschließlich auf die Arbeit mit dieser Komponente. Wenn Sie nicht mit der VCL-Komponente arbeiten, können Sie diesen Abschnitt überspringen.

4.1 Einbindung der Komponente

Die Einbindung erfolgt mit Hilfe eines Packages. Sollten Sie mit einer Delphi Version kleiner Version 6 arbeiten so verwenden Sie das Package ListLabel20PreDelphi6.dpk und bei einer Delphi-Version ab Version 6 verwenden Sie das Package ListLabel20.dpk. Sie finden beide Dateien in Ihrem List & Label Installationsverzeichnis unterhalb von "..\Programmierbare Beispiele und Deklarationen\Delphi". Weitere Informationen zur Installation des Packages in Ihre IDE finden Sie in der Onlinehilfe Ihrer Entwicklungsumgebung.

Im Anschluss an die Installation des Packages werden automatisch mehrere Symbole im Komponentenbereich der Werkzeugleiste angelegt.

Sie können nun beginnen, List & Label durch die angebotenen Eigenschaften individuell an Ihre Bedürfnisse anzupassen und die benötigte Programmlogik zu implementieren. Hierzu bieten sich drei unterschiedliche Ansätze an:

- Datenbindung
- Die einfachen Print- und Design-Methoden
- Eine eigene, iterative Druckschleife

Die beiden erstgenannten Möglichkeiten werden nachfolgend vorgestellt. Der iterative Ansatz entspricht weitgehend der direkten Verwendung der DLL und ist somit durch die allgemeine Beschreibung der List & Label API abgedeckt.

4.2 Datenbindung

Für die Datenbindung mit Hilfe der List & Label VCL-Komponente existiert ein extra Control. Dieses erbt von der "normalen" Komponente alle Eigenschaften und erweitert diese um die Möglichkeiten zur direkten Datenbindung. Über die Eigenschaft DataSource kann nun eine Datenquelle vom Typ TDataSource angegeben werden.

4.2.1 Bindung von List & Label an eine Datenquelle

Die Datenbindung erfolgt über die Eigenschaft DataSource. Sie können diese programmatisch zuweisen oder aber alternativ über das Eigenschaftsfenster in Ihrer IDE. Haben Sie hier bereits eine DataSource in Ihrem Formular angelegt, so können Sie dies über das Eigenschaftsfenster auswählen. Die notwendige Verknüpfung wird automatisch erzeugt.

Sie können nun den Programmcode zum Start des Designs und Drucks implementieren. Nehmen Sie hierzu beispielsweise im Click-Ereignis eines neuen Buttons den Methoden-

aufruf Print bzw. Design ohne zusätzliche Parameter auf. Die Daten der zugewiesenen Quelle werden automatisch zur Verfügung gestellt.

```
// Designer anzeigen
DBL20_1.AutoDesign('Invoice');

// Druck durchführen
DBL20_1.AutoPrint('Invoice','');
```

Sofern Sie den Standardablauf des datengebundenen Drucks modifizieren wollen, stehen Ihnen eine Reihe von Eigenschaften zur Verfügung. Diese beginnen mit Auto… und sind im Daten-Bereich des Eigenschaftsfensters zu finden.

Eigenschaft	Beschreibung
AutoProjectFile	Dateiname des zu verwendenden Druckprojekts
AutoDestination	Druckformat, zum Beispiel Drucker, Vorschau, PDF, HTML und so weiter
AutoProjectType	Typ des Druck-Projekts (Liste, Karteikarte, Etikett)
AutoFileAlsoNew	Projektneuanlage bei Designeraufruf möglich
AutoShowPrintOptions	Anzeige der Druckoptionen beim Druckstart
AutoShowSelectFile	Anzeige des Dateiauswahl-Dialoges bei Druck und Designer
AutoBoxType	Art der Fortschrittsbox

4.2.2 Arbeiten mit Master-Detail-Datensätzen

In Verbindung mit der Datenbindung und Listenprojekten kann List & Label automatisch vorhandene Relationen zwischen mehreren Tabellen auswerten und übernehmen.

Die Art der Datenübergabe wird mithilfe der Eigenschaft *AutoMasterMode* festgelegt. Die zugrundeliegende Enumeration stellt folgende Werte zur Verfügung:

- None: Es werden keine Master-Detail-Relationen ausgewertet
- AsFields: Master- und Detaildaten werden parallel als Felder angemeldet. Es lassen sich dadurch Gruppierungen, Statistiken und Übersichten realisieren.
- AsVariables: Die Masterdaten werden als Variablen, die Detaildaten als Felder angemeldet. Nach jedem Masterdatensatz wird das Projekt intern mittels *LlPrintResetProjectState()* zurückgesetzt. Auf diese Weise lassen sich mehrere identische Reports mit unterschiedlichen Daten hintereinander in einem Job drucken, zum Beispiel mehrere Rechnungen.

Bitte beachten Sie auch die mitgelieferten Beispiele zur Datenbindung.

4.2.3 Weitere Möglichkeiten der Datenbindung

Die Datenbindung der Komponente stellt Ihnen vier unterschiedliche Ereignisse zur Verfügung, mithilfe derer Sie den Ablauf beeinflussen können. Die Tabelle zeigt eine Übersicht:

Ereignis	Beschreibung
AutoDefineNewPage	Das Ereignis wird für jede neue Seite aufgerufen und erlaubt die Anmeldung von zusätzlichen Variablen für diese Seite. Die Eigenschaft IsDesignMode der Ereignisargumente gibt an, ob es sich um den Design-Modus handelt.
AutoDefineNewLine	Dieses Ereignis wird für jede neue Zeile vor der automatischen Anmeldung der datengebundenen Felder aufgerufen. Analog zu AutoDefineNewPage können Sie hier zusätzliche Felder anmelden.
AutoDefineVariable	Für jede automatisch mittels Datenbindung angelegte Variable wird dieses Ereignis aufgerufen. Über die Eigenschaften Name und Value der Ereignisargumente können Sie den Namen sowie den Inhalt jeder einzelnen Variable individuell vor der Übergabe zum Drucken manipulieren.
AutoDefineField	Analog zu AutoDefineVariable für Felder
AutoDefineTable	Dieses Ereignis wird für jede Tabelle aufgerufen, die über *LlDbAddTable()* angemeldet wird. Sie können z.B. den Namen ändern oder die Übergabe unterdrücken.
AutoDefineTableSortOrder	Dieses Ereignis wird für jede Sortierung aufgerufen, die über *LlDbAddTableSortOrder()* angemeldet wird. Sie können z.B. den Namen ändern oder die Übergabe unterdrücken.
AutoDefineTableRelation	Dieses Ereignis wird für jede DataRelation aufgerufen, die über *LlDbAddTableRelation()* angemeldet wird. Sie können z.B. den Namen ändern oder die Übergabe unterdrücken.

Beachten Sie bitte, dass Sie bei Verwendung dieser Events das Sender-Objekt auf den jeweiligen Komponententypen casten müssen, wenn Sie mit der auslösenden Komponenteninstanz arbeiten wollen. Ansonsten kann es bei DrillDown oder Designervorschau zu Problemen kommen.

Beispiel:

```
procedure TForm1.DBL20_1AutoDefineNewPage(Sender: TObject;
  IsDesignMode: Boolean);
begin
```

```
  (sender as TDBL20_).LlDefineVariable('MyCustomVariableName',
'MyCustomVariableValue');
end;
```

4.3 Einfache Print- und Design-Methoden

4.3.1 Funktionsweise

Die Methoden implementieren eine standardisierte Druckschleife, die für den Großteil der einfacheren Anwendungen direkt verwendbar ist, wenn Sie die Daten nicht per DataBinding übergeben. Die Daten werden bei diesem Ansatz innerhalb der Ereignisse *DefineVariables* und *DefineFields* an List & Label übergeben. Auf diese Weise können beliebige Datenquellen individuell angebunden werden. Die Ereignisargumente erlauben den Zugriff auf nützliche Informationen wie die übergebenen Benutzerdaten, den Design-Mode und so weiter. Über die Eigenschaft *IsLastRecord* wird der Druckschleife mitgeteilt, dass der letzte Datensatz erreicht wurde. Solange dies nicht der Fall ist, wird das jeweilige Ereignis wiederholt aufgerufen, um die Daten abzufragen.

Die *Design*-Methode stellt den Designer in einem modalen Pop-up Fenster dar, welches Ihr Anwendungsfenster überlagert.

Zusätzliche Optionen lassen sich im Ereignis *LLSetPrintOptions* festlegen. Intern wird dieses Ereignis nach dem Aufruf von *LlPrintWithBoxStart()* aber vor dem eigentlichen Druck ausgelöst.

Eine sehr einfache Verwendung der Methode *Print* sieht wie folgt aus:

Delphi:

```
procedure TForm1.LLDefineVariables(Sender: TObject; UserData: Integer;
  IsDesignMode: Boolean; var Percentage: Integer;
  var IsLastRecord: Boolean; var EventResult: Integer);
var
  i: integer;
begin
     For i:= 0 to (DataSource.FieldCount-1) do
     begin
          LL.LlDefineVariableFromTField(DataSource.Fields[i]);
     end;
     if not IsDesignMode then
     begin
          Percentage:=Round(DataSource.RecNo/DataSource.RecordCount*100);
          DataSource.Next;
          if DataSource.EOF=True then IsLastRecord:=true;
     end;
end;
```

4.3.2 Verwendung des UserData-Parameters

Die Methoden *Print* und *Design* erlauben die Übergabe eines Parameters *UserData* vom Typ integer. Mithilfe dieses Parameters können Sie in den Ereignis verschiedene Daten für List & Label bereitstellen. So wäre es z.B. möglich in den Events anhand des Parameters sowohl Daten für den Rechnungsdruck als auch für eine Kundenliste bereit zu stellen.

4.4 Übergabe von ungebundenen Variablen und Feldern

Die Übergabe von Variablen und Feldern entspricht dem regulären Prinzip von List & Label. Für die Anmeldung stehen drei "API-Varianten" zur Verfügung.

API	Beschreibung
LlDefineVariable()	Definiert eine Variable vom Typ LL_TEXT und deren Inhalt.
LlDefineVariableExt()	Wie oben und zusätzlich kann der List & Label Datentyp mit übergeben werden.
LlDefineVariableExtHandle()	Wie oben, wobei der Inhalt nun ein Handle sein muss.

Ein Beispiel für die Anmeldung einer Variablen vom Typ Text sieht folgendermaßen aus:

```
LL.LlDefineVariableExt(´MeineVariable', ´Inhalt', LL_TEXT);
```

Sie finden die Konstanten für die List & Label Datentypen in der Unit cmbtll20.pas in Ihrem List & Label Installationsverzeichnis wieder.

4.4.1 Bilder

Um Bilder zu übergeben, die als Datei auf dem System vorhanden sind, verwenden Sie

```
LL.LlDefineVariableExt ('Picture', <Dateipfad>, LL_DRAWING);
```

Die Übergabe von Grafiken im Speicher (nur für BMP, EMF) erfolgt mittels der API *LlDefineVariableExtHandle()*. Um beispielsweise die Grafik als Bitmap oder Metafile darzustellen, verwenden Sie folgenden Aufruf:

```
LL.LlDefineVariableExtHandle('Picture', BufferImage.picture.bitmap.handle,
LL_DRAWING_HBITMAP);
```

oder

```
LL.LlDefineVariableExtHandle('Picture', BufferImage.picture.metafile.handle,
LL_DRAWING_HEMETA);
```

4.4.2 Barcodes

Die Übergabe von Barcodes wird durch die Verwendung der Konstante *LL_BARCODE...* erreicht. Ein Beispiel für die Anmeldung einer Barcodevariablen vom Typ EAN13 sieht wie folgt aus:

```
LL.LlDefineVariableExt('EAN13', '123456789012',LL_BARCODE_EAN13);
```

4.5 Auswahl der Sprache

Der List & Label Designer liegt in zahlreichen Sprachen vor. Die Komponente unterstützt Sie somit intensiv bei der Realisierung von mehrsprachigen Desktop-Applikationen. Es gibt zwei Möglichkeiten, List & Label die zu verwendende Sprache mitzuteilen:

1. Weisen Sie der Eigenschaft *Language* die entsprechende Sprache zu:

   ```
   LL.Language = ltDeutsch;
   ```

2. Stellen Sie die Sprache direkt an der Komponente ein

Hinweis: Bitte beachten Sie, dass Sie zur Anzeige der jeweiligen Sprache das entsprechende Language Kit benötigen. Welche Kits derzeit zu welchen Konditionen verfügbar sind, erfahren Sie in unserem Online-Shop unter www.combit.net.

4.6 Arbeiten mit Ereignissen

List & Label bietet eine Vielzahl von Callbacks an, die bei der List & Label VCL Komponente als Ereignisse implementiert wurden.

Eine Umfangreiche Beschreibung der zur Verfügung stehenden Events finden Sie in der mitgelieferten Onlinehilfe zur VCL-Komponente.

4.7 Anzeigen einer Vorschaudatei

Für die VCL existiert ein VCL-Vorschaucontrol. Dieses bietet spezialisierte Möglichkeiten zur Verwendung des List & Label Vorschauformates. Es ist z.B. möglich aus dem Control heraus einen PDF-Export zu starten. Zusätzlich kann das Control an die eigenen Bedürfnisse angepasst werden, indem Toolbar-Buttons über Eigenschaften des Controls aus- bzw. eingeblendet werden können. Es ist auch möglich auf die Click-Ereignisse der Buttons zu reagieren und evtl. eigene Behandlungsroutinen zu hinterlegen.

4.8 Arbeiten mit Vorschau-Dateien

Die List & Label Storage-API erlaubt den Zugriff auf die LL-Vorschaudateien. Sie können allgemeine Informationen oder die einzelnen Seiten abfragen, mehrere Dateien zusammenfügen und Benutzerdaten abspeichern. Zur Verwendung der Storage-API müssen Sie die Unit cmbtls20.pas in Ihr Projekt einbinden.

4.8.1 Öffnen einer Vorschaudatei

Sie können die Vorschaudatei mit Hilfe der Funktion *LlStgsysStorageOpen()* öffnen. Über eine ganze Reihe von weiteren Funktionen stehen nun allgemeine Informationen über die Datei zur Verfügung.

Delphi:

```
var
  hStg:  HLLSTG;
begin
  hStg := LlStgsysStorageOpen('c:\test.ll','',False, False)
end;
```

4.8.2 Zusammenführen mehrerer Vorschaudateien

Sie können mehrere Vorschaudateien zusammenführen. Hierzu müssen Sie zunächst die Zieldatei öffnen. Da ein schreibender Zugriff notwendig ist, müssen Sie für den zweiten Parameter *ReadOnly* "False" übergeben. Anschließend können Sie mit Hilfe der Funktion *LlStgSysAppend()* die Dateien zusammen führen.

Delphi:

```
var
  hStgOrg, hStgAppend:  HLLSTG;
begin
  hStgOrg := LlStgsysStorageOpen('c:\test1.ll','',False, True);
  hStgAppend := LlStgsysStorageOpen('c:\test2.ll','',False, True);
  LlStgsysAppend(hStgOrg, hStgAppend);
  LlStgsysStorageClose(hStgOrg);
  LlStgsysStorageClose(hStgAppend);
end;
```

4.8.3 Debugging

Das Debugging der VCL-Komponente können Sie entweder direkt an der Komponente aktivieren, indem Sie die Eigenschaft *DebugMode* auf "1" setzen, oder aber im Quellcode über z.B.:

```
LL.DebugMode := LL_DEBUG_CMBTLL;
```

Weitere Informationen über das Debugtool Debwin finden Sie in Kapitel "Fehlersuche mit Debwin".

4.9 Erweiterung des Designers

List & Label bietet vielfältige Möglichkeiten, den Designer zu erweitern. Hierzu zählen unter anderem die verschiedenen Ereignisse der Komponente die beispielsweise einen Eingriff in das Menü und die angebotenen Funktionen erlauben. Doch die Möglichkeiten gehen noch weiter.

4.9.1 Eigene Funktionen dem Formelassistent hinzufügen

Eine der wichtigsten und mächtigsten Möglichkeiten des Designers sind der Formelassistent und die dort angebotenen Funktionen. Mit Hilfe einer speziellen List & Label Komponente für die VCL ist es zudem möglich, ganz individuelle Funktionen in den Designer einzubinden.

Zum Hinzufügen einer neuen Funktion fügen Sie in der Entwicklungsumgebung diese Komponente auf einem Formular ein. Im Eigenschaftsfenster dieser Komponente können Sie nun die notwendigen Parameter einstellen.

Eigenschaft	Beschreibung
Name	Der eindeutige Name der Designerfunktion
Description	Eine zusätzliche Beschreibung der Funktion für den Formelassistenten
GroupName	Die Gruppe in der die Funktion im Formelassistenten angezeigt wird
Visible	Gibt an, ob die Funktion im Assistenten angezeigt wird oder nicht
MinimumParameters	Die minimale Anzahl von Parametern. Gültig sind Werte zwischen 0 und 4.
MaximumParameters	Die maximale Anzahl von Parametern. Gültig sind auch hier Werte zwischen 0 und 4. Der Wert muss gleich oder größer der minimalen Anzahl sein. Eine größere Anzahl ergibt optionale Parameter.
Parameter1 – 4	Jeder der bis zu vier Parameter kann individuell konfiguriert werden.
Type	Der Datentyp des Parameters
Description	Eine Beschreibung des Parameters für die Tooltip-Hilfe im Designer
ResultType	Der Datentyp des Rückgabewerts

Mithilfe der Eigenschaften können Sie die neue Designerfunktion individuell einstellen. Um die Funktion schließlich zum Leben zu erwecken, müssen Sie das Ereignis *OnEvaluateFunction* behandeln. Über die Ereignisargumente erhalten Sie Zugriff auf die vom Benutzer eingegebenen Parameter. Um beispielsweise die römische Ziffer zurück zu liefern, verwenden Sie folgende Zeilen:

Delphi:

```
procedure TDesExtForm.RomanNumberEvaluateFunction(Sender: TObject;
        var ResultType: TLl20XFunctionParameterType;
        var ResultValue: OleVariant;
        var DecimalPositions: Integer; const ParameterCount: Integer;
```

```
        const Parameter1, Parameter2, Parameter3, Parameter4: OleVariant);
begin
        ResultValue:=ToRoman(Parameter1);
end;
```

Zwei weitere Ereignisse erlauben Ihnen optional eine weitergehende Anpassung der Funktion. Über *OnCheckFunctionSyntax* können Sie eine Syntaxprüfung vornehmen. Hier können Sie die Datentypen der Parameter überprüfen und beispielsweise sicherstellen, dass die Parameter in einem bestimmten Bereich liegen. Über *OnParameterAutoComplete* ist es möglich, verschiedene Vorschlagswerte für das AutoComplete-Feature des Formelassistenten vorzugeben.

4.9.2 Eigene Objekte dem Designer hinzufügen

Analog zur Erweiterung der Designer-Funktionen können Sie auch eigene Objektarten definieren und an List & Label anmelden. Die neuen Objekte stehen dem Benutzer anschließend in gewohnter Weise über die Symbolleiste links und das Menü zur Verfügung.

Zum Hinzufügen steht Ihnen auch hier eine spezielle Komponente (TLI20XObject) zur Verfügung.

Nach dem Hinzufügen dieser Komponente zu Ihrem Formular können Sie die Eigenschaften des neuen Objekts im Eigenschaftsfenster der Komponente festlegen. Die Tabelle zeigt eine Übersicht.

Eigenschaft	Beschreibung
Name	Der eindeutige Name des Objekts
Description	Diese Beschreibung wird im Designer angezeigt. Sie darf Leerzeichen enthalten, sollte jedoch nicht länger als 30 Zeichen lang sein.
Icon	Das Icon des Objekts, das im Designer in der Symbolleiste und im Menü angezeigt wird. Es sollte sich um ein 16x16 Pixel großes Icon handeln.

Die Komponente bietet Ihnen drei Ereignisse an. Zunächst wird bei der Anlage eines neuen Objekts durch den Benutzer das Ereignis *OnInitialCreation* ausgelöst. Falls gewünscht können Sie dem Benutzer hier einen Einstiegsdialog anzeigen. Dies kann beispielsweise ein Assistent sein, der dem Benutzer die Konfiguration des neuen Objekts vereinfacht. Bietet sich die Verwendung im konkreten Fall nicht an, verzichten Sie einfach auf die Behandlung des Ereignisses.

Die folgenden Zeilen initialisieren das Objekt sobald das neue Objekt erstmals auf dem Arbeitsbereich platziert wird.

Delphi:

```
procedure TDesExtForm.GradientFillObjectInitialCreation(Sender: TObject;
        ParentHandle: Cardinal);
```

```
begin
        with Sender as TLl20XObject do
        begin
                Properties.AddProperty('Color1', '255');
                Properties.AddProperty('Color2', '65280');
        end;
end;
```

Das Ereignis *OnEdit* wird ausgelöst, wenn der Benutzer doppelt auf das neu eingefügte Objekt klickt oder aber den Eintrag "Eigenschaften" aus dem Kontextmenü wählt.

Nachdem der Benutzer das Objekt editiert hat, werden Sie von List & Label aufgefordert, das Objekt darzustellen. Es wird hierzu das Ereignis *OnDraw* ausgelöst. Über die Ereignisargumente erhalten Sie einen *TCanvas* sowie ein *TRect* des Objekts. Sie können nun mit den bekannten Methoden im Arbeitsbereich zeichnen. Hierbei ist selbstverständlich auch der Zugriff auf die hinterlegten Objekteigenschaften möglich beziehungsweise sinnvoll.

5. Programmierung per API

Ein guter Start ist auch die Arbeit mit einem unserer Beispiele, die wir für viele Programmiersprachen mitliefern. Sie finden diese in der Startmenügruppe Ihrer List & Label-Installation in der Untergruppe "Beispiele". Für viele weitere Sprachen werden Deklarationsdateien mitgeliefert, die eine möglichst einfache Einbindung ermöglichen – hier ist es dann häufig möglich, sich an den Beispielen für andere Programmiersprachen zu orientieren.

Für die Quellcode-Ausschnitte in diesem Kapitel wurde C/C++ als Programmiersprache verwendet, die Syntax lässt sich jedoch sehr leicht auf andere Programmiersprachen analog übertragen.

5.1 Programmierschnittstelle

5.1.1 Dynamic Link Libraries

Funktionsprinzip

Eine DLL (Dynamic Link Library, auch Dynamic Library oder Dynalink Library) ist eine Funktionsbibliothek, d.h. eine Ansammlung von Routinen in einer Datei, von der bei Bedarf Teile durch den Windows-Kernel nachgeladen werden. Das DLL-Prinzip ermöglicht es, dass mehrere Programme auf die in der DLL enthaltenen Funktionen zurückgreifen können, die DLL aber nur einmal im Speicher vorhanden ist.

Natürlich können DLLs auch auf die Funktionen anderer DLLs zurückgreifen, wie das in Windows ständig geschieht.

List & Label verwendet wiederum mehrere DLLs, die für spezifische Aufgaben verwendet werden.

Verwendung einer DLL

Windows muss in der Lage sein, die DLL selbsttätig zu finden und zu laden. Hierzu muss sie im Windows-Pfad, Windows-System-Pfad, dem Programmpfad oder auf einem beliebigen Pfad, der sich in dem Systemsuchpfad befindet, vorhanden sein.

Das gleiche gilt analog für die von List & Label (indirekt) verwendeten DLLs.

Auf jedem Rechner, auf dem die Endapplikation installiert wird, muss auch die DLL in einem der oben angesprochenen Verzeichnisse vorhanden sein!

Aus Kompatibilitätsgründen sollten unter Windows 7 und Windows 8 keine Dateien in das Windows-Verzeichnis kopiert werden.

Detaillierte Hinweise hierzu finden Sie im Kapitel "Redistribution Ihrer Anwendung".

Einbindung der DLL-Routinen

Routinen aus DLLs können einfach aus vielen Programmiersprachen heraus aufgerufen werden, sei es nun aus C, C++, Delphi, Visual Basic, Xbase und anderen mehr.

Damit diese Aufrufe in einem Programm nicht zu einer Fehlermeldung des Linkers führen, müssen diese Funktionen dem Linker, Compiler oder Interpreter bekannt gemacht, d.h. deklariert werden.

Hier unterscheiden sich prinzipbedingt die verschiedenen Sprachen wie C, Delphi und Visual Basic. Extremfälle sind Makrosprachen wie Excel, in denen die Deklaration relativ kompliziert ist.

Einbindung über Import-Libraries

Um die API-Funktionen nutzen zu können, muss Ihre Anwendung die Deklarationsdatei für die cmll20.dll (C++: cmbtll20.h) inkludieren, und zwar nach dem "#include <windows.h>"-Statement bzw. den precompiled Header-Dateien, da Windows-Variablentypen in den Deklarationen verwendet werden und ggf. die zugehörige LIB-Datei (C++: cmll20.lib) linken. Um die speziellen Vorschaufunktionen (siehe Kapitel "Verwaltung der Preview-Dateien") nutzen zu können, muss Ihre Anwendung die Deklarationsdatei für die cmls20.dll (C++: cmbtls20.h) inkludieren und ggf. die zugehörige LIB-Datei (C++: cmls20.lib) linken.

Unter den mitgelieferten Programmen befindet sich eine Import-Library, CMLL20.LIB. Diese muss dem Linker als Library angeboten werden. Sie sorgt dafür, dass der Linker die Verbindung zur DLL und deren Funktionen einbaut.

Sie können die Importbibliothek auch mit dem Programm IMPLIB, das bei Ihrem SDK oder möglicherweise auch dem Compiler beiliegt, erstellen. Manche Compiler-Hersteller (z.B. Watcom/Sybase oder Borland) liefern ein solches Tool mit aus, ein entsprechendes Tool liegt dem Microsoft-Compiler nicht bei.

Auch ein dynamisches Laden der DLLs ist möglich – eine Beschreibung hierfür finden Sie in der Online-Knowledgebase unter http://support.combit.net.

Wichtiges zu den Funktionsparametern

Die Übergabe von Zeichenketten von der DLL an das Anwenderprogramm ist immer durch die Übergabe eines Zeigers auf einen Speicherbereich und als weiteren Parameter einen Integer-Wert für dessen Länge (in Zeichen) definiert. Es werden grundsätzlich nur so viele Zeichen kopiert, dass es keinen Überlauf in diesem Bereich gibt, die Zeichenketten sind immer ´\0´-terminiert. Bei Bedarf (wenn der Puffer zu klein ist) wird der übergebene String also gekürzt und unvollständig zurückgegeben. Achten Sie auf den Fehlercode *LL_ERR_BUFFERTOOSMALL*.

Die Parameter werden soweit möglich, auf Korrektheit überprüft. Während der Programmentwicklung lohnt es sich also, den Debug-Modus einzuschalten (siehe *LlSetDebug()*) und die Rückgabewerte zu überprüfen. Später können Sie dann die Parameterüberprüfung explizit ausschalten (*LL_OPTION_NOPARAMETERCHECK*).

Bei Delphi ist zu beachten, dass die Routinen null-terminierte Zeichenketten benötigen und zurückgeben, wie es bei Windows-Funktionen üblich ist. Im Bedarfsfall müssen die Pascal-String zu C-String Konvertierungsroutinen benutzt werden.

Bei Visual Basic sollte beim DLL-Zugriff das '\0' Zeichen zur Weiterverarbeitung entfernt werden (normalerweise beim OCX nicht nötig). Parameter werden überall ByVal übergeben. Es empfiehlt sich, Zeichenketten/Puffer vor Gebrauch durch

```
Dim lpszBuffer As String * 255
```

auf eine gewisse Größe (hier 255 Bytes) zu initialisieren. Dies ist auch durch eine Zuweisung wie

```
lpszBuffer$ = space$(255)
```

möglich, das aber mehr Zeit und Code benötigt. Wichtig ist nur, dass der Platz reserviert wird, so dass die DLL nicht in unbenutzte Bereiche schreibt, ansonsten wäre ein GPF die Folge.

Ansonsten ist zu Visual Basic für die Verwendung der DLL-Schnittstelle anzumerken, dass manche Funktionen prinzipiell nicht unterstützt werden können; im Normalfall werden diese aber auch nicht benötigt. Sofern in diesem Handbuch als Übergabewert NULL oder nil verwendet wird, sollte (je nach Datentyp) in Visual Basic "" (Leerstring) oder 0 übergeben werden.

Bei C oder C++ muss die C-Konvention beachtet werden, so dass Sie in Quelltexten für '\', z.B. in Pfadangaben, dieses doppelt eingeben müssen: "c:\\temp.lbl" statt "c:\temp.lbl".

Beispiel:

```
INT    nSize = 16000;
LPWSTR pszBuffer = new TCHAR[nSize];
INT    nResult = <API>(hJob,...,pszBuffer,nSize);

if (nResult == LL_ERR_BUFFERTOOSMALL)
    {
    nSize = <API>(hJob,...,NULL,0);

    ASSERT(nSize > 0);
    delete[] pszBuffer;
    pszBuffer = new TCHAR[nSize];
    nResult = <API>(hJob,...,pszBuffer,nSize);
    }
...

delete[] pszBuffer;
```

5.1.2 Allgemeines zum Rückgabewert

- 0 (oder bei manchen Funktionen positive Rückgabewerte) bedeutet im Allgemeinen, dass die Funktion erfolgreich war.
- Ein negativer Rückgabewert signalisiert bis auf Ausnahmefälle einen Fehler, der über die entsprechenden Fehlerkonstanten den Grund bezeichnet.

5.2 Programmiergrundlagen

5.2.1 Datenbankunabhängiges Konzept

List & Label arbeitet bei der Programmierung per API datenbankunabhängig, d.h. List & Label selbst greift nicht direkt auf die Datenbank zu und besitzt auch keine eigenen Datenbanktreiber. Dieses Konzept bietet eine ganze Reihe enormer Vorteile.

Vorteile:

- Kein unnötiger Ballast durch doppelt mitgeführte Datenbanktreiber, dadurch kann ein Geschwindigkeitsvorteil sowie ein geringerer Platzbedarf der Module erreicht werden.
- Flexibler Einsatz, da genaue Kontrolle der Daten.
- Arbeiten auch ohne Vorhandensein einer Datenbank möglich.
- Arbeiten mit seltenen Datenbanksystemen möglich.
- Einfaches Mischen unterschiedlicher Datenquellen, z.B. Datenbankdaten und programminterne Variablen.
- Datenbankdaten können vor dem Ausdruck noch einfach manipuliert werden.

Nachteil:

- Es muss tatsächlich etwas programmiert werden, d.h. List & Label müssen die Daten übergeben werden. Dies funktioniert aber nach einem sehr einfachen Prinzip und ist somit für die meisten Standardfälle mit relativ wenig Code-Schreibarbeit verbunden.

5.2.2 Der List & Label Job

Damit List & Label die einzelnen Anwendungen, die mit List & Label drucken möchten, unterscheiden kann, ist ein sog. Jobmanagement erforderlich: Jede Anwendung, die eine Funktionalität von List & Label nutzen möchte (Druck, Designer, etc.) muss dazu vorher einen Job öffnen (*LlJobOpen()*, *LlJobOpenLCID()*) und das zurückerhaltene Jobhandle dann bei allen anderen List & Label Funktionsaufrufen mit übergeben.

```
HLLJOB hJob = LlJobOpen(CMBTLANG_GERMAN);
...
LlDefineVariable(hJob, "Vorname", "Otto");
```

```
LlDefineVariable(hJob, "Name", "Normalverbraucher");
...
```

5.2.3 Variablen, Felder und Datentypen

Das Anmelden und Definieren von Variablen samt Inhalt geschieht mit der List & Label Funktion *LlDefineVariable[Ext]()* und das Anmelden und Definieren von Feldern samt Inhalt erfolgt über die Funktion *LlDefineField[Ext]()*. Für die Namensvergabe gelten dabei die allgemeinen Hinweise von Kapitel "Hinweise zu Variablen- und Feldnamen".

List & Label erlaubt die Spezifikation folgender Variablen- bzw. Feldtypen. Da die API-Funktionen zur Übergabe der Werte einen String als Wert erwartet, müssen die tatsächlichen Werte vor der Übergabe ggf. in eine Zeichenkette umgewandelt werden.

Beachten Sie bitte die Hinweise zu NULL-Werten gemäß Kapitel "Übergabe von NULL-Werten".

```
HLLJOB hJob = LlJobOpen(CMBTLANG_GERMAN);
...
LlDefineVariable(hJob, "Vorname", "Otto");
LlDefineVariable(hJob, "Name", "Normalverbraucher");
LlDefineVariableExt(hJob, "ISBN", "40|15589|97531", LL_BARCODE_EAN13, NULL);
LlDefineVariableExt (hJob, "Foto", "c:\\dwg\\test.bmp", LL_DRAWING, NULL);
...
```

Text

Konstante:

LL_TEXT

Beispielinhalt:

"abcdefg"

Bemerkung:

Dieser kann spezielle, auf Wortumbrüche spezialisierte Zeichen beinhalten. Diese sind:

Wert	Bedeutung
LL_CHAR_NEWLINE, 0x0d, 0x0a	Textobjekt: wird zu einem Leerzeichen, wenn im Designer "kein Umbruch" ("Abschneiden") eingestellt wird. Tabellenfeld: Umbruch wird erzwungen. "Das hier"+*chr$(LL_CHAR_NEWLINE)*+"wird getrennt"
LL_CHAR_PHANTOMSPACE	Das Zeichen wird ignoriert, wenn kein Umbruch gewünscht wird. Damit kann man auch andere Zeichen als Umbruchzeichen deklarieren: bei "Dies

Wert	Bedeutung
	ist ein Doppel-"+*chr$(LL_CHAR_PHANTOMSPACE)*+"Wort" kann bei Bedarf hinter dem Trennstrich getrennt werden.
LL_CHAR_LOCK	Wird vor Leerzeichen oder Tabs gestellt, und bedeutet, dass bei diesen kein Umbruch stattfinden kann: "Hier"+*chr$(LL_CHAR_LOCK)*+" bitte nicht"

Die Codes dieser Zeichen können im Bedarfsfall per Option verändert werden (*LL_OPTION_xxxREPRESENTATIONCODE*).

Numerisch

Konstante:

LL_NUMERIC

Beispielinhalt:

"3.1415", "2.5e3"

Bemerkung:

Exponentialschreibweise ist erlaubt (2.5e3 entspricht 2.5*10^3). Nicht erlaubt sind Tausendertrennzeichen bei der Übergabe, z.B. "1.420.000,00". Beachten Sie, dass das Dezimaltrennzeichen bei Verwendung dieser Konstante bei der Übergabe immer "." sein sollte.

Bei folgendem Untertyp für numerische Werte können sie die Zahl direkt übergeben, wie Ihre Anwendung sie lokalisiert erstellt:

Konstante:

LL_NUMERIC_LOCALIZED

Beispielinhalt:

"1.255,00"

Bemerkung:

Hier wird die Zahl als "lokalisierte" Zahl interpretiert, also z.B. "123.456,00" in einer deutschen Betriebssystem-Umgebung. Intern geschieht die Umwandlung über die OLE-API *VarR8FromStr()*.

Konstante:

LL_NUMERIC_INTEGER

Beispielinhalt:

"5"

Bemerkung:

Übergibt eine Integer-Zahl (Zahl ohne Nachkommastellen). Mit Integer-Zahlen kann schneller gerechnet werden, sie werden im Design und Druck ohne Nachkommastellen ausgegeben.

Datum

Konstante:

LL_DATE

Beispielinhalt:

"2451158.5" (entspricht dem 11.12.1998 12:00 mittags), "1.5.2000" mit *LL_DATE_DMY* Format oder "20000501" für *LL_DATE_YYYYMMDD* Format

Bemerkung:

Datumswerte werden im Julianischen Format erwartet. Das Julianische Datum spezifiziert ein Datum, indem als numerischer Wert die Anzahl vergangener Tage seit dem 01. Januar -4713 angegeben werden.

Der Nachkommaanteil ist der Bruchteil eines Tags, über den Stunden, Minuten und Sekunden berechnet werden.

Viele Programmiersprachen besitzen ebenfalls einen speziellen Datentyp für Datumswerte. Die Repräsentation erfolgt meist analog zum Julianischen Datum, häufig jedoch mit einem anderen Startdatum. Dies bedeutet in diesem Fall, dass noch ein Offset hinzuaddiert werden muss. Um dies zumindest für die Programmiersprachen Visual Basic, Visual FoxPro und Delphi zu umgehen, gibt es in List & Label zusätzlich folgende spezielle Datumsvarianten:

Konstante:

LL_DATE_OLE, LL_DATE_MS, LL_DATE_DELPHI_1, LL_DATE_DELPHI, LL_DATE_VFOXPRO

Bemerkung:

Dadurch kann direkt der Zahlenwert einer Datumsvariablen als String an List & Label übergeben werden, ohne dass in Ihrem Programm eine Umrechnung in das Julianische Datum erfolgen muss - List & Label erledigt das für Sie.

Um eine Übergabe des Datums zu erleichtern, gibt es zusätzlich noch folgende Datentypen, die die Formatierung eines Datums in nicht-julianischer Form ermöglichen:

Konstante:

LL_DATE_DMY, LL_DATE_MDY, LL_DATE_YMD, LL_DATE_YYYYMMDD.

Bemerkung:

Bei den ersten drei Formaten müssen die Zahlen für Tag, Monat und Jahr jeweils durch Punkt ('.'), Schräg- ('/') oder Bindestrich ('-') getrennt sein.

Ebenfalls in nicht-julianischer Form wird das Datum bei dem folgenden Typ erwartet:

Konstante:

LL_DATE_LOCALIZED

Bemerkung:

Hier wird das Datum als "lokalisiertes" Datum interpretiert, also z.B. "1.5.2001" in einer deutschen Betriebssystem-Umgebung. Intern geschieht die Umwandlung über die OLE-API *VarDateFromStr()*.

Logisch

Konstante:

LL_BOOLEAN

Beispielinhalt:

"T"

Bemerkung:

"T","Y","J","t","y","j","1" entsprechen "wahr", alle übrigen Werte entsprechen "falsch".

RTF-formatierter Text

Konstante:

LL_RTF

Beispielinhalt:

"{\rtf1\ansi[...]Hallo {\b Welt}!\par}"

Bemerkung:

Der Variableninhalt muss mit "{\rtf" anfangen und anschließend RTF-formatierten Text enthalten.

Wichtig: Bitte beachten Sie, dass die Darstellung von RTF-Texten aus Variablen/Feldern auf Inhalte ausgelegt ist, welche mit Hilfe des Microsoft RTF-Controls erzeugt wurden. Sie können diese Inhalte beispielsweise mit der Windows Anwendung "Wordpad" generieren. Inhalte, die in Microsoft Word erzeugt wurden, sind unter Umständen nicht mit dem vom Control verwendeten RTF-Standard kompatibel und sollten deshalb auch nicht verwendet werden.

HTML-formatierter Text

Konstante:

LL_HTML

Beispielinhalt:

"<html><body>Hallo Welt</body></html>"

Bemerkung:

List & Label nutzt für die Darstellung von HTML-Inhalten eine eigene Komponente, die einen begrenzten Satz an CSS-Eigenschaften unterstützt. Die korrekte Wiedergabe ganzer Webseiten steht nicht im Zentrum, vielmehr handelt es sich um die Möglichkeit, schnell und einfach simple HTML-Streams auszugeben.

Zeichnung

Konstante:

LL_DRAWING

Beispielinhalt:

"c:\temp\sunny.wmf"

Bemerkung:

Variableninhalt ist der Name einer Grafikdatei (C/C++: doppelten '\\' bei Pfadangaben benutzen!).

Konstante:

LL_DRAWING_HMETA, LL_DRAWING_HEMETA, LL_DRAWING_HBITMAP, LL_DRAWING_HICON

Bemerkung:

Variableninhalt ist ein Handle auf eine entsprechende Grafik im Speicher (kann nur über *LlDefineVariableExtHandle()* oder *LlDefineFieldExtHandle()* definiert werden).

Barcode

Konstante:

LL_BARCODE

Beispielinhalt:

"Barcodetext"

Bemerkung:

Variableninhalt ist der Text, der in einem Barcode erscheinen kann. Die erlaubten Formatierungen des Textes und die erlaubten Zeichen sind in der Online-Hilfe beschreiben.

Konstante:

alle Konstanten der Deklarationsdatei, die mit *LL_BARCODE_... beginnen.*

Benutzergezeichnetes Objekt

Konstante:

LL_DRAWING_USEROBJ, LL_DRAWING_USEROBJ_DLG

Bemerkung:

Dieses Objekt wird per Callback/Event vom Programmierer selbst gezeichnet. Bei *LL_DRAWING_USEROBJ_DLG* kann der Programmierer im Designer auch einen eigenen Eigenschaftsdialog für das Objekt bereitstellen.

Die Benutzung dieses Variablentyps gehört zur sehr fortgeschrittenen Programmierung und wird an anderer Stelle in diesem Handbuch behandelt.

5.3 Aufruf des Designers

5.3.1 Grundschema

Der Aufruf des Designers sieht, in Pseudo-Sprache ausgedrückt, folgendermaßen aus (die Funktionen mit '*' sind optionale Aufrufe, die nicht unbedingt benötigt werden):

```
<öffne Job>
        (LlJobOpen, LlJobOpenLCID)
<definiere List & Label-Voreinstellungen>*
        (LlSetOption,
         LlSetOptionString,
         LlSetDebug,
         LlSetFileExtensions,
         LlSetNotificationMessage,
         LlSetNotificationCallback)
<zu bearbeitende Datei bestimmen lassen>*
        LlSelectFileDlgTitleEx
<definiere Variablen>
        (LlDefineVariableStart,
         LlDefineVariable,
         LlDefineVariableExt,
         LlDefineVariableExtHandle)
<definiere Felder>*     (nur LL_PROJECT_LIST)
        (LlDefineFieldStart,
         LlDefineField,
```

```
         LlDefineFieldExt,
         LlDefineFieldExtHandle)
<sperre Funktionen>*
        (LlDesignerProhibitAction,
         LlDesignerProhibitFunction)
<Aufruf des Designers>
        (LlDefineLayout)
<schließe Job>
        (LlJobClose)
```

Zur Jobverwaltung (*LlJobOpen[LCID]()* und *LlJobClose()*) reicht es auch aus, den Job am Programmanfang anzufordern und am Programmende freizugeben, und dann diesen Job für Designeraufrufe und Ausdrucke gleichermaßen zu verwenden. Ein Job-Handle kann über die ganze Laufzeit der Applikation behalten werden, so dass man es erst zum Schluss wieder freigeben muss.

Aus Übersichtsgründen empfehlen wir, globale Einstellungen, die für alle List & Label-Aufrufe gelten sollen, ein einziges Mal nach LlJobOpen[LCID]() zu tätigen, und die lokalen Einstellungen wie Menü-Sperreinträge direkt vor Aufruf des Designers oder des Drucks vorzunehmen.

5.3.2 Erläuterung

Wenn die Einstellung von List & Label Optionen gewünscht ist, müssen diese vor Aufruf des Designers vorgenommen werden.

Normalerweise wird nun der Benutzer über eine Dialogbox gefragt, welche Datei er bearbeiten möchte. In unserem Fall nehmen wir an, dass er ein Etikett bearbeiten will.

Wichtig ist, dass der Puffer für den Dateinamen vorinitialisiert wurde - entweder auf einen leeren String (""), oder auf einen Dateinamenvorschlag, incl. Pfad:

```
TCHAR   aczProjectFile[_MAX_PATH];
_tcscpy(aczProjectFile, "c:\\mylabel.lbl");
LlSelectFileDlgTitleEx(hJob, hWindow, "Etikett auswählen", LL_PROJECT_LABEL,
        aczProjectFile, _MAX_PATH, NULL);
```

Natürlich können Sie den Aufruf auch mit einer eigenen Dialogbox realisieren, oder Sie können den Dateinamen ohne Benutzerabfrage dem Designer übergeben, falls nicht gewünscht ist, dass der Benutzer wählen kann.

Jetzt müssen List & Label die möglichen Variablen mitgeteilt werden, damit es diese dem Benutzer in der Variablenliste zur Verfügung stellt. Sonst könnte der Benutzer nur festen Text in die Objektdefinitionen übernehmen.

Zuerst wird der Variablenpuffer gelöscht (falls schon Variablen definiert wurden, aber der Aufruf ist zur Sicherstellung eines leeren Variablenpuffers empfehlenswert):

```
LlDefineVariableStart(hJob);
```

Jetzt kann man die Variablen auf mehrere Arten angeben. Wenn der Designer zu einer Variablen eine Beispiel-Übersetzung kennt, wird diese im Preview-Fenster statt des Variablennamens verwendet, um eine realistischere Preview-Darstellung zu gewährleisten.

```
LlDefineVariable(hJob, "Vorname", "Otto");
LlDefineVariable(hJob, "Name", "Normalverbraucher");
```

Im Preview wird der auf dem Designer-Arbeitsblatt stehende Ausdruck

Vorname+" "+Name

in die Ausgabe

Otto Normalverbraucher

umgesetzt.

Die erweiterte Variablendefinition mit *LlDefineVariableExt()* wird benutzt, um andere Variablentypen als Text, z.B. für Barcode-Objekte oder Zeichnungen zu definieren.

Wenn auch Listenobjekte gebraucht werden, also ein Projekt des Typs *LL_PROJECT_LIST* bearbeitet werden soll, muss der Programmierer die hier möglichen Felder zur Verfügung stellen. Dies geschieht analog zu oben (auch z.B. Barcode-Felder und Zeichnungen sind als Tabellenspalten möglich), nur dass die Funktionsnamen nun statt 'Variable' 'Field' enthalten:

```
LlDefineFieldStart(hJob);
LlDefineField(hJob, "Buch");
LlDefineField(hJob, "ISBN");
LlDefineFieldExt(hJob, "ISBN", "40|15589|97531", LL_BARCODE_EAN13, NULL);
LlDefineFieldExt(hJob, "Foto", "c:\\dwg\\test.bmp", LL_DRAWING, NULL);
```

Vor dem Aufruf von *LlDefineLayout()* können dem Benutzer über *LlDesignerProhibitAction()* Menüpunkte gesperrt werden, oder verhindert werden, dass der Designer minimiert wird. Dies macht beispielsweise der Aufruf von

```
LlDesignerProhibitAction(hJob, LL_SYSCOMMAND_MINIMIZE);
```

Jetzt ist alles so weit definiert, dass der Benutzer sein Etikett, seine Karteikarte oder Liste definieren kann, indem der Designer aufgerufen wird:

```
LlDefineLayout(hJob, hParentWindow, "Test-Titel", LL_PROJECT_LABEL, "test");
```

Für eine Liste muss entsprechend die Konstante *LL_PROJECT_LIST* bzw. für eine Karteikarte *LL_PROJECT_CARD* verwendet werden.

Beachten Sie, dass prinzipbedingt im Vorschaufenster des Designers mit nur einem (immer gleichen) Datensatz gearbeitet wird. Es ist aber möglich, auch im Designer eine Echtdatenvorschau direkt anzubieten, siehe Kapitel "Direkter Druck und Export aus dem Designer".

Es empfiehlt sich, generell den Fehlercode von Funktionsaufrufen auszuwerten.

5.4 Der Druckvorgang

Konkrete Quellcode-Beispiele finden Sie für die verschiedenen Programmiersprachen in Ihrem List & Label Installationsverzeichnis im Ordner "Programmierbare Beispiele und Deklarationen". Nachfolgende Überlegungen in Pseudocode behandelt.

5.4.1 Die Datenversorgung

List & Label arbeitet bei der Ansteuerung per API datenbankunabhängig. Das bedeutet, dass die Anwendungen bzw. Sie als Programmierer für die Datenversorgung zuständig sind. Sie teilen also List & Label per Funktionsaufruf mit, welche Daten(felder) von Ihrer Anwendung überhaupt als druckbare Daten zur Verfügung gestellt werden (bspw. "Ein Feld namens <Name>, ein Feld namens <Vorname>" etc.) und welchen Inhalt dieses Feld hat. Woher Sie letztlich zur Druckzeit die Inhalte der Datenfelder bekommen, spielt für List & Label überhaupt keine Rolle. In den meisten Fällen dürfte vermutlich durch Sie ein Lesezugriff auf ein entsprechendes Datensatz-Feld einer Datenbank erfolgen.

Um den Designer zur Gestaltung der Druckformulare in Ihre Anwendung einzubauen, melden Sie per Funktionsaufruf jedes Ihrer vorhandenen bzw. gewünschten Datenfelder einmal bei List & Label an. Dabei können Sie neben dem Datenfeld-Namen auch noch optional einen Datentyp (z.B. Text, Numerisch, Logisch, Datum, etc.) übergeben, der bspw. für die Behandlung der Datenfelder in Formeln u.ä. im Designer relevant wird. Sie können außerdem einen Beispiel-Feldinhalt übergeben, der zur Designzeit für die Darstellung im Arbeitsbereich genutzt wird. Wenn Sie eine Echtdatenvorschau unterstützen wollen, berücksichtigen Sie die Hinweise im Kapitel "Direkter Druck und Export aus dem Designer".

Zur Druckzeit erfolgt die Datenübergabe im Prinzip analog, außer, dass anstatt des Beispiel-Feldinhaltes von Ihnen der Echtdaten-Feldinhalt übergeben werden muss. Dies geschieht für alle Felder, während Sie insgesamt über alle Ihre zu druckenden Datensätze iterieren.

5.4.2 Echtdatenvorschau oder Druck?

Prinzipiell läuft Ihre Druckschleife immer gleich ab, unabhängig davon, ob auf Drucker (*LL_PRINT_NORMAL*), Vorschau (*LL_PRINT_PREVIEW*) oder Datei (*LL_PRINT_FILE*) gedruckt wird. Die Unterscheidung wird lediglich in einem Parameter beim Start des Druckvorganges programmierseitig festgelegt (siehe *LlPrint[WithBox]Start()*). Sie können allerdings diese Entscheidung auch dem (End-)Anwender überlassen (*LL_PRINT_EXPORT*), indem Sie ihm im Drucker-Dialog von *LlPrintOptionsDialog()* eine Auswahlmöglichkeit des Druckziels inklusive aller Exportmodule anbieten.

5.4.3 Grundlegender Ablauf

Zunächst wird ein List & Label Job geöffnet (*LlJobOpen[LCID]()*) und ggf. anschließend globale List & Label Optionen (*LlSetOption()*) eingestellt.

Nun muss List & Label der Beginn des Druckvorgangs mitgeteilt werden (*LlPrint[With]BoxStart()*). Dabei wird außerdem spezifiziert, welche Etiketten- bzw. Formular-Definitionsdatei genommen werden soll. Zu diesem Zeitpunkt wird List & Label die angegebene Definitionsdatei öffnen und parsen. Dabei erfolgt auch eine syntaktische Überprüfung aller verwendeten Variablen, Felder und Formel-Ausdrücke. Dies bedeutet allerdings, dass List & Label bereits zu diesem Zeitpunkt alle von Ihnen zur Verfügung gestellten Variablen und Felder kennen muss. Sie müssen also vor dem Aufruf von *LlPrint[With]BoxStart()* alle Variablen und Felder mit den Funktionen *LlDefineVariable[Ext]()* und *LlDefineField[Ext]()* definiert haben.

Da es zu diesem Zeitpunkt nur um die Namen und Typen und nicht um aktuelle Inhalte geht, können Sie direkt dieselbe Routine verwenden, mit der Sie alle Felder und Variablen für den Designer anmelden (z.B. mit einem Beispieldateninhalt, der keine Rolle spielt, oder auch mit dem Inhalt des ersten Datensatzes).

Nach der optionalen Anzeige einer Druckerauswahlbox (*LlPrintOptionsDialog()*) erfolgt nun die eigentliche Druckschleife.

Ein Druckvorgang hat also prinzipiell folgendes Schema (Die Funktionen mit '*' sind optionale Aufrufe, die nicht unbedingt benötigt werden):

```
<öffne Job>
        (LlJobOpen, LlJobOpenLCID)
<definiere List & Label-Voreinstellungen>*
        (LlSetOption,
         LlSetOptionString,
         LlSetFileExtensions,
         LlSetNotificationMessage,
         LlSetNotificationCallback)
<Auswahl einer Datei>*
        (LlSelectFileDlgTitleEx)
<Ausdruck>      (siehe unten)
<schließe Job>
        (LlJobClose)
```

Ausdrucken von Etiketten und Karteikarten

Für den Etiketten- oder Karteikarten-Ausdruck (*LL_PROJECT_LABEL, LL_PROJECT_CARD*) sieht nun <Ausdruck> folgendermaßen aus:

```
<definiere alle vorhandenen Variablen>
        (LlDefineVariableStart,
         LlDefineVariable,
         LlDefineVariableExt,
         LlDefineVariableExtHandle)
<definiere Optionen>*
        (LlSetPrinterDefaultsDir)
<starte Ausdruck>
        (LlPrintStart,
         LlPrintWithBoxStart)
<definiere Druckoptionen>*
```

```
        (LlPrintSetOption,
         LlPrintSetOptionString,
         LlPreviewSetTempPath)
<lasse Benutzer Optionen verändern>*
        (LlPrintOptionsDialog,
         LlPrintOptionsDialogTitle,
         LlPrintSelectOffsetEx,
         [LlPrinterSetup])
<definiere unveränderliche Variablen>*
        (LlDefineVariable,
         LlDefineVariableExt,
         LlDefineVariableExtHandle)
<hole Druckerinfo für Fortschritts-Box>*
        (LlPrintGetOption,
         LlPrintGetOptionString,
         LlPrintGetPrinterInfo)
<überspringe nicht zu druckende Etiketten>*

<solange Daten zu drucken und kein Fehler oder Abbruch>
{
        <gib Fortschritts-Meldung>*
                (LlPrintSetBoxText,
                 LlPrintGetCurrentPage,
                 LlPrintGetOption)
        <definiere veränderliche Variablen>
                (LlDefineVariable,
                 LlDefineVariableExt,
                 LlDefineVariableExtHandle)
        <drucke Objekte>
                (LlPrint)
        <keine Warnung, kein Abbruch: nächster Datensatz>
}

<beende Ausdruck>
        (LlPrintEnd)
```

Ausdrucken von Listen

Und für den Listenausdruck (*LL_PROJECT_LIST*) wird <Ausdruck> wie folgt ersetzt:

```
<definiere alle möglichen Variablen>
        (LlDefineVariableStart,
         LlDefineVariable,
         LlDefineVariableExt,
         LlDefineVariableExtHandle)
<definiere alle möglichen Felder>
        (LlDefineFieldStart,
         LlDefineField,
         LlDefineFieldExt,
         LlDefineFieldExtHandle)
<definiere Optionen>*
        (LlSetPrinterDefaultsDir)
```

```
<starte Ausdruck>
        (LlPrintStart,
         LlPrintWithBoxStart)
<definiere Druckoptionen>*
        (LlPrintSetOption,
         LlPrintSetOptionString,
         LlPreviewSetTempPath)
<lasse Benutzer Optionen verändern>*
        (LlPrintOptionsDialog,
         LlPrintOptionsDialogTitle,
         LlPrintSelectOffsetEx,
         [LlPrinterSetup])
<definiere unveränderliche Variablen>
        (LlDefineVariable,
         LlDefineVariableExt,
         LlDefineVariableExtHandle)
<drucke Variablen>       (drucke alle Objekte)
         (LlPrint)
<solange "Seite-Voll"-Warnung (LL_WRN_REPEAT_DATA) wiederholen>
        (LlPrint)

<wiederhole>
{
        <definiere Felder>
                (LlDefineField,
                 LlDefineFieldExt,
                 LlDefineFieldExtHandle)
        <drucke Zeile>
                (LlPrintFields)
        <solange -"Seite-Voll"-Warnung (LL_WRN_REPEAT_DATA) wiederholen>
                <definiere seitenspezifische Variablen>*
                (LlDefineVariable,
                 LlDefineVariableExt,
                 LlDefineVariableExtHandle)
                (LlPrint)
                (LlPrintFields)
        <gehe zum nächster Datensatz>
        <gib Fortschritts-Meldung>*
        (LlPrintSetBoxText,
         LlPrintGetCurrentPage,
         LlPrintGetOption)
}
<bis
 -Fehler oder
 -Dateiende oder
 -Abbruch durch Benutzer
>
<Drucke abschließende Fußzeilen und angehängte Objekte>
        (LlPrintFieldsEnd)
<solange "Seite-Voll"-Warnung (LL_WRN_REPEAT_DATA) wiederholen>
        (LlPrintFieldsEnd)
```

```
<beende Ausdruck>
        (LlPrintEnd)}
```

Es empfiehlt sich grundsätzlich, den Fehlercode auszuwerten, insbesondere Funktionen, welche eine Benutzerinteraktion auslösen, können z.B. *LL_ERR_USER_ABORTED* zurückgeben, wenn der Benutzer auf den Abbruch-Button drückt!

5.4.4 Erläuterungen

Druck-Beginn: Einlesen der Projektdatei

Bevor man den Druck starten kann, muss man erst wissen, welches Projekt geladen werden soll und welche Variablen ihm zur Verfügung gestellt werden sollen.

Nach der optionalen Frage an den Benutzer nach der Projektdatei über *LlSelectFileDlgTitleEx()* müssen alle Variablen/Felder definiert werden (analog zum Designer-Aufruf), die dieses Projekt haben könnte. Wenn List & Label auf einen Ausdruck stößt, in dem eine unbekannte Variable vorkommt, beendet List & Label den Ladevorgang (und damit den Druckvorgang) und gibt den entsprechenden Fehlercode zurück.

Das eigentliche Einlesen der Projektdatei wird dabei gestartet durch:

```
LlPrintWithBoxStart(hJob, LL_PROJECT_LABEL, aczProjectFile, LL_PRINT_NORMAL,
                LL_BOXTYPE_BRIDGEMETER, hWindow, "Mein Test");
```

Wenn von dieser Funktion kein Fehler zurückgegeben wurde, hat List & Label jetzt die Definition des Projekts eingelesen und ist bereit zum Ausdruck. Der Drucker ist aber zu diesem Zeitpunkt noch nicht initialisiert, das kommt erst bei dem ersten Aufruf einer Funktion, die eine Ausgabe provoziert.

Denn jetzt kann man erst noch die Druckparameter einstellen. Wenn man dem Benutzer die Änderung der Druckparameter gestatten will, ruft man den Dialog dafür mit

```
LlPrintOptionsDialog(hJob, hWindow, "Druck-Parameter");
```

auf. Über *LlSetOption()* und *LlSetOptionString()* werden vor diesem Aufruf programmeigene Standardwerte vorgegeben, man kann beispielsweise unerwünschte Auswahlfelder im Dialog unterdrücken, z.B. über

```
LlPrintSetOption(hJob, LL_PRNOPT_COPIES, LL_COPIES_HIDE);
```

für die Abfrage nach der Zahl der Kopien.

Wenn man (siehe unten) schon jetzt weiß, wie viele Datensätze ausgegeben werden, könnte man z.B. auch als Titelzeile anzeigen lassen, wie viele Datensätze oder Seiten zu drucken sind.

Wenn im Dialog "Änderungen permanent" angekreuzt worden ist, wird die gewählte Druckereinstellung in eine sog. "Drucker-Definitionsdatei" gespeichert (Schreib- und Löschrechte müssen vorhanden sein), die alten Einstellungen gehen verloren. Ursprüng-

lich wird die Druckereinstellung im Designer unter Projekt > Seitenlayout bestimmt. Wenn die Drucker-Definitionsdatei nicht vorhanden ist, wird als Druckereinstellung der Windows-Standarddrucker verwendet. Weiterführende Informationen zur Drucker-Definitionsdatei finden Sie im Kapitel "Die List & Label Dateien".

Wichtige Hinweise für Listenprojekte

Variablen sind bei Listenprojekten Werte, die für eine Seite gleich bleiben und Felder übernehmen die datensatzabhängigen Daten. Diese druckt man dann mit *LlPrintFields()*.

Bei dem Aufruf von *LlPrint()* werden die Objekte gezeichnet, die nicht Listen oder nicht an Listen angehängt sind. Wenn die Option *LL_OPTION_DELAYTABLEHEADER* nicht gesetzt ist, werden dann auch die Listenköpfe gedruckt, ansonsten kommen diese erst beim ersten Aufruf von *LlPrintFields()* auf das Papier. Danach erwartet List & Label die Definition der Datensätze.

Bei jedem *LlPrintFields()* wird getestet, ob der auszugebende Datensatz noch auf derselben Seite in die Liste passt. Wenn er nicht vollständig gedruckt werden konnte, meldet die Funktion *LL_WRN_REPEAT_DATA* zurück - dann muss man daran denken, den Satzzeiger nicht zu erhöhen, da genau dieser Datensatz auf der folgenden Seite erneut gedruckt werden soll.

Sind die Listen gefüllt, muss man jetzt die Variablen für angehängte Objekte definieren, bevor man *LlPrint()* aufruft, denn bei diesem *LlPrint()* werden nun die angehängten Objekte gefüllt, die neue Seite begonnen und - siehe oben - wieder die Objekte der neuen Seite inklusive Listenköpfe gedruckt.

Ein vorzeitiger Seitenumbruch ist möglich, indem man einfach zum gewünschten Zeitpunkt *LlPrint()* aufruft – dieser Aufruf beendet die aktuelle Seite, wenn diese schon bedruckt ist.

Kopien

Es gibt prinzipiell zwei verschiedene Bedeutungen für "Kopien".

a) Kopien von Etiketten und evtl. Karteikarten sollen meist nicht auf verschiedenen Seiten, sondern auf hintereinander liegenden Etiketten sein. Um diese Art zu unterstützen, fragen Sie vor dem ersten *LlPrint()* die Zahl der Kopien ab, weil Sie einfach jeden Datensatz entsprechend oft ausgeben müssen, und setzen den Kopienzähler auf 1 (damit List & Label nicht den Drucker anweist, entsprechend Kopien zu drucken!).

```
// Benutzer kann Kopienanzahl ändern...:
LlPrintOptionsDialog(hJob, hWnd, "Druck...");

nCopies = LlPrintGetOption(hJob, LL_PRNOPT_COPIES);
LlPrintSetOption(hJob, LL_PRNOPT_COPIES, 1);
```

Bei dem Ausdruck muss dann jedes Etikett entsprechend oft ausgegeben werden:

```
for (nCopy = 1; (nCopy < nCopies) && (nError == 0); ++nCopy)
{
```

```
        nError = LlPrint(hJob);
        // unterstütze AUTOMULTIPAGE (meist bei Karteikarten)
        while (nError == LL_WRN_REPEAT_DATA)
                nError = LlPrint(hJob);
}
```

b) Wirkliche Seitenkopien, d.h. mehrere identische Seiten, meist bei Reports. Diese werden direkt von List & Label behandelt, so dass hier keine spezielle Unterstützung seitens des Entwicklers notwendig ist.

Optimierung der Geschwindigkeit

a) Programmoptimierung

Zuerst einmal kann man Variablendefinitionen, die über den Ausdruck hinweg konstant sein sollen, aus der Druckschleife herausziehen. Wenn Sie also bei Listen immer Ihren Firmennamen als Briefkopf ausgeben wollen, definieren Sie diesen am besten außerhalb der Schleife vor *LlPrintWithBoxStart()*.

b) Wird die Variable / das Feld verwendet?

Man kann außerdem abfragen, welche Variablen oder Felder in den Ausdrücken verwendet werden. Wenn das Angebot an Variablen oder Feldern größer ist als die tatsächlich verwendete Zahl, oder die Beschaffung der Datenwerte aufwändig ist (Unterabfragen, Berechnungen etc.), dann lohnt sich der Einsatz dieser Funktionen. Der Aufruf von *LlGetUsedIdentifiers()* liefert alle im Projekt verwendeten Variablen und Felder. *LlGetUsedIdentifiersEx()* erlaubt darüber hinaus, nach der Art (Variable oder Feld) zu unterscheiden.

Sie sollten diese Funktion vor dem Druckstart aufrufen und nur die Felder bzw. Variablen aus Ihrer Datenquelle übergeben, die auch wirklich verwendet werden.

c) Globaler "Dummy"-Job

Einige der von List & Label verwendeten Systembibliotheken (z.B. riched20.dll) scheinen unter bestimmten Umständen Ressourcenverluste zu verursachen. Diese sind sehr klein, fallen aber bei jedem Laden und Entladen der DLL an.

Diese DLLs werden von List & Label bei jedem Öffnen bzw. Schließen des "ersten" Jobs ge- bzw. entladen. Insofern sollten Sie in Ihrer Applikation ein häufiges *LlJobOpen()* / *LlJobClose()* vermeiden oder aber zu Beginn einen Dummy-Job öffnen und diesen bis zum Ende geöffnet halten. Damit wird das ständige Laden und Entladen der DLLs umgangen, und neben einer dadurch erreichten Geschwindigkeitsoptimierung werden auch die Ressourcenverluste nicht mehr auftreten.

5.5 Drucken relationaler Daten

List & Label bietet Ihnen komfortable Möglichkeiten, Druckprojekte mit mehreren Datenbanktabellen (hierarchische Reports) zu gestalten. Dies stellt die für den Anwender kom-

fortabelste Art dar, mit mehreren Tabellen, Kreuztabellen und Charts zu arbeiten. Grundsätzlich verwenden Sie für solche Projekte den Projekttypen *LL_PROJECT_LIST.* Die Projekttypen *LL_PROJECT_LABEL* oder *LL_PROJECT_CARD* unterstützen genau eine Tabelle und eine beliebige Anzahl Sortierungen für diese Tabelle, die genauso wie bei *LL_PROJECT_LIST* Projekten gesetzt und abgefragt werden können.

Im Folgenden wird "Tabelle" als Synonym für eine Gruppe zusammengehöriger Felder im List & Label-Designer bzw. im Objekte-Toolfenster verwendet. Sie sind hierbei nicht an "echte" Datenbanken gebunden, auch Klassenarrays oder dynamisch erzeugte Daten können eine "Tabelle" darstellen, z.B. alle Member-Variablen einer Klasse. Beachten Sie, dass Sie im List & Label Designer dennoch nur mit einem einzigen Berichtscontainer-Objekt arbeiten. Dieses kann aber mehrere Unterobjekte wie Tabellen, Kreuztabellen und Charts enthalten.

Bei der Ausgabe im Druck sind sie ebenfalls nicht an die Tabellenausgabe gebunden – mit dem gleichen Code werden auch Kreuztabellen sowie Chartobjekte (auch in Tabellenspalten) gefüllt.

Sobald Sie einzelne Tabellen über *LlDbAddTable()* hinzugefügt haben, können Ihre Anwender im Designer mit dem Objekte-Toolfenster die Struktur des Containers bearbeiten. Weitere Hinweise zum Design finden Sie im entsprechenden Kapitel des Designerhandbuchs. Dieses Kapitel konzentriert sich auf die Fragen der Ansteuerung solcher Reports.

Für viele Programmiersprachen sind Beispiele für die Ansteuerung von List & Label bei Verwendung mehrerer Tabellen im Lieferumfang enthalten.

5.5.1 Benötigte API-Funktionen

Die API-Funktionen, die Sie für die Ansteuerung dieser Funktionalität benötigen beginnen einheitlich mit *LlDb...* bzw. *LlPrintDb...* Sie können Tabellen hinzufügen (*LlDbAddTable()*), Sortierungen innerhalb der Tabellen zur Verfügung stellen (*LlDbAddTableSortOrder()*) und Beziehungen zwischen Tabellen definieren (*LlDbAddTableRelation()*).

Zur Druckzeit können Sie dann die derzeit aktive Tabelle abfragen (*LlPrintDbGetCurrentTable()*) sowie analog die gerade aktive Beziehung und Sortierung erhalten (*LlPrintDbGetCurrentTableSortOrder(), LlPrintDbGetCurrentTableRelation()*). Ausführliche Beispiele für die Verwendung dieser Funktionen finden Sie im Laufe dieses Kapitels.

5.5.2 Aufruf des Designers

Zunächst müssen alle Tabellen bei List & Label bekannt gemacht werden, die Sie dem Benutzer für das Design zur Verfügung stellen möchten:

```
LlDbAddTable(hJob, "", ""); // evtl. vorhandene Tabellen löschen
LlDbAddTable(hJob, "Orders", "Bestellungen");
LlDbAddTable(hJob, "OrderDetails", "Bestellposten");
```

Der erste Parameter ist wie üblich das Job-Handle des List & Label-Jobs. Der zweite Parameter ist die TableID, dies ist der Wert, den Sie beim Druckvorgang von *LlPrintDb-*

GetCurrentTable() zurückgeliefert bekommen. Der dritte Parameter ist der Anzeigename der Tabelle im Designer. Wenn Sie NULL bzw. einen Leerstring übergeben, sind TableID und Anzeigename identisch.

Eine besondere Rolle kommt dabei dem Tabellennamen "LLStaticTable" zu. Dieser ist reserviert und kann für das Einfügen von 'statischen' Inhalten (feste Texte oder Inhalte von Variablen, Chartunterschriften etc.) verwendet werden. Eine solche statische Tabelle steht dann als "Freier Inhalt" im Datenquellen-Auswahldialog des Designers zur Verfügung und kann vom Benutzer nur mit Datenzeilen befüllt werden. In Ihrem Code müssen Sie entsprechend auf die Tabelle reagieren - wie wird im Druck-Unterkapitel erläutert.

Im nächsten Schritt werden die Beziehungen zwischen den Tabellen definiert. List & Label unterscheidet hierbei nicht direkt zwischen unterschiedlichen Beziehungstypen (n:m, 1:n) – Sie melden eine Beziehung einfach mit einer ID an, die Sie hinterher im Druck abfragen können:

```
LlDbAddTableRelation(hJob, "OrderDetails", "Orders",
        "Orders2OrderDetails", NULL);
```

Mit diesem Befehl haben Sie bekanntgemacht, dass "OrderDetails" im Designer als Untertabelle für die Tabelle "Orders" zur Verfügung stehen soll. In diesem Falle wurde nur die ID der Relation übergeben, diese erscheint dann auch im Designer.

Zuletzt können Sie noch Sortierungen übergeben, die für die jeweiligen Tabellen zur Auswahl stehen sollen. Auch hier erhält wieder jede Sortierung eine eindeutige ID, die Sie im Druck abfragen können:

```
LlDbAddTableSortOrder(hJob, "Orders", "OrderDate ASC",
        "Order Date [+]");
LlDbAddTableSortOrder(hJob, "Orders", "OrderDate DESC",
        "Order Date [-]");
```

Nun kann der Benutzer im Designer eine dieser Sortierungen sowie die Grundeinstellung "unsortiert" auswählen. Wenn Sie die Tabellen über *LlDbAddTableEx()* anmelden können Sie auch Unterstützung für mehrfache Sortierungen signalisieren.

Der restliche Ablauf des Designeraufrufs ist analog zum "normalen" Aufruf, d.h. das gesamte Schema eines Designeraufrufs mit mehreren Tabellen würde so aussehen:

```
<öffne Job>
        (LlJobOpen, LlJobOpenLCID)
<definiere List & Label-Voreinstellungen>
        (LlSetOption,
         LlSetOptionString,
         LlSetDebug,
         LlSetFileExtensions,
         LlSetNotificationMessage,
         LlSetNotificationCallback)
<zu bearbeitende Datei bestimmen lassen>
        LlSelectFileDlgTitleEx
```

```
<definiere Datenstruktur>
        (LlDbAddTable,
         LlDbAddTableRelation,
         LlDbAddTableSortOrder)
<definiere Variablen>
        (LlDefineVariableStart,
         LlDefineVariable,
         LlDefineVariableExt,
         LlDefineVariableExtHandle)
<definiere Felder>
        (LlDefineFieldStart,
         LlDefineField,
         LlDefineFieldExt,
         LlDefineFieldExtHandle)
<sperre Funktionen>
        (LlDesignerProhibitAction,
         LlDesignerProhibitFunction)
<Aufruf des Designers>
        (LlDefineLayout)
<schließe Job>
        (LlJobClose)
```

Alle Felder einer Tabelle müssen in der Form "<TabellenID>.<Feldname>" benannt werden, damit List & Label diese korrekt den einzelnen Tabellen zuordnen kann. Stellen Sie also sicher, dass Sie jedem Feldnamen die TabellenID voranstellen.

Wenn Sie zusätzlich zu den Feldern einer Tabelle alle Felder von 1:1 verknüpften Tabellen zur Verfügung stellen möchten, beachten Sie bitte die Hinweise im Kapitel "Umgang mit 1:1-Relationen"

5.5.3 Steuerung des Druckvorgangs

Die Erzeugung von hierarchischen Reports mit List & Label erfolgt im Prinzip analog wie im vorigen Kapitel beschrieben. Über die Funktion *LlPrintDbGetCurrentTable()* kann abgefragt werden, welche Tabelle im Augenblick befüllt werden muss. Diese wird dann – wie gehabt – über *LlDefineField[Ext]()* und *LlPrintFields()* ausgegeben. Je nach Druckvorlage können zwei Sonderfälle auftreten:

- Nach der Ausgabe einer Tabelle kann der Benutzer eine zweite Tabelle platziert haben
- Der Benutzer kann auch zur aktuellen Tabelle eine Untertabelle eingefügt haben

Diese beiden Fälle werden in den beiden nächsten Abschnitten betrachtet.

Mehrere unabhängige Tabellen

Ein Beispiel hierfür wäre eine Liste der Kunden, die von einer Chartauswertung der Angestellten gefolgt werden soll. Beide Tabellen sind nicht voneinander abhängig. Die Druckschleife für eine solche Ausgabe ist der aus dem vorigen Kapitel sehr ähnlich, mit

einem Unterschied. Der Abschluss des Drucks einer Tabelle erfolgt über *LlPrintFieldsEnd()*, Sie bekommen an dieser Stelle aber möglicherweise den Rückgabewert *LL_WRN_TABLECHANGE.* Dies bedeutet, dass im Layout noch eine weitere zu druckende Tabelle vorhanden ist. Am Einfachsten teilen Sie die Druckschleife so auf, dass Sie verschiedene Unterroutinen haben.

Der erste Teil meldet die Daten und Datenstruktur an, startet den Druckjob und initialisiert die erste Seite, so dass mit dem Druck einer Tabelle begonnen werden kann. Die optionalen Teile der Druckschleife sind hier aus Übersichtlichkeitsgründen nicht enthalten, diese sind analog zu den im letzten Kapitel beschriebenen.

```
<definiere Datenstruktur>
        (LlDbAddTable,
         LlDbAddTableRelation,
         LlDbAddTableSortOrder)
<definiere alle möglichen Variablen>
        (LlDefineVariableStart,
         LlDefineVariable,
         LlDefineVariableExt,
         LlDefineVariableExtHandle)
<definiere alle möglichen Felder>
        (LlDefineFieldStart,
         LlDefineField,
         LlDefineFieldExt,
         LlDefineFieldExtHandle)
         LlSetPrinterDefaultsDir
<starte Ausdruck>
        (LlPrintStart,
         LlPrintWithBoxStart)
<definiere Optionen>
        (LlPrintSetOption,
         LlPrintSetOptionString,
         LlPreviewSetTempPath)
<definiere unveränderliche Variablen>
        (LlDefineVariable,
         LlDefineVariableExt,
         LlDefineVariableExtHandle)
<drucke Variablen>       (drucke alle Objekte)
                (LlPrint)
<solange Warnung wiederholen>
                (LlPrint)
```

Der zweite Teil der Druckschleife benötigt eine Hilfsfunktion. Diese druckt die Daten einer einzelnen (Datenbank-)Tabelle:

```
Funktion DruckeTabelle(DataTable Datenobjekt)
{
        // DataTable ist ein geeignetes Datenzugriffsobjekt, z.B. eine
        // Datenbanktabelle, ein Klassenarray o.ä.
```

```
        <wiederhole>
        {
                <definiere Felder von DataTable>
                        (LlDefineField,
                         LlDefineFieldExt,
                         LlDefineFieldExtHandle)
                <drucke Zeile>
                        (LlPrintFields)
                <solange Warnung wiederholen>
                        (LlPrint,
                         LlPrintFields)
                <nächster Datensatz in DataTable>
        }
        <bis letzter Datensatz in DataTable erreicht>

        <Fußzeile drucken>
                (Rückgabewert = LlPrintFieldsEnd)
        <solange Warnung "Seite voll" wiederholen>
                (Rückgabewert = LlPrintFieldsEnd)
        <Ergebnis = Rückgabewert>
}
```

Als Rückgabewert erhält man die Information, ob eine weitere Tabelle folgt (*LlPrintFieldsEnd()* liefert dann *LL_WRN_TABLECHANGE* zurück), oder ob der Druck abgeschlossen werden kann (Rückgabewert 0).

Damit kann der zweite Teil des Drucks, also der Teil nach der Initialisierung der ersten Seite, wie folgt realisiert werden:

```
<wiederhole>
{
        <Hole aktuellen Tabellennamen>
                (LlPrintDbGetCurrentTable)
        <Hole aktuelle Sortierung>
                (LlPrintDbGetCurrentTableSortOrder)
        <Generiere ein passendes DataTable-Objekt>
        <Rückgabewert=DruckeTabelle(DataTable)>
}
<bis Rückgabewert <> LL_WRN_TABLECHANGE>

<beende Ausdruck>
        (LlPrintEnd)
```

Wenn Sie die "LLStaticTable"-Tabelle für freien Inhalt angemeldet haben und *LlPrintDbGetCurrentTable()* diese Tabelle als aktuelle Tabelle liefert, muss Ihre Druckschleife darauf mit dem Druck einer einzelnen Datenzeile über LlPrintFields() reagieren. Im Beispiel oben könnten Sie für den Fall von "LLStaticTable" einfach ein DataTab-

le-Objekt mit nur einem beliebigen Datensatz erzeugen, dann läuft der Druck automatisch korrekt.

Dieser Code erlaubt bereits die beliebige Abfolge von mehreren Tabellen hintereinander. Im folgenden Abschnitt wird eine Erweiterung für den Druck von Untertabellen vorgenommen.

Einfache 1:n-Relationen

Das typische Beispiel für diesen Fall ist die bereits angesprochene 1:n-Beziehung Bestellung – Bestellposten. In diesem Fall werden nach jeder Datenzeile mit Bestell-Daten n Bestellpositionen ausgegeben. Die Ausgabe einer Zeile erfolgt mit *LlPrintFields()*. Analog zum Verhalten von *LlPrintFieldsEnd()* im letzten Abschnitt erhalten Sie – wenn der Benutzer eine Untertabelle platziert hat – nun auch hier *LL_WRN_TABLECHANGE* zurück und müssen darauf entsprechend reagieren. Sie können die Tabellenbeziehung über *LlPrintDbGetCurrentRelation()* und den Namen der Kindtabelle über *LlPrintDbGetCurrentTableName()* erfragen. Mit diesen Informationen können Sie dann wiederum die Hilfsfunktion *DruckeTabelle()* aus dem vorigen Abschnitt aufrufen. Dieser Aufruf muss direkt nach dem *LlPrintFields()* erfolgen – also aus der Funktion *DruckeTabelle()* selbst. Die Funktion muss also dahingehend abgeändert werden, dass sie sich selbst rekursiv aufruft:

```
Funktion DruckeTabelle(DataTable Datenobjekt)
{
        // DataTable ist ein geeignetes Datenzugriffsobjekt, z.B. eine
        // Datenbaktabelle, ein Klassenarray o.ä.

        <wiederhole>
        {
                <definiere Felder von DataTable>
                        (LlDefineField,
                         LlDefineFieldExt,
                         LlDefineFieldExtHandle)
                <drucke Zeile>
                        (LlPrintFields)
                <solange Warnung wiederholen>
                        (LlPrint,
                         Rückgabewert = LlPrintFields)
                <solange Rückgabewert LL_WRN_TABLECHANGE wiederholen>
                {
                        <Hole aktuellen Tabellennamen>
                         (LlPrintDbGetCurrentTable)
                        <Hole aktuelle Beziehung>
                         (LlPrintDbGetCurrentTableRelation)
                        <Hole aktuelle Sortierung>
                         (LlPrintDbGetCurrentTableSortOrder)
                        <Generiere ein passendes Kind-DataTable-Objekt>
                        <Rückgabewert=DruckeTabelle(Kind-DataTable)>
                }
                <nächster Datensatz in DataTable>
        }
```

```
        <bis letzter Datensatz in DataTable erreicht>

        <Fußzeile drucken>
                (Rückgabewert = LlPrintFieldsEnd)
        <solange Warnung "Seite voll" wiederholen>
                (Rückgabewert = LlPrintFieldsEnd)
        <Ergebnis = Rückgabewert>
}
```

Damit können nun auch beliebige Abfolgen von Untertabellen ausgegeben werden. Die Rekursion stellt hierbei sicher, dass dies beliebig "tief" funktioniert, d.h. der Code kann in dieser Form auch mehrstufige Relationen korrekt ansteuern.

Die rekursive Druckschleife

Für eine vollständige Druckschleife, die sowohl Folgetabellen als auch Untertabellen korrekt unterstützt muss nun nichts mehr getan werden – der Code aus den beiden letzten Abschnitten stellt sicher, dass der komplette Baum der Tabellenstruktur durchlaufen wird.

Insofern bleibt nur noch Feinarbeit zu leisten – z.B. die Anzeige eines Fortschrittsbalkens. Da das Layout beliebig komplex werden kann, kann nicht mehr einfach die aktuelle Position innerhalb der Datenquelle zur Gesamtzahl der Datensätze in Verhältnis gesetzt werden. Dieser Ansatz funktioniert schon dann nicht mehr korrekt, wenn der Benutzer zwei Tabellen hintereinander im Designer platziert. Daher bietet List & Label über die Funktion *LlPrintDbGetRootTableCount()* die Möglichkeit, die Anzahl der Tabellen auf der obersten Ebene ("Root") zu bestimmen. Dann können Sie die Fortschrittsanzeige immer dann aktualisieren, wenn Sie auf dieser Ebene einen Datensatz ausgeben.

Für den maximal pro Tabelle verfügbaren Prozentsatz gilt

```
INT nMaxPerc = 100/LlPrintDbGetRootTableCount();
```

Wenn Sie die Root-Tabellen von 0.. *LlPrintDbGetRootTableCount()-1* durchindizieren, können Sie den Gesamt-Prozentsatz dann als

```
INT nPercTotal = nMaxPerc*nIndexCurrentTable+(nPerc/100*nMaxPerc);
```

berechnen, wobei *nPerc* der Prozentposition innerhalb der aktuellen Tabelle entspricht. Für die eigentliche Aktualisierung der Prozentanzeige kann dann die *DruckeTabelle()*-Funktion aus dem letzten Abschnitt angepasst werden. Durch einen weiteren Eingangsparameter kann die augenblickliche Rekursionstiefe bestimmt werden – ist diese 0 wird gerade ein "Root"-Datensatz gedruckt und die Anzeige kann aktualisiert werden:

```
Funktion DruckeTabelle(DataTable Datenobjekt, Rekursionstiefe Tiefe)
{
        <wiederhole>
        {
                <definiere Felder von DataTable>
                ...
                <wenn Tiefe==0 Fortschrittsanzeige aktualisieren>
                        (LlPrintDbGetRootTableCount, LlPrintSetBoxText)
```

```
                <drucke Zeile>
                        (LlPrintFields)
                <solange Warnung wiederholen>
                        (LlPrint,
                        Rückgabewert = LlPrintFields)
                <solange Rückgabewert LL_WRN_TABLECHANGE wiederholen>
                {
                        ...
                        <Generiere ein passendes Kind-DataTable-Objekt>
                        <Rückgabewert=DruckeTabelle(Kind-DataTable, Tiefe+1)>
                }
                ...
        }
}
```

Übergabe der Master-Daten als Variablen

Im Falle einer Bestellung mit den zugehörigen Bestellposten kann es auch erwünscht sein, die "Master"-Daten, d.h. in diesem Beispiel die Daten der Orders-Tabelle als Variablen zu übergeben. So kann der Adressat dann z.B. in einem Textobjekt ausgegeben werden, die Posten im Tabellenobjekt.

Damit Ihnen die "OrderDetails"- Tabelle mit der benötigten Relation zum Einfügen im Tabellenstruktur-Dialog angeboten wird, müssen Sie zuvor mitteilen, dass Sie die Master-Daten als Variablen verwalten und somit schon in der äußersten Ebene des Tabellenobjekts die 1:n-verknüpfte OrderDetails- Tabelle zur Verfügung stellen möchten. Hierzu verwenden Sie *LlDbSetMasterTable()*. Die benötigten Aufrufe wären also

```
LlDbAddTable(hJob, "Orders", "Bestellungen");
LlDbAddTable(hJob, "OrderDetails", "Bestellposten");
LlDbAddTableRelation(hJob, "OrderDetails", "Orders",
        "Orders2OrderDetails", NULL);
LlDbSetMasterTable(hJob, "Orders");
```

Der Ablauf der Druckschleife erfolgt dann analog zu oben, allerdings müssen Sie jetzt ggf. schon auf der äußersten Ebene (vgl. Kap. Mehrere unabhängige Tabellen) die geeignete Kind- Datenbanktabelle bereitstellen:

```
<wiederhole>
{
        <Hole aktuellen Tabellennamen>
                (LlPrintDbGetCurrentTable)
        <Hole aktuelle Sortierung>
                (LlPrintDbGetCurrentTableSortOrder)
        <Hole aktuelle Beziehung>
                (LlPrintDbGetCurrentTableRelation)
        <Wenn Beziehung leer>
                <Generiere ein passendes DataTable-Objekt>
        <sonst>
                <Generiere ein passendes Kind-DataTable-Objekt>
        <Rückgabewert=DruckeTabelle(DataTable)>
}
```

```
<bis Rückgabewert <> LL_WRN_TABLECHANGE>

<beende Ausdruck>
        (LlPrintEnd)
```

5.5.4 Umgang mit 1:1-Relationen

Meist werden 1:1-Relationen zur Berichtserstellung bereits per Datenbankabfrage über ein SQL JOIN so zusammengefasst, dass die Daten innerhalb derselben Tabelle zur Verfügung stehen. Sollte dies nicht der Fall sein oder sollten Sie dies nicht wünschen, so können Sie 1:1-Relationen auch visuell im Variablenfenster innerhalb der Felder der Elterntabelle einblenden. Hierzu müssen Sie die Felder mit einer besonderen Syntax anmelden.

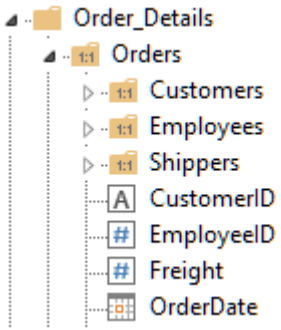
1:1 Relation ohne Schlüsselfeldangabe

Wenn die Schlüsselfelder für die Relation nicht relevant sind, es also nur eine 1:1-Relation zwischen den beiden beteiligten Tabellen gibt, können Sie die Felder wie folgt anmelden:

<Elterntabelle>:<verknüpfte Tabelle>.<Feldname>, also z.B.

OrderDetails:Orders.OrderDate

Damit erscheint innerhalb der OrderDetails-Hierarchie im Variablenfenster ein Ordner mit dem Feld OrderDate der Orders-Tabelle:



Sie müssen nun natürlich auch dafür sorgen, dieses Feld für jeden Datensatz, den Sie in der OrderDetails-Tabelle durchlaufen, mit dem zugehörigen Inhalt zu füllen.

1:1 Relation mit Schlüsselfeldangabe

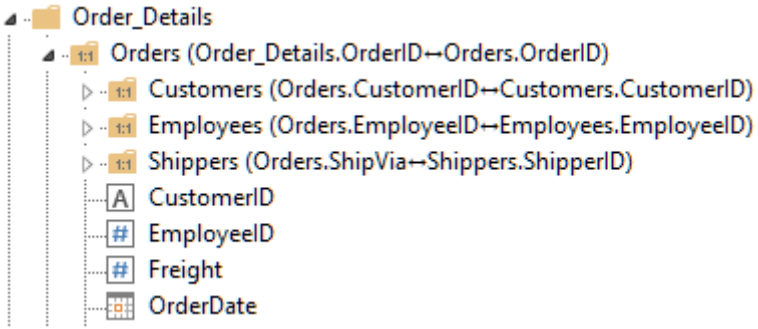
Im Falle mehrfacher 1:1-Verknüpfungen ist es für den Benutzer wichtig zu sehen, über welches der Schlüsselfelder die Verknüpfung vorgenommen wird. In diesem Falle können Sie die Felder wie folgt anmelden:

<Elterntabelle>.<Schlüsselfeld Elterntabelle>@<verknüpfte Tabelle>.<Schlüsselfeld verknüpfte Tabelle>:<Feldname>, also z.B.

OrderDetails.OrderID@Orders.OrderID:OrderDate

(SQL Äquivalent: "SELECT OrderDate FROM Orders WHERE OrderDetails.OrderID=Orders.OrderID")

Nun erscheint im Variablenfenster neben der Verknüpfung auch die Schlüsselfeldangabe:



Auch hier muss das Feld für jeden Datensatz, den Sie in der OrderDetails-Tabelle durchlaufen, mit dem zugehörigen Inhalt gefüllt werden.

Performance-Tipps

Gerade beim Umgang mit 1:1-Relationen sollten Sie unbedingt prüfen, ob der Benutzer überhaupt ein Feld der verknüpften Tabelle verwendet hat. Sie können dies durch Verwendung der Wildcard-Option bei *LlPrintIsFieldUsed()* erreichen. Um etwa zu sehen, ob ein Feld der 1:1 verknüpften Tabelle "Orders" innerhalb der "OrderDetails"- Tabelle verwendet wurde, können Sie

```
LlPrintIsFieldUsed(hJob, "OrderDetails.OrderID@Orders.OrderID*");
```

verwenden. Erhalten Sie hier einen Rückgabewert von 0, so ist kein Feld der Orders-Tabelle verwendet und Sie brauchen die Inhalte auch nicht bereitzustellen.

5.6 Callbacks und Notifications

Dieses Kapitel ist nur für Anwendungen interessant, die nicht über .NET, OCX- oder VCL-Controls auf List & Label zugreifen. Entwickler, die das .NET, OCX- oder VCL-Control verwenden, können dieses Kapitel überspringen.

5.6.1 Überblick

Folgendes Prinzip steckt hinter den Begriffen "Callbacks und Notifications": wenn List & Label etwas nicht weiß, fragt es einfach Ihr Programm. Sie müssen nicht alle Antworten vorprogrammieren, sondern nur die, zu denen Sie Anfragen ausdrücklich wünschen.

Beispielsweise gibt es programmdefinierbare Objekte sog. User-Objekte, die von List & Label wie eine "Blackbox" behandelt werden. Wenn List & Label solch ein Objekt ausgeben muss, wendet es sich an Ihr Programm mit der Bitte, diese Aufgabe durchzuführen. Diese Art "Objekt-Container" kann verwendet werden, um Sonderwünsche nach speziellen Objekten, beispielsweise extern erstellte Charts, erfüllen zu können. Sie müssen hier keine komplette Schnittstelle, sondern nur eine einzige Routine zur Verfügung stellen.

Aber auch bei der Datenausgabe kann etwas gezaubert werden - über die Callback-Möglichkeit kann man Daten auf eine Seite hinzufügen, die vom Programm gesteuert sind (und somit vom Benutzer auch nicht im Designer entfernbar), man kann Objekte kurzerhand verstecken (das kann man aber auch über *LlPrintEnableObject()* oder die Darstellungsbedingung eines Objekts) oder objektspezifische Bemalung ausgeben.

Um Callbacks/Events zu nutzen, muss eine der folgenden Möglichkeiten implementiert werden:

- man definiert eine Callback-Routine, deren Adresse man List & Label über *LlSetNotificationCallback()* mitteilt, oder
- man reagiert auf von List & Label gesendete Nachrichten (Windows Messages). Diese werden von List & Label an das Fenster, das bei *LlDefineLayout()* und *LlPrintWithBoxStart()* angegeben wird, geschickt.

In beiden Fällen bekommt man dann ausführlichere Informationen über die durchzuführende Aufgabe.

Die nachfolgenden Kapitel beschreiben, wie man eine solche Callback Routine implementiert. Eine Übersicht über alle verfügbaren Callbacks finden Sie in Kapitel "Referenz der Callback-Notifications".

5.6.2 User-Objekte

Sollte Ihnen in List & Label eine Objektart fehlen, dann lässt sich List & Label über sog. User-Objekte erweitern.

Wenn Sie in Ihrem Programm eine Variable über

```
LlDefineVariableExt(hJob, <Name>, <Inhalt>, LL_DRAWING_USEROBJ, NULL);
```

definiert haben, kann der Benutzer im Designer ein Objekt definieren, das mit dieser Variablen zusammenhängt. Dies geschieht analog zu normalen *LL_DRAWING*-Variablen.

Wenn List & Label dieses Objekt im Preview oder beim Druck ausgeben soll, ruft es über den Callback *LL_CMND_DRAW_USEROBJ* Ihr Programm auf, um diese Aufgabe an Ihr Programm weiterzugeben.

Dasselbe können Sie auch für Tabellenfelder durchführen, damit der Benutzer die Möglichkeit hat, sich ein User-Objekt in die Tabelle einzubauen:

```
LlDefineFieldExt(hJob, <Name>, <Inhalt>, LL_DRAWING_USEROBJ, NULL);
```

Für Variablen besteht zusätzlich die Möglichkeit, User-Objekte zu definieren, deren Parameter der Benutzer im Designer ändern kann. Die Objekte werden mit dem *LL_DRAWING_USEROBJ_DLG*-Typ definiert:

```
LlDefineVariableExt(hJob, <Name>, <Inhalt>, LL_DRAWING_USEROBJ_DLG, NULL);
```

Bearbeitbare User-Objekte können nicht in Tabellen eingefügt werden.

Wenn der Benutzer nun im Designer in den Objekt-Eigenschaften des Bildes die Eigenschaften der Variable bearbeitet, dann werden Sie über den *LL_EDIT_USEROBJ*-Callback aufgefordert, einen dazugehörigen Dialog aufzubauen, in dem die Parameter, die zu dem Objekt gehören, geändert werden können. Diese Parameter werden mit den anderen Objekt-Informationen automatisch in der Projektdefinitionsdatei gespeichert und zur Auswertung beim *LL_DRAW_USEROBJ*-Callback übergeben, so dass Ihr Programm sich nicht weiter um die Speicherung der Parameter kümmern muss.

Bitte beachten Sie die Hinweise des Artikels http://support.microsoft.com/kb/99109 in der Microsoft Developer Network Knowledge Base, wenn Sie in Callback-Objekten DDBs (device dependent bitmaps) drucken wollen: Wandeln Sie die DDB unbedingt erst in eine DIB (device independent bitmap), wenn sie nicht kompatibel mit dem Drucker-DC ist.

5.6.3 Definition einer Callback-Routine

Eine Callback-Routine wird wie ein normaler Windows-Callback definiert. Dafür benötigte spezielle Einstellungen, wie z.B. Compiler-Switches entnehmen Sie bitte der Dokumentation Ihres Compilers.

Die allgemeine Form des Callbacks ist in C-Notation

```
LRESULT CALLBACK _extern LLCallback(INT nMsg, LPARAM lParam,UINT_PTR lUserParam);
```

bzw. in Delphi-Notation:

```
function LLCallback(nMsg: integer; lParam: longint, lUserParam: longint) :
                longint; external;
```

Die Funktion kann direkt übergeben werden:

```
LlSetNotificationCallback(hJob,(FARPROC)LLCallback);
```

Ab jetzt kann Ihre Routine von List & Label aufgerufen werden, wenn dies nötig ist.

Wichtig ist, am Programmende den Callback wieder auf NULL zu setzen:

```
LlSetNotificationCallback(hJob,(FARPROC)NULL);
```

5.6.4 Datenübergabe an die Callback-Routine

Der Wert des *nMsg*-Parameters unterscheidet die verschiedenen Aufgaben. Die Werte sind die Konstanten, die mit *LL_CMND_xxxx* beginnen, z.B. *LL_CMND_TABLEFIELD* zum Zeichnen des Hintergrunds eines Tabellenfeldes, oder *LL_INFO_xxx* sowie *LL_NTFY_xxx*-Nachrichten.

Abhängig von der Aufgabe, die Ihr Programm zu erledigen hat, erhält der Parameter *lParam* unterschiedliche Bedeutungen. Die einzelnen Bedeutungen stehen weiter unten bei den Aufgaben beschrieben. Es sind meist Strukturen (Records), auf die *lParam* zeigt, der Wert muss also über eine Typkonvertierung in einen Strukturzeiger verwandelt werden:

```
LRESULT CALLBACK _extern LLCallback(INT wParam, LPARAM lParam,
    UINT_PTR lUserParam)
{
        PSCLLTABLEFIELD pSCF;

        switch (wParam)
        {
                case LL_CMND_TABLEFIELD:
                        pSCF = (PSCLLTABLEFIELD)lParam;
                        // do something using pSCF;
                        break;
        }
        return(0);
}
```

Die Funktion muss immer einen definierten Wert zurückgeben. Wenn nicht anders gefordert, muss dieser Wert Null sein.

lUserParam ist der über

```
llSetOption(hJob, LL_OPTION_CALLBACKPARAMETER, <Wert>);
```

übergebene Wert.

In objektorientierten Sprachen kann so ein Zeiger ("this", "self") übergeben werden.

5.6.5 Datenübergabe per Nachricht

Zu einer Nachricht gehören drei Parameter: nMsg, wParam und lParam in der folgenden Definition Ihres Nachrichten-Callbacks (nennt sich hier Fensterroutine, ist aber nichts anderes als ein Callback!)

```
LRESULT WINAPI MyWndProc(HWND hWnd, UINT nMsg, WPARAM wParam, LPARAM lParam);
```

Der Nachrichtenwert, den List & Label benutzt, kann über *LlGetNotificationMessage()* abgefragt werden. Alternativ könnte man über *LlSetNotificationMessage()* einen anderen aussuchen.

wParam ist hier unsere Aufgaben-Konstante und *lParam* zeigt auf eine Struktur des Typs *scLlCallback*:

```
struct scLlCallback
{
        int      _nSize;
        LPARAM   _lParam;
        LRESULT  _lResult;
        UINT_PTR _lUserParameter;
}
```

In dieser Struktur stecken nun die erforderlichen *_lParam* (als Parameterwert) und *_lResult* (als Rückgabewert).

```
nLLMessage = LlGetNotificationMessage(hJob);
//...
//...in the window procedure...
if (wMsg == nLLMessage)
{
        PSCLLCALLBACK    pSC;
        PSCLLTABLEFIELD pSCF;
        pSC = (PSCCALLBACK)lParam;
        switch (wParam)
        {
                case LL_CMND_TABLEFIELD:
                        pSCF = (PSCLLTABLEFIELD)pSC->_lParam;
                        // do something;
                        pSC._lResult = 0;
                        break;
        }
}
```

_lUserParam ist der über

```
LlSetOption(hJob, LL_OPTION_CALLBACKPARAMETER, <Wert>);
```

übergebene Wert.

In objektorientierten Sprachen kann so ein Zeiger ("this", "self") übergeben werden.

Wenn kein Rückgabewert gefordert wird, braucht das *_lResult*-Feld nicht verändert zu werden, es steht als Voreinstellung auf Null.

5.6.6 Weitere Hinweise

In diversen Callback Strukturen für Zeichnungsoperationen sind zwei Device Context Variablen enthalten. Beide sind identisch und lediglich aus Kompatibilitätsgründen vorhanden.

Wenn Sie irgendein GDI-Objekt in diesen DC selektieren oder andere Änderungen vornehmen, z.B. des Mapping-Modes, sollten Sie die Änderungen vor der Beendigung der

Routine wieder rückgängig machen (siehe auch die Windows API-Funktionen *SaveDC()* und *RestoreDC()*.

5.7 Fortgeschrittene Programmierung

5.7.1 Direkter Druck und Export aus dem Designer

Einführung

Sie haben die Möglichkeit, die Vorschau im Designer mit den "echten" Daten zu versorgen, so dass die Anwender den Report so sehen, wie er bei der Ausgabe aussehen wird. Zudem besteht die Möglichkeit, aus dem Designer heraus zu drucken oder zu exportieren.

Für C++ ist bereits ein vollständig implementiertes Beispiel in Quellcode-Form vorhanden. Sie finden es in der 'Beispiele' Programmgruppe für 'Visual C++' unter dem Namen 'Echtdatenpreview im Designer und Drilldown'.

Ihre Entwicklungsumgebung muss folgende Voraussetzungen erfüllen, damit dieses Feature unterstützt werden kann:

- Sie können auf Callbacks reagieren (s. Kapitel Callbacks und Notifications)
- Sie können einen Thread mit einer Druckprozedur starten und haben Synchronisationselemente wie Mutex, Critical Section oder ähnliches zur Verfügung.

Die von Ihrem Code durchzuführende Aufgabe besteht darin, Ihren Echtdatendruck bzw. -Export auszuführen, dies aber - zumindest für die Vorschau - in einem getrennten Thread. Hierfür gibt es über einen Callback Informationen über die Aufgabe (Start, Abbruch, Ende, Abfrage des Zustands). Dabei wird ein Zeiger auf eine *scLIDesignerPrintJob* Struktur übergeben, die alle für die jeweilige Aktion benötigten Daten enthält. Es sind nur geringe Änderungen gegenüber der normalen Druck/Exportausgabe nötig.

Vorbereitung

Um List & Label mitzuteilen, dass Sie die Echtdatenversorgung entsprechend der obigen Anforderungen durchführen können, müssen Sie eine oder beide der folgenden Optionen setzen:

- *LL_OPTION_DESIGNERPREVIEWPARAMETER* für die Echtdatenvorschau
- *LL_OPTION_DESIGNEREXPORTPARAMETER* für den Export aus dem Designer

Den Wert, den Sie in diesen Optionen übergeben, können Sie selbst definieren, beispielsweise als Zeiger auf interne Datenstrukturen oder Objekte. Sie bekommen diesen im Callback unverändert wieder geliefert (*scLIDesignerPrintJob._nUserParam*). Wichtig ist für List & Label nur, dass er nicht 0 oder -1 ist.

Über den Callback *LL_NTFY_DESIGNERPRINTJOB* informiert List & Label Sie über die durchzuführende Aktion. Dieser Callback wird immer im Kontext des Designer-Threads (dies ist der Thread, von dem aus Sie *LIDefineLayout()* aufgerufen haben) aufgerufen.

Wenn Sie Werte aus der Struktur, wie zum Beispiel *_nUserParam*, im Thread verwenden, sorgen Sie bitte dafür, dass der Thread sie ausgewertet oder kopiert hat, bevor Sie aus dem Event-Handler wieder die Kontrolle an List & Label übergeben, da danach die Struktur nicht mehr gültig ist - dies gilt für alle Callbacks!

Aufgaben

Nun zu den einzelnen Aufgaben, die Ihnen durch den Callback gestellt werden, sie werden durch unterschiedliche Werte von *scLIDesignerPrintJob._nFunction* unterschieden. Die symbolischen Konstanten für die möglichen Werte beginnen dabei alle mit LL_DESIGNERPRINTCALLBACK...:

Start-Event (..._PREVIEW_START/..._EXPORT_START)

Wenn Sie diesen Event erhalten, müssen Sie einen Thread erzeugen und die Start-Parameter an diesen übergeben.

Dieser Thread erzeugt einen neuen List & Label-Job, und führt in diesem neuen Job im Wesentlichen Ihre "ganz normale" Druckschleife aus. Abweichend zu Ihrer normalen Druckschleife müssen Sie lediglich noch folgende Änderungen durchführen:

- Setzen Sie vor dem Druckstart die Option *LL_OPTIONSTR_ORIGINALPROJECTFILENAME* auf den Pfad, der in der Struktur mitgeliefert wurde.
- Setzen Sie nach dem Druckstart über *LIPrintSetOption(hJob, LL_PRNOPT_LASTPAGE, _nPages)* die Anzahl der maximal zu druckenden Seiten, die in der Callback-Struktur übergeben wurde.
- Überprüfen Sie nach jedem *LIPrint()*, ob die Seitenzahl überschritten wurde, und rufen Sie in diesem Fall *LIPrintAbort()* auf. Diese Optimierung kann bzw. sollte auch für den normalen Druck benutzt werden. In diesem Fall sollten Sie aber nicht abbrechen, sondern den Druck regulär über *LIPrintFieldsEnd()* beenden.
- Signalisieren Sie über den in der Callback-Struktur mitgelieferten Event *_hEvent* an List & Label den Threadzustand, einmal zu Beginn und einmal am Ende des Threads. Wichtig ist, dass dies so synchronisiert ist, dass darauffolgende QUEST_JOBSTATE-Aufrufe (s.u.) den entsprechenden Zustand RUNNING/STOPPED korrekt melden. Da Sie den Event aus dem Thread heraus aufrufen, können Sie nicht den Thread-Zustand verwenden, sondern müssen eine entsprechende Variable verwenden. Dieser Prozesszustand muss für jeden Thread getrennt verwaltet werden.
- Löschen Sie die übergebene Projektdatei nach dem Druck

Zusätzlich sind noch folgende Punkte zu beachten:

Vorschau

- Übergeben Sie vor dem Aufruf von *LIPrint(WithBox)Start* das Fensterhandle, das über die Callback-Struktur übergeben wurde per *LIAssociatePreviewControl(hJob, hWnd, 1*) an List & Label, so dass der Druckjob darüber informiert wird, wo er die Daten darstellen soll.
- Nach dem Ende des Drucks, also nach *LIPrintEnd()* rufen Sie *LIAssociatePreviewControl(hJob,NULL,1)* auf, damit das Vorschau-Control die Kontrolle über die

Vorschaudatei erhält. Falls der Druck fehlgeschlagen ist, muss der letzte Parameter 0 sein, damit das Preview-Control leer ist und nicht das letzte Projekt anzeigt.

Export

- Benutzen Sie *LlPrintWithBoxStart()*, damit eine Fortschrittsbox dargestellt wird. Als Parent-Fensterhandle benutzen Sie hier ebenfalls das über die Callback-Struktur übergebene Fensterhandle.
- Wenn der _pszExportFormat-Member der Struktur gesetzt ist, hat der Anwender im Menü des Ribbons (ab Windows Vista) einen Direktexport gewählt. In diesem Falle sollte Ihr Code per *LlPrintSetOptionString(hJob, LL_PRNOPTSTR_EXPORT, _pszExportFormat)*; das gewünschte Format vorselektieren und keinen Druckoptionsdialog (*LlPrintOptionsDialog()*) anzeigen, wenn _ bWithoutDialog auf TRUE steht.

Abbruch-Event (..._PREVIEW_ABORT/..._EXPORT_ABORT)

Wenn Sie diesen Event erhalten, sollten Sie einfach *LlPrintAbort()* für den Druckjob des Preview-/Export-Threads aufrufen. Die Druckschleife im Thread sorgt dann für die korrekte Abarbeitung.

Finalisieren-Event (..._PREVIEW_FINALIZE/..._EXPORT_FINALIZE)

Wird auf jeden Fall aufgerufen, damit Sie interne Datenstrukturen freigeben können.

Statusabfrage-Event (..._PREVIEW_QUEST_JOBSTATE/..._EXPORT_QUEST_JOBSTATE)

Diese Aufgabe benutzt List & Label, um die Toolbar-Icons und die Menüeinträge aktuell zu halten. Geben Sie *LL_DESIGNERPRINTTHREAD_STATE_RUNNING* zurück, wenn Ihr Thread arbeitet, ansonsten liefern Sie *LL_DESIGNERPRINTTHREAD_STATE_STOPPED*.

Ablauf

Natürlich können Sie mehrere Start-Events erhalten. Vor jedem Start überprüft List & Label, ob schon ein Druck läuft, und stoppt diesen gegebenenfalls per Abbruch-Event.

Designer-Thread:	Druck-Thread:
Start-Event: • kopiert die Startparameter des Callbacks • startet den Druck-Thread und wartet auf das Signal, dass dieser bereit ist (Event).	
	startet: • setzt Prozesszustand intern auf RUNNING • signalisiert Zustandsänderung per *SetEvent(hEvent)* an List & Label • signalisiert Bereitschaft

kehrt an List & Label zurück	

Ab jetzt laufen Designer und Preview/Export parallel.

Üblicher Designer-Ablauf. **Abbruch** • ruft *LlPrintAbort()* für den Druckjob auf und kehrt zurück **Statusabfrage** • gibt den Wert des Prozesszustands zurück **Finalisieren** • ruft im Bedarfsfall *LlPrintAbort()* auf und wartet auf das Ende des Threads	• erzeugt neuen Job • startet Druckschleife mit oben erwähnten Änderungen Wenn Druck fertig: • setzt Prozesszustand intern auf STOPPED • signalisiert Zustandsänderung per *SetEvent(hEvent)* an List & Label • beendet Job • löscht Projektdatei

Am besten ist es, wenn Sie für jeden der beiden Ausgabetypen eine eigene Struktur haben, und die Adresse der Struktur über *LL_OPTION_DESIGNERPREVIEWPARAMETER* und *LL_OPTION_DESIGNEREXPORTPARAMETER* an List & Label übergeben. Diese Struktur enthält dann sinnvollerweise:

- einen eigenen Zeiger auf ein Objekt, das die Datenquelle verwaltet (wenn nötig bzw. möglich)
- ein Synchronisationsobjekt (CRITICAL_SECTION)
- das Thread-Handle des Arbeiter-Threads
- das Job-Handle des Arbeits-Threads
- Variablen als Kopie der Startparameter

Wenn Sie die Datenversorgung nur in einem Thread durchführen können weil z.B. die Datenquelle single-threaded ist, müssen Sie die Option *LL_OPTION_DESIGNERPRINT_SINGLETHREADED* auf *TRUE* setzen. Dies wird für List & Label dann dazu benutzt, dass während der Preview-Berechnung kein Export möglich ist und umgekehrt.

5.7.2 Drilldown-Berichte in der Vorschau

Drilldown Reporting bezeichnet die Navigation in hierarchischen Daten durch verschiedene Detaillevel hindurch.

Zunächst wird nur eine obere Ebene auf Vorschau ausgegeben (z.B. "Kunden") und durch einen Klick auf einen Kunden dann ein neuer Bericht (z.B. "Bestellungen") geöffnet, der im Normalfall Details zu dem Datensatz enthält, auf den man geklickt hat. So klickt man sich nach und nach herunter (daher die Bezeichnung) bis man z.B. bei den einzelnen Produkten gelandet ist. Der Vorteil liegt in der Performance, durch die schrittweise Spezialisierung findet man auch in komplexen und großen Datenbeständen schnell genau

die Informationen, die man sucht. Drilldown funktioniert in List & Label für die Vorschau, und für Tabellenzeilen oder -felder.

Voraussetzung für dieses Feature ist, dass Ihre Entwicklungsumgebung auf Callbacks oder Fenster-Nachrichten reagieren kann (siehe Kapitel "Callbacks und Notifications").

Die .NET und VCL-Komponenten unterstützen im datengebundenen Modus automatisch dieses Feature, sobald eine Datenquelle angebunden wird, die Hierarchien unterstützt und auch zurückgesetzt werden kann (die meisten DataProvider erfüllen diese Voraussetzungen). Wenn Sie die Komponente so verwenden, brauchen Sie den Inhalt dieses Kapitels nicht zu berücksichtigen.

Für C++ ist bereits ein vollständig implementiertes Beispiel in Quellcode-Form vorhanden. Sie finden es in der 'Beispiele' Programmgruppe für 'Visual C++' unter dem Namen 'Echtdatenpreview im Designer und Drilldown'.

Für eine verbesserte Benutzerakzeptanz empfiehlt es sich bei anderen Entwicklungsumgebungen das Drilldown durch den Einsatz von Threads zu implementieren, so dass die Arbeit dazu im Hintergrund stattfinden kann. Dies ist aber nicht zwingend notwendig. Sie müssen in diesem Fall einen Thread mit einer Druckprozedur starten können und benötigen Synchronisationselemente wie Mutex, Critical Section oder ähnliches.

Die von Ihrem Code durchzuführende Aufgabe besteht darin, Ihren Echtdatendruck mit einer passend gefilterten Datenquelle auszuführen. Hierfür gibt es über einen Callback Informationen über die Aufgabe (Start und Ende eines Drilldown-Berichts). Dabei wird ein Zeiger auf eine *scLlDrillDownJob* Struktur übergeben, die alle für die jeweilige Aktion benötigten Daten enthält. Es sind nur geringe Änderungen gegenüber dem normalen Druck notwendig.

Vorbereitungen

Um List & Label mitzuteilen, dass Sie Drilldown-Berichte entsprechend der obigen Anforderungen durchführen können, müssen Sie die Option *LL_OPTION_DRILLDOWNPARAMETER* auf einen Wert ungleich 0 setzen.

Beachten Sie aber, dass Sie diese Option für jeden Druckjob setzen müssen, der Drilldown ermöglichen soll:

```
// activate Drilldown for current LL-Job
::LlSetOption(hJob, LL_OPTION_DRILLDOWNPARAMETER,
                (LPARAM)&oMyDrillDownParameters);
```

Um Drilldown für den LL-Job wieder zu deaktivieren, müssen Sie diese Option einfach auf den Wert 'NULL' setzen:

```
// deactivate Drilldown for current LL-Job
::LlSetOption(hJob, LL_OPTION_DRILLDOWNPARAMETER, NULL);
```

Den Wert, den Sie in dieser Option übergeben, können Sie selbst definieren, beispielsweise als Zeiger auf interne Datenstrukturen oder Objekte. Sie bekommen diesen im

START- und FINALIZE-Callback unverändert wieder geliefert (*scLlDrillDown-Job._nUserParam)*, um ihn dort benutzen zu können. Wichtig ist für List & Label nur, dass er nicht 0 ist.

Über den Callback *LL_NTFY_VIEWERDRILLDOWN* (Beschreibung siehe Kapitel "Callbacks und Notifications") informiert List & Label Sie über die durchzuführende Aktion. Dieser Callback wird immer im Kontext des Vorschau-Threads aufgerufen, unabhängig davon, ob er aus der Echtdatenvorschau im Designer oder aus dem direkten Vorschaudruck heraus aufgerufen wurde.

Wenn Sie Werte aus der Struktur, wie zum Beispiel _nUserParam, im Thread verwenden, sorgen Sie bitte dafür, dass der Thread sie ausgewertet oder kopiert hat, bevor Sie aus dem Event-Handler wieder die Kontrolle an List & Label übergeben, da danach die Struktur nicht mehr gültig ist - dies gilt für alle Callbacks!

Aufgaben

Nun zu den einzelnen Aufgaben, die Ihnen durch den Callback gestellt werden, sie werden durch unterschiedliche Werte von *scLlDrillDownJob._nFunction* unterschieden:

Start-Event (LL_DRILLDOWN_START)

Wenn Sie einen START-Event erhalten, können Sie einen Thread erzeugen und die Start-Parameter an diesen übergeben. Wenn Ihre Entwicklungsumgebung keine Threads unterstützt, können Sie auch im Hauptthread einen Druck starten, dann ist allerdings die Oberfläche Ihres Programms während der Erstellung nicht bedienbar.

Der Rückgabewert des Callbacks (bzw. der *_lReply*-Member der *scLlCallback*-Strucktur bei Nachrichten) sollte auf eine von Ihnen vergebene eindeutige Zahl gesetzt werden, so dass Sie im FINALIZE-Event den Drilldown-Berichten dem richtigen Thread zuordnen können.

Beispiel:

```
…
LRESULT lResult = 0;
case LL_DRILLDOWN_START:
{
        scLlDrillDownJob* pDDJob = (scLlDrillDownJob*)lParam;
        … StartSubreport(pDDJob); …
        // generate new Drilldown-JobID
        lResult = ++m_nUniqueDrillDownJobID;
}
case LL_DRILLDOWN_FINALIZE:
{
        scLlDrillDownJob* pDDJob = (scLlDrillDownJob*)lParam;
        if (pDDJob->_nID == 0)
        {
                // clean up
        }
        else
        {
```

```
                // clean up the corresponding job
        }
}
…
return (lResult);
}
…
```

Dieser Thread erzeugt also nach dem Kopieren der Parameter einen neuen List & Label-Job, und führt in diesem neuen Job im Wesentlichen die "ganz normale" Druckschleife aus. Abweichend zu Ihrer normalen Druckschleife müssen Sie lediglich vor dem Aufruf von *LlPrintStart()* folgende Änderungen durchführen:

- Setzen Sie die Option *LL_OPTIONSTR_PREVIEWFILENAME* auf den Pfad, der in der Struktur mit *_pszPreviewFileName* mitgeliefert wurde.

  Beispiel:

  ```
  // set preview filename
  ::LlSetOptionString(pMyDrillDownParameters->m_hLlJob,
              LL_OPTIONSTR_PREVIEWFILENAME,
              pMyDrillDownParameters ->m_sPreviewFileName);
  ```

- Übergeben Sie die *_hAttachInfo*, die über die Callback-Struktur übergeben wurde an List & Label, so dass der Druckjob darüber informiert wird, wo er die Daten darstellen soll.

  Beispiel:

  ```
  // attach viewer
  ::LlAssociatePreviewControl(pMyDrillDownParameters->m_hLlJob,
              (HWND)pMyDrillDownParameters->_hAttachInfo,
              LL_ASSOCIATEPREVIEWCONTROLFLAG_DELETE_ON_CLOSE |
              LL_ASSOCIATEPREVIEWCONTROLFLAG_HANDLE_IS_ATTACHINFO);
  ```

Finalisieren-Event (LL_DRILLDOWN_FINALIZE)

Wird für Drilldown-Jobs aufgerufen, sofern diese abgebrochen werden, damit Sie interne Datenstrukturen freigeben können. Sie sollten in diesem Event einen ggf. noch laufenden Job abbrechen, indem Sie für diesen *LlPrintAbort()* aufrufen.

Wenn in der von List & Label übergebenen *scLlDrillDownJob*-Struktur der *_nID*-Member auf 0 gesetzt ist, können alle aktiven Drilldown-Jobs beendet und aufgeräumt werden. Dies geschieht beim Beenden der Vorschau.

Datenquelle aufbereiten

Um auch die richtigen Daten für den Drilldown-Bericht zur Verfügung zu stellen, müssen Sie kleinere Anpassungen an der Bereitstellung Ihrer Daten in der Druckschleife vornehmen.

Relation(en)

Es müssen entsprechende Relationen angemeldet sein. Für Drilldown-Berichte verwenden Sie dazu die API *LlDbAddTableRelationEx()*. Diese hat zwei zusätzliche Parameter – *pszKeyField* und *pszParentKeyField*. Diese stehen für das Schlüsselfeld der Child-Tabelle und das Schlüsselfeld der Eltern-Tabelle, damit eine eindeutige Zuordnung der Datensätze in der Child-Tabelle auf den Datensatz der Eltern-Tabelle erfolgen kann.

Weitere Informationen finden Sie bei der Beschreibung der API *LlDbAddTableRelationEx()*.

Beachten Sie, dass die Schlüsselfelder mit dem Tabellennamen identifiziert werden müssen; bspw. "Customers.CustomerID".

Beispiel:

Relation zwischen den beiden Tabellen 'Customers' und 'Orders' aus der mitgelieferten Northwind-Datenbank für Drilldown anmelden.

```
// add relation
…
CString sParentField = pMyDrillDownParameters->_pszSubreportTableID +
        _T(".")+pMyDrillDownParameters->_pszKeyField; //Orders.CustomerID
CString sChildField = pMyDrillDownParameters->_pszTableID + _T(".") +
        pMyDrillDownParameters->_pszSubreportKeyField; //Customers.OrderID
::LlDbAddTableRelationEx(hJob,
        pMyDrillDownParameters->_pszSubreportTableID,   // "Orders"
        pMyDrillDownParameters->_pszTableID,            // "Customers"
        pMyDrillDownParameters->_pszRelationID, _T(""),
        sParentField, sChildField);
…
```

Datenquelle

Für den jeweiligen Drilldown-Bericht müssen Sie Ihre Datenquelle auf eine andere Art aufbereiten. Sie fragen dann ja nur noch spezialisierte Daten ab, genau diejenigen, die mit dem Eltern-Datensatz verknüpft sind, auf den der Anwender geklickt hat.

Beispielsweise möchten Sie eine Drilldown-Struktur "Customers" zu "Orders" erstellen. Dann sollen Ihnen in der Eltern-Tabelle nur die Daten der "Customers" angezeigt werden. Beim Klick auf einen bestimmten "Customer" soll nun der Drilldown-Bericht erstellt werden, der dann die spezialisierten Daten für diesen "Customer" enthält; nämlich seine "Orders". Und hierfür müssen Sie die Datenquelle für den Drilldown-Bericht entsprechend anpassen – eben alle "Orders" von einem bestimmten "Customer". Alle notwendigen Daten für die Filterung der Child-Tabelle sind in der Drilldown-Struktur 'scLlDrillDownJob' enthalten.

5.7.3 Unterstützung des Berichtsparameter-Auswahlbereichs in der Vorschau

Der Berichtsparameter-Druck wird standardmäßig bereits automatisch unterstützt. Wenn Sie jedoch ein erneutes Rendern aus dem Vorschaufenster unterstützen möchten, müssen Sie List & Label diese Unterstützung signalisieren, indem Sie einige zusätzliche Optionen setzen. Der Einfachheit halber verwendet dieses Feature denselben Callback wie der Drilldown-Druck, so dass Sie in den meisten Fällen Ihren bestehenden Code weiterverwenden können. Sie müssen lediglich sicherstellen, dass der behandelnde Code auch dann funktioniert, wenn kein Drilldown-Filter in der übergebenen Struktur gesetzt wurde (z.B. sind alle Tabellen IDs und Schüsselfeldwerte leer).

Vorbereitungen

Um den Berichtsparameter-Druck in List & Label zu unterstützen, setzen Sie die Option *LL_OPTION_REPORT_PARAMETERS_REALDATAJOBPARAMETER* auf einen Wert ungleich 0.

Bitte beachten Sie dabei, dass Sie diese Option für jeden LL-Job setzen müssen, der den Berichtsparameter-Druck unterstützen soll:

```
// Berichtsparameter-Auswahlbereich für aktuellen LL-Job aktivieren
::LlSetOption(hJob, LL_OPTION_REPORT_PARAMETERS_REALDATAJOBPARAMETER,
                (LPARAM)&oMyReportParameters);
```

Um den Berichtsparameter-Druck für diesen LL-Job zu deaktivieren, setzen Sie die Option auf NULL:

```
// Berichtsparameter-Auswahlbereich für aktuellen LL-Job deaktivieren
::LlSetOption(hJob, LL_OPTION_REPORT_PARAMETERS_REALDATAJOBPARAMETER, NULL);
```

Der mit dieser Option übergebene Parameter kann frei verwendet werden, z.B. als Zeiger auf eine interne Datenstruktur oder Objekte. Dieser Parameter wird unverändert an den Callback für Ihre Verwendung übergeben (*scLlDrillDownJob._nUserParam)*. Bitte stellen Sie sicher, dass der Parameter nicht 0 oder NULL ist, es sei denn Sie möchten den Berichtsparameter-Druck deaktivieren.

Mit dem Callback *LL_NTFY_VIEWERDRILLDOWN* (siehe Kapitel Drilldown-Berichte in der Vorschau für weitere Informationen) benachrichtigt List & Label über die aktuelle Aufgabe. Dieser Callback wird immer im Kontext des Vorschau-Threads aufgerufen, ungeachtet, ob er vom Designer oder dem Vorschaudruck initiiert wurde.

Stellen Sie bei Verwendung von Struktur-Members wie z.B. _nUserParam, sicher, dass der Thread diese vorher evaluiert oder kopiert hat, bevor Sie die Kontrolle zurück an List & Label übergeben, da die Struktur nicht länger gültig wäre – dies gilt für alle Callbacks!

5.7.4 Unterstützung von ausklappbaren Bereichen in der Vorschau

Wenn dieses Feature unterstützt wird, können Elemente im Berichtscontainer dynamisch im Vorschaufenster ausgeklappt und zugeklappt werden. Der Einfachheit halber verwendet dieses Feature denselben Callback wie der Drilldown-Druck, so dass Sie in den meisten Fällen Ihren bestehenden Code weiterverwenden können. Sie müssen lediglich sicherstellen, dass der behandelnde Code auch dann funktioniert, wenn kein Drilldown-Filter in der übergebenen Struktur gesetzt wurde (z.B. sind alle Tabellen IDs und Schüsselfeldwerte leer).

Vorbereitungen

Um die ausklappbaren Bereiche in List & Label zu unterstützen, setzen Sie die Option *LL_OPTION_EXPANDABLE_REGIONS_REALDATAJOBPARAMETER* auf einen Wert ungleich 0.

Bitte beachten Sie dabei, dass Sie diese Option für jeden LL-Job setzen müssen, der ausklappbare Bereiche unterstützen soll:

```
// Ausklappbare Bereiche für aktuellen LL-Job aktivieren
::LlSetOption(hJob, LL_OPTION_EXPANDABLE_REGIONS_REALDATAJOBPARAMETER,
                (LPARAM)&oMyExpandableRegions);
```

Um die ausklappbaren Bereiche für diesen LL-Job zu deaktivieren, setzen Sie die Option auf NULL:

```
// Ausklappbare Bereiche für aktuellen LL-Job deaktivieren
::LlSetOption(hJob, LL_OPTION_EXPANDABLE_REGIONS_REALDATAJOBPARAMETER, NULL);
```

Der mit dieser Option übergebene Parameter kann frei verwendet werden, z.B. als Zeiger auf eine interne Datenstruktur oder Objekte. Dieser Parameter wird unverändert an den Callback für Ihre Verwendung übergeben (*scLIDrillDownJob._nUserParam)*. Bitte stellen Sie sicher, dass der Parameter nicht 0 oder NULL ist, es sei denn Sie möchten die ausklappbaren Bereiche deaktivieren.

Mit dem Callback *LL_NTFY_VIEWERDRILLDOWN* (siehe Kapitel Drilldown-Berichte in der Vorschau für weitere Informationen) benachrichtigt List & Label über die aktuelle Aufgabe. Dieser Callback wird immer im Kontext des Vorschau-Threads aufgerufen, ungeachtet, ob er vom Designer oder dem Vorschaudruck initiiert wurde.

Stellen Sie bei Verwendung von Struktur-Members wie z.B. _nUserParam, sicher, dass der Thread diese vorher evaluiert oder kopiert hat, bevor Sie die Kontrolle zurück an List & Label übergeben, da die Struktur nicht länger gültig wäre – dies gilt für alle Callbacks!

5.7.5 Unterstützung von interaktiver Sortierung in der Vorschau

Wenn dieses Feature unterstützt wird, können Felder der Berichtscontainer Tabellenkopfzeile dazu verwendet werden, zwischen verschiedenen Sortierungen im Vorschaufenster zu wechseln. Der Einfachheit halber verwendet dieses Feature denselben Call-

back wie der Drilldown-Druck, so dass Sie in den meisten Fällen Ihren bestehenden Code weiterverwenden können. Sie müssen lediglich sicherstellen, dass der behandelnde Code auch dann funktioniert, wenn kein Drilldown-Filter in der übergebenen Struktur gesetzt wurde (z.B. sind alle Tabellen IDs und Schüsselfeldwerte leer).

Vorbereitungen

Um die interaktive Sortierung in List & Label zu unterstützen, setzen Sie die Option *LL_OPTION_INTERACTIVESORTING_REALDATAJOBPARAMETER* auf einen Wert ungleich 0.

Bitte beachten Sie dabei, dass Sie diese Option für jeden LL-Job setzen müssen, der die interaktive Sortierung unterstützen soll:

```
// Interaktive Sortierung für aktuellen LL-Job aktivieren
::LlSetOption(hJob, LL_OPTION_INTERACTIVESORTING_REALDATAJOBPARAMETER,
                (LPARAM)&oMyInteractiveSortings);
```

Um die interaktive Sortierung für diesen LL-Job zu deaktivieren, setzen Sie die Option auf NULL:

```
// Interaktive Sortierung für aktuellen LL-Job deaktivieren
::LlSetOption(hJob, LL_OPTION_INTERACTIVESORTING_REALDATAJOBPARAMETER, NULL);
```

Der mit dieser Option übergebene Parameter kann frei verwendet werden, z.B. als Zeiger auf eine interne Datenstruktur oder Objekte. Dieser Parameter wird unverändert an den Callback für Ihre Verwendung übergeben (*scLIDrillDownJob._nUserParam)*. Bitte stellen Sie sicher, dass der Parameter nicht 0 oder NULL ist, es sei denn Sie möchten die ausklappbaren Bereiche deaktivieren.

Mit dem Callback *LL_NTFY_VIEWERDRILLDOWN* (siehe Kapitel Drilldown-Berichte in der Vorschau für weitere Informationen) benachrichtigt List & Label über die aktuelle Aufgabe. Dieser Callback wird immer im Kontext des Vorschau-Threads aufgerufen, ungeachtet, ob er vom Designer oder dem Vorschaudruck initiiert wurde.

Stellen Sie bei Verwendung von Struktur-Members wie z.B. _nUserParam, sicher, dass der Thread diese vorher evaluiert oder kopiert hat, bevor Sie die Kontrolle zurück an List & Label übergeben, da die Struktur nicht länger gültig wäre – dies gilt für alle Callbacks!

5.7.6 Ansteuerung von Chart- und Kreuztabellen-Objekten

Am einfachsten können Chart- und Kreuztabellenobjekte angesteuert werden, indem Sie – zusammen mit Tabellen – im Berichtscontainer eingefügt werden. Die Ansteuerung solcher Berichte wird im Kapitel "Drucken relationaler Daten" detailliert erläutert. Für Etiketten- oder Karteikartenprojekte kann aber auch eine separate Ansteuerung von solchen Objekten interessant sein.

Im Folgenden ist nur von "Charts" die Rede, die Ansteuerung funktioniert für Kreuztabellen analog.

Für die Datenversorgung von Chart-Objekten gibt es neben der oben genannten zwei weitere Methoden. Welche von beiden Sie verwenden möchten, stellen Sie über die Option *LL_OPTION_USECHARTFIELDS* ein. Grundsätzlich funktioniert der Druck mit Charts dann analog zu dem Tabellendruck, d.h. Sie definieren zunächst eine Datenreihe, die Sie an die Chartobjekte übergeben möchten und übergeben diese dann mit einem Aufruf an die Chart-Objekte.

Der Standardmodus (Voreinstellung)

Dieser Modus steht Ihnen nur für Listenprojekte zur Verfügung und benötigt keine Änderungen an Ihren bestehenden Projekten. Die Chart-Objekte werden hierbei mit den gleichen Daten wie die Tabellenobjekte versorgt, ein *LlPrintFields()* übergibt die Daten sowohl an Tabellen- als auch an Chartobjekte, wobei diese die Werte zunächst nur aufnehmen und – im Gegensatz zu Tabellen – nicht gleich drucken. Dieser Modus ist aus Kompatibilitätsgründen zu älteren Versionen noch enthalten und ermöglicht eine einfache Verwendung von Chartobjekten z.B. hinter Tabellen verkettet.

Der erweiterte Modus

Diesen Modus aktivieren Sie durch Setzen der Option *LL_OPTION_USECHARTFIELDS* auf TRUE. In diesem Falle stehen Ihnen neben den Variablen und Feldern noch spezielle Chart-Felder zur Verfügung. Diese können Sie analog zu Feldern über API-Aufrufe deklarieren. Ihre Benutzer können damit

- Chart-Objekte an beliebigen Stellen im Ausdruck verwenden
- Chart-Objekte auf Etiketten/Karteikarten verwenden

Ein *LlPrintFields()* hat in diesem Modus keinen Einfluss auf die Chart-Objekte, der analoge Befehl hierfür ist im erweiterten Modus *LlPrintDeclareChartRow().* Durch diesen API-Aufruf werden die gegenwärtig definierten Daten an die Chart-Objekte übergeben. Welche Chart-Objekte dabei angesprochen werden sollen, kann durch den Parameter von *LlPrintDeclareChartRow()* bestimmt werden:

Wert	Bedeutung
LL_DECLARECHARTROW_-FOR_OBJECTS	Die Daten werden an alle Chart-Objekte übergeben, die nicht in Tabellenspalten enthalten sind.
LL_DECLARECHARTROW_-FOR_TABLECOLUMNS	Die Daten werden an alle Chart-Objekte übergeben, die sich in Tabellenspalten befinden (aus Kompatibilitätsgründen weiter unterstützt).

Für Charts in einem Etikettenprojekt würden Sie in Pseudocode wie folgt vorgehen:

```
<starte Ausdruck>
        (LlPrintStart,
         LlPrintWithBoxStart)
<solange
        - kein Fehler und nicht fertig>
{
```

```
        <definiere Variablen>
        <solange
                - kein Fehler
                - nicht fertig> (z.B. i = 1..12)
        {
                <Definiere Chartfelder (z.B. Monat = Monatsname[i])>
                <sende Chart-Felder an Chartobjekte>
                        (LlPrintDeclareChartRow(LL_DECLARECHARTROW_FOR_OBJECTS))
        }
        <drucke Objekte>
                (LlPrint)
        <keine Warnung, kein Abbruch: nächster Datensatz>
}

<beende Ausdruck>
        (LlPrintEnd)
```

Wie üblich müssen alle verwendeten Chart-Felder auch vor dem Designeraufruf angemeldet werden, damit Sie dem Benutzer überhaupt zur Verfügung stehen.

5.8 Verwendung der DOM-API (ab Professional Edition)

Um Projektdateien dynamisch zur Laufzeit zu erstellen oder auch um bestehende Projektdateien per Code zu bearbeiten, können Sie ab der Professional Edition mit den List & Label DOM-Funktionen arbeiten.

Sofern Sie mit der .NET oder VLC Komponente arbeiten, stellt diese Ihnen ein komfortables und typsicheres Objektmodell für den DOM Zugriff zur Verfügung. In diesem Fall können Sie dieses Kapitel überspringen und sich anstatt dessen für einen schnellen Einstieg eines der mitgelieferten DOM Beispiele ansehen.

5.8.1 Grundlagen

Jedes "Objekt" innerhalb einer Projektdatei hat ein eigenes Handle ("DOM-Handle"). Die Funktionen des DOM-API verwenden dieses Handle, um Objekte eindeutig zu identifizieren. "Objekte" in diesem Sinne sind hierbei alle Designerobjekte, aber auch andere Elemente wie Hilfslinien, Projektparameter etc.. Der im Lieferumfang befindliche DOM-Viewer erlaubt einen schnellen Überblick über alle Objekte, deren Wert und weitere Eigenschaften. Zusätzlich dazu können mit dem Viewer Eigenschaften bzw. Werte geändert und im Projekt gespeichert werden. Über die Clipboard Funktion kann jedes Objekt bzw. Eigenschaft zur weiteren Verwendung in die Zwischenablage kopiert werden.

Die für das DOM-API relevanten Funktionen unterteilen sich in 2 Gruppen: zunächst können Projektdateien geladen, erzeugt und gespeichert werden. Dafür stehen die Funktionen *LlProjectOpen(), LlProjectClose()* und *LlProjectSave()* zur Verfügung. Dabei liefert die Funktion *LlDomGetProject()* (aufzurufen nach *LlProjectOpen()*) das DOM-Handle für das Projektobjekt zurück. Dieses liefert dann die Basis für die Verwendung der weiteren Funktionen.

DOM-Funktionen

LlDomGetObject

Mit dieser Funktion können vom Projekt-Objekt wichtige Unterobjekte erhalten werden. Um z.B. die Objektliste zu erhalten, kann

```
LlProjectOpen(hJob, LL_PROJECT_LIST, "c:\\filename.lst", LL_PRJOPEN_AM_READONLY);
HLLDOMOBJ hProj;
LlDomGetProject(hJob, &hProj);
HLLDOMOBJ hObjList;
INT nRet = LlDomGetObject(hProj, "Objects", &hObjList);
```

verwendet werden. Die weiteren verfügbaren Objekte entsprechen den Einträgen innerhalb der Baumstruktur im DOM-Viewer: "Layout", "ProjectParameters", "Settings", "SumVars" und "UserVars". Eine Beschreibung der einzelnen Objekte mit allen Eigenschaften finden Sie im Referenzkapitel, hier soll das Prinzip der Arbeit mit den DOM-Funktionen im Vordergrund stehen.

LlDomGetSubobjectCount

Dient dazu, die Anzahl der Unterobjekte in der angegebenen Liste abzufragen. Um etwa die Anzahl der Objekte im Projekt zu erfragen, verwendet man

```
INT nObjCount;
INT nRet = LlDomGetSubobjectCount(hObjList, &nObjCount);
```

LlDomGetSubobject

Liefert das DOM-Handle des angegebenen Unterobjektes zurück. Parameter sind neben dem DOM-Handle der Liste der Index (0-basiert) und einen Zeiger für die Rückgabe des Handles. Der Code für ein DOM-Handle auf das erste Objekt in der Projektdatei lautet

```
HLLDOMOBJ hObj;
INT nRet = LlDomGetSubobject(hObjList, 0, &hObj);
```

LlDomCreateSubobject

Erzeugt ein neues Unterobjekt in der angegebenen Liste. Parameter sind das Listenhandle, die Einfügeposition, der gewünschte Typ sowie einen Handle-Zeiger für das neue Objekt. Um ein neues Textobjekt am Anfang der Objektliste einzufügen, verwendet man

```
HLLDOMOBJ hObj;
INT nRet = LlDomCreateSubobject(hObjList, 0, _T("Text"), &hObj);
```

Innerhalb der Objektliste z.B. können Sie mit Hilfe dieser Funktion die folgenden Objekte erzeugen:

Objekttyp:	Benötigter dritter Parameter:
Linie	"Line"
Rechteck	"Rectangle"
Ellipse	"Ellipse"
Zeichnung	"Drawing"
Text	"Text"
Formularvorlage	"Template"
Barcode	"Barcode"
RTF	"RTFText"
HTML	"LLX:LLHTMLObject"
Berichtscontainer (kann Tabellen, Charts und Kreuztabellen enthalten)	"ReportContainer"
Messinstrument	"Gauge"
PDF	"PDF"

Die weiteren möglichen Werte für andere Listen (z.B. Feldliste innerhalb einer Tabelle) finden Sie in der Onlinehilfe der DOM-Viewer Anwendung.

LlDomDeleteSubobject

Löscht das angegebene Unterobjekt. Um z.B. das erste Objekt der Objektliste zu löschen benutzt man

```
INT nRet = LlDomDeleteSubobject(hObjList, 0);
```

LlDomSetProperty

Erlaubt das Setzen einer Eigenschaft für das angegebene Objekt. Um z.B. den Seitenumbruch für ein Textobjekt zu erlauben, benötigt man

```
INT nRet = LlDomSetProperty(hObj, _T("AllowPageWrap"), _T("True"));
```

WICHTIG: Der Übergabeparameter für den Wert muss eine gültige List & Label-Formel sein. Eine Besonderheit ergibt sich hierdurch für Eigenschaften, die Zeichenketten enthalten (z.B. der Inhalt eines Text-Absatzes): Zeichenketten müssen ja innerhalb des

Designers ihrerseits in Anführungszeichen gesetzt werden, um als gültige Formel verwendbar zu sein. Um also den festen Text "combit" zu übergeben muss der Parameter "'combit'" verwendet werden. Dies gilt auch z.B. für feste Fontnamen, auch hier muss z.B. "'Verdana'" übergeben werden, nicht "Verdana",

Beispiel:

```
LlDomSetProperty(hObj, _T("Contents"), _T("'") + sProjectTitle + _T("'"));
```

Um die Werte von verschachtelten Eigenschaften wie den der Farbe einer Füllung zu setzen, kann der Eigenschaftsname "<Elterneigenschaft>.<Kindeigenschaft>" verwendet werden, also z.B.

```
INT nRet = LlDomSetProperty(hObj, _T("Filling.Color"), _T("LL.Color.Black"));
```

LlDomGetProperty

Liest den Wert einer Eigenschaft aus. Es empfiehlt sich, wie üblich zunächst durch Übergabe eines NULL-Puffers die benötigte Pufferlänge zu ermitteln und dann einen ausreichend großen Puffer zu allozieren:

```
INT nBufSize = LlDomGetProperty(hObj, _T("AllowPageWrap"), NULL, 0);
TCHAR* pszBuffer = new TCHAR[nBufSize];
INT nRet = LlDomGetProperty(hObj, _T("AllowPageWrap"), pszBuffer, nBufSize);
…
delete[] pszBuffer;
```

Zur Vereinfachung können Objekte (nicht aber Listen!) mit Hilfe des Punktes als Hierarchietrenner auch "durchtunnelt" werden, wie z.B.:

```
...
// Auslesen der Seitenkoordinaten der ersten Seite
LlDomGetProperty(hRegion,
        _T("Paper.Extent.Horizontal"),
        pszContainerPositionWidth, nBufSize);
```

Einheiten

Viele Eigenschaften enthalten Informationen über Größen, Breiten etc. Diese werden – wenn Sie als feste Zahl übergeben werden – als SCM-Einheiten (1/1000 mm) interpretiert und zurückgeliefert und sind somit vom gewählten Einheitensystem unabhängig. Um ein Objekt an einer (festen) Position 5 mm vom linken Rand entfernt zu platzieren, würde man

```
INT nRet = LlDomSetProperty(hObj, _T("Position.Left"), _T("5000"));
```

verwenden. Wenn die Eigenschaft allerdings keinen festen Wert, sondern eine Formel enthalten soll, muss die Funktion *UnitFromSCM* verwendet werden, um unabhängig von den Einheiten zu sein. Einen Bundsteg mit einem Einzug von 10 mm auf ungeraden und 5 mm auf geraden Seiten würde man über

```
INT nRet = LlDomSetProperty(hObj, _T("Position.Left"),
                _T("Cond(Odd(Page()), UnitFromSCM(10000), UnitFromSCM(5000))"));
```

realisieren.

5.8.2 Beispiele

Textobjekt anlegen

Der folgende Code erzeugt ein neues Projekt, fügt ein Textobjekt und darin einen neuen Absatz mit dem Inhalt "DOM" ein und speichert das Projekt:

```
HLLJOB hJob = LlJobOpen(-1);

// Neues Projekt erzeugen
LlProjectOpen(hJob,LL_PROJECT_LIST,"c:\\simple.lst",
  LL_PRJOPEN_CD_CREATE_ALWAYS | LL_PRJOPEN_AM_READWRITE);

HLLDOMOBJ hProj;
LlDomGetProject(hJob, &hProj);

// Objektliste holen
HLLDOMOBJ hObjList;
LlDomGetObject(hProj, "Objects", &hObjList);

// Textobjekt erzeugen
HLLDOMOBJ hObj;
LlDomCreateSubobject(hObjList, 0, _T("Text"), &hObj);
LlDomSetProperty(hObj, _T("Name"), _T("My new Textobject"));

// Absatzliste holen
HLLDOMOBJ hObjParagraphList;
LlDomGetObject(hObj, _T("Paragraphs"), &hObjParagraphList);

// Neuen Absatz erzeugen und Inhalt anlagen
HLLDOMOBJ hObjParagraph;
LlDomCreateSubobject(hObjParagraphList, 0, _T("Paragraph"), &hObjParagraph);
LlDomSetProperty(hObjParagraph, _T("Contents"), _T("'DOM'"));

// Projekt speichern
LlProjectSave(hJob, NULL);
LlProjectClose(hJob);

LlJobClose(hJob);
```

Tabelle anlegen

Dieses Beispiel zeigt die Erzeugung eines Tabellenobjektes innerhalb eines Berichtscontainers und legt darin eine neue Datenzeile und drei Spalten an.

Beachten Sie, dass Sie auch dann, wenn Sie nicht die APIs zur Ansteuerung des Berichtscontainers (*LlDbAddTable()* etc.) verwenden einen Berichtscontainer mit genau einer Tabelle anlegen müssen.

```
HLLJOB hJob = LlJobOpen(-1);

// Neues Projekt erzeugen
LlProjectOpen(hJob, LL_PROJECT_LIST, "c:\\simple.lst",
        LL_PRJOPEN_CD_CREATE_ALWAYS | LL_PRJOPEN_AM_READWRITE);

HLLDOMOBJ hProj;
LlDomGetProject(hJob, &hProj);

// Objektliste holen
HLLDOMOBJ hObjList;
LlDomGetObject(hProj, "Objects", &hObjList);

// Berichtscontainer erzeugen und Eigenschaften setzen
HLLDOMOBJ hObjReportContainer;
LlDomCreateSubobject(hObjList, 0, _T("ReportContainer"),&hObjReportContainer);
LlDomSetProperty(hObjReportContainer,_T("Position.Left"), _T("27000"));
LlDomSetProperty(hObjReportContainer,_T("Position.Top"),  _T("103500"));
LlDomSetProperty(hObjReportContainer,_T("Position.Width"), _T("153400"));
LlDomSetProperty(hObjReportContainer,_T("Position.Height"), _T("159500"));

// Unterobjektliste holen und Tabelle darin anlegen
HLLDOMOBJ hObjSubItems;
LlDomGetObject(hObjReportContainer, _T("SubItems"), & hObjSubItems);
HLLDOMOBJ hObjTable;
LlDomCreateSubobject(hObjSubItems, 0, _T("Table"), &hObjTable);

// Zeilenliste holen
HLLDOMOBJ hObjTableLines;
LlDomGetObject(hObjTable , _T("Lines"), &hObjTableLines);

// Datenzeilenliste holen
HLLDOMOBJ hObjTableData;
LlDomGetObject(hObjTableLines , _T("Data"), &hObjTableData);

// Neue Zeilendefinition anlegen
HLLDOMOBJ hObjTableLine;
LlDomCreateSubobject(hObjTableData, 0, _T("Line"), &hObjTableLine);
LlDomSetProperty(hObjTableLine,_T("Name"), _T("My new table line"));

// Kopfzeilenliste holen
HLLDOMOBJ hObjTableHeader;
LlDomGetObject(hObjTableLines , _T("Header"), &hObjTableHeader);

// Neue Zeilendefinition anlegen
HLLDOMOBJ hObjTableHeaderLine;
```

```
LlDomCreateSubobject(hObjTableHeader, 0, _T("Line"), &hObjTableHeaderLine);

// Feldliste für Kopfzeilen holen
HLLDOMOBJ hObjTableHeaderFields;
LlDomGetObject(hObjTableHeaderLine , _T("Fields"), &hObjTableHeaderFields);

// Feldliste für Datenzeilen holen
HLLDOMOBJ hObjTableDataFields;
LlDomGetObject(hObjTableLine , _T("Fields"), &hObjTableDataFields);

TCHAR aczVarName[1024];
int nItemCount = 3;

// Tabelle mit nItemCount Spalten anlegen
for (int i=0; i < nItemCount; i++)
{
        // Spalteninhalt für die Kopfzeile bestimmen. Beachten Sie
        // die Hochkommata – dadurch wird fester Text (z.B. "Field1") als
        // Inhalt übergeben
        _stprintf(aczVarName, "'Field%d'", i);

        // Neues Feld in Kopfzeile anlegen und Eigenschaften setzen
        HLLDOMOBJ hObjHeaderField;
        LlDomCreateSubobject(hObjTableHeaderFields, 0, _T("Text"),
                &hObjHeaderField);
        LlDomSetProperty(hObjHeaderField, _T("Contents"), aczVarName);
        LlDomSetProperty(hObjHeaderField,_T("Filling.Style"), _T("1"));
        LlDomSetProperty(hObjHeaderField,_T("Filling.Color"),
                _T( "RGB(204,204,255)"));
        LlDomSetProperty(hObjHeaderField,_T("Font.Bold"), _T("True"));
        LlDomSetProperty(hObjHeaderField,_T("Width"), _T("50000"));

        // Spalteninhalt für die Datenzeile bestimmen. Jetzt ohne
        // die Hochkommata – dadurch wird der Feldinhalt (z.B. Field1) als
        // Inhalt übergeben
        _stprintf(aczVarName, "Field%d", i);

        // Neues Feld in Datenzeile anlegen und Eigenschaften setzen
        HLLDOMOBJ hObjDataField;
        LlDomCreateSubobject(hObjTableDataFields, 0, _T("Text"),
                &hObjDataField);

        LlDomSetProperty(hObjDataField,_T("Contents"), aczVarName);
        LlDomSetProperty(hObjDataField,_T("Width"), _T("50000"));
}

// Projekt speichern
LlProjectSave(hJob, NULL);
LlProjectClose(hJob);
LlJobClose(hJob);
```

Projektparameter setzen

Der nachfolgende Code setzt Projektparameter in ein bestehendes List & Label Projekt für den Fax- und Mailversand:

```
HLLJOB hJob = LlJobOpen(-1);

LlProjectOpen(hJob, LL_PROJECT_LIST, "c:\\simple.lst",
                LL_PRJOPEN_CD_OPEN_EXISTING | LL_PRJOPEN_AM_READWRITE);

HLLDOMOBJ hProj;
LlDomGetProject(hJob, &hProj);

// Fax parameter:
LlDomSetProperty(hProj, _T("ProjectParameters.LL.FAX.RecipName.Contents"),
                _T("'sunshine agency'"));
LlDomSetProperty(hProj, _T("ProjectParameters.LL.FAX.RecipNumber.Contents"),
                _T("'555-555 555'"));
LlDomSetProperty(hProj,_T("ProjectParameters.LL.FAX.SenderCompany.Contents"),
                _T("'combit'"));
LlDomSetProperty(hProj,_T("ProjectParameters.LL.FAX.SenderName.Contents"),
                _T("'Mustermann'"));

// Mail parameter:
LlDomSetProperty(hProj, _T("ProjectParameters.LL.MAIL.Subject.Contents"),
                _T("'Your request'"));
LlDomSetProperty(hProj, _T("ProjectParameters.LL.MAIL.From.Contents"),
                _T("'info@combit.net'"));
LlDomSetProperty(hProj, _T("ProjectParameters.LL.MAIL.To.Contents"),
                _T("'info@sunshine-agency.net'"));

// Projekt speichern
LlProjectSave(hJob, NULL);
LlProjectClose(hJob);
LlJobClose(hJob);
```

6. API Referenz

6.1 Referenz der Funktionen

LlAddCtlSupport

Syntax:

```
INT LlAddCtlSupport(HWND hWnd, UINT nFlags, LPCTSTR lpszInifile);
```

Aufgabe:

Übertragen von combit-Dialogstilen auf andere Applikationen.

Parameter:

hWnd: Window-Handle des Programms

nFlags:

Wert	Bedeutung
LL_CTL_ADDTOSYSMENU	Erweiterung des Systemmenüs des Programms um eine Dialogstilartauswahl
LL_CTL_ALSOCHILDREN	Automatische Übertragung der Dialogstile auf die Child-Fenster
LL_CTL_CONVERTCONTROLS	Damit auch bei Thunderframe Controls (VB) korrekt gezeichnet werden.

Bei Bedarf jeweils mit 'oder' verknüpfbar.

lpszInifile: Name eines Registry-Schlüssels. Empfehlung: nicht verwenden, sondern leer lassen für Voreinstellung.

Rückgabewert:

Fehlercode

Hinweise:

Ermöglicht es, auch z.B. die Serverapplikation mit anderen Dialogstilarten zu versorgen.

Beispiel:

```
LlAddCtlSupport(hWMain, LL_CTL_ALSOCHILDREN | LL_CTL_CONVERTCONTROLS,
"");
```

LlAssociatePreviewControl

Syntax:

```
INT LlAssociatePreviewControl(HLLJOB hJob, HWND hWndControl,
   UINT nFlags);
```

Aufgabe:

Ordnet einen LL-Job einem Vorschaufenster zu.

Parameter:

hJob: List & Label Job-Handle

hWnd: Fensterhandle

nFlags:

Wert	Bedeutung
LL_ASSOCIATEPREVIEW-CONTROLFLAG_DELETE_-ON_CLOSE	Preview-Datei nach Beenden der Vorschau automatisch löschen
*LL__*ASSOCIATEPREVIEW-CONTROLFLAG_HANDLE_-IS_ATTACHINFO	Informiert die API, dass das übergebene Fensterhandle ein Zeiger auf eine Struktur mit Drilldown-Informationen ist
LL_ASSOCIATEPREVIEW-CONTROLFLAG_PRV_-REPLACE	Wenn LL_ASSOCIATEPREVIEWCONTROLFLAG_-HANDLE_IS_ATTACHINFO nicht gesetzt ist: gibt an, dass diese Vorschau die momentane Vorschau ersetzt
LL_ASSOCIATEPREVIEW-CONTROLFLAG_PRV_-ADD_TO_CONTROL_STACK	Wenn LL_ASSOCIATEPREVIEWCONTROLFLAG_-HANDLE_IS_ATTACHINFO nicht gesetzt ist: gibt an, dass diese Vorschau im aktuellen Vorschautab hinzugefügt wird
LL_ASSOCIATEPREVIEW-CONTROLFLAG_PRV_-ADD_TO_CONTROL_IN_-TAB	Wenn LL_ASSOCIATEPREVIEWCONTROLFLAG_-HANDLE_IS_ATTACHINFO nicht gesetzt ist: gibt an, dass diese Vorschau in einem neuen Tab angelegt wird

Bei Bedarf jeweils mit 'oder' verknüpfbar.

Rückgabewert:

Fehlercode

Hinweise:

Siehe Kapitel Direkter Druck und Export aus dem Designer

Beispiel:

Siehe Kapitel Direkter Druck und Export aus dem Designer

LlCreateSketch

Syntax:

```
INT LlCreateSketch(HLLJOB hJob, UINT nObjType, LPCTSTR lpszObjName);
```

Aufgabe:

Erzeugt die Skizzendatei für ein Projekt, die in dem Dateiauswahl-Dialog angezeigt werden kann.

Parameter:

hJob: List & Label Job-Handle

nObjType:

Wert	Bedeutung
LL_PROJECT_LABEL	für Etiketten
LL_PROJECT_CARD	für Karteikarten
LL_PROJECT_LIST	für Listen

lpszObjName: Projektdateiname mit Pfadangabe

Hinweise:

Dieser Funktion kann z.B. genutzt werden, um Skizzen-Dateien automatisiert, z.B. im Rahmen eines Setup-Programms, zu erstellen.

Rückgabewert:

Fehlercode

Siehe auch:

LlSelectFileDlgTitleEx

LlDbAddTable

Syntax:

```
INT LlDbAddTable(HLLJOB hJob, LPCTSTR pszTableID,
   LPCTSTR pszDisplayName);
```

Aufgabe:

Meldet eine Tabelle oder ein Datenbankschema für das Design und den Druck an. Die Tabelle steht dem Benutzer dann im Designer zur Verfügung und kann zur Druckzeit von List & Label angefordert werden.

Parameter:

hJob: List & Label Job-Handle

pszTableID: ID der Tabelle. Diese wird bei *LlPrintDbGetCurrentTable()* zurückgeliefert, wenn die Tabelle gedruckt werden soll. Wenn Sie einen Leerstring oder NULL übergeben, wird der Tabellenpuffer gelöscht.

pszDisplayName: Name der Tabelle wie er im Designer angezeigt werden soll. Wird nur zur Anzeige verwendet und nicht im Projektfile gespeichert. Kann auch NULL sein, in diesem Falle wird der zweite Parameter auch für die Anzeige im Designer verwendet.

Rückgabewert:

Fehlercode

Hinweise:

Wenn ein Tabellenname mit "." angegeben wird, wird daraus ein Schema erzeugt.

Beachten Sie die Hinweise im Kapitel "Drucken relationaler Daten".

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

LlDbAddTable(hJob, "", NULL);
LlDbAddTable(hJob, "Orders", NULL);
LlDbAddTable(hJob, "OrderDetails", NULL);
LlDbAddTable(hJob, " HumanResources.Employee", NULL); // scheme info
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LlDbAddTableSortOrder, LlDbAddTableRelation, LlPrintDbGetCurrentTable, LlPrintDbGetCurrentTableSortOrder, LlPrintDbGetCurrentTableRelation

LlDbAddTableEx

Syntax:

```
INT LlDbAddTableEx(HLLJOB hJob, LPCTSTR pszTableID,
   LPCTSTR pszDisplayName, UINT nOptions);
```

Aufgabe:

Meldet eine Tabelle oder ein Datenbankschema für das Design und den Druck an und erlaubt die Übergabe weiterer Optionen. Die Tabelle steht dem Benutzer dann im Designer zur Verfügung und kann zur Druckzeit von List & Label angefordert werden.

Parameter:

hJob: List & Label Job-Handle

pszTableID: ID der Tabelle. Diese wird bei *LlPrintDbGetCurrentTable()* zurückgeliefert, wenn die Tabelle gedruckt werden soll. Wenn Sie einen Leerstring oder NULL übergeben, wird der Tabellenpuffer gelöscht.

pszDisplayName: Name der Tabelle wie er im Designer angezeigt werden soll. Wird nur zur Anzeige verwendet und nicht im Projektfile gespeichert. Kann auch NULL sein, in diesem Falle wird der zweite Parameter auch für die Anzeige im Designer verwendet.

nOptions: Eine Kombination der folgenden Flags:

Wert	Bedeutung
LL_ADDTABLEOPT_SUPPORTSSTACKEDSORTORDERS	Im Designer werden für diese Tabelle mehrfache Sortierungen unterstützt. *LlDbGetCurrentTableSortOrder()* liefert in diesem Falle eine tabulatorgetrennte Liste von Sortierungen zurück.
LL_ADDTABLEOPT_SUPPORTSADVANCEDFILTERING	Unterstützung für die Übersetzung von Filterausdrücken in native Syntax. Siehe die Dokumentation des *LL_QUERY_EXPR2HOSTEXPRESSION* Callbacks und *LlPrintDbGetCurrentTableFilter().*

Rückgabewert:

Fehlercode

Hinweise:

Wenn ein Tabellenname mit "." angegeben wird, wird daraus ein Schema erzeugt.

Beachten Sie die Hinweise im Kapitel "Drucken relationaler Daten".

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

LlDbAddTable(hJob, "", NULL);
LlDbAddTable(hJob, "Orders", NULL);
LlDbAddTable(hJob, "OrderDetails", NULL);
LlDbAddTable(hJob, " HumanResources.Employee", NULL); // scheme info
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LlDbAddTableSortOrder, LlDbAddTableRelation, LlPrintDbGetCurrentTable, LlPrintDbGetCurrentTableSortOrder, LlPrintDbGetCurrentTableRelation

LlDbAddTableRelation

Syntax:

```
INT LlDbAddTableRelation(HLLJOB hJob, LPCTSTR pszTableID,
   LPCTSTR pszParentTableID, LPCTSTR pszRelationID,
   LPCTSTR pszRelationDisplayName);
```

Aufgabe:

Meldet eine Beziehung zwischen zwei Tabellen für das Design und den Druck an. Die Tabelle steht dem Benutzer dann im Designer als Untertabelle der Elterntabelle zur Verfügung und kann zur Druckzeit von List & Label angefordert werden.

Parameter:

hJob: List & Label Job-Handle

pszTableID: ID der Kindtabelle. Diese muss mit der bei *LlDbAddTable()* übergebenen ID übereinstimmen.

pszParentTableID: ID der Elterntabelle. Diese muss mit der bei *LlDbAddTable()* übergebenen ID übereinstimmen.

pszRelationID: ID der Relation. Diese wird bei *LlPrintDbGetCurrentTableRelation()* zurückgeliefert, wenn die Kindtabelle gedruckt werden soll.

pszRelationDisplayName: Name der Relation wie er im Designer angezeigt wird. Wird nur zur Anzeige verwendet und nicht im Projektfile gespeichert. Kann auch NULL sein, in diesem Falle wird der zweite Parameter auch für die Anzeige im Designer verwendet.

Rückgabewert:

Fehlercode

Hinweise:

Beachten Sie die Hinweise im Kapitel "Drucken relationaler Daten". Zum Aufrufzeitpunkt müssen die Eltern- und Kindtabelle bereits mit *LlDbAddTable()* angemeldet worden sein.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

LlDbAddTable(hJob, "Orders", NULL);
LlDbAddTable(hJob, "OrderDetails", NULL);
LlDbAddTableRelation(hJob, "OrderDetails", "Orders",
   "Orders2OrderDetails", NULL);
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LlDbAddTable, LlDbAddTableSortOrder, LlPrintDbGetCurrentTable, LlPrintDbGetCurrentTableSortOrder, LlPrintDbGetCurrentTableRelation

LlDbAddTableRelationEx

Syntax:

```
INT LlDbAddTableRelationEx(HLLJOB hJob, LPCTSTR pszTableID,
   LPCTSTR pszParentTableID, LPCTSTR pszRelationID,
```

```
LPCTSTR pszRelationDisplayName, LPCTSTR pszKeyField,
LPCTSTR pszParentKeyField);
```

Aufgabe:

Meldet eine Beziehung zwischen zwei Tabellen für das Design und den Druck an, insbesondere bei der Verwendung von Drilldown-Funktionalität. Die Tabelle steht dem Benutzer dann im Designer als Untertabelle der Elterntabelle zur Verfügung und kann zur Druckzeit von List & Label angefordert werden. Durch die Parameter *pszKeyField* und *pszParentKeyField* kann eine eindeutige Zuordnung für Drilldown erfolgen.

Parameter:

hJob: List & Label Job-Handle

pszTableID: ID der Kindtabelle. Diese muss mit der bei *LIDbAddTable()* übergebenen ID übereinstimmen.

pszParentTableID: ID der Elterntabelle. Diese muss mit der bei *LIDbAddTable()* übergebenen ID übereinstimmen.

pszRelationID: ID der Relation. Diese wird bei *LIPrintDbGetCurrentTableRelation()* zurückgeliefert, wenn die Kindtabelle gedruckt werden soll.

pszRelationDisplayName: Name der Relation wie er im Designer angezeigt wird. Wird nur zur Anzeige verwendet und nicht im Projektfile gespeichert. Kann auch NULL sein, in diesem Falle wird der zweite Parameter auch für die Anzeige im Designer verwendet.

pszKeyField: Schlüsselfeld der Kindtabelle, mehrere Schlüsselfelder als TAB-getrennte Liste

pszParentKeyField: Schlüsselfeld der Elterntabelle, mehrere Schlüsselfelder als TAB-getrennte Liste

Rückgabewert:

Fehlercode

Hinweise:

Beachten Sie die Hinweise im Kapitel "Drucken relationaler Daten". Zum Aufrufzeitpunkt müssen die Eltern- und Kindtabelle bereits mit *LIDbAddTable()* angemeldet worden sein.

Beispiel:

Siehe Kapitel Direkter Druck und Export aus dem Designer

Siehe auch:

LIDbAddTable, LIDbAddTableSortOrder, LIPrintDbGetCurrentTable, LIPrintDbGetCurrentTableSortOrder, LIPrintDbGetCurrentTableRelation

LlDbAddTableSortOrder

Syntax:

```
INT LlDbAddTableSortOrder(HLLJOB hJob, LPCTSTR pszTableID,
   LPCTSTR pszSortOrderID, LPCTSTR pszSortOrderDisplayName);
```

Aufgabe:

Meldet eine Sortierung einer Tabelle für das Design und den Druck an. Die Sortierung steht dem Benutzer dann im Designer zur Verfügung und kann zur Druckzeit von List & Label angefordert werden.

Parameter:

hJob: List & Label Job-Handle

pszTableID: ID der Tabelle, für die die Sortierung bereitgestellt wird. Diese muss mit der bei *LlDbAddTable()* übergebenen ID übereinstimmen.

pszSortOrderID: ID der Sortierung. Diese wird bei *LlPrintDbGetCurrentTableSortOrder()* zurückgeliefert, wenn die Tabelle mit der entsprechenden Sortierung gedruckt werden soll.

pszSortOrderDisplayName: Name der Sortierung wie er im Designer angezeigt werden soll. Wird nur zur Anzeige verwendet und nicht im Projektfile gespeichert. Kann auch NULL sein, in diesem Falle wird der zweite Parameter auch für die Anzeige im Designer verwendet.

Rückgabewert:

Fehlercode

Hinweise:

Beachten Sie die Hinweise im Kapitel "Drucken relationaler Daten". Zum Aufrufzeitpunkt muss die Tabelle bereits mit *LlDbAddTable()* angemeldet worden sein.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

LlDbAddTable(hJob, "Orders", NULL);
LlDbAddTableSortOrder(hJob, "Orders", "Name ASC", "Name [+]");
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LlDbAddTable, LlDbAddTableRelation, LlPrintDbGetCurrentTable, LlPrintDbGetCurrentTableSortOrder, LlPrintDbGetCurrentTableRelation

LlDbAddTableSortOrderEx

Syntax:

```
INT LlDbAddTableSortOrder(HLLJOB hJob, LPCTSTR pszTableID,
   LPCTSTR pszSortOrderID, LPCTSTR pszSortOrderDisplayName,
   LPCTSTR pszField);
```

Aufgabe:

Meldet eine Sortierung einer Tabelle für das Design und den Druck an. Die Sortierung steht dem Benutzer dann im Designer zur Verfügung und kann zur Druckzeit von List & Label angefordert werden. Zusätzlich können die sortierungsrelevanten Felder Tab-separiert übergeben werden, so dass diese bei *LlGetUsedIdentifiers()* berücksichtigt werden können.

Parameter:

hJob: List & Label Job-Handle

pszTableID: ID der Tabelle, für die die Sortierung bereitgestellt wird. Diese muss mit der bei *LlDbAddTable()* übergebenen ID übereinstimmen.

pszSortOrderID: ID der Sortierung. Diese wird bei *LlPrintDbGetCurrentTableSortOrder()* zurückgeliefert, wenn die Tabelle mit der entsprechenden Sortierung gedruckt werden soll.

pszSortOrderDisplayName: Name der Sortierung wie er im Designer angezeigt werden soll. Wird nur zur Anzeige verwendet und nicht im Projektfile gespeichert. Kann auch NULL sein, in diesem Falle wird der zweite Parameter auch für die Anzeige im Designer verwendet.

pszField: Tab-separierte Liste der sortierungsrelevanten Felder, sofern diese bei *LlGetUsedIdentifiers()* berücksichtigt werden sollen.

Rückgabewert:

Fehlercode

Hinweise:

Beachten Sie die Hinweise im Kapitel "Drucken relationaler Daten". Zum Aufrufzeitpunkt muss die Tabelle bereits mit *LlDbAddTable()* angemeldet worden sein.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

LlDbAddTable(hJob, "Orders", NULL);
LlDbAddTableSortOrderEx(hJob, "Orders", "Name ASC", "Name [+]",
   "Orders.Name");
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LIDbAddTableSortOrder, LIDbAddTable, LIDbAddTableRelation, LIPrintDbGetCurrentTable, LIPrintDbGetCurrentTableSortOrder, LIPrintDbGetCurrentTableRelation

LIDbSetMasterTable

Syntax:

```
INT LlDbSetMasterTable(HLLJOB hJob, LPCTSTR pszTableID);
```

Aufgabe:

Meldet eine Tabelle als Master-Tabelle an. Findet Verwendung, wenn die Master-Daten (z.B. der Adressat einer Rechnung) als Variablen definiert werden, damit die passenden Unterdaten (z.B. die Rechnungsposten) auch direkt im Berichtscontainer eingefügt werden können, ohne die Rechnungs-Tabelle als Elterntabelle zu verwenden.

Parameter:

hJob: List & Label Job-Handle

pszTableID: ID der Tabelle, die als Mastertabelle verwendet werden soll. Diese muss mit der bei LIDbAddTable übergebenen ID übereinstimmen.

Rückgabewert:

Fehlercode

Hinweise:

Beachten Sie die Hinweise im Kapitel "Drucken relationaler Daten". Zum Aufrufzeitpunkt muss die Tabelle bereits mit *LIDbAddTable()* angemeldet worden sein.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

LlDbAddTable(hJob, "Orders", NULL);
LlDbSetMasterTable(hJob, "Orders");
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LIDbAddTable, LIDbAddTableRelation, LIPrintDbGetCurrentTable, LIPrintDbGetCurrentTableSortOrder, LIPrintDbGetCurrentTableRelation

LIDebugOutput

Syntax:

```
void LlDebugOutput(INT nIndent, LPCTSTR pszText);
```

Aufgabe:

Gibt den angegebenen Text auf Debwin aus.

Parameter:

nIndent: Einrückung für Folgezeilen

pszText: auszugebender Text

Hinweise:

Der Parameter *nIndent* ist für die Einrückung der Folgezeilen zuständig, d.h. ein Befehl mit Einrückung +1 sollte irgendwann mit einem Befehl mit -1 "gepaart" werden.

So können Verschachtelungen Ihrer Ausgaben im Debug-Output realisiert werden.

Beachten Sie, dass Sie – um die Ausgaben auch zu sehen – die Debug-Ausgaben über *LlSetDebug()* einschalten müssen.

Beispiel:

```
HLLJOB hJob;
LlSetDebug(LL_DEBUG_CMBTLL);
LlDebugOutput(+1, "Versionsnummer holen:");
hJob = LlJobOpen(0);
v = LlGetVersion(LL_VERSION_MAJOR);
LlJobClose(hJob);
LlDebugOutput(-1, "Versionsnummer geholt");
```

gibt in etwa folgendes auf dem Debugging-Output aus:

```
Versionsnummer holen:
 @LlJobOpen(0)=1
 @LlGetVersion(1)=20
 @LlJobClose(1)
Versionsnummer geholt
```

Siehe auch:

LlSetDebug, Debwin

LlDefineChartFieldExt

Syntax:

```
INT LlDefineChartFieldExt(HLLJOB hJob, LPCTSTR lpszName,
   LPCTSTR lpszCont, INT lPara, LPVOID lpPara);
```

Aufgabe:

Definiert ein Chartfeld und dessen Inhalt.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Feldname

lpszCont: Zeiger auf Zeichenkette mit Feldinhalt

lPara: Feldtyp

lpPara: für spätere Erweiterungen, muss NULL sein oder Zeiger auf Leerstring

Rückgabewert:

Fehlercode

Hinweise:

Es gelten die allgemeinen Hinweise von Kapitel "Hinweise zu Variablen- und Feldnamen".

lpPara muss, wie oben beschrieben, entweder NULL oder ein Zeiger auf 0 (Leerstring) sein.

Siehe auch:

LIDefineChartFieldStart, LL_OPTION_VARSCASESENSITIVE

LIDefineField

Syntax:

```
INT LlDefineField(HLLJOB hJob, LPCTSTR lpszName, LPCTSTR lpszCont);
```

Aufgabe:

Definiert ein Listenfeld und dessen Inhalt.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Feldname

lpszCont: Zeiger auf Zeichenkette mit Feldinhalt

Rückgabewert:

Fehlercode

Hinweise:

Es gelten die allgemeinen Hinweise von Kapitel "Hinweise zu Variablen- und Feldnamen".

Diese Funktion definiert ein Textfeld, sie kann beliebig mit den anderen *LIDefineField...()*-Funktionen gemischt werden.

LIDefineField() ist identisch mit *LIDefineFieldExt(..., LL_TEXT, NULL)*.

Folgende Felder sind von List & Label vorgegeben.

Feld	Bedeutung
LL.CountDataThisPage	Numerisch, Fußzeilenfeld, definierte Datensätze pro Seite
LL.CountData	Numerisch, Fußzeilenfeld, definierte Datensätze gesamt
LL.CountPrintedDataThisPage	Numerisch, Fußzeilenfeld, gedruckte Datensätze pro Seite
LL.CountPrintedData	Numerisch, Fußzeilenfeld, gedruckte Datensätze gesamt
LL.FCountDataThisPage	Numerisch, definierte Datensätze pro Seite
LL.FCountData	Numerisch, definierte Datensätze gesamt
LL.FCountPrintedDataThisPage	Numerisch, gedruckte Datensätze pro Seite
LL.FCountPrintedData	Numerisch, gedruckte Datensätze gesamt

Der Unterschied von "definierten" zu "gedruckten" Datensätzen besteht darin, dass der Benutzer einen Satzfilter auf die Tabelle anwenden kann, so dass mit jedem vom Programm gesendeten Datensatz sich die "definierten" Zahlen erhöhen, aber nicht unbedingt die "gedruckten".

Beispiel:

```
HLLJOB hJob;

hJob = LlJobOpen(0);
LlDefineFieldStart(hJob);
LlDefineField(hJob, "Name", "Normalverbraucher");
LlDefineField(hJob, "Vorname", "Otto");
LlDefineFieldExt(hJob, "Ort", "Konstanz", LL_TEXT, NULL);
LlDefineFieldExt(hJob, "PLZ", "*78462*", LL_BARCODE_3OF9);
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LlDefineFieldStart, LlDefineField, LlDefineFieldExt, LlDefineFieldExtHandle, LL_OPTION_VARSCASESENSITIVE

LlDefineFieldExt

Syntax:

```
INT LlDefineFieldExt(HLLJOB hJob, LPCTSTR lpszName, LPCTSTR lpszCont,
   INT lPara, LPVOID lpPara);
```

Aufgabe:

Definiert ein Listenfeld und dessen Inhalt.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Feldname

lpszCont: Zeiger auf Zeichenkette mit Feldinhalt

lPara: Feldtyp, bei Bedarf ´oder´ verknüpft mit (s.u.)

lpPara: für spätere Erweiterungen, muss NULL sein oder Zeiger auf Leerstring

Rückgabewert:

Fehlercode

Hinweise:

Es gelten die allgemeinen Hinweise von Kapitel "Hinweise zu Variablen- und Feldnamen".

Die von List & Label vorgegebenen Felder sind bei *LlDefineField()* aufgeführt.

lPara ver'oder't mit *LL_TABLE_FOOTERFIELD* stellt Felddefinitionen nur für den Listenfuß zur Verfügung. Der Listenfuß ist dynamisch an den Listenkörper gekoppelt und eignet sich so z.B. für dynamische Rechnungen als Summen oder Zwischensummenzeile.

lPara ver'oder't mit *LL_TABLE_HEADERFIELD* stellt Felddefinitionen nur im Listenheader zur Verfügung, entsprechend *LL_TABLE_GROUPFIELD* nur im Gruppenbereich, *LL_TABLE_GROUPFOOTERFIELD* nur im Gruppenfußbereich, *LL_TABLE_BODYFIELD* nur im Listendatenbereich.

Wenn keine Veroderung vorgenommen wird, tauchen die Felder in allen Tabellenbereichen zur Auswahl auf.

lpPara muss, wie oben beschrieben, entweder NULL oder ein Zeiger auf 0 (Leerstring) sein.

Beispiel:

```
HLLJOB hJob;

hJob = LlJobOpen(0);
LlDefineFieldStart(hJob);
LlDefineField(hJob, "Name", "Normalverbraucher");
LlDefineField(hJob, "Vorname", "Otto");
LlDefineFieldExt(hJob, "Ort", "Konstanz", LL_TEXT, NULL);
LlDefineFieldExt(job, "Anzahl_Seite","1",LL_TABLE_FOOTERFIELD Or LL_-
TEXT, NULL)
LlDefineFieldExt(hJob, "PLZ", "*78462*", LL_BARCODE_3OF9);
LlDefineFieldExt(hJob, "Foto", "c:\\fotos\\norm.bmp", LL_DRAWING);
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LlDefineFieldStart, LlDefineField, LlDefineFieldExtHandle, LL_OPTION_VARS-CASESENSITIVE

LlDefineFieldExtHandle

Syntax:

```
INT LlDefineFieldExtHandle(HLLJOB hJob, LPCTSTR lpszName,
   HANDLE hContents, INT lPara, LPVOID lpPara);
```

Aufgabe:

Definiert ein Listenfeld und dessen Inhalt.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Feldname

hContents: Handle vom Typ *HMETAFILE*, *HENHMETAFILE*, *HBITMAP* oder *HICON.*

lPara: *LL_DRAWING_HMETA, LL_DRAWING_HEMETA*, *LL_DRAWING_HICON* oder *LL_DRAWING_HBITMAP*

lpPara: für spätere Erweiterungen, muss NULL oder "" (Leerstring) sein (siehe *LlDefineFieldExt()*)

Rückgabewert:

Fehlercode

Hinweise:

Es gelten die allgemeinen Hinweise von Kapitel "Hinweise zu Variablen- und Feldnamen".

Diese Funktion definiert ein Textfeld, sie kann beliebig mit den anderen *LlDefineField...()*-Funktionen gemischt werden.

Die von List & Label vorgegebenen Felder sind bei *LlDefineField()* aufgeführt.

Das Handle muss so lange gültig sein, wie es gebraucht wird, also während der gesamten Layout-Definition oder bis nach *LlPrintFields()* bzw. *LlPrint()*.

Nach der Verwendung kann bzw. sollte es über die übliche Windows-API-Funktion gelöscht werden.

Beispiel:

```
HLLJOB hJob;
HMETAFILE hMeta;
HDC   hMetaDC;
INT   i;
hMetaDC = CreateMetaFile(NULL); // Fieberkurve
```

```
selectObject(hMetaDC,GetStockObject(NULL_PEN));
Rectangle(hMetaDC, 0, 0, LL_META_MAXX, LL_METY_MAXY);
for (i = 0; i < 10; ++i)
{
   MoveTo(hMetaDC,0,MulDiv(i, LL_META_MAXY, 10));
   LineTo(hMetaDC,MulDiv(i, LL_META_MAXX, 100),
         MulDiv(i, LL_META_MAXY, 10);
}
MoveTo(hMetaDC, 0, MulDiv(((100*i) & 251) % 100, LL_META_MAXY,100));
for (i = 0; i < 10; ++i)
   LineTo(hMetaDC,MulDiv(i, LL_META_MAXX, 10),
         MulDiv(((100*i) & 251) % 100, LL_META_MAXY, 100));
hMeta = CloseMetaFile(hMetaDC);

hJob = LlJobOpen(0);
LlDefineFieldStart(hJob);
LlDefineField(hJob, "Name", "Normalverbraucher");
LlDefineField(hJob, "Vorname", "Otto");
LlDefineFieldExt(hJob, "Ort", "Konstanz", LL_TEXT, NULL);
LlDefineFieldExtHandle(hJob, "Erfolgs-Chart", hMeta,
      LL_DRAWING_HMETA, NULL);
<... etc ...>
LlJobClose(hJob);
DeleteObject(hMeta);
```

Siehe auch:

LIDefineFieldExt, LIDefineFieldStart, LIDefineField, LL_OPTION_VARS-CASESENSITIVE

LIDefineFieldStart

Syntax:

```
void LlDefineFieldStart(HLLJOB hJob);
```

Aufgabe:

Leert den DLL-internen Feldpuffer, um alte Definitionen zu löschen.

Parameter:

hJob: List & Label Job-Handle

Hinweise:

Die Hinweise zu *LIDefineVariableStart()* gelten auch für diese Funktion.

Wenn die Funktion *LIPrintIsFieldUsed()* verwendet wird, darf diese Funktion nur vor dem Laden des Projekts aufgerufen werden, da *LIDefineFieldStart()* auch den "Used"-Status zurücksetzt. Wir empfehlen aber ohnehin die Verwendung von *LIGetUsedIdentifiers.*

In keinem Fall darf die Funktion innerhalb der Druckschleife aufgerufen werden!

Beispiel:

```
HLLJOB hJob;

hJob = LlJobOpen(0);
```

```
LlDefineFieldStart(hJob);
LlDefineField(hJob, "Name", "Normalverbraucher");
LlDefineField(hJob, "Vorname", "Otto");
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LIDefineField, LIDefineFieldExt, LIDefineFieldExtHandle, LL_OPTION_VARS-CASESENSITIVE

LIDefineLayout

Syntax:

```
INT LlDefineLayout(HLLJOB hJob, HWND hWnd, LPCTSTR lpszTitle,
   UINT nObjType, LPCTSTR lpszObjName);
```

Aufgabe:

Aufruf des interaktiven Designers, welcher in einem modalen Pop-up Fenster Ihr Anwendungsfenster (siehe *hWnd*-Parameter) überlagert.

Parameter:

hJob: List & Label Job-Handle

hWnd: Window-Handle des aufrufenden Fensters

lpszTitle: Fenstertitel

nObjType:

Wert	Bedeutung
LL_PROJECT_LABEL	für Etiketten
LL_PROJECT_CARD	für Karteikarten
LL_PROJECT_LIST	für Listen

jeweils bei Bedarf mit 'oder' verknüpft mit:

Wert	Bedeutung
LL_FIXEDNAME	Sperrt die Menüpunkte 'Neu' und 'Laden' (besser: über *LIDesignerProhibitAction()*)
LL_NOSAVEAS	Sperrt den Menüpunkt ´Speichern Als´ (besser: über *LIDesignerProhibitAction()*)
LL_NONAMEINTITLE	Verhindert das Anhängen des Dateinamens an den Fenstertitel

lpszObjName: Dateiname des gewünschten Objekts

Rückgabewert:

Fehlercode

Hinweise:

Das Window-Handle wird dazu verwendet, das aufrufende Programm(fenster) zu deaktivieren.

Falls dies nicht gewünscht ist, kann auch NULL übergeben werden. In diesem Fall hat dann das aufrufende Programm für das ordnungsgemäße Beenden des Layout-Editors zu sorgen, falls der Benutzer das Hauptprogramm abbricht. Dieses Vorgehen wird jedoch **ausdrücklich nicht empfohlen**.

Bei Iconisierung des List & Label-Designers wird das aufrufende Programm auch automatisch iconisiert, bei der darauffolgenden Restaurierung wird auch der Designer wieder mit restauriert.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

LlDefineVariableStart(hJob);
LlDefineVariable(hJob, "Name", "Normalverbraucher");
LlDefineVariable(hJob, "Vorname", "Otto");
LlDefineVariable(hJob, "PIN", "40|08150|78462", LL_BARCODE_EAN13, NULL);
LlDefineLayout(hJob, hWndMain, "Test", LL_PROJECT_LABEL, "test.lbl")
LlJobClose(hJob);
```

Siehe auch:

LlDesignerProhibitAction, LlSetOption, LlSetFileExtensions

LlDefineSumVariable

Syntax:

```
INT LlDefineSumVariable(HLLJOB hJob, LPCTSTR lpszName,
   LPCTSTR lpszCont);
```

Aufgabe:

Definiert einen Summenvariableninhalt.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Feldname

lpszCont: Zeiger auf Zeichenkette mit Feldinhalt

Rückgabewert:

Fehlercode

Hinweise:

Der Feldinhalt muss numerisch sein.

Die Benutzung dieser Funktion ist nur sinnvoll, wenn der Endanwender nicht den Designer benutzen darf/kann, da das Ergebnis einer Summenvariablen in diesem Falle nicht unbedingt mit den Designereinstellungen konform wäre.

Beispiel:

```
HLLJOB hJob;

hJob = LlJobOpen(0);
LlDefineSumVariable(hJob, "@Summe15", "14");
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LlGetSumVariableContents

LlDefineVariable

Syntax:

```
INT LlDefineVariable(HLLJOB hJob, LPCTSTR lpszName, LPCTSTR lpszCont);
```

Aufgabe:

Definiert eine Variable vom Typ *LL_TEXT* und deren Inhalt.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Variablenname

lpszCont: Zeiger auf Zeichenkette mit Variableninhalt

Rückgabewert:

Fehlercode

Hinweise:

Es gelten die allgemeinen Hinweise von Kapitel "Hinweise zu Variablen- und Feldnamen".

Diese Funktion definiert eine Textvariable, sie kann beliebig mit den anderen *LlDefineVariable...()*-Funktionen gemischt werden.

LlDefineVariable() ist identisch mit *LlDefineVariableExt(..., LL_TEXT, NULL)*.

Von List & Label sind schon folgende Variablen vorgegeben:

Variable	Bedeutung
LL.CountDataThisPage	Numerisch, definierte Datensätze pro Seite
LL.CountData	Numerisch, definierte Datensätze gesamt

Variable	Bedeutung
LL.CountPrintedDataThisPage	Numerisch, gedruckte Datensätze pro Seite
LL.CountPrintedData	Numerisch, gedruckte Datensätze gesamt
LL.SortStrategy	Zeichenkette, Sortierausdruck
LL.FilterExpression	Zeichenkette, Filterausdruck

Der Unterschied von "definierten" zu "gedruckten" Datensätzen besteht darin, dass der Benutzer einen Filter auf die Datensätze anwenden kann, so dass mit jedem vom Programm gesendeten Datensatz sich die "definierten" Zahlen erhöhen, aber nicht unbedingt die "gedruckten" (letztere Werte werden nur dann erhöht, wenn der Datensatz gedruckt wurde).

Beispiel:

```
HLLJOB hJob;

hJob = LlJobOpen(0);
LlDefineVariableStart(hJob);
LlDefineVariable(hJob, "Name", "Normalverbraucher");
LlDefineVariable(hJob, "Vorname", "Otto");
LlDefineVariableExt(hJob, "Ort", "Konstanz", LL_TEXT, NULL);
LlDefineVariableExt(hJob, "PLZ", "*78462*", LL_BARCODE_3OF9 , NULL);
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LlDefineVariableStart, LlDefineVariableExt, LlDefineVariableExtHandle, LlGetVariableContents, LlGetVariableType, LL_OPTION_VARSCASESENSITIVE

LlDefineVariableExt

Syntax:

```
INT LlDefineVariableExt(HLLJOB hJob, LPCTSTR lpszName,
   LPCTSTR lpszCont, INT lPara, LPVOID lpPara);
```

Aufgabe:

Definiert eine Variable und deren Inhalt.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Variablenname

lpszCont: Zeiger auf Zeichenkette mit Variableninhalt

lPara: Variablentyp

lpPara: für spätere Erweiterungen, muss NULL oder "" (Leerstring) sein

Rückgabewert:

Fehlercode

Hinweise:

Es gelten die allgemeinen Hinweise von Kapitel "Hinweise zu Variablen- und Feldnamen".

Diese Funktion kann beliebig mit den anderen *LlDefineVariable...()*-Funktionen gemischt werden.

Die von List & Label vorgegebenen Variablen sind bei *LlDefineVariable()* aufgeführt.

Beispiel:

```
hJob = LlJobOpen(0);
LlDefineVariableStart(hJob);
LlDefineVariableExt(hJob, "Ort", "Konstanz", LL_TEXT, NULL);
LlDefineVariableExt(hJob, "PLZ", "*78462*", LL_BARCODE_3OF9, NULL);
LlDefineVariableExt(hJob, "Foto", "i.bmp", LL_DRAWING, NULL);
LlJobClose(hJob);
```

Siehe auch:

LlDefineVariableStart, LlDefineVariable, LlDefineVariableExtHandle, LlGetVariableContents, LlGetVariableType, LL_OPTION_VARSCASESENSITIVE

LlDefineVariableExtHandle

Syntax:

```
INT LlDefineVariableExtHandle(HLLJOB hJob, LPSTR lpszName,
   HANDLE hContents, INT lPara, LPVOID lpPara);
```

Aufgabe:

Definiert eine Variable und deren Inhalt.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Variablenname

hContents: Handle (*HMETAFILE, HENHMETAFILE, HICON* oder *HBITMAP*)

lPara: *LL_DRAWING_HMETA, LL_DRAWING_HEMETA*, *LL_DRAWING_HICON* oder *LL_DRAWING_HBITMAP*

lpPara: für spätere Erweiterungen, muss NULL oder "" (Leerstring) sein (siehe LlDefineFieldExt)

Rückgabewert:

Fehlercode

Hinweise:

Es gelten die allgemeinen Hinweise von Kapitel "Hinweise zu Variablen- und Feldnamen".

Diese Funktion kann beliebig mit den anderen *LlDefineVariable...()*-Funktionen gemischt werden.

Das Handle muss so lange gültig sein, wie es gebraucht wird, also während der gesamten Layout-Definition oder bis nach *LlPrintFields()* bzw. *LlPrint()*.

Nach der Verwendung kann bzw. sollte es über die übliche API-Funktion gelöscht werden.

Beispiel:

```
HLLJOB hJob;
HMETAFILE hMeta;
HDC hMetaDC;
INT i;

hMetaDC = CreateMetaFile(NULL);// Fieberkurve
SelectObject(hMetaDC, GetStockObject(NULL_PEN));
Rectangle(hMetaDC, 0, 0, LL_META_MAXX, LL_META_MAXY);

for (i = 0; i < 10; ++i)
{
   MoveTo(hMetaDC,0,MulDiv(i, LL_META_MAXY, 10));
   LineTo(hMetaDC,MulDiv(i, LL_META_MAXX, 100), MulDiv(i, LL_META_-
MAXY,10);
}
MoveTo(hMetaDC,0,MulDiv(((100*i) & 251) % 100, LL_META_MAXY,100));
for (i = 0; i < 10; ++i)
   LineTo(hMetaDC,MulDiv(i, LL_META_MAXX, 10),MulDiv(((100*i) & 251) %
100,
            LL_META_MAXY,100));
hMeta = CloseMetaFile(hMetaDC);

hJob = LlJobOpen(0);
LlDefineVariableStart(hJob);
LlDefineVariable(hJob, "Name", "Normalverbraucher");
LlDefineVariable(hJob, "Vorname", "Otto");
LlDefineVariableExtHandle(hJob, "Chart", hMeta, LL_DRAWING_HMETA, NULL);
LlDefineVariableExt(hJob, "PLZ", "*78462*", LL_BARCODE_3OF9, NULL);
<... etc ...>
LlJobClose(hJob);
DeleteObject(hMeta);
```

Siehe auch:

LlDefineVariableStart, LlDefineVariable, LlDefineVariableExt, LlGetVariableContents, LlGetVariableType, LL_OPTION_VARSCASESENSITIVE

LlDefineVariableStart

Syntax:

```
void LlDefineVariableStart(HLLJOB hJob);
```

Aufgabe:

Leert den internen Variablenpuffer, um alte Definitionen zu löschen.

Parameter:

hJob: List & Label Job-Handle

Hinweise:

Muss nicht unbedingt aufgerufen werden. Da jedoch bei jedem *LlDefineVariable...()* die interne Variablenliste nach einer schon vorhandenen Variablen desselben Namens und Typs durchsucht wird, kann dies durch diese Funktion etwas verschnellert werden. Andernfalls braucht man nur die Variablen, deren Inhalt sich ändert, anzugeben, da dann der alte Inhalt der Variable überschrieben wird, die Inhalte der übrigen Variablen aber erhalten bleiben.

Wenn die Funktion *LlPrintIsVariableUsed()* verwendet wird, darf diese Funktion nur vor dem Laden des Projekts aufgerufen werden, da *LlDefineVariableStart()* auch den "Used"-Status zurücksetzt.

In keinem Fall darf die Funktion innerhalb der Druckschleife aufgerufen werden!

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);
LlDefineVariableStart(hJob);
LlDefineVariable(hJob, "Name", "Normalverbraucher");
LlDefineVariable(hJob, "Vorname", "Otto");
<...etc ...>
LlDefineVariable(hJob, "Vorname", "Friedrich");
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LlDefineVariable, LlDefineVariableExt, LlDefineVariableExtHandle, LlGetVariableContents, LlGetVariableType

LlDesignerAddAction

Syntax:

```
INT LlDesignerAddAction(HLLJOB hJob, UINT nID, UINT nFlags,
   LPCTSTR pszMenuText, LPCTSTR pszMenuHierarchy,
   LPCTSTR pszTooltipText, UINT nIcon, LPVOID pvReserved);
```

Aufgabe:

Erweitert den Designer und wahlweise auch die Toolbar des Designers um eigene Menüpunkte. Im Unterschied zur Verwendung des Callbacks *LL_CMND_MODIFYMENU* kann hier auch eine Schaltfläche mit wählbarem Icon zur Toolbar hinzugefügt werden. Dieser Befehl muss vor *LlDefineLayout()* aufgerufen werden.

Parameter:

hJob: List & Label Job-Handle

nID: Menü-ID für die neu hinzuzufügende Aktion. Diese ID erhalten Sie im Callback *LL_CMND_SELECTMENU*, wenn der Benutzer den zugehörigen Menüpunkt oder Button ausgewählt hat. Benutzerdefinierte IDs sollten im Bereich zwischen 10100 und 10999 liegen.

nFlags: Kombination (Veroderung) der folgenden Flags:

Wert	Bedeutung
LLDESADDACTIONFLAG_ADD_TO_TOOLBAR	Zusätzlich zum Menüeintrag eine Schaltfläche zur Toolbar hinzufügen.
LLDESADDACTION_MENUITEM_APPEND	Der Menüpunkt wird hinter dem in pszMenuHierarchy bezeichneten Eintrag eingefügt.
LLDESADDACTION_MENUITEM_INSERT	Der Menüpunkt wird vor dem in pszMenuHierarchy bezeichneten Eintrag eingefügt.

sowie wahlweise zusätzlich einen Keycode als Shortcut und eine Kombination (veroderung) der folgenden Flags als Modifikator:

Wert	Bedeutung
LLDESADDACTION_ACCEL_CONTROL	Tastaturkürzel ist STRG+Keycode.
LLDESADDACTION_ACCEL_SHIFT	Tastaturkürzel ist UMSCHALT+Keycode.
LLDESADDACTION_ACCEL_ALT	Tastaturkürzel ist ALT+Keycode.
LLDESADDACTION_ACCEL_VIRTKEY	Sollte immer gesetzt sein.

pszMenuText: Menütext ohne Tastaturshortcut (dieser wird automatisch ergänzt). Sie können aber das "&"-Zeichen für die Vergabe der Kurztaste bei Menünavigation verwenden. Um Untermenüpunkte anzulegen verwenden Sie "." als Hierarchietrenner. Um z.B. ein Menü "Vorlagen" mit dem Unterpunkt "Rechnung" anzulegen, verwenden Sie "Vorlagen.Rechnung" als Menütext.

pszMenuHierarchy: Menühierarchie des neuen Menüpunkts. Die Angabe erfolgt in der Form "<Ebene>.<Ebene>..." wobei "Ebene" jeweils der 0-basierende Index des Menüeintrages ist. Um z.B. in das "Bearbeiten"-Menü an erster Stelle einen neuen Eintrag einzufügen, verwenden Sie "1.0" und *LLDESADDACTION_MENUITEM_INSERT.*

pszTooltipText: Text für den Tooltip der Toolbarschaltfläche. Wird nur ausgewertet, wenn das Flag *LLDESADDACTIONFLAG_ADD_TO_TOOLBAR* gesetzt wird. Darf NULL sein.

nIcon: Icon-ID für die Schaltfläche. Wird nur ausgewertet, wenn das Flag *LLDESADDACTIONFLAG_ADD_TO_TOOLBAR* gesetzt wird. Verwenden Sie das Programm IconSelector.exe (im Tools-Verzeichnis) um die verfügbaren Icons mit ihren IDs angezeigt zu bekommen.

pvReserved: Für zukünftige Erweiterungen, muss NULL sein.

Rückgabewert:

Fehlercode

Hinweise:

Um die eigentliche Aktion auszuführen muss der *LL_CMND_SELECTMENU*-Callback behandelt werden.

Siehe auch:

LlDefineLayout

LlDesignerFileOpen

Syntax:

```
INT LlDesignerFileOpen(HLLJOB hJob, LPCTSTR pszFileName,
   UINT nFlags);
```

Aufgabe:

Öffnet bei geöffnetem Designer die angegebene Projektdatei.

Parameter:

hJob: List & Label Job-Handle

pszFileName: Projektdateiname mit Pfadangabe

nFlags: Kombination (Veroderung) jeweils eines Flags aus den folgenden zwei Gruppen:

Wert	Bedeutung
LL_DESFILEOPEN_OPEN_EXISTING	Datei muss bereits existieren, sonst wird Fehlercode zurückgeliefert.
LL_DESFILEOPEN_CREATE_ALWAYS	Datei wird immer neu erzeugt. Wenn schon vorhanden wird der Inhalt gelöscht.
LL_DESFILEOPEN_CREATE_NEW	Datei wird neu erzeugt, wenn nicht vorhanden. Wenn Datei bereits existiert wird

Wert	Bedeutung
	Fehlercode zurückgeliefert.
LL_DESFILEOPEN_OPEN_ALWAYS	Wenn Datei vorhanden, wird der Inhalt verwendet, sonst wird Datei neu erzeugt.
LL_DESFILEOPEN_OPEN_IMPORT	Importiert eine bestehende Datei in ein bereits geöffnetes Projekt.

Wert	Bedeutung
LL_DESFILEOPENFLAG_SUPPRESS_SAVEDIALOG	Die gerade geöffnete Datei wird vor dem Laden des neuen Projekts ohne Benutzerinteraktion gespeichert.
LL_DESFILEOPENFLAG_SUPPRESS_SAVE	Die gerade geöffnete Datei wird ohne Speichern geschlossen. **Alle Änderungen seit dem letzten Speichervorgang gehen damit verloren!**
LL_DESFILEOPENFLAG_DEFAULT	Die gerade geöffnete Datei wird – wenn nötig – nach Benutzerauswahl gespeichert oder verworfen, bevor das neue Projekt geladen wird.

Rückgabewert:

Fehlercode

Hinweise:

Die Funktion kann nur innerhalb eines Designer-Events verwendet werden. Typisch ist die Verwendung in Verbindung mit *LlDesignerAddAction()* um bestimmte Abläufe zu automatisieren.

Siehe auch:

LlDesignerFileSave

LlDesignerFileSave

Syntax:

```
INT LlDesignerFileSave(HLLJOB hJob, UINT nFlags);
```

Aufgabe:

Speichert bei geöffnetem Designer die gerade geöffnete Projektdatei.

Parameter:

hJob: List & Label Job-Handle

nFlags: Für spätere Erweiterungen, muss "0" (*LL_DESFILESAVE_DEFAULT*) sein.

Rückgabewert:

Fehlercode

Hinweise:

Die Funktion kann nur innerhalb eines Designer-Events verwendet werden. Typisch ist die Verwendung in Verbindung mit *LlDesignerAddAction()* um bestimmte Abläufe zu automatisieren.

Siehe auch:

LlDesignerFileOpen

LlDesignerGetOptionString

Syntax:

```
INT LlDesignerGetOptionString(HLLJOB hJob, INT nMode, LPTSTR pszBuffer,
   UINT nBufSize);
```

Aufgabe:

Fragt bei geöffnetem Designer diverse Einstellungen ab.

Parameter:

hJob: List & Label Job-Handle

nMode: Optionsindex, siehe *LlDesignerSetOptionString()*

pszBuffer: Puffer für Rückgabewert, darf NULL sein (s.u.)

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode oder benötigte Puffergröße, wenn pszBuffer NULL ist.

Hinweise:

Die für den Mode-Parameter gültigen Werte sind bei *LlDesignerSetOptionString()* aufgeführt.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlDesignerSetOptionString

LlDesignerInvokeAction

Syntax:

```
INT LlDesignerInvokeAction(HLLJOB hJob, INT nMenuIndex);
```

Aufgabe:

Löst bei geöffnetem Designer die angegebene Aktion/den angegebenen Menüpunkt aus.

Parameter:

hJob: List & Label Job-Handle

nMenuIndex: Funktionsindex. Die verfügbaren Funktionen finden Sie in der Datei MENUID.TXT.

Rückgabewert:

Fehlercode

Hinweise:

Wenn die Funktion eingesetzt werden soll, muss sie innerhalb eines Designer-Events verwendet werden. Typisch ist die Verwendung in Verbindung mit *LlDesignerAddAction()* um bestimmte Abläufe zu automatisieren.

Siehe auch:

LlDefineLayout, LlDesignerAddAction

LlDesignerProhibitAction

Syntax:

```
INT LlDesignerProhibitAction(HLLJOB hJob, INT nMenuIndex);
```

Aufgabe:

Verhindert eine Benutzeraktion im Designer, indem es Menüpunkte sperrt (versteckt).

Parameter:

hJob: List & Label Job-Handle

nMenuIndex: Funktionsindex:

Wert	Bedeutung
0	Das Menü wird zurückgesetzt und dadurch der Ausgangszustand herbeigeführt. Bei *LlJobOpen[LCID]()* wird dies automatisch aufgerufen; bei mehreren *LlDefineLayout()*-Aufrufen mit verschiedenen Sperreinträgen wird diese Funktion also gebraucht, wenn zwischendurch der Job nicht freigegeben und wieder angefordert wird, sonst addieren sich die Sperreinträge.
LL_SYSCOMMAND_MINIMIZE	Das Designer-Fenster kann nicht minimiert (iconisiert) werden

Wert	Bedeutung
LL_SYSCOMMAND_-MAXIMIZE	Der Designer kann nicht maximiert werden.
andere Werte	Hier können die Menü-IDs der zu sperrenden Menüs angegeben werden. Die entsprechenden IDs finden Sie (für die englische Version) in der Datei MENUID.TXT in Ihrer List & Label Installation.

Rückgabewert:

Fehlercode

Hinweise:

Wenn die Funktion eingesetzt werden soll, muss sie vor der *LlDefineLayout()*-Funktion aufgerufen werden.

Der Aufruf kann mehrfach hintereinander für verschiedene Funktionsindex-Werte aufgerufen werden, da die Einträge zu einer Sperreintragsliste hinzugefügt werden, die bei dem Aufruf von *LlDefineLayout()* ausgewertet wird.

Sie können Menüidentifikationen auch in den Callbacks *LL_CMND_ENABLEMENU* und *LL_CMND_MODIFYMENU* durchführen.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

LlDefineVariableStart(hJob);
LlDefineVariable(hJob, "Name", "Normalverbraucher");
LlDefineVariable(hJob, "Vorname", "Otto");
LlDefineVariable(hJob, "PIN", "40|08150|78462", LL_BARCODE_EAN13, NULL);
LlDesignerProhibitAction(hJob, LL_SYSCOMMAND_MINIMIZE);
LlDesignerProhibitAction(hJob, 515); // no "save as"
LlDefineLayout(hJob,hWndMain, "Test", LL_PROJECT_LABEL, "test.lbl");
LlJobClose(hJob);
```

Siehe auch:

LlDefineLayout, Callback LL_CMND_ENABLEMENU, Callback LL_CMND_-MODIFYMENU

LlDesignerProhibitEditingObject

Syntax:

```
INT LlDesignerProhibitEditingObject(HLLJOB Job, LPCTSTR pszObject);
```

Aufgabe:

Verhindert, dass das übergebene Objekt im Designer bearbeitet werden kann.

Parameter:

hJob: List & Label Job-Handle

pszObject: Objektname.

Rückgabewert:

Fehlercode

Hinweise:

Über NULL oder einen Leerstring wird die Liste der zu sperrenden Objekte gelöscht.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

LlDesignerProhibitEditingObject(hJob, "DemoText");
...
LlJobClose(hJob);
```

Siehe auch:

-

LlDesignerProhibitFunction

Syntax:

```
INT LlDesignerProhibitFunction(HLLJOB Job, LPCTSTR pszFunction);
```

Aufgabe:

Verhindert, dass die übergebene Funktion im Funktionsassistenten angeboten wird.

Parameter:

hJob: List & Label Job-Handle

pszFunction: Funktionsname.

Rückgabewert:

Fehlercode

Hinweise:

Über NULL oder einen Leerstring wird die Liste der zu unterdrückenden Funktionen gelöscht.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

LlDesignerProhibitFunction(hJob, "");
LlDesignerProhibitFunction(hJob, "CStr$");
```

```
...
LlJobClose(hJob);
```

Siehe auch:

-

LIDesignerRefreshWorkspace

Syntax:

```
INT LlDesignerRefreshWorkspace(HLLJOB hJob);
```

Aufgabe:

Löst im Designer eine Aktualisierung aller Toolfenster, Menüpunkte etc. aus. Verwenden Sie diese Funktion, um nach Änderungen am Objektmodell per DOM bei geöffnetem Designer sicherzustellen, dass der Designer diese Änderungen auch sofort darstellt.

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Fehlercode

Hinweise:

Die Funktion kann nur innerhalb eines Designer-Events verwendet werden. Typisch ist die Verwendung in Verbindung mit *LIDesignerAddAction()*.

Siehe auch:

LIDefineLayout, LIDesignerAddAction

LIDesignerSetOptionString

Syntax:

```
INT LlDesignerSetOptionString (HLLJOB hJob, INT nMode, LPCTSTR
pszValue);
```

Aufgabe:

Setzt diverse Einstellungen im geöffneten List & Label Designer.

Parameter:

hJob: List & Label Job-Handle

nMode: Folgende Werte sind als Funktionsindex möglich:

LL_DESIGNEROPTSTR_PROJECTFILENAME

Der Name des gerade geöffneten Projekts. Wenn Sie durch eine Aktion eine Datei neu angelegt haben, können Sie dieser auf diese Weise einen Namen zuordnen. Ansonsten entspricht das Setzen einem "Speichern unter…".

LL_DESIGNEROPTSTR_ WORKSPACETITLE

Bestimmt den Inhalt der Titelzeile im Designer. Sie können im Text den Formatierungsplatzhalter %s verwenden, um den Projektnamen anzuzeigen.

LL_DESIGNEROPTSTR_ PROJECTDESCRIPTION

Setzt die Projektbeschreibung, die auch im "Datei öffnen"-Dialog angezeigt wird.

pszValue: neuer Wert

Rückgabewert:

Fehlercode

Siehe auch:

LlDesignerGetOptionString

LlDlgEditLineEx

Syntax:

```
INT LlDlgEditLineEx(HLLJOB Job, HWND hWnd, LPTSTR pszBuffer,
   UINT nBufSize, UINT nParaTypes, LPCTSTR pszTitle,
   BOOL bTable, LPVOID pReserved);
```

Aufgabe:

Diese Funktion steht nur in der Enterprise Edition zur Verfügung! Startet den List & Label Formelassistenten unabhängig vom Designer. Dadurch können List & Label Formeln auch an vom Druck unabhängigen Stellen der Applikation verwendet werden.

Parameter:

hJob: List & Label Job-Handle

hWnd: Fensterhandle des aufrufenden Programms

pszBuffer: Puffer für Rückgabewert

nBufSize: Größe des Puffers

nParaTypes: erwarteter Rückgabetyp. Ein oder mehrere veroderte *LL_FCTPARATYPE_...* Werte

pszTitle: Fenstertitel. Beachten Sie, dass das Wort "bearbeiten" durch List & Label angehängt wird.

bTable: bestimmt, ob nur Variablen (*FALSE*) oder Felder (*TRUE*) zur Verfügung stehen sollen

pReserved: reserviert, muss NULL oder leer (") sein.

Hinweise:

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Rückgabewert:

Fehlercode

LlDomCreateSubobject

Syntax:

```
INT LlDomCreateSubobject(HLLDOMOBJ hDOMObj, INT nPosition,
   LPCTSTR pszType, PHLLDOMOBJ phDOMSubObj);
```

Aufgabe:

Erzeugt ein neues Unterobjekt innerhalb der angegebenen DOM-Liste. Ausführliche Anwendungsbeispiele finden Sie im Kapitel "DOM-Funktionen".

Parameter:

hDomObj: DOM-Handle der Liste

nPosition: Index (0-basierend) des einzufügenden Elements. Alle folgenden Elemente werden um eine Position nach hinten verschoben.

pszType: gewünschter Elementtyp, z.B. "Text" um in der Objektliste ein neues Textobjekt anzulegen

phDOMSubObj: Zeiger auf DOM-Handle für die Rückgabe

Rückgabewert:

Fehlercode

Siehe auch:

Kapitel "DOM-Funktionen"

LlDomDeleteSubobject

Syntax:

```
INT LlDomDeleteSubobject(HLLDOMOBJ hDOMObj, INT nPosition);
```

Aufgabe:

Löscht ein Unterobjekt aus der angegebenen DOM-Liste. Ausführliche Anwendungsbeispiele finden Sie im Kapitel "DOM-Funktionen".

Parameter:

hDomObj: DOM-Handle der Liste

nPosition: Index (0-basierend) des zu löschenden Elements. Alle folgenden Elemente werden um eine Position nach vorne verschoben.

Rückgabewert:

Fehlercode

Siehe auch:

Kapitel "DOM-Funktionen"

LlDomGetObject

Syntax:

```
INT LlDomGetObject(HLLDOMOBJ hDOMObj, LPCTSTR pszName,
   PHLLDOMOBJ phDOMSubObj);
```

Aufgabe:

Liefert ein Unterobjekt des angegebenen DOM-Objektes, wird z.B. verwendet, um vom Projekt die Objektliste zu erfragen. Ausführliche Anwendungsbeispiele finden Sie im Kapitel "DOM-Funktionen".

Parameter:

hDomObj: DOM-Handle des "Eltern"-Objektes

pszName: Name des gewünschten Unterobjektes, z.B. "Objects"

phDOMSubObj: Zeiger auf DOM-Handle für die Rückgabe

Rückgabewert:

Fehlercode

Siehe auch:

Kapitel "DOM-Funktionen"

LlDomGetProject

Syntax:

```
INT LlDomGetProject(HLLJOB hJob, PHLLDOMOBJ phDOMObj);
```

Aufgabe:

Liefert das aktuell geladene Projektobjekt. Kann z.B. nach *LlPrint(WithBox)Start* verwendet werden, um das Projekt während des Ausdrucks mit DOM-Funktionen zu verändern. Um Projekte neu zu erstellen oder *vor* dem Ausdruck zu bearbeiten verwenden Sie *LlProjectOpen(), LlDomGetProject(), LlProjectSave() und LlProjectClose()*. Ausführliche Anwendungsbeispiele finden Sie im Kapitel "DOM-Funktionen".

Parameter:

hJob: List & Label Job-Handle

phDOMSubObj: Zeiger auf DOM-Handle für die Rückgabe

Rückgabewert:

Fehlercode

Siehe auch:

LlProjectOpen, LlProjectSave, LlProjectClose , Kapitel "DOM-Funktionen"

LlDomGetProperty

Syntax:

```
INT LlDomGetProperty(HLLDOMOBJ hDOMObj, LPCTSTR pszName,
   LPTSTR pszBuffer, UINT nBufSize);
```

Aufgabe:

Liefert den Inhalt der angegebenen Eigenschaft zurück. Ausführliche Anwendungsbeispiele finden Sie im Kapitel "DOM-Funktionen".

Parameter:

hDomObj: DOM-Handle des abzufragenden Objekts

pszName: Name der gewünschten Eigenschaft, z.B. "Condition" um von einem Objekt die Darstellungsbedingung zu erfragen.

pszBuffer: Speicherbereich, in den der Parameter geschrieben werden soll. Kann NULL sein (siehe Hinweise)

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode bzw. benötigte Puffergröße

Hinweise:

Wenn pszBuffer NULL ist, ist der Rückgabewert die Länge des benötigten Puffers (in TCHARS, also BYTEs im SBCS/MBCS-Fall und WCHARs bei UNICODE) inklusive der String-Terminierung.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

Kapitel "DOM-Funktionen"

LlDomGetSubobject

Syntax:

```
INT LlDomGetSubobject(HLLDOMOBJ hDOMObj, INT nPosition,
   PHLLDOMOBJ phDOMSubObj);
```

Aufgabe:

Liefert das angegebene Unterelement der DOM-Liste zurück. Ausführliche Anwendungsbeispiele finden Sie im Kapitel "DOM-Funktionen".

Parameter:

hDomObj: DOM-Handle der abzufragenden Liste

nPosition: Index (0-basierend) des gewünschten Elements

phDOMSubObj: Zeiger auf DOM-Handle für die Rückgabe

Rückgabewert:

Fehlercode

Siehe auch:

Kapitel "DOM-Funktionen"

LlDomGetSubobjectCount

Syntax:

```
INT LlDomGetSubobjectCount(HLLDOMOBJ hDOMObj, _LPINT pnCount);
```

Aufgabe:

Liefert die Anzahl der Elemente in der angegebenen DOM-Liste zurück. So kann z.B. die Anzahl der Objekte in der Objektliste ermittelt werden. Ausführliche Anwendungsbeispiele finden Sie im Kapitel "DOM-Funktionen".

Parameter:

hDomObj: DOM-Handle der abzufragenden Liste

pnCount: Zeiger für die Rückgabe

Rückgabewert:

Fehlercode

Siehe auch:

Kapitel "DOM-Funktionen"

LlDomSetProperty

Syntax:

```
INT LlDomSetProperty(HLLDOMOBJ hDOMObj, LPCTSTR pszName,
   LPCTSTR pszValue);
```

Aufgabe:

Setzt die angegebene Eigenschaft auf den übergebenen Wert. Ausführliche Anwendungsbeispiele finden Sie im Kapitel "DOM-Funktionen".

Parameter:

hDomObj: DOM-Handle des abzufragenden Objekts

pszName: Name der gewünschten Eigenschaft, z.B. "Condition" um die Darstellungsbedingung eines Objektes zu setzen

pszValue: Neuer Wert der Eigenschaft

Rückgabewert:

Fehlercode

Siehe auch:

Kapitel "DOM-Funktionen"

LlEnumGetEntry

Syntax:

```
HLISTPOS LlEnumGetEntry(HLLJOB hJob, HLISTPOS hPos, LPTSTR pszNameBuf,
   UINT nNameBufsize, LPTSTR pszContBuf, UINT nContBufSize,
   _LPHANDLE pHandle, _LPINT pType);
```

Aufgabe:

Liefert den Inhalt und den Namen einer Variable bzw. eines Feldes.

Parameter:

hJob: List & Label Job-Handle

hPos: Das Handle der momentanen Variable/des momentanen Felds

pszNameBuf, nNameBufsize: beschreiben einen Puffer, in dem der Variablen/Feldbezeichner gespeichert werden soll

pszContBuf, nContBufSize: beschreiben einen Puffer, in dem der Inhalt gespeichert werden soll. pszContBuf darf NULL sein.

pHandle: Zeiger auf ein HANDLE, in dem ein etwaiges Handle gespeichert werden soll. Darf NULL sein (siehe *LlDefineVariableExtHandle()* und *LlDefineFieldExtHandle()*).

pType: Zeiger auf INT, in dem der Typ gespeichert werden soll (*LL_TEXT*, ...). Darf NULL sein.

Rückgabewert:

Fehlercode

Hinweise:

Während der Iteration darf nicht *LlDefineVariableStart()* oder *LlDefineFieldStart()* aufgerufen werden!

Über die Enumeratoren kann man die Variablen- und Feldliste durchlaufen und die Definitionen der vorhandenen Variablen und Felder samt Typen und Inhalten abfragen.

Folgendes Beispiel durchläuft die Variablenliste und gibt alle Variablen aus (*LL_TYPEMASK* ist die Addition aller möglichen Variablentypen):

```
HLISTPOS hPos = LlEnumGetFirstVar(hJob, LL_TYPEMASK);
while (hPos != NULL)
{
   TCHAR szName[64+1];
   TCHAR szContents[256+1];
   LlEnumGetEntry(hJob, hPos, szName, sizeof(szName), szContents,
sizeof(szContents),
            NULL, NULL);
   printf("%s - %s\n",szName,szContents);
   hPos = LlEnumGetNextEntry(hJob, hPos, LL_TYPEMASK);
}
```

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlEnumGetFirstVar, LlEnumGetFirstField, LlEnumGetNextEntry

LlEnumGetFirstChartField

Syntax:

```
HLISTPOS LlEnumGetFirstChartField(HLLJOB hJob, UINT nFlags);
```

Aufgabe:

Liefert das erste definierte Chart-Feld. Dabei muss der Name des Feldes nicht bekannt sein. Die Felder werden in der Reihenfolge wie sie bei List & Label angemeldet wurden zurückgeliefert.

Parameter:

hJob: List & Label Job-Handle

nFlags: Flags für die erlaubten Feldtypen: *LL_TEXT, LL_BOOLEAN, LL_BARCODE, LL_DRAWING, LL_NUMERIC, LL_DATE, LL_RTF, LL_HTML*

Rückgabewert:

Handle oder NULL, wenn kein Feld vorhanden.

Hinweise:

Während einer Iteration darf *LlDefineChartFieldStart()* nicht aufgerufen werden.

Siehe auch:

LlEnumGetFirstField, LlEnumGetFirstVar, LlEnumGetNextEntry, LlEnumGetEntry

LlEnumGetFirstField

Syntax:

```
HLISTPOS LlEnumGetFirstField(HLLJOB hJob, UINT nFlags);
```

Aufgabe:

Liefert das erste definierte Feld. Dabei muss der Name des Feldes nicht bekannt sein. Die Felder werden in der Reihenfolge wie sie bei List & Label angemeldet wurden zurückgeliefert.

Parameter:

hJob: List & Label Job-Handle

nFlags: Flags für die erlaubten Feldtypen: *LL_TEXT, LL_BOOLEAN, LL_BARCODE, LL_DRAWING, LL_NUMERIC, LL_DATE, LL_RTF, LL_HTML*

Rückgabewert:

Handle oder NULL, wenn kein Feld vorhanden.

Hinweise:

Während einer Iteration darf *LlDefineFieldStart()* nicht aufgerufen werden.

Siehe auch:

LlEnumGetFirstChartField, LlEnumGetFirstVar, LlEnumGetNextEntry, LlEnumGetEntry

LlEnumGetFirstVar

Syntax:

```
HLISTPOS LlEnumGetFirstVar(HLLJOB hJob, UINT nFlags);
```

Aufgabe:

Liefert die erste definierte Variable. Dabei muss der Name der Variable nicht bekannt sein. Die Variablen werden in der Reihenfolge wie sie bei List & Label angemeldet wurden zurückgeliefert.

Parameter:

hJob: List & Label Job-Handle

nFlags: Flags für die erlaubten Variablentypen: *LL_TEXT, LL_BOOLEAN, LL_BARCODE, LL_DRAWING, LL_NUMERIC, LL_DATE, LL_RTF, LL_HTML*

Rückgabewert:

Handle oder NULL, wenn keine Variable vorhanden.

Hinweise:

Während einer Iteration darf *LlDefineVariableStart()* nicht aufgerufen werden.

Es werden keine internen Variablen ausgegeben.

Siehe auch:

LlEnumGetFirstField, LlEnumGetNextEntry, LlEnumGetEntry

LlEnumGetNextEntry

Syntax:

```
HLISTPOS LlEnumGetNextEntry(HLLJOB hJob, HLISTPOS hPos, UINT nFlags);
```

Aufgabe:

Liefert das nächste definierte Feld / Variable. Die Iterierung wir über *LlEnumGetFirstVar()* oder *LlEnumGetFirstField()* begonnen und mit dieser Funktion fortgesetzt.

Parameter:

hJob: List & Label Job-Handle

hPos: Das Handle der momentanen Variable/des momentanen Felds

nFlags: Flags für die erlaubten Variablentypen: *LL_TEXT, LL_BOOLEAN, LL_BARCODE, LL_DRAWING, LL_NUMERIC, LL_DATE, LL_RTF, LL_HTML*

Rückgabewert:

Handle für die nächste Variable/Feld oder NULL, wenn keine passende Variable/kein passendes Feld mehr vorhanden.

Hinweise:

Während einer Iteration darf *LlDefineVariableStart()* oder *LlDefineFieldStart()* nicht aufgerufen werden.

Siehe auch:

LlEnumGetFirstVar, LlEnumGetFirstField, LlEnumGetEntry

LlExprError

Syntax:

```
void LlExprError(HLLJOB hJob, LPTSTR lpBuffer, UINT nBufferSize);
```

Aufgabe:

Gibt den Grund des Fehlers in einem Ausdruck-String von *LlExprParse()* im Klartext zurück.

Parameter:

hJob: List & Label Job-Handle

lpBuffer: Zeiger auf Puffer, in den der Fehlertext geschrieben werden soll

nBufferSize: Größe des Puffers, empfohlen: ca. 250 Zeichen

Rückgabewert:

Fehlercode

Hinweise:

Die Funktion muss sofort nach *LlExprParse()* aufgerufen werden.

LlExprError() führt beim Laden (auch zum Druck) eines Projekts nur bedingt zum Erfolg, da der interne Formelparser evtl. beim Aufruf der Funktion schon eine völlig andere Formel geparst hat und somit evtl. gar keinen Fehler mehr "aktiv" hat, kann man über den Callback *LL_NTFY_EXPRERROR* Fehlermeldungen sammeln und dann nach *LlPrintStart()* dem Benutzer melden.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Beispiel:

Siehe *LlExprParse()*

Siehe auch:

LlExprParse, LlExprEvaluate, LlExprType, LlExprFree

LlExprEvaluate

Syntax:

```
INT LlExprEvaluate(HLLJOB hJob, HLLEXPR lpExpr, LPTSTR lpBuffer,
   UINT nBufferSize);
```

Aufgabe:

Evaluiert einen Ausdruck.

Parameter:

hJob: List & Label Job-Handle

lpExpr: Der vom dazugehörigen *LlExprParse()* zurückgegebene Zeiger

lpBuffer: Zeiger auf Puffer, in den der berechnete Wert geschrieben werden soll

nBufferSize: Größe des Puffers

Rückgabewert:

Fehlercode

Hinweise:

Der Puffer wird immer mit einer nullterminierten Zeichenkette gefüllt.

Abhängig vom Typ des Ergebnisses ist der Pufferinhalt wie folgt zu interpretieren:

Typ	Bedeutung
LL_EXPRTYPE_STRING	Der Pufferinhalt ist die Ergebnis-Zeichenkette
LL_EXPRTYPE_DOUBLE	Der Pufferinhalt ist die entsprechende Darstellung des Wertes, für Pi z.B. "3.141592". Der Wert wird immer mit 6 Nachkommastellen ausgegeben.
LL_EXPRTYPE_DATE	Der Pufferinhalt ist die entsprechende Darstellung des julianischen Wertes, z.B. "21548263".
LL_EXPRTYPE_BOOL	Der Puffer enthält entweder die Zeichenkette "TRUE" oder "FALSE".
LL_EXPRTYPE_DRAWING	Der Puffer enthält den Namen der Zeichnungs-Variablen/des Zeichnungs-Feldes(!), nicht den Inhalt.
LL_EXPRTYPE_BARCODE	Der Puffer enthält den Wert, der als Barcode zu interpretieren wäre.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Beispiel:

Siehe *LlExprParse()*

Siehe auch:

LlExprParse, LlExprType, LlExprError, LlExprFree

LlExprFree

Syntax:

```
void LlExprFree (HLLJOB hJob, HLLEXPR lpExpr);
```

Aufgabe:

Gibt die mit *LlExprParse()* erzeugte Baumstruktur wieder frei.

Parameter:

hJob: List & Label Job-Handle

lpExpr: Der vom dazugehörigen *LlExprParse()* zurückgegebene Zeiger

Hinweise:

Um Speicherverluste zu vermeiden, muss die Funktion aufgerufen werden, wenn ein über *LlExprParse()* erzeugter Baum nicht mehr benötigt wird und freigegeben werden soll.

Beispiel:

Siehe *LlExprParse()*

Siehe auch:

LlExprParse, LlExprEvaluate, LlExprType, LlExprError

LlExprGetUsedVars

Syntax:

```
INT LlExprGetUsedVars(HLLJOB hJob, HLLEXPR lpExpr, LPSTR pszBuffer, UINT
nBufSize);
```

Aufgabe:

Gibt die in der mit LlExprParse() berechneten Formel verwendeten Variablen und Felder zurück (Tab-separiert).

Parameter:

hJob: List & Label Job-Handle

lpExpr: Der vom dazugehörigen *LlExprParse()* zurückgegebene Zeiger

pszBuffer: Puffer für Rückgabewert

nBufSize: Größe des Puffers

Hinweise:

Entspricht "LlExprGetUsedVarsEx" mit Parameter bOrgName = TRUE.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LIExprParse, LIExprEvaluate, LIExprType, LIExprError

LIExprGetUsedVarsEx

Syntax:

```
INT LlExprGetUsedVarsEx(HLLJOB hLlJob, HLLEXPR lpExpr, LPSTR pszBuffer,
UINT nBufSize, BOOL bOrgName);
```

Aufgabe:

Gibt die in der mit LIExprParse() berechneten Formel verwendeten Variablen und Felder zurück (Tab-separiert).

Parameter:

hJob: List & Label Job-Handle

lpExpr: Der vom dazugehörigen *LIExprParse()* zurückgegebene Zeiger

pszBuffer: Puffer für Rückgabewert

nBufSize: Größe des Puffers

bOrgName: TRUE: globale Feldnamen, FALSE: lokalisierte Feldnamen

Hinweise:

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LIExprParse, LIExprEvaluate, LIExprType, LIExprError

LIExprParse

Syntax:

```
LPVOID LlExprParse(HLLJOB hJob, LPCTSTR lpExprText, BOOL bTableFields);
```

Aufgabe:

Testet den Ausdruck auf Korrektheit und erstellt eine interne Datenstruktur für diesen Ausdruck.

Parameter:

hJob: List & Label Job-Handle

lpExprText: Ausdruck

bTableFields: TRUE: Bezug auf Felder (und Variablen), FALSE: Bezug nur auf Variablen

Rückgabewert:

Zeiger auf List & Label interne Datenstruktur

Hinweise:

Wenn ein Fehler zurückgemeldet wird (Adresse = NULL), kann man über *LlExprError()* die Fehlerbeschreibung abgefragt werden.

Die über *LlDefineVariable()* definierten Variablen können mit in den Ausdruck eingebaut werden, wenn *bTableFields* auf FALSE ist, ansonsten werden die über *LlDefineField()* definierten Felder in den Ausdruck mit einbezogen.

Wenn der Ausdruck mehrmals benötigt wird, empfiehlt es sich, diesen einmal über *LlExprParse()* übersetzen zu lassen und dann die Berechnungen durchzuführen und erst am Schluss den Baum wieder freizugeben.

Wenn beim Aufruf dieser Funktionen ein Fehler im Ausdruck erkannt wird, wird der entsprechende Fehler-Callback *LL_NTFY_EXPRERROR* aufgerufen.

Beispiel:

```
LPVOID lpExpr;
char     lpszErrortext[128];
char     lpszBuf[20];
long     lDateOne;
long     lDateTwo;

LlDefineVariable(hJob, "Datum", "29.2.1964", LL_TEXT);
lpExpr = LlExprParse(hJob, "DateToJulian(DATE(Datum))", FALSE);

if (lpExpr)
{
   if (LlExprType(hJob, lpExpr) != LL_EXPRTYPE_DOUBLE)
   {
      // da stimmt was nicht, muss numerisch sein!
   }
   LlExprEvaluate(hJob, lpExpr, lpszBuf, sizeof(lpszBuf));
   lDateOne = atol(lpszBuf);
   // lDateOne hat nun das julianische Datum vom 29.2.1964
   LlDefineVariable(hJob, "Datum", "28.10.2013", LL_TEXT);
   LlExprEvaluate(hJob, lpExpr,  lpszBuf, sizeof(lpszBuf));
   lDateTwo = atol(lpszBuf);
   // lDateTwo hat nun das julianische Datum vom 28.10.2013
   LlExprFree(hJob, LpExpr);
}
else
{
   // Fehler!
   LlExprError(hJob, lpszErrortext, sizeof(lpszErrortext));
}
```

Siehe auch:

LlExprEvaluate, LlExprType, LlExprError, LlExprFree

LlExprType

Syntax:

```
INT LlExprType(HLLJOB hJob, HLLEXPR lpExpr);
```

Aufgabe:

Evaluiert den Typ eines Ausdrucks.

Parameter:

hJob: List & Label Job-Handle

lpExpr: Der vom dazugehörigen *LlExprParse()* zurückgegebene Zeiger

Rückgabewert:

Typ des Ergebnisses:

Wert	Bedeutung
LL_EXPRTYPE_STRING	Zeichenkette
LL_EXPRTYPE_DOUBLE	Numerischer Wert
LL_EXPRTYPE_DATE	Datum
LL_EXPRTYPE_BOOL	Bool´scher Wert
LL_EXPRTYPE_DRAWING	Zeichnung
LL_EXPRTYPE_BARCODE	Barcode

Beispiel:

Siehe *LlExprParse()*

Siehe auch:

LlExprParse, LlExprEvaluate, LlExprError, LlExprFree

LlGetChartFieldContents

Syntax:

```
INT LlGetChartFieldContents(HLLJOB hJob, LPCTSTR lpszName,
   LPTSTR lpszBuffer, UINT nBufSize);
```

Aufgabe:

Gibt den Inhalt des entsprechenden Chart-Felds zurück.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Feldname

lpszBuffer: Zeiger auf Puffer, in den der Inhalt gespeichert werden soll

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode (meist *LL_ERR_UNKNOWN_FIELD* oder 0)

Hinweise:

Diese Funktion kann in Callback-Routinen verwendet werden, um Chartfeld-Inhalte abzufragen.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlDefineChartFieldStart, LlDefineChartFieldExt, LlGetFieldType

LlGetDefaultPrinter

Syntax:

```
INT LlGetDefaultPrinter(LPTSTR pszPrinter, LLPUINT pnPrinterBufferSize,
   _PDEVMODE pDevMode, LLPUINT pnDevModeBufSize, UINT nOptions)
```

Aufgabe:

Liefert den Namen des Standard-Druckers und füllt eine DEVMODE Struktur gemäß seinen Standard-Einstellungen.

Parameter:

pszPrinter: Puffer für den Druckernamen. Kann NULL sein (siehe Hinweise).

pnPrinterBufferSize: Größe des Puffers (in Zeichen)

pDevMode: Zeiger auf einen Puffer für die DEVMODE Struktur. Kann NULL sein (siehe Hinweise).

pnDevModeBufSize: Größe des Pufferbereichs (in Bytes).

nOptions: Reserviert, muss 0 sein.

Rückgabewert:

Fehlercode bzw. benötigte Puffergröße

Hinweise:

Wenn *pszPrinter* und *pDevMode* NULL sind, wird die benötigte Puffergröße in *pnPrinterBufferSize* und *pnDevModeBufSize* gespeichert.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlSetPrinterToDefault, LlSetPrinterInPrinterFile

LlGetDefaultProjectParameter

Syntax:

```
INT LlGetDefaultProjectParameter(HLLJOB hLlJob, LPCTSTR pszParameter,
   LPTSTR pszBuffer, INT nBufSize, _LPUINT pnFlags);
```

Aufgabe:

Fragt den Default-Wert eines Projekt-Parameters ab (siehe auch Kapitel Projekt-Parameter)

Parameter:

hJob: List & Label Job-Handle

pszParameter: Name des Parameters. Kann NULL sein (siehe Hinweise)

pszBuffer: Speicherbereich, in den der Parameter geschrieben werden soll. Kann NULL sein (siehe Hinweise)

nBufSize: Größe des Pufferbereichs, auf den *pszBuffer* zeigt (in TCHARs).

pnFlags: Zeiger auf einen UINT, den Typ des Parameters (Werte siehe *LlSetDefaultProjectParameter()*). Kann NULL sein, wenn der Wert nicht benötigt wird.

Rückgabewert:

Fehlercode bzw. benötigte Puffergröße

Hinweise:

Wenn *pszParameter* NULL ist, wird eine Semikolon-separierte Liste aller USER-Parameter zurückgegeben.

Wenn *pszBuffer* NULL ist, ist der Rückgabewert die Länge des benötigten Puffers (in TCHARS, also BYTEs im SBCS/MBCS-Fall und WCHARs bei UNICODE) inklusive der String-Terminierung.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlSetDefaultProjectParameter, LlPrintSetProjectParameter, LlPrintGetProjectParameter

LlGetErrortext

Syntax:

```
INT LlGetErrortext(INT nError, LPTSTR lpszBuffer, UINT nBufSize);
```

Aufgabe:

Liefert eine lokalisierte Fehlermeldung für den übergebenen Fehlercode.

Parameter:

nError: Fehlercode

lpszBuffer: Zeiger auf Puffer, in den die Meldung gespeichert werden soll

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode bzw. benötigte Puffergröße

Hinweise:

Diese Funktion kann verwendet werden, um eine Fehlermeldung anzuzeigen. Häufigere Fehler sind z.B. *LL_ERR_EXPRESSION* (-23) oder *LL_ERR_NOPRINTER* (-11). Wenn bereits ein Job geöffnet wurde erfolgt die Ausgabe in der Sprache des jeweiligen Jobs, ansonsten wird die Sprache des ersten gefundenen Sprachkits verwendet.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

LlGetFieldContents

Syntax:

```
INT LlGetFieldContents(HLLJOB hJob, LPCTSTR lpszName, LPTSTR lpszBuffer,
   UINT nBufSize);
```

Aufgabe:

Gibt den Inhalt des entsprechenden Felds zurück.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Feldname

lpszBuffer: Zeiger auf Puffer, in den der Inhalt gespeichert werden soll

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode (meist *LL_ERR_UNKNOWN_FIELD* oder 0)

Hinweise:

Diese Funktion kann in Callback-Routinen verwendet werden, um Variableninhalte abzufragen.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LIDefineFieldStart, LIDefineFieldExt, LIDefineFieldExtHandle, LIGetFieldType

LIGetFieldType

Syntax:

```
INT LlGetFieldType(HLLJOB hJob, LPCTSTR lpszName);
```

Aufgabe:

Gibt den Typ des entsprechenden Felds (oder Chart-Felds) zurück.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Feldname

Rückgabewert:

Feldtyp (positiv), oder Fehlercode (negativ)

Hinweise:

Diese Funktion kann in Callback-Routinen verwendet werden, um Variableninhalte abzufragen.

Siehe auch:

LIDefineFieldStart, LIDefineFieldExt, LIDefineFieldExtHandle, LIGetFieldContents

LIGetNotificationMessage

Syntax:

```
UINT LlGetNotificationMessage(HLLJOB hJob);
```

Aufgabe:

Rückgabe der Nachrichtennummer für Callback-(USER-)Objekte.

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

eingestellte Nachrichtennummer

Hinweise:

Die voreingestellte Nachrichtennummer hat den Wert der Funktion *RegisterWindowMessage("cmbtLLMessage").*

Höhere Priorität hat die Callback-Funktion. Wenn diese definiert ist, wird keine Nachricht gesendet

Beispiel:

```
HLLJOB hJob;
INT    wMsg;

LlSetDebug(TRUE);
hJob = LlJobOpen(0);
v = LlGetNotificationMessage(hJob);
LlJobClose(hJob);
```

Siehe auch:

LlSetNotificationMessage, LlSetNotificationCallback

LlGetOption

Syntax:

```
INT_PTR LlGetOption(HLLJOB hJob, INT nMode);
```

Aufgabe:

Fragt diverse Einstellungen und Parameter (s.u.) in List & Label ab.

Parameter:

hJob: List & Label Job-Handle

nMode: Optionsindex, siehe *LlSetOption()*

Rückgabewert:

Der Wert der entsprechenden Option

Hinweise:

Die für den Mode-Parameter gültigen Werte sind bei *LlSetOption()* aufgeführt. Hinzu kommen folgende **neue** oder im Rückgabewert **modifizierte** Optionen:

LL_OPTION_DEFPRINTERINSTALLED

Gibt zurück, ob es im Betriebssystem einen Default-Drucker gibt.

LL_OPTION_LANGUAGE

Über diese Option können Sie die eingestellte Sprache abfragen. Siehe *LlJobOpen()* und *LlJobOpenLCID()*.

LL_OPTION_HELPAVAILABLE

LOWORD: Einstellung von *LlSetOption()*

HIWORD: Testet, ob die Hilfedatei benutzbar ist: TRUE: benutzbar, FALSE: nicht benutzbar

Beispiel:

```
HLLJOB hJob;
UINT32 n;
```

```
hJob = LlJobOpen(0);
n = LlGetOption(hJob, LL_OPTION_LANGUAGE, TRUE);
// ....
LlJobClose(hJob);
```

Siehe auch:

LlSetOption

LlGetOptionString

Syntax:

```
INT LlGetOptionString(HLLJOB hJob, INT nMode, LPTSTR pszBuffer,
   UINT nBufSize);
```

Aufgabe:

Fragt diverse Einstellungen in List & Label ab.

Parameter:

hJob: List & Label Job-Handle

nMode: Optionsindex, siehe *LlSetOptionString()*

pszBuffer: Puffer für Rückgabewert

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode

Hinweise:

Die für den Mode-Parameter gültigen Werte sind bei *LlSetOptionString()* aufgeführt.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlSetOptionString

LlGetPrinterFromPrinterFile

Syntax:

```
INT LlGetPrinterFromPrinterFile (HLLJOB hJob, UINT nObjType,
   LPCTSTR pszObjName, INT nPrinter, LPTSTR pszPrinter,
   LLPUINT pnSizePrn, _PDEVMODE pDM, LLPUINT pnSizeDm);
```

Aufgabe:

Ermöglicht es, die Druckerkonfiguration aus der Druckerbeschreibungsdatei zu lesen.

Parameter:

hJob: List & Label Job-Handle

nObjType: *LL_PROJECT_LABEL, LL_PROJECT_LIST* oder *LL_PROJECT_CARD*

pszObjName: Dateiname des Projekts

nPrinter: Druckerindex (0 für Erstseiten-Drucker, 1 für Folgeseiten-Drucker) Wenn Sie Werte ab 100 übergeben (bspw. in einer Schleife solange bis Sie *LL_ERR_PARAMETER* als Rückgabewert bekommen), können Sie damit den Drucker für die Layout-Bereiche (in der Reihenfolge wie sie unter Projekt > Seitenlayout hinzugefügt wurden) abfragen. Sollte das Projekt nur einen Drucker enthalten muss *nPrinter* den Wert -1 erhalten.

pszPrinter: Zeiger auf einen Puffer, in den der Druckername gespeichert werden soll. Wenn *pszPrinter* NULL ist und *pnSizePrn* nicht NULL ist, wird die Größe des benötigten Platzes in **pnSizePrn* gespeichert.

pnSizePrn: Puffergröße des Bereichs, auf den *pszPrinter* zeigt (Größe in Zeichen, also muss bei der Unicode-API die doppelte Anzahl in Bytes reserviert werden).

pDM: Zeiger auf einen Puffer, in dem die DEVMODE-Struktur des Druckers abgelegt wird. Wenn *pDM* NULL ist und *pnSizeDm* nicht NULL ist, wird die Größe des benötigten Platzes in **pnSizeDm* gespeichert.

pnSizeDm: Puffergröße des Bereichs, auf den *pDM* zeigt.

Rückgabewert:

Fehlercode

Hinweise:

Die *DEVMODE*-Struktur ist in der Windows-API definiert.

Durch die Möglichkeit unterschiedliche Druck-Bereiche im Designer definieren zu können, ist die praktische Nutzbarkeit dieser Funktion sehr stark eingeschränkt. Wir empfehlen daher, über das LL Objektmodell gemäß Kapitel ("Verwendung der DOM-API (ab Professional Edition)" auf die Bereiche und die dort eingestellten Drucker zuzugreifen.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlSetPrinterInPrinterFile

LlGetProjectParameter

Syntax:

```
INT LlGetProjectParameter(HLLJOB hJob, LPCTSTR lpszProjectName,
 LPCTSTR lpszParameter, lpszLPTSTR lpszBuffer, UINT nBufSize);
```

Aufgabe:

Liefert den Wert eines Projektparameters im angegebenen Projekt zurück. Bei Projektparametern, die Formeln verwenden, wird die (nicht evaluierte) Formel zurückgeliefert.

Parameter:

hJob: List & Label Job-Handle

lpszProjectName: Zeiger auf Zeichenkette mit Projektname

lpszParameter: Zeiger auf Zeichenkette mit Parametername

lpszBuffer: Zeiger auf Puffer, in den der Inhalt gespeichert werden soll

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode

Beispiel:

```
HLLJOB hJob;
TCHAR Buffer[1024];
hJob = LlJobOpen(0);

LlSetDefaultProjectParameter(hJob, "QueryString",
   "SELECT * FROM PRODUCTS", LL_PARAMETERFLAG_SAVEDEFAULT);
// Designeraufruf
…

// anschließend vor Druckstart
LlGetProjectParameter(hJob, "c:\\repository\\report.lst", "QueryString",
Buffer, 1024);
<... etc ...>
LlJobClose(hJob);
```

Hinweise:

Mit Hilfe dieser Funktion lassen sich vor dem Druckstart die Werte der Projektparameter auslesen. Dies ist insbesondere dann nützlich, wenn Sie eigene Projektparameter angemeldet haben, die dem Benutzer im Designer eine Parametrisierung der Druckausgabe erlauben.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlSetDefaultProjectParameter

LlGetSumVariableContents

Syntax:

```
INT LlGetSumVariableContents(HLLJOB hJob, LPCTSTR lpszName,
   LPTSTR lpszBuffer, UINT nBufSize);
```

Aufgabe:

Gibt den Inhalt der gewünschten Summenvariablen zurück.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Namen der Summenvariablen

lpszBuffer: Zeiger auf Puffer, in den der Inhalt gespeichert werden soll

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode (*LL_ERR_UNKNOWN_FIELD* oder 0)

Hinweise:

Diese Funktion kann in Callback-Routinen verwendet werden, um Summenvariableninhalte abzufragen.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlDefineSumVariable, LlGetUserVariableContents, LlGetVariableContents

LlGetUsedIdentifiers

Syntax:

```
INT LlGetUsedIdentifiers(HLLJOB hJob, LPCTSTR lpszProjectName,
   LPTSTR lpszBuffer, UINT nBufSize);
```

Aufgabe:

Liefert die im angegebenen Projekt verwendeten Variablen, Felder und Chartfelder als semikolon-separierte Liste zurück.

Parameter:

hJob: List & Label Job-Handle

lpszProjectName: Zeiger auf Zeichenkette mit Projektname

lpszBuffer: Zeiger auf Puffer, in den der Inhalt gespeichert werden soll

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode

Hinweise:

Mit Hilfe dieser Funktion lassen sich vor dem Druckstart die tatsächlich benötigten Felder, Chartfelder und Variablen ermitteln. Dadurch brauchen auch nur diese angemeldet zu werden, was zu erheblichen Performancegewinnen führen kann.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlGetUsedIdentifiersEx

LlGetUsedIdentifiersEx

Syntax:

```
INT LlGetUsedIdentifiers(HLLJOB hJob, LPCTSTR lpszProjectName,
   UINT nIdentifierTypes, LPTSTR lpszBuffer, UINT nBufSize);
```

Aufgabe:

Liefert die im angegebenen Projekt verwendeten Felder, Chartfelder, Variablen, Tabellen oder Relationen zurück.

Parameter:

hJob: List & Label Job-Handle

lpszProjectName: Zeiger auf Zeichenkette mit Projektname

nIdentifierTypes: Gewünschte Typen für die Rückgabe. Die folgenden Werte können verodert werden:

Wert	Bedeutung
LL_USEDIDENTIFIERSFLAG_VARIABLES	Variablen
LL_USEDIDENTIFIERSFLAG_FIELDS	Felder
LL_USEDIDENTIFIERSFLAG_CHARTFIELDS	Chart-Felder
LL_USEDIDENTIFIERSFLAG_TABLES	Tabellen (vgl. LlDbAddTable)
LL_USEDIDENTIFIERSFLAG_RELATIONS	Relationen (vgl. LlDbAddTableRelation)

lpszBuffer: Zeiger auf Puffer, in den der Inhalt gespeichert werden soll

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode

Hinweise:

Mit Hilfe dieser Funktion lassen sich vor dem Druckstart die tatsächlich benötigten Felder, Chartfelder, Variablen, Tabellen und Relationen ermitteln. Dadurch brauchen auch nur diese angemeldet zu werden, was zu erheblichen Performancegewinnen führen kann.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlGetUsedIdentifiers

LlGetUserVariableContents

Syntax:

```
INT LlGetUserVariableContents(HLLJOB hJob, LPCTSTR lpszName,
   LPTSTR lpszBuffer, UINT nBufSize);
```

Aufgabe:

Gibt den Inhalt der gewünschten Benutzervariablen zurück.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Feldname

lpszBuffer: Zeiger auf Puffer, in den der Inhalt gespeichert werden soll

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode (meist *LL_ERR_UNKNOWN_FIELD* oder 0)

Hinweise:

Diese Funktion kann in Callback-Routinen verwendet werden, um Benutzervariableninhalte abzufragen.

Der Typ einer Benutzervariablen kann über *LlGetVariableType()* oder *LlGetFieldType()* abgefragt werden.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlGetSumVariableContents, LlGetVariableContents

LlGetVariableContents

Syntax:

```
INT LlGetVariableContents(HLLJOB hJob, LPCTSTR lpszName,
   LPTSTR lpszBuffer, UINT nBufSize);
```

Aufgabe:

Gibt den Inhalt der entsprechenden Variablen zurück.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Variablenname

lpszBuffer: Zeiger auf Puffer, in den der Inhalt gespeichert werden soll

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode (meist *LL_ERR_UNKNOWNVARIABLE* oder 0)

Hinweise:

Diese Funktion kann in Callback-Routinen verwendet werden, um Variableninhalte abzufragen.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlDefineVariableStart, LlDefineVariableExt, LlDefineVariableExtHandle, LlGetVariableType

LlGetVariableType

Syntax:

```
INT LlGetVariableType(HLLJOB hJob, LPCTSTR lpszName);
```

Aufgabe:

Gibt den Typ der entsprechenden Variablen zurück.

Parameter:

hJob: List & Label Job-Handle

lpszName: Zeiger auf Zeichenkette mit Variablenname

Rückgabewert:

Variablentyp (positiv), oder Fehlercode (negativ)

Hinweise:

Diese Funktion kann in Callback-Routinen verwendet werden, um Variablentypen abzufragen.

Siehe auch:

LlDefineVariableStart, LlDefineVariableExt, LlDefineVariableExtHandle, LlGetVariableContents

LlGetVersion

Syntax:

```
INT LlGetVersion(INT nCmd);
```

Aufgabe:

Rückgabe der Versionsnummer von List & Label.

Parameter:

Wert	Bedeutung
LL_VERSION_MAJOR	Rückgabe der Haupt-Versionsnummer, z.B. 20
LL_VERSION_MINOR	Rückgabe der Unter-Versionsnummer, z.B. 13, (13 bedeutet 013, da die Unter-Versionsnummer bei List & Label immer 3-stellig ist)

Rückgabewert:

siehe Parameter

Beispiel:

```
INT   v;
v = LlGetVersion(LL_VERSION_MAJOR);
```

LlJobClose

Syntax:

```
void LlJobClose(HLLJOB hJob);
```

Aufgabe:

Schließen des DLL-Jobs

Parameter:

hJob: List & Label Job-Handle

Hinweise:

Diese Funktion muss am Ende aufgerufen werden (paarweise mit *LlJobOpen()* oder *LlJobOpenLCID()*), d.h. nach Benutzung der List & Label-DLL oder bei Beendigung Ihres Programms.

Beispiel:

```
HLLJOB hJob;

hJob = LlJobOpen(1);
LlDefineVariableStart(hJob);
LlDefineVariable(hJob, "Name", "Normalverbraucher");
LlDefineVariable(hJob, "Vorname", "Otto");
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LlJobOpen, LlJobOpenLCID

LlJobOpen

Syntax:

```
HLLJOB LlJobOpen(INT nLanguage);
```

Aufgabe:

Öffnet den DLL-Job. Fast alle DLL-Befehle benötigen den Rückgabewert dieser Funktion als Parameter.

Parameter:

nLanguage: gewählte Sprache für Benutzer-Interaktionen

Wert	Bedeutung
CMBTLANG_DEFAULT	im System voreingestellte Sprache
CMBTLANG_GERMAN	deutsch
CMBTLANG_ENGLISH	englisch

Weitere Konstanten in den Deklarationsdateien.

wenn dieser Parameter mit dem Wert *LL_JOBOPENFLAG_NOLLXPRELOAD* ver"odert" wird, werden die List & Label-Extensions nicht (vor-)geladen.

Rückgabewert:

Ein Handle, das bei den meisten Funktionen als Parameter benötigt wird, um auf die applikationsspezifischen Daten zugreifen zu können.

Ein gültiger Wert ist größer als 0.

Hinweise:

Aus Übersichtsgründen empfehlen wir, globale Einstellungen, die für alle List & Label-Aufrufe gelten sollen, ein einziges Mal nach *LlJobOpen()* zu tätigen (z.B. Dialogdesign oder Callback-Modi).

Die C?LL20-DLL benötigt die sprachabhängigen Teile in einer separaten DLL, z.B. C?LL2000.LNG oder C?LL2001.LNG, die je nach Sprach-Einstellung benutzt werden. Lesen Sie hierzu auch Kapitel 1.

Wenn List & Label nicht mehr benötigt wird, sollte der Job über die Funktion *LlJobClose()* wieder freigegeben werden, um der DLL eine Chance zu geben, die internen Variablen zu diesem Job freigeben zu können.

Viele weitere Sprachen sind als Sprachmodule zusätzlich erhältlich. Fragen Sie unseren Vertrieb.

Beispiel:

```
HLLJOB hJob;

hJob = LlJobOpen(CMBTLANG_GERMAN);
LlDefineVariableStart(hJob);
LlDefineVariable(hJob, "Name", "Normalverbraucher");
LlDefineVariable(hJob, "Vorname", "Otto");
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LlJobOpenLCID, LlJobClose, LlSetOption, LlSetOptionString

LlJobOpenLCID

Syntax:

```
HLLJOB LlJobOpenLCID(_LCID nLCID);
```

Aufgabe:

Siehe *LlJobOpen()*.

Parameter:

nLCID: Windows-Locale-ID gilt nur für das Benutzerinterface

Rückgabewert:

Siehe *LlJobOpen()*.

Hinweise:

Ruft *LlJobOpen()* auf mit der zur LCID gehörigen CMBTLANG_...-Konstanten.

Beispiel:

```
HLLJOB hJob;

hJob = LlJobOpenLCID(LOCALE_USER_DEFAULT);
```

```
LlDefineVariableStart(hJob);
LlDefineVariable(hJob, "Name", "Normalverbraucher");
LlDefineVariable(hJob, "Vorname", "Otto");
<... etc ...>
LlJobClose(hJob);
```

Siehe auch:

LlJobOpen, LlJobClose, LL_OPTION_LCID, LL_OPTION_CODEPAGE, WIN-API: GetACP

LlJobStateRestore

Syntax:

```
INT LlJobStateRestore(HLLJOB hLlJob, _PISTREAM pStream,UINT nFlags);
```

Aufgabe:

Wird z.B. vom .NET DesignerControl verwendet, um den Status eines Jobs auf einem Rechner auf einem anderen Rechner wiederherzustellen. Dabei werden je nach übergebenen Flags unter anderem die Variablen- und Feldlisten und die Datenbankstruktur aus dem Stream deserialisiert, so dass ein anschließender Aufruf von *LlDefineLayout()* die aus dem Stream gelesenen Strukturen im Designer anbietet.

Parameter:

hJob: List & Label Job-Handle

pStream: Stream der von einem vorherigen Aufruf von *LlJobStateSave()* geschrieben wurde. Das Format des Streams ist proprietär und kann jederzeit geändert werden.

nFlags: Kombination aus LL_JOBSTATEFLAG_...-Werten. Diese bestimmen, welche Werte aus dem Stream gelesen werden sollen (Variablen, Felder, Chartfelder, Datenbankstruktur, Übersetzungstabellen, allgemeine Jobeinstellungen). Um alle vorhandenen Werte zu deserialisieren, verwenden Sie LL_JOBSTATEFLAG_ALL.

Rückgabewert:

Fehlercode

Siehe auch:

LlJobStateSave

LlJobStateSave

Syntax:

```
INT LlJobStateRestore(HLLJOB hLlJob, _PISTREAM pStream,UINT nFlags,
 bool bPacked);
```

Aufgabe:

Wird z.B. vom .NET DesignerControl verwendet, um den Status eines Jobs auf einem Rechner zu speichern. Dabei werden je nach übergebenen Flags unter anderem die Variablen- und Feldlisten und die Datenbankstruktur in den Stream serialisiert.

Parameter:

hJob: List & Label Job-Handle

pStream: Stream in den geschrieben wird. Das Format des Streams ist proprietär und kann jederzeit geändert werden.

nFlags: Kombination aus LL_JOBSTATEFLAG_...-Werten. Diese bestimmen, welche Werte in den Stream geschrieben werden sollen (Variablen, Felder, Chartfelder, Datenbankstruktur, Übersetzungstabellen, allgemeine Jobeinstellungen). Um alle vorhandenen Werte zu serialisieren, verwenden Sie LL_JOBSTATEFLAG_ALL.

bPacked: Bestimmt, ob der Inhalt des Streams komprimiert werden soll.

Rückgabewert:

Fehlercode

Siehe auch:

LlJobStateRestore

LlLocAddDesignLCID

Syntax:

```
INT LlLocAddDesignLCID(HLLJOB hJob, LCID nLCID);
```

Aufgabe:

Fügt eine Sprache für das Projekt-Design hinzu. Für die angemeldeten Sprachen können dann über *LlLocAddDictionaryEntry()* Übersetzungen vorgenommen werden.

Parameter:

hJob: List & Label Job-Handle

nLCID: Locale-ID. Die erste übergebene Locale-ID wird im Designer als Basissprache verwendet. In dieser Sprache sollten über *LlLocAddDictionaryEntry()* die Wörterbuchschlüssel übergeben werden. Wird 0 übergeben, so werden alle vorhandenen Sprachen entfernt.

Rückgabewert:

Fehlercode

Siehe auch:

LlLocAddDictionaryEntry

LlLocAddDictionaryEntry

Syntax:

```
INT LlLocAddDictionaryEntry(HLLJOB hJob, LCID nLCID, LPCTSTR pszKey,
   LPCTSTR pszValue, UINT nType);
```

Aufgabe:

Fügt einen Eintrag zu einem der Wörterbücher hinzu. Die Wörterbücher erlauben das Lokalisieren von Projekten bzw. Designer-Elementen.

Parameter:

hJob: List & Label Job-Handle

nLCID: Locale-ID, für die der Eintrag hinzugefügt wird. Diese muss zuvor über *LlLocAddDesignLCID()* angemeldet worden sein.

pszKey: Schlüssel für das Wörterbuch (Originaltext in der Basissprache).

pszValue: Wert für das Wörterbuch (Übersetzung).

nType: Wörterbuchtyp.

Wert	Bedeutung
LL_DICTIONARY_TYPE_STATIC	Statischer (fester) Text
LL_DICTIONARY_TYPE_IDENTIFIER	Feld oder Variablenname
LL_DICTIONARY_TYPE_TABLE	Tabellenname
LL_DICTIONARY_TYPE_RELATION	Relationsname
LL_DICTIONARY_TYPE_SORTORDER	Sortierungsname

Rückgabewert:

Fehlercode

Hinweise:

Verwenden Sie diese Funktion, um sprachübergreifend mit dem gleichen Projekt arbeiten zu können. Sobald über *LlLocAddDesignLCID()* mehrere Locale-IDs angemeldet wurden, steht in der Toolbar des Designers eine neue Schaltfläche zur Sprachwahl zur Verfügung. *LL_DICTIONARY_TYPE_STATIC* erlaubt es, mit Hilfe der Translate$-Designerfunktion feste Texte zu lokalisieren, für diese wird im Designer dann Intellisense-Eingabeunterstützung gegeben. Sie können innerhalb der statischen Texte bis zu 3 Formatierungsplatzhalter verwenden, die mit {0}, {1} bzw. {2} bezeichnet werden können.

Für die Ausgabe zur Druckzeit wird als Voreinstellung die threadspezifische Locale-ID (in der Regel die Systemsprache) verwendet. Wenn Sie für den Druck eine bestimmte Voreinstellung vorgeben möchten, verwenden Sie *LL_OPTION_LCID*. Diese Voreinstellung kann durch das Design überschrieben werden, wenn im Designer eine entsprechende Projekteinstellung vorgenommen wurde.

Übergeben Sie NULL für *pszKey* und *pszValue*, dann werden alle Wörterbucheinträge für alle Wörterbücher gelöscht.

Beispiel:

```
HLLJOB hJob;

hJob = LlJobOpen(CMBTLANG_DEFAULT);

// Sprachen hinzufügen
LlLocAddDesignLCID(hJob, 7); // Deutsch als Basissprache
LlLocAddDesignLCID(hJob, 9); // Englisch als Übersetzungssprache

// Übersetzungen hinzufügen
LlLocAddDictionaryEntry(hJob, 9, "Artikelnummer", "ArticleNumber",
   LL_DICTIONARY_TYPE_IDENTIFIER);
LlLocAddDictionaryEntry(hJob, 9, "Preis", "Price",
   LL_DICTIONARY_TYPE_IDENTIFIER);
LlLocAddDictionaryEntry(hJob, 9, "Seite {0} von {1}", "Page {0} of {1}",
   LL_DICTIONARY_TYPE_STATIC);

LlDefineVariableStart(hJob);
LlDefineVariable(hJob, "Artikelnummer", "12345");
LlDefineVariable(hJob, "Preis", "123");

// Designeraufruf usw.
...
LlJobClose(hJob);
```

Siehe auch:

LlLocAddDesignLCID, LL_OPTION_LCID

LlPreviewDeleteFiles

Syntax:

```
INT LlPreviewDeleteFiles(HLLJOB hJob, LPCTSTR lpszObjName,
   LPCTSTR lpszPath);
```

Aufgabe:

Löscht die beim Vorschaudruck angelegten Preview-Dateien.

Parameter:

hJob: List & Label Job-Handle

lpszObjName: gültiger Projektdateiname ohne Pfadangabe

lpszPath: gültiger Pfad der Preview-Dateien mit abschließendem Backslash "\"

Rückgabewert:

Fehlercode

Hinweise:

Sollte immer nach einem *LlPreviewDisplay()* aufgerufen werden, da die erstellten Vorschaudateien im Allgemeinen nur momentane Gültigkeit haben.

Sollte natürlich nicht (sofort) aufgerufen werden, wenn Sie die Vorschaudateien archivieren oder später (z.B. im Hintergrund) drucken möchten.

Siehe auch:

LlPreviewDisplay, LlPreviewDisplayEx, LlPreviewSetTempPath

LlPreviewDisplay

Syntax:

```
INT LlPreviewDisplay(HLLJOB hJob, LPCTSTR lpszObjName,
   LPCTSTR lpszPath, HWND hWnd);
```

Aufgabe:

Startet das separat aufzurufende Preview-Fenster für eine Vorschau.

Parameter:

hJob: List & Label Job-Handle

lpszObjName: gültiger Projektdateiname ohne Pfadangabe

lpszPath: gültiger Pfad der Vorschau-Dateien mit abschließendem Backslash "\"

hWnd: Fensterhandle des aufrufenden Programms

Rückgabewert:

Fehlercode

Hinweise:

Wenn Pfad NULL oder "" ist, wird der Pfad genommen, in dem die Definitionsdatei gespeichert ist.

LlPreviewDisplay() ruft intern *LlPreviewDisplayEx()* mit *LL_PRVOPT_PRN_ASKPRINTERIFNEEDED* auf.

Siehe auch:

LlPrintStart, LlPrintWithBoxStart, LlPreviewDeleteFiles, LlPreviewSetTempPath, LlPreviewDisplayEx

LlPreviewDisplayEx

Syntax:

```
INT LlPreviewDisplayEx(HLLJOB hJob, LPCTSTR lpszObjName,
   LPCTSTR lpszPath, HWND hWnd, UINT nOptions, LPVOID pOptions);
```

Aufgabe:

Startet das separate Preview-Fenster für eine Vorschau mit der Möglichkeit spezielle Druckerkonfigurationen vorzugeben.

Parameter:

hJob: List & Label Job-Handle

lpszObjName: gültiger Projektdateiname mit Pfadangabe

lpszPath: gültiger Pfad der Preview-Dateien mit abschließendem Backslash "\"

hWnd: Fensterhandle des aufrufenden Programms

nOptions:

Wert	Bedeutung
LL_PRVOPT_PRN_USEDEFAULT	Der Preview benutzt den Default-Drucker als Ausgabemedium
LL_PRVOPT_PRN_ASKPRINTERIFNEEDED	Der Preview versucht, die in dem Druckfile gespeicherten Drucker zu finden. Wenn diese nicht gefunden werden, kann der Benutzer die Drucker wählen
LL_PRVOPT_PRN_ASKPRINTERALWAYS	Es erscheint in jedem Fall ein Druckerauswahl-Dialog, in dem der Benutzer seine Druckerzuordnung bestimmen kann

pOptions: Optionen für zukünftige Versionen, jetzt NULL oder ""!

Rückgabewert:

Fehlercode

Hinweise:

Unabhängig vom Designer aufzurufendes Fenster mit der Möglichkeit, eine Druckvorschau zu realisieren.

Wenn Pfad NULL oder "" ist, wird der Pfad genommen, in dem die Definitionsdatei gespeichert ist.

Beispiel:

Siehe Programmbeispiel im Kapitel "Programmierung"

Siehe auch:

LlPreviewDeleteFiles, LlPreviewSetTempPath, LlPreviewDisplay

LlPreviewSetTempPath

Syntax:

```
INT LlPreviewSetTempPath(HLLJOB hJob, LPCTSTR lpszPath);
```

Aufgabe:

Setzt einen Pfad für die Druckvorschau-Datei(en).

Parameter:

hJob: List & Label Job-Handle

lpszPath: gültige Pfadangabe mit abschließendem Backslash "\"

Rückgabewert:

Fehlercode

Hinweise:

Wenn Pfad NULL oder "" ist, wird der Pfad genommen, in dem die Definitionsdatei gespeichert ist. In diesem Pfad wird die Vorschaudatei gespeichert. Der Dateiname ist immer der Projektname, die Dateiendung ist immer ".LL". Die Vorschaudateien liegen im Storage-Format vor und können so gegebenenfalls weiterverarbeitet oder zu Archivierungszwecken gespeichert werden.

Dieser Befehl muss VOR dem ersten *LlPrint()* aufgerufen werden.

Siehe auch:

LlPrintStart, LlPrintWithBoxStart, LlPreviewDisplay, LlPreviewDisplayEx

LlPrint

Syntax:

```
INT LlPrint(HLLJOB hJob);
```

Aufgabe:

Ausgabe aller Objekte auf dem Drucker.

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Fehlercode

Hinweise:

Es werden alle Objekte ausgegeben, von einem Tabellenobjekt die Kopfzeile (siehe *LL_OPTION_DELAYTABLEHEADER*). Eine Tabelle muss danach mit *LlPrintFields()*-Aufrufen gefüllt werden. Durch *LlPrint()* wird der Seitenvorschub ausgelöst.

Karteikarten/Etikettenprojekte: So lange *LlPrint() LL_WRN_REPEAT_DATA* zurückliefert, muss es noch einmal aufgerufen werden, da sich dann durch einen Umbruch das Objekt auf einer neuen Seite fortsetzt.

Beispiel:

Siehe Programmbeispiel im Kapitel "Programmierung"

Siehe auch:

LlPrintFields, LlPrintEnableObject

LlPrintAbort

Syntax:

```
INT LlPrintAbort(HLLJOB hJob);
```

Aufgabe:

Beendet Ausdruck (evtl. unvollendete Seite bleibt unvollendet oder wird evtl. nicht ausgedruckt).

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Fehlercodes

Hinweise:

Wird benötigt, um den Druck durch das Programm abzubrechen, wenn nicht *LlPrintWithBoxStart()* verwendet wird.

Der Unterschied zum ´normalen´ Ende, d.h. dazu, nicht mehr *LlPrint()* oder *LlPrintFields()* aufzurufen, liegt darin, dass noch im Druckertreiber stehende Daten verworfen werden, so dass der Druck möglicherweise mitten auf einer Seite beendet wird.

Danach folgende *LlPrint....()*-Aufrufe geben als Rückgabewert *LL_USER_ABORTED* zurück.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

if (LlPrintStart(hJob, LL_PROJECT_LABEL, "test", LL_PRINT_NORMAL) == 0)
{
   for all data records
   {<... etc...>
      if (bDataError)
         LlPrintAbort(hJob);
   }
   LlPrintEnd(hJob);
}
```

```
else
   MessageBox(NULL, "Fehler", "List & Label",MB_OK);
LlJobClose(hJob);
```

Siehe auch:

LlPrintStart, LlPrintWithBoxStart, LlPrintEnd

LlPrintCopyPrinterConfiguration

Syntax:

```
INT LlPrintCopyPrinterConfiguration(HLLJOB hJob, LPCTSTR lpszFilename,
   INT nFunction);
```

Aufgabe:

Ermöglicht die Speicherung bzw. das Restaurieren der Druckerkonfiguration.

Parameter:

hJob: List & Label Job-Handle

lpszFilename: Dateiname der Druckerkonfiguration (P-Datei)

nFunction: auszuführende Aktion.

Aktion	Bedeutung
LL_PRINTERCONFIG_SAVE	speichert die Druckerkonfiguration des zum Druck geöffneten Projekts in eine andere Datei
LL_PRINTERCONFIG_-RESTORE	kopiert die durch *LL_PRINTERCONFIG_SAVE* gespeicherte Druckerkonfiguration wieder zu dem aktuellen geöffneten Druckprojekt.

Rückgabewert:

Fehlercode (immer 0)

Hinweise:

***LL_PRINTERCONFIG_RESTORE* muss nach *LlPrint[WithBox]Start()* und vor *LlPrint()* aufgerufen werden!**

Beispiel:

Folgendes Prinzip sollte z.B. bei selbsterstellten Kopien auf temporär geänderten Drucker benutzt werden (da ansonsten nur die erste Kopie auf den geänderten Drucker geschrieben wird):

```
for each copy
{
   LlPrintWithBoxStart(...)
   if (erste Kopie)
   {
      LlPrintOptionsDialog(...);
      LlPrintCopyPrinterConfiguration("curconfig.~~~",
         LL_PRINTERCONFIG_SAVE);
   }
```

```
   else
   {
      LlPrintCopyPrinterConfiguration("curconfig.~~~",
         LL_PRINTERCONFIG_RESTORE);
   }
   .. LlPrint(), LlPrintFields(), ...
}
```

Siehe auch:

LIPrintStart, LIPrintWithBoxStart, LISetPrinterToDefault, LIPrintStart, LIPrintWithBoxStart, LISetPrinterInPrinterFile, LIGetPrinterFromPrinterFile, LISetPrinterDefaultsDir

LIPrintDbGetRootTableCount

Syntax:

```
INT LlPrintDbGetRootTableCount(HLLJOB hJob);
```

Aufgabe:

Liefert die Anzahl der Tabellen in der obersten Hierarchieebene zurück. Dieser Wert kann verwendet werden, um eine Fortschrittsanzeige zu realisieren.

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Anzahl der Tabellen

Hinweise:

Beachten Sie die Hinweise im Kapitel "Drucken relationaler Daten".

Siehe auch:

LIDbAddTable, LIDbAddTableRelation, LIDbAddTableSortOrder, LIPrintDbGetCurrentTable, LIPrintDbGetCurrentTableSortOrder, LIPrintDbGetCurrentTableRelation

LIPrintDbGetCurrentTable

Syntax:

```
INT LlPrintDbGetCurrentTable(HLLJOB hJob, LPTSTR pszTableID,
   UINT nTableIDLength, BOOL bCompletePath);
```

Aufgabe:

Fragt die ID der aktuell zu druckenden Tabelle ab.

Parameter:

hJob: List & Label Job-Handle

pszTableID: Puffer für Rückgabewert

nTableIDLength: Größe des Puffers

bCompletePath: Bestimmt, ob die Tabellen-ID mit allen Hierarchestufen (z.B. "Orders > OrderDetails") oder nur die Tabellen-ID (z.B. "OrderDetails") zurückgeliefert wird.

Rückgabewert:

Fehlercode

Hinweise:

Beachten Sie die Hinweise im Kapitel "Drucken relationaler Daten".

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LIDbAddTable, LIDbAddTableRelation, LIDbAddTableSortOrder, LIPrintDbGetCurrentTableSortOrder, LIPrintDbGetCurrentTableRelation

LIPrintDbGetCurrentTableFilter

Syntax:

```
INT LlPrintDbGetCurrentTableFilter(HLLJOB hJob,
   PVARIANT pvFilter, PVARIANT pvParams);
```

Aufgabe:

Diese Funktion gibt den aktuellen Tabellenfilter in der nativen Syntax der Datenquelle zurück. Die Übersetzung muss dabei mit dem *LL_QUERY_EXPR2HOSTEXPRESSION* Callback durchgeführt werden. Dieser Callback wird für jeden Teil des Filterausdrucks, der im Designer verwendet wird, ausgelöst.

Parameter:

hJob: List & Label Job-Handle

pvFilter: Dieser Parameter bekommt den übersetzten Filterausdruck. Wie bei VARIANTs gewohnt muss dieser vor (VariantInit()) und nach der Verwendung (VariantClear()) freigegeben werden.

pvParams: Wenn der Filterausdruck Parameter verwendet (siehe Callback Dokumentation), bekommt dieses Argument ein VARIANTARRAY mit den Parameterwerten. Wie bei VARIANTs gewohnt muss dieser vor (VariantInit()) und nach der Verwendung (VariantClear()) freigegeben werden.

Rückgabewert:

Fehlercode

Siehe auch:

LIDbAddTable, LL_QUERY_EXPR2HOSTEXPRESSION

LlPrintDbGetCurrentTableRelation

Syntax:

```
INT LlPrintDbGetCurrentTableRelation(HLLJOB hJob, LPTSTR pszRelationID,
   UINT nRelationIDLength);
```

Aufgabe:

Fragt die ID der aktuell zu druckenden Unterrelation ab.

Parameter:

hJob: List & Label Job-Handle

pszRelationID: Puffer für Rückgabewert

nRelationIDLength: Größe des Puffers

Rückgabewert:

Fehlercode

Hinweise:

Beachten Sie die Hinweise im Kapitel "Drucken relationaler Daten".

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LIDbAddTable, LIDbAddTableRelation, LIDbAddTableSortOrder, LlPrintDbGetCurrentTable, LlPrintDbGetCurrentTableSortOrder

LlPrintDbGetCurrentTableSortOrder

Syntax:

```
INT LlPrintDbGetCurrentTableSortOrder(HLLJOB hJob,
   LPTSTR pszSortOrderID, UINT nSortOrderIDLength);
```

Aufgabe:

Fragt die ID der aktuell zu druckenden Sortierung ab.Wenn mehrfache Sortierungen unterstützt werden (s. *LlDbAddTableEx()),* wird eine tabulatorgetrennte Liste von Sortierungen zurückgeliefert.

Parameter:

hJob: List & Label Job-Handle

pszSortOrderID: Puffer für Rückgabewert

nSortOrderIDLength: Größe des Puffers

Rückgabewert:

Fehlercode

Hinweise:

Beachten Sie die Hinweise im Kapitel "Drucken relationaler Daten".

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlDbAddTable, LlDbAddTableRelation, LlDbAddTableSortOrder, LlPrintDbGetCurentTable, LlPrintDbGetCurrentTableRelation

LlPrintDeclareChartRow

Syntax:

```
INT LlPrintDeclareChartRow(HLLJOB hJob, UINT nFlags);
```

Aufgabe:

Diese Funktion teilt den im Projekt enthaltenen Chart-Objekten mit, dass Werte für diese bereitstehen.

Parameter:

hJob: List & Label Job-Handle

nFlags: bestimmt, für welchen Typ von Chart-Objekten Werte zur Verfügung stehen.

Rückgabewert:

Fehlercode

Hinweise:

Ein oder mehrere der folgenden Flags können für den *nFlags* Parameter benutzt werden:

LL_DECLARECHARTROW_FOR_OBJECTS: teilt Chart-Objekten mit, dass sie sich die Daten speichern sollen

LL_DECLARECHARTROW_FOR_TABLECOLUMNS: teilt Charts in Tabellenspalten mit, dass sie sich die Daten speichern sollen

Bitte beachten Sie die Hinweise im Kapitel "Ansteuerung von Chart- und Kreuztabellen-Objekten".

Durch diesen Befehl werden Charts nicht gedruckt, sondern es wird Ihnen nur mitgeteilt, dass neue Daten für sie zur Verfügung stehen. Erst bei dem Befehl *LlPrint()* (für Chart-Objekte) bzw. dem Befehl *LlPrintFields()* (für Charts in Tabellen) werden diese ausgegeben.

Beispiel:

```
// while data to put into chart object...
   ... LlDefineChartFieldExt(...);
   LlPrintDeclareChartRow(hJob, LL_DECLARECHARTROW_FOR_OBJECTS);
// now print chart object
ret = LlPrint();
```

Siehe auch:

LlDefineChartFieldExt, LlDefineChartFieldStart

LlPrintDidMatchFilter

Syntax:

```
INT LlPrintDidMatchFilter(HLLJOB hJob);
```

Aufgabe:

Gibt an, ob der zuletzt gedruckte Datensatz dem vom Benutzer eingegebenen Filter (aus dem Designer) entsprochen hat, also wirklich ausgedruckt wurde.

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Fehlercode (wenn negativ), 0: wurde nicht ausgedruckt, 1: wurde ausgedruckt

Hinweise:

Diese Funktion kann erst nach *LlPrint()* bzw. *LlPrintFields()* aufgerufen werden.

Beispiel:

```
ret = LlPrint();
if (ret == 0 && LlPrintDidMatchfilter(hJob))
   ++nCountOfPrintedRecords;
```

Siehe auch:

LlPrintGetFilterExpression, LlPrintWillMatchFilter, Callback LL_NTFY_FAILS_FILTER

LlPrintEnableObject

Syntax:

```
INT LlPrintEnableObject(HLLJOB hJob, LPCTSTR lpszObject, BOOL bEnable);
```

Aufgabe:

Ermöglicht Druck des Objekts oder verhindert diesen, indem das Objekt übergangen wird.

Parameter:

hJob: List & Label Job-Handle

lpszObject: Objektname, siehe Hinweise

bEnable: TRUE: Objekt druckbar, FALSE: Objekt wird übergangen

Rückgabewert:

Fehlercode (wichtig!)

Hinweise:

Der Objektname kann " (also leer) sein, um alle Objekte anzusprechen, ansonsten muss es der vom Benutzer eingegebene Name des Objekts sein, mit einem ':' davor.

Wenn der Benutzer im Designer Objekte und Objektnamen verändern kann, ist es wichtig, den Rückgabewert abzufragen, um testen zu können, ob es das Objekt überhaupt gibt!

Besonders wichtig ist diese Funktion für das Auffüllen mehrerer unabhängiger Tabellen. Vor *LlPrint()* müssen auf jeden Fall alle Tabellen enabled sein.

Beispiel:

```
LlPrintEnableObject(hJob, "", TRUE);
LlPrintEnableObject(hJob, ":AuthorList", FALSE);
```

Siehe auch:

LlPrint, LlPrintFields

LlPrintEnd

Syntax:

```
INT LlPrintEnd(HLLJOB hJob, INT nPages);
```

Aufgabe:

Beendet den Druckjob.

Parameter:

hJob: List & Label Job-Handle

nPages: Zahl der gewünschten Seitenvorschübe nach dem Druck

Rückgabewert:

Fehlercode

Hinweise:

Der Druckjob wird beendet und der Drucker-Device-Kontext geschlossen. Falls benötigt, werden Seitenvorschübe an den Ausdruck angehängt.

Benutzen Sie immer *LlPrintEnd()*, wenn Sie ein *LlPrintStart()* oder *LlPrintWithBoxStart()* verwendet haben, auch wenn diese Befehle mit einem Fehler abgebrochen wurden, ansonsten kann es zu Speicherverlusten kommen.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

if (LlPrintStart(hJob, LL_PROJECT_LABEL, "test", LL_PRINT_NORMAL) == 0)
{
   <... etc...>
   LlPrintEnd(hJob,0);
}
else
   MessageBox(NULL, "Fehler", "List & Label", MB_OK);
LlJobClose(hJob);
```

Siehe auch:

LlPrintStart, LlPrintWithBoxStart

LlPrinterSetup

Syntax:

```
INT LlPrinterSetup(HLLJOB hJob, HWND hWnd, UINT nObjType,
   LPCTSTR lpszObjName);
```

Aufgabe:

Öffnet ein Druckerauswahlfenster, und speichert die Benutzerauswahl in die Definitionsdatei.

Parameter:

hJob: List & Label Job-Handle

hWnd: Window-Handle des aufrufenden Programms

nObjType:

Wert	Bedeutung
LL_PROJECT_LABEL	Etiketten
LL_PROJECT_LIST	Listen
LL_PROJECT_CARD	Karteikarten

lpszObjName: Dateiname

Rückgabewert:

Fehlercode

Hinweise:

Einfachere Variante von *LlPrintOptionsDialog()*. Diese Funktion muss vor *LlPrint[WithBox]Start()* aufgerufen werden, da sie die Druckerdefinitionsdatei direkt verändert.

Von der Verwendung dieser Funktion wird abgeraten, *LlPrintOptionsDialog()* ist der Dialog der ersten Wahl.

Siehe auch:

LlPrintStart, LlPrintWithBoxStart, LlPrintOptionsDialog, LlPrintOptionsDialogTitle, LlPrintGetPrinterInfo

LlPrintFields

Syntax:

```
INT LlPrintFields(HLLJOB hJob);
```

Aufgabe:

Ausgabe einer Tabellenzeile.

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Fehlercode oder Anweisung

Hinweise:

Mit *LlPrintFields()* wird in allen Tabellen eine Tabellenzeile mit den zum Zeitpunkt des Aufrufs definierten Feldern ausgefüllt.

Der Rückgabewert *LL_WRN_REPEAT_DATA* informiert darüber, dass für diesen Datensatz eine neue Seite angefangen werden muss, da der schon definierte Datensatz nicht mehr auf die Seite passen würde. Bei dem dann für die nächste Seite folgenden *LlPrint()* darf dementsprechend nicht auf den nächsten Datensatz gewechselt werden.

Wenn Sie über *LlDbAddTable()* mehrere Tabellen zum Design anbieten, kann der Rückgabewert auch *LL_WRN_TABLECHANGE* sein. Dies bedeutet, dass der Benutzer eine Untertabelle platziert hat, die gedruckt werden soll. Sie können dann über *LlPrintDbGetCurrentTable()* erfragen, welche Tabelle zu drucken ist. Beachten Sie auch die Hinweise im Kapitel "Drucken relationaler Daten".

Beispiel:

Siehe Programmbeispiel im Kapitel "Programmierung"

Siehe auch:

LIPrint, LIPrintEnableObject

LlPrintFieldsEnd

Syntax:

```
INT LlPrintFieldsEnd(HLLJOB hJob);
```

Aufgabe:

Bewirkt den Druck bzw. den Versuch die Fußzeile der letzten Seite und angehängte Objekte zu drucken.

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Fehlercode oder Anweisung

Hinweise:

Nur nötig bei Listen-Projekten!

Wird benötigt, um sicherzustellen, dass die Fußzeile auch gedruckt werden kann, auch wenn keine anderen Daten auf der Seite vorhanden sind.

Solange der Rückgabewert *LL_WRN_REPEAT_DATA* ist, konnten die Fußzeile oder angehängte Objekte nicht mehr auf die letzte Seite gedruckt werden. *LlPrintFieldsEnd()* muss dann ein weiteres Mal aufgerufen werden, um die Fußzeile oder die angehängten Objekte dann auf einer eigenen Seite auszugeben. Ab diesem Zeitpunkt liefert *LastPage()* aus dem Designer TRUE.

Wenn Sie über *LlDbAddTable()* mehrere Tabellen zum Design anbieten, kann der Rückgabewert auch *LL_WRN_TABLECHANGE* sein. Dies bedeutet, dass der Benutzer auf derselben Hierarchie-Ebene eine weitere Tabelle platziert hat, die gedruckt werden soll. Sie können dann über *LlPrintDbGetCurrentTable()* erfragen, welche Tabelle zu drucken ist. Sie auch die Hinweise im Kapitel "Drucken relationaler Daten".

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);
if (LlPrintStart(hJob, LL_PROJECT_LIST, "test", LL_PRINT_NORMAL) == 0)
{
   <... etc...>
   <Daten fertig>
   while (LlPrintFieldsEnd(hJob) == LL_WRN_REPEAT_DATA)
   {
      <Evtl. Variablen definieren>
      // Benutzer sollte abbrechen können:
      LlPrintUpdateBox(hJob);
   }
```

```
   LlPrintEnd(hJob,0);
}
else
   MessageBox(NULL, "Fehler", "List & Label", MB_OK);
LlJobClose(hJob);
```

Siehe auch:

LlPrintEnd

LlPrintGetChartObjectCount

Syntax:

```
INT LlPrintGetChartObjectCount(HLLJOB hJob, UINT nType);
```

Aufgabe:

Abfrage der Anzahl der im Projekt vorhandenen Charts mit der angegebenen Platzierung.

Parameter:

hJob: List & Label Job-Handle

nType: Ort des abzufragenden Charts

Rückgabewert:

Fehlercode oder Anzahl der Charts

Hinweise:

Einer der folgenden Werte muss für nType angegeben werden:

LL_GETCHARTOBJECTCOUNT_CHARTOBJECTS: Gibt die Anzahl der Chart-Objekte zurück. Dieses beinhaltet nicht Tabellen, die Charts enthalten.

LL_GETCHARTOBJECTCOUNT_CHARTOBJECTS_BEFORE_TABLE: Gibt die Anzahl der Chart-Objekte zurück, die sich in der Druckreihenfolge vor Tabellen befinden.

LL_GETCHARTOBJECTCOUNT_CHARTCOLUMNS: Gibt die Anzahl der Charts in Tabellenspalten zurück.

Diese Funktion kann dazu verwendet werden, die Druckschleife zu optimieren. Weitere Hinweise hierzu sowie die Anwendung finden Sie im Kapitel "Ansteuerung von Chart- und Kreuztabellen-Objekten".

Siehe auch:

LlPrint

LlPrintGetCurrentPage

Syntax:

```
INT LlPrintGetCurrentPage(HLLJOB hJob);
```

Aufgabe:

Abfrage der momentanen Seitennummer.

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Fehlercode oder Seitennummer

Hinweise:

Die Seitennummer entspricht dem Wert, der von *Page()* im Designer zurückgegeben wird.

Siehe auch:

LlPrint

LlPrintGetFilterExpression

Syntax:

```
INT LlPrintGetFilterExpression(HLLJOB hJob, LPTSTR pszBuffer,
   UINT nBufSize);
```

Aufgabe:

Fragt die vom Benutzer in dem Projekt (Designer) eingegebene Filterbedingung ab.

Parameter:

hJob: List & Label Job-Handle

pszBuffer: Puffer für Rückgabewert

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode

Hinweise:

Diese Funktion kann erst nach dem Einlesen eines Projekts, also nach *LlPrint[WithBox]Start()* benutzt werden.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlPrintWillMatchFilter, LlPrintDidMatchFilter

LlPrintGetItemsPerPage

Syntax:

```
INT LlPrintGetItemsPerPage(HLLJOB hJob);
```

Aufgabe:

Gibt die Zahl der Etiketten einer Seite zurück (Spaltenzahl * Zeilenzahl) entsprechend den Einstellungen des Projekts.

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Zahl der Etiketten oder Fehlercode (negativ).

Hinweise:

Bei *LL_PROJECT_LIST*-Objekten wird immer 1 zurückgegeben.

Kann zur Berechnung der Ausgabe-Gesamtseitenzahl verwendet werden.

LlPrintGetOption

Syntax:

```
INT LlPrintGetOption (HLLJOB hJob, INT nIndex);
```

Aufgabe:

Gibt verschiedene Druckoptionen zurück, die z.B. bei dem Aufruf von *LlPrintOptionsDialog()* vom Benutzer gesetzt wurden.

Parameter:

hJob: List & Label Job-Handle

nIndex: s.u.

Rückgabewert:

vom Benutzer gewählte Einstellung

Hinweise:

Zusätzlich zu den bei *LlPrintSetOption()* einstellbaren Werten gibt es noch weitere Optionen:

LL_PRNOPT_COPIES_SUPPORTED

Gibt zurück, ob die über *LL_PRNOPT_COPIES* eingestellte Kopienzahl durch den Drucker selbst unterstützt wird. Dies macht üblicherweise nur bei Listenprojekten Sinn.

Wichtig: Diese Abfrage bewirkt gleichzeitig, dass - sofern der Drucker Kopien unterstützt - die über *LL_PRNOPT_COPIES* eingestellten Kopien vom Drucker übernommen werden.

Wenn nicht, muss *LL_PRNOPT_COPIES* auf 1 gesetzt werden und die Kopien dann evtl. "von Hand" gedruckt werden.

LL_PRNOPT_DEFPRINTERINSTALLED

Gibt zurück, ob es im Betriebssystem einen Default-Drucker gibt.

LL_PRNOPT_JOBID

Abgefragt nach *LlPrint()* gibt diese Option die Druckjobnummer des Spoolers zurück. Wenn mehrere Drucker im Projekt definiert sind oder mehrere Druckjobs ausgegeben werden, können nach den *LlPrint()*-Aufrufen verschiedenen IDs zurückgeliefert werden, es sollte also nach jedem *LlPrint()* abgefragt werden.

Mit dieser ID kann man über Windows-API-Funktionen die tatsächliche Ausführung eines Druckjobs überwachen.

LL_PRNOPT_PRINTORDER

Gibt die gewählte Druckreihenfolge des Projekts zurück. Default: *LL_PRINTORDER_HORZ_LTRB*

LL_PRNOPT_UNIT

Gibt die verwendete Maßeinheit zurück, vorgegeben durch die Systemeinstellungen. Rückgabewerte sind eine der folgenden Konstanten:

Wert	Bedeutung
LL_UNITS_MM_DIV_10	1/10 mm
LL_UNITS_MM_DIV_100	1/100 mm (Voreinstellung auf metrischen Systemen)
LL_UNITS_MM_DIV_1000	1/1000 mm
LL_UNITS_INCH_DIV_100	1/100 inch
LL_UNITS_INCH_DIV_1000	1/1000 inch (Voreinstellung auf britischen Systemen)
LL_UNITS_SYSDEFAULT_LORES	Voreinstellung geringe Auflösung auf dem System
LL_UNITS_SYSDEFAULT_HIRES	Voreinstellung hohe Auflösung auf dem System
LL_UNITS_SYSDEFAULT	Voreinstellung auf dem System

LL_PRNOPT_USE2PASS

Liefert zurück, ob der Druck ein Two-Pass-Verfahren verwendet, d.h. ob im Projekt die Gesamtseitenzahl ausgeben wird.

Beispiel:

siehe Programmbeispiel bei *LlGetOption*

Siehe auch:

LlPrintSetOption, LlPrintOptionsDialog

LlPrintGetOptionString

Syntax:

```
INT LlPrintGetOptionString(HLLJOB hJob, INT nMode, LPTSTR pszBuffer,
   UINT nBufSize);
```

Aufgabe:

Fragt diverse Einstellungen in List & Label ab.

Parameter:

hJob: List & Label Job-Handle

nMode: Optionsindex

pszBuffer: Puffer für Rückgabewert

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode

Hinweise:

Die für den Mode-Parameter gültigen Werte sind bei *LlPrintSetOptionString()* aufgeführt.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlPrintSetOptionString

LlPrintGetPrinterInfo

Syntax:

```
INT LlPrintGetPrinterInfo(HLLJOB hJob, LPTSTR lpszPrn, UINT nPrnBufSize,
   LPTSTR lpszPort, UINT nPortBufSize);
```

Aufgabe:

Rückgabe von Informationen über den Zieldrucker des Druckprojekts

Parameter:

hJob: List & Label Job-Handle

lpszPrn: Puffer für Druckername

nPrnBufSize: Länge des durch *lpszPrn* bezeichneten Puffers

lpszPort: Puffer für Druckerport

nPortBufSize: Länge des durch *lpszPort* bezeichneten Puffers

Rückgabewert:

Fehlercode

Hinweise:

Beispiele für Druckernamen sind ´HP Deskjet 500´ oder ´NEC P6´, für Druckerport ´LPT2:´ oder ´\\server\printer1´.

Bei einem Export enthält der Druckername die Beschreibung des Exportmoduls und der Port ist leer.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlPrintStart, LlPrintWithBoxStart

LlPrintGetProjectParameter

Syntax:

```
INT LlPrintGetProjectParameter(HLLJOB hLlJob, LPCTSTR pszParameter,
   BOOL bEvaluated, LPTSTR pszBuffer, INT nBufSize, _LPUINT pnFlags)
```

Aufgabe:

Über diese Funktion kann man den Wert eines Projekt-Parameters abfragen

Parameter:

hJob: List & Label Job-Handle

pszParameter: Name des Parameters. Kann NULL sein (siehe Hinweise).

pszBuffer: Speicherbereich, in den der Parameter geschrieben werden soll. Kann NULL sein (siehe Hinweise)

bEvaluated: Gibt an, ob der Wert vor der Rückgabe berechnet werden soll oder nicht, falls der Parameter den Type *LL_PARAMETERFLAG_FORMULA* besitzt.

nBufSize: Größe des Pufferbereichs, auf den pszBuffer zeigt (in TCHARs).

Rückgabewert:

Fehlercode bzw. benötigte Puffergröße

Hinweise:

Diese Funktion kann erst nach *LlPrint[WithBox]Start()* aufgerufen werden!

Wenn *pszParameter* NULL ist, wird eine Semikolon-separierte Liste aller USER-Parameter zurückgegeben.

Wenn *pszBuffer* NULL ist, ist der Rückgabewert die Länge des benötigten Puffers (in TCHARS, also BYTEs im SBCS/MBCS-Fall und WCHARs bei UNICODE) inklusive der String-Terminierung.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlSetDefaultProjectParameter, LlGetDefaultProjectParameter, LlPrintGetProjectParameter

LlPrintIsChartFieldUsed

Syntax:

```
INT LlPrintIsChartFieldUsed(HLLJOB hJob, LPCTSTR lpszFieldName);
```

Aufgabe:

Gibt an, ob das angegebene Chart-Feld von dem geladenen Projekt verwendet wird.

Parameter:

hJob: List & Label Job-Handle

lpszFieldName: Feldname

Rückgabewert:

Wert	Bedeutung
1	Feld wird verwendet
0	Feld wird nicht verwendet
LL_ERR_UNKNOWN	Feld nicht definiert

Hinweise:

Diese Funktion kann erst nach *LlPrintStart()* oder *LlPrintWithBoxStart()* aufgerufen werden.

Diese Funktion setzt voraus, dass *LL_OPTION_NEWEXPRESSIONS* auf TRUE steht (Voreinstellung).

Ein Aufruf von *LlDefineChartFieldStart()* löscht die "Benutzt"-Flags, so dass diese Funktion direkt danach immer *LL_ERR_UNKNOWN* zurückmeldet, daher darf *LlDefineChartFieldStart()* nur vor *LlPrint[WithBox]Start()* verwendet werden.

Statt eines einfachen Feldnamens kann auch eine Wildcard-Suche verwendet werden. Hinweise hierzu finden Sie bei *LlPrintIsFieldUsed()*.

Beispiel:

```
if (LlPrintIsChartFieldUsed(hJob, "Name")==1)
   LlDefineChartFieldExt(hJob, "Name",<...>);
```

Siehe auch:

LlPrintStart, LlPrintWithBoxStart, LlPrintIsVariableUsed, LlPrintIsFieldUsed

LlPrintIsFieldUsed

Syntax:

```
INT LlPrintIsFieldUsed(HLLJOB hJob, LPCTSTR lpszFieldName);
```

Aufgabe:

Gibt an, ob das angegebene Feld von dem geladenen Projekt verwendet wird. Um die Abfrage schon vor dem Druck für alle Felder durchzuführen, sollten Sie besser *LlGetUsedIdentifiers* verwenden.

Parameter:

hJob: List & Label Job-Handle

lpszFieldName: Feldname

Rückgabewert:

Wert	Bedeutung
1	Feld wird verwendet
0	Feld wird nicht verwendet
LL_ERR_UNKNOWN	Feld nicht definiert

Hinweise:

Diese Funktion kann erst nach *LlPrintStart()* oder *LlPrintWithBoxStart()* aufgerufen werden.

Diese Funktion setzt voraus, dass *LL_OPTION_NEWEXPRESSIONS* auf TRUE steht (Voreinstellung).

Ein Aufruf von *LlDefineFieldStart()* löscht die "Benutzt"-Flags, so dass diese Funktion direkt danach immer *LL_ERR_UNKNOWN* zurückmeldet, daher darf *LlDefineFieldStart()* nur vor *LlPrint[WithBox]Start()* verwendet werden.

Statt eines einfachen Feldnamens kann auch eine Wildcard-Suche verwendet werden. Dies ist immer dann nützlich, wenn Sie Ihre Felder hierarchisch anmelden, z.B. alle Felder der Tabelle "Artikel" in der Form "Artikel.Nr", "Artikel.Bezeichnung" usw. Um zu überprüfen, ob die Artikel-Tabelle überhaupt benötigt wird, können Sie dann für den Parameter lpszFieldName "Artikel*" übergeben.

Beispiel:

```
if (LlPrintIsFieldUsed(hJob, "Name")==1)
   LlDefineFieldExt(hJob, "Name",<...>);
```

Siehe auch:

LlPrintStart, LlPrintWithBoxStart, LlPrintIsVariableUsed, LlPrintIsChartFieldUsed

LlPrintIsVariableUsed

Syntax:

```
INT LlPrintIsVariableUsed(HLLJOB hJob, LPCTSTR lpszName);
```

Aufgabe:

Gibt an, ob die angegebene Variable von dem geladenen Projekt verwendet wird. Beachten Sie die Hinweise bei *LlPrintIsFieldUsed*.

Parameter:

hJob: List & Label Job-Handle

lpszName: Variablenname

Rückgabewert:

Wert	Bedeutung
1	Variable wird verwendet
0	Variable wird nicht verwendet
LL_ERR_UNKNOWN	Variable nicht definiert

Hinweise:

Diese Funktion kann erst nach *LlPrintStart()* oder *LlPrintWithBoxStart()* aufgerufen werden.

Diese Funktion setzt voraus, dass *LL_OPTION_NEWEXPRESSIONS* auf TRUE steht (Voreinstellung).

Ein Aufruf von *LlDefineVariableStart()* löscht die Flags, so dass diese Funktion direkt danach immer *LL_ERR_UNKNOWN* zurückmeldet, daher darf *LlDefineVariableStart()* nur vor *LlPrint[WithBox]Start()* verwendet werden.

Statt eines einfachen Variablennamens kann auch eine Wildcardsuche verwendet werden. Hinweise hierzu finden Sie bei *LlPrintIsFieldUsed()*.

Beispiel:

```
if (LlPrintIsVariableUsed(hJob, "Name")==1)
   LlDefineVariabledExt(hJob, "Name",<...>);
```

Siehe auch:

LlPrintStart, LlPrintWithBoxStart, LlPrintIsFieldUsed

LlPrintOptionsDialog

Syntax:

```
INT LlPrintOptionsDialog(HLLJOB hJob, HWND hWnd, LPCTSTR lpszText);
```

Aufgabe:

Ruft ein Druckoptionsauswahlfenster auf und ermöglicht es dem Benutzer, druckspezifische Einstellungen vorzunehmen.

Parameter:

hJob: List & Label Job-Handle

hWnd: Window-Handle des aufrufenden Programms

lpszText: im Dialog oben auszugebender Text, z.B. ´Es werden nun 55 Etiketten gedruckt´

Rückgabewert:

Fehlercode

Hinweise:

Es werden folgende Optionen abgefragt:

- Drucker (oder Referenzdrucker für Export)
- Export-Ziel
- Seitenzahl der ersten Seite (optional)
- Zahl der gewünschten Exemplare (optional)
- Anfangsposition bei *LL_PROJECT_LABEL*, *LL_PROJECT_CARD*, wenn mehr als ein Etikett/ eine Karteikarte pro Seite vorhanden sind (optional)
- Ausgabemedium (optional)
- Seitenbereich von.. bis... (optional)

Voreinstellungswerte können mit *LlPrintSetOption()* definiert werden. Diese Funktion kann erst nach *LlPrintStart()* / *LlPrintWithBoxStart()*, muss aber vor dem ersten Aufruf von *LlPrint()* aufgerufen werden.

Diese Funktion ruft intern *LlPrintOptionsDialogTitle()* mit NULL als pszTitle auf.

Die Funktion *LlPrinterSetup()* liefert eine Möglichkeit, einen Druckauswahldialog ohne weitere Einstellungen aufzurufen, wird aber nicht empfohlen.

Siehe auch:

LlPrintSetOption, LlPrintGetOption, LlPrintOptionsDialogTitle, LlPrinterSetup

LlPrintOptionsDialogTitle

Syntax:

```
INT LlPrintOptionsDialogTitle(HLLJOB hJob, HWND hWnd, LPCTSTR pszTitle,
   LPCTSTR pszText);
```

Aufgabe:

Ruft ein Druckoptionsauswahlfenster auf und ermöglicht es dem Benutzer, druckspezifische Einstellungen vorzunehmen.

Parameter:

hJob: List & Label Job-Handle

hWnd: Window-Handle des aufrufenden Programms

pszTitle: Dialogtitel

pszText: im Dialog oben auszugebender Text, z.B. ´Es werden nun 55 Etiketten gedruckt´

Rückgabewert:

Fehlercode

Hinweise:

Die Funktion ist fast identisch zu *LlPrintOptionsDialog()*, nur kann man hierüber auch den Dialogtitel festlegen.

Siehe auch:

LlPrintSetOption, LlPrintGetOption, LlPrintOptionsDialog, LlPrinterSetup

LlPrintResetProjectState

Syntax:

```
INT LlPrintResetProjectState (HLLJOB hJob);
```

Aufgabe:

Setzt den Status des Druckjobs zurück, so als ob gerade *LlPrint[WithBox]Start()* aufgerufen werden wäre.

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Fehlercode

Hinweise:

Über diese Funktion kann man den Druck-Status des gesamten Projekts "zurücksetzen" (Objektstati, Seitennummern, Benutzer- und Summenvariablen etc.), d.h. die folgenden Druck-Befehle arbeiten wieder mit einem "frischen" Projekt.

Dies kann z.B. genutzt werden, um Serienbriefe zu erstellen. Der Projektstatus kann nach jedem Brief zurückgesetzt werden, um den nächsten Brief zu drucken. Außerdem sind so alle Drucke ohne Zusatzaufwand in einer Vorschaudatei enthalten.

Beispiel:

```
<starte Druckjob>
<solange noch ein Brief zu drucken ist>
   {
   <hole Datensatz>
   <drucke einen Brief>
   LlPrintResetProjectState(hJob)
   <auf nächsten Datensatz gehen>
   }
<beende Druckjob>
```

Siehe auch:

LlPrintResetObjectState

LlPrintSelectOffsetEx

Syntax:

```
INT LlPrintSelectOffsetEx (HLLJOB hJob, HWND hWnd);
```

Aufgabe:

Öffnet das Auswahlfenster für die Anfangsposition.

Parameter:

hJob: List & Label Job-Handle

hWnd: Window-Handle des aufrufenden Programms

Rückgabewert:

Fehlercode

Hinweise:

Nicht für Listenprojekte!

Diese Funktion kann erst nach *LlPrintStart()* oder *LlPrintWithBoxStart()* aufgerufen werden. Dieser Dialog fragt die Position des Anfangsetiketts ab. Der Offset kann mit *LlPrintSetOption()* und *LL_PRNOPT_OFFSET* vorher gesetzt und hinterher abgefragt werden.

Wenn die Projektdatei nur ein Etikett hat, wird 0 zurückgegeben.

LlPrintSetBoxText

Syntax:

```
INT LlPrintSetBoxText (HLLJOB hJob, LPCTSTR lpszText, INT nPercentage);
```

Aufgabe:

Setzt Text und Fortschrittsanzeige in der Abbruch-Dialogbox.

Parameter:

hJob: List & Label Job-Handle

lpszText: Text, der im Textfeld erscheinen soll

nPercentage: Fortschritt in Prozent

Rückgabewert:

Fehlercode

Hinweise:

Um den Text mehrzeilig zu machen, können LineFeeds ('\x0a', also das ASCII Zeichen 10) eingefügt werden.

Unveränderte Texte oder NULL-Pointer werden nicht neu gezeichnet, um ein Flimmern zu verhindern, unveränderte Prozentwerte oder ´-1´ werden ebenfalls ignoriert.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

if (LlPrintWithBoxStart(hJob, LL_PROJECT_LABEL, "test", LL_PRINT_NORMAL,
          LL_BOXTYPE_NORMALMETER, hWnd, "Ausdruck") == 0)
{
   LlPrintSetBoxText(hJob, "bin gleich soweit...", 0);
   <... etc...>
   LlPrintSetBoxText(hJob, " fertig", 100);
   LlPrintEnd(hJob);
}
else
   MessageBox(NULL, "Fehler", "List & Label", MB_OK);
LlJobClose(hJob);
```

Siehe auch:

LlPrintWithBoxStart, LlPrintUpdateBox, LlPrint

LlPrintSetOption

Syntax:

```
INT LlPrintSetOption (HLLJOB hJob, INT nIndex, INT nValue);
```

Aufgabe:

Setzt verschiedene Druckoptionen, z.B. um die Zahl der gewünschten Kopien (und evtl. die Anfangsseite) voreinzustellen.

Parameter:

hJob: List & Label Job-Handle

nIndex:

LL_PRNOPT_COPIES

Zahl der gewünschten Kopien. Ein Wert von *LL_COPIES_HIDE* versteckt die Abfragebox im Optionsdialog. Die Verarbeitung von Kopien ist im Kapitel über den Druck beschrieben.

Voreinstellung: 1

LL_PRNOPT_PAGE

Seitenzahl, mit der List & Label die erste Seite ausdrucken soll. Wenn sie nicht eingegeben werden können soll, muss *LL_PAGE_HIDE* als Wert übergeben werden.

Voreinstellung: 1

LL_PRNOPT_OFFSET

Position (nur beim Etikettendruck) des ersten Etiketts, abhängig von der eingestellten Druckreihenfolge.

Voreinstellung: 0

LL_PRNOPT_FIRSTPAGE

Die vom Benutzer gewählte Startseite. Wenn "Alle" gewählt wurde, ist die erste Druckseite mit der Startseite identisch.

Voreinstellung: MIN_INT

LL_PRNOPT_LASTPAGE

Seitennummer der letzten zu druckenden Seite.

Voreinstellung: MAX_INT

LL_PRNOPT_JOBPAGES

Hiermit stellen sie die Zahl der Seiten pro Druckjob ein, wenn Sie mit dem Flag *LL_PRINT_MULTIPLE_JOBS* drucken.

Voreinstellung: 16

LL_PRNOPT_PRINTDLG_ONLYPRINTERCOPIES

Wenn diese Option auf TRUE gesetzt wird, kann man die Kopien im Druckdialog nur dann einstellen, wenn der Drucker auch von sich aus Kopien unterstützt.

Voreinstellung: FALSE.

LL_PRNOPT_UNITS

Der Rückgabewert ist identisch mit dem von *LlGetOption(..., LL_OPTION_UNITS).*

nValue: Setzt die dem *nIndex* entsprechende Option

Rückgabewert:

Fehlercode

Siehe auch:

LlPrintStart, LlPrintWithBoxStart, LlPrintGetOption, LlPrintOptionsDialog

LlPrintSetOptionString

Syntax:

```
INT LlPrintSetOptionString(HLLJOB hJob, INT nIndex, LPCTSTR pszValue);
```

Aufgabe:

Setzt diverse Einstellungen in List & Label.

Parameter:

hJob: List & Label Job-Handle

nIndex: Folgende Werte sind als Funktionsindex möglich:

LL_PRNOPTSTR_EXPORT

Gibt das gewünschte bzw. im Dialog voreingestellte Exportmedium an (z.B. "RTF", "HTML", "PDF", ...)

LL_PRNOPTSTR_ ISSUERANGES

Eine Zeichenkette kann zur Angabe des gewünschten Ausfertigungsbereichs voreingestellt werden, z.B. "1,3-4,10-".

LL_PRNOPTSTR_ PAGERANGES

Eine Zeichenkette kann zur Angabe des gewünschten Druckbereichs, wie sie im Druckdialog eingestellt werden kann, voreingestellt werden, z.B. "1,3-4,10-".

LL_PRNOPTSTR_PRINTDST_FILENAME

Hier kann ein Dateiname voreingestellt werden, in den die Druckausgabe geschieht, sofern *LL_PRINT_FILE* bzw. das Ausgabemedium *LL_DESTINATION_FILE* durch den Endanwender bzw. bei *LlPrint[WithBox]Start* gewählt wurde.

LL_PRNOPTSTR_PRINTJOBNAME

Hierüber kann die Bezeichnung des Druckjobs eingestellt werden, die im Druckerspooler erscheint.

Diese muss vor dem ersten Aufruf von *LlPrint()* angegeben werden.

pszValue: neuer Wert

Rückgabewert:

Fehlercode

Beispiel:

```
HLLJOB hJob;

hJob = LlJobOpen(0);
// LlPrintStart(...);
LlPrintSetOptionString(hJob, LL_PRNOPTSTR_PRINTDST_FILENAME,
            "c:\\tmp\\ll.prn");
// ....
// LlPrintEnd();
LlJobClose(hJob);
```

Siehe auch:

LlPrintGetOptionString

LlPrintSetProjectParameter

Syntax:

```
INT LlPrintSetProjectParameter(HLLJOB hLlJob, LPCTSTR pszParameter,
   LPCTSTR pszValue, UINT nFlags)
```

Aufgabe:

Ändert den Wert eines Projekt-Parameters (siehe auch Kapitel Projekt-Parameter)

Parameter:

hJob: List & Label Job-Handle

pszParameter: Name des Parameters

pszValue: Wert des Parameters

nFlags: Typ des Parameters (siehe *LlSetDefaultProjectParameter()*). Wird nur benutzt, wenn der Parameter neu ist.

Rückgabewert:

Fehlercode

Hinweise:

Diese Funktion kann erst nach *LlPrint[WithBox]Start()* aufgerufen werden!

Siehe auch:

LlSetDefaultProjectParameter, LlGetDefaultProjectParameter, LlPrintGetProjectParameter

LlPrintStart

Syntax:

```
INT LlPrintStart (HLLJOB hJob, UINT nObjType, LPCTSTR lpszObjName,
   INT nPrintOptions, INT nReserved);
```

Aufgabe:

Öffnet das Projekt zum Drucken.

Parameter

hJob: List & Label Job-Handle

nObjType: *LL_PROJECT_LABEL, LL_PROJECT_LIST* oder *LL_PROJECT_CARD*

lpszObjName: Der Dateiname des Projekts

nPrintOptions: Druck-Optionen:

Wert	Bedeutung
LL_PRINT_NORMAL	Ausgabe auf Drucker
LL_PRINT_PREVIEW	Ausgabe auf Preview-Dateien
LL_PRINT_FILE	Ausgabe in Datei
LL_PRINT_EXPORT	Als Ausgabemedium wird ein Export-Modul voreingestellt, welches anschließend über *LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT)* festgelegt werden kann.

kann mit *LL_PRINT_MULTIPLE_JOBS* ver'oder't werden, damit der Druckjob in mehrere kleinere Einzeljobs gesplittet wird und der Druck dadurch schon beginnen kann. Die Seitenanzahl, nach der der Job gesplittet werden soll, kann mit *LlPrintSetOption()* eingestellt werden.

nReserved: Für zukünftige Erweiterungen

Rückgabewert:

Fehlercode

Hinweise:

Bitte unbedingt den Rückgabewert auswerten!

nPrintOptions kann mit *LL_PRINT_MULTIPLE_JOBS* ver'oder't werden, damit der Druckjob in mehrere kleinere Einzeljobs gesplittet wird. Die Seitenanzahl, nach der der Job gesplittet werden soll, kann mit *LlPrintSetOption()* eingestellt werden.

Es wird keine Fortschrittsanzeige durch List & Label dargestellt, dies geschieht über *LlPrintWithBoxStart()*.

Achten Sie darauf, dass Sie in diesem Fall eine eigene Nachrichtenschleife (Message Loop) implementieren müssen, damit Ihre Anwendung während des Druckvorgangs noch "reagiert" (z.B. sich die Fenster bei einem Anwendungswechsel neu zeichnen etc.) und ein entsprechendes Multitasking auf dem System noch möglich ist.

Mit "Nachrichtenschleife" ist die Message-Loop

```
while (PeekMessage(hWindow,&msg,0,0,PM_REMOVE))
{
  TranslateMessage(&msg);
  DispatchMessage(&msg);
}
<wenn Abbruch gewünscht>
{
  LlPrintAbort(hJob);
}
```

gemeint, die eingesetzt werden sollte, wenn nicht die Abbruch-Box von List & Label benutzt wird, da ansonsten alle anderen Programme während des Ausdrucks keine optimale Rechnerzeit zugeteilt bekommen.

Beim Drücken eines etwaigen eigenen Abbruch-Buttons muss *LlPrintAbort(HLLJOB)* aufgerufen werden. Dadurch wird bei allen folgenden *LlPrint....*-Aufrufen immer der Fehlercode *LL_ERR_USER_ABORTED* zurückgegeben.

Bei Delphi können Sie statt dieser Schleife *Application.ProcessMessages*, bei Visual Basic *DoEvents* aufrufen.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

if(LlPrintStart(hJob, LL_PROJECT_LABEL, "test", LL_PRINT_NORMAL)== 0)
{
   <... etc ...>
   LlPrintEnd(hJob);
}
else
```

```
   MessageBox(NULL, "Fehler", "List & Label", MB_OK);
LlJobClose(hJob);
```

Siehe auch:

LIPrintWithBoxStart, LIPrintEnd, LIPrintSetOption

LIPrintUpdateBox

Syntax:

```
INT LlPrintUpdateBox(HLLJOB hJob);
```

Aufgabe:

Ermöglicht ein Neuzeichnen der Abbruch-Dialogbox

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Fehlercode

Hinweise:

Wenn Sie langwierige Operationen während des Drucks ausführen, können Sie über diese Funktion ermöglichen, dass sich die Abbruch-Box (wenn nötig) wieder vollständig zeichnet, oder auch auf einen Button-Druck "flüssig" reagiert. Zu diesem Zweck ruft diese Funktion die Nachrichtenbearbeitungsschleife der Abbruch-Box auf.

LIPrintSetBoxText() ruft auch diese Nachrichtenbearbeitungsschleife auf, so dass die Funktion *LIPrintUpdateBox()*nur in seltenen Fällen benötigt wird.

Siehe auch:

LIPrintWithBoxStart, LIPrintSetBoxText

LIPrintWillMatchFilter

Syntax:

```
INT LlPrintWillMatchFilter (HLLJOB hJob);
```

Aufgabe:

Gibt an, ob der momentane Datensatz den vom Benutzer eingegebenen Filter entspricht, also bei der nächsten Ausdruckfunktion (*LIPrint()* oder *LIPrintFields()*) ausgedruckt wird.

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Wert	Bedeutung
negativ	Fehlercode
0	wird nicht ausgedruckt
1	wird ausgedruckt

Hinweise:

Diese Funktion kann erst nach *LlPrintStart()* oder *LlPrintWithBoxStart()* aufgerufen werden.

Die Funktion berechnet den Filterwert anhand der momentan definierten Daten (Variablen bzw. Felder).

Beispiel:

```
if (LlPrintWillMatchfilter(hJob))
   ....
```

Siehe auch:

LlPrintGetFilterExpression, LlPrintDidMatchFilter

LlPrintWithBoxStart

Syntax:

```
INT LlPrintWithBoxStart (HLLJOB hJob, UINT nObjType,
   LPCTSTR lpszObjName, INT nPrintOptions, INT nBoxType,
   HWND hWnd, LPCTSTR lpszTitle);
```

Aufgabe:

Öffnet das Projekt zum Drucken mit Abbruch-Fenster.

Parameter

hJob: List & Label Job-Handle

nObjType: *LL_PROJECT_LABEL, LL_PROJECT_LIST* oder *LL_PROJECT_CARD*

lpszObjName: Der Dateiname des Projekts

nPrintOptions: Druck-Optionen:

Wert	Bedeutung
LL_PRINT_NORMAL	Ausgabe auf Drucker
LL_PRINT_PREVIEW	Ausgabe auf Preview-Dateien
LL_PRINT_FILE	Ausgabe in Datei
LL_PRINT_EXPORT	Als Ausgabemedium wird ein Export-Modul voreingestellt, welches anschließend über *LlPrint-*

Wert	Bedeutung
	SetOptionString(LL_PRNOPTSTR_EXPORT) festgelegt werden kann.

Diese Optionen können jeweils mit den folgenden Flags verodert werden:

Wert	Bedeutung
LL_PRINT_MULTIPLE_JOBS	Ausgabe in mehreren kleinen Druckjobs
LL_PRINT_REMOVE_-UNUSED_VARS	Vom Projekt nicht benötigte Variablen und Felder werden nach dem Druckstart aus dem internen Puffer gelöscht. Dies kann die folgende Übergabe von Variablen und Feldern deutlich beschleunigen, ist aber nur notwendig, wenn die benötigten Daten nicht zuvor über LlGetUsedIdentifiers() abgefragt werden.

Die Druckoptionen beeinflussen den Wert von *LL_OPTIONSTR_EXPORTS_-ALLOWED*.

nBoxType:

Wert	Bedeutung
LL_BOXTYPE_STDABORT	Abbruch-Box mit Systemfortschrittsanzeige
LL_BOXTYPE_-NORMALMETER	Abbruch-Box mit Balken-Fortschrittsanzeige
LL_BOXTYPE_-BRIDGEMETER	Abbruch-Box mit Brücken-Fortschrittsanzeige
LL_BOXTYPE_EMPTYABORT	Abbruch-Box mit Text
LL_BOXTYPE_STDWAIT	Box mit Systemfortschrittsanzeige, kein Abbruchbutton
LL_BOXTYPE_-NORMALWAIT	Box mit Balken-Fortschrittsanzeige, kein Abbruchbutton
LL_BOXTYPE_BRIDGEWAIT	Box mit Brücken-Fortschrittsanzeige, kein Abbruchbutton
LL_BOXTYPE_EMPTYWAIT	Box mit Text, kein Abbruchbutton
LL_BOXTYPE_NONE	Keine Fortschrittsbox

Beachten Sie, dass sich der Boxtype-Parameter unter Windows Vista und neuer nicht auswirkt – bei diesen Systemen wird die Standardfortschrittsbox des Betriebssystems genutzt.

hWnd: Fenster-Handle des aufrufenden Programms (für die Dialog-Box)

lpszTitle: Titel der Dialogbox, erscheint auch als Text im Druck-Manager

Rückgabewert:

Fehlercode

Hinweise:

Bitte unbedingt den Rückgabewert auswerten!

Es wird eine anwendungsmodale Fortschrittsanzeige dargestellt, deren Titel durch den oben angegebenen Parameter definiert wird. In der Dialogbox befindet sich ein Prozent-Meter-Control und ein 2-zeiliger statischer Text, die beide über *LlPrintSetBoxText()* gesetzt werden können, um dem Benutzer den Druckfortschritt anzuzeigen, und bei Anforderung (s.u.) noch ein Abbruch-Button.

Falls keine Fortschrittsanzeige durch List & Label dargestellt werden soll, verwenden Sie anstatt dessen *LlPrintStart()*.

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);

if (LlPrintWithBoxStart(hJob, LL_PROJECT_LABEL, "test", LL_PRINT_NORMAL,
         LL_BOXTYPE_NORMALMETER, hWnd, "Ausdruck") == 0)
{
   LlPrintSetBoxText(hJob, "Drucke...", 0);
   <... etc...>
   LlPrintEnd(hJob, 0);
}
else
   MessageBox(NULL, "Fehler", "App", MB_OK);
LlJobClose(hJob);
```

Siehe auch:

LlPrintStart, LlPrintEnd, LlPrintSetBoxText

LlProjectClose

Syntax:

```
HLLDOMOBJ LlProjectClose(HLLJOB hJob);
```

Aufgabe:

Diese Funktion steht erst ab der Professional Edition zur Verfügung! Schließt ein geöffnetes Projekt und gibt die zugehörige Projektdatei wieder frei. Die Datei wird dabei nicht gespeichert! Ausführliche Anwendungsbeispiele finden Sie im Kapitel "DOM-Funktionen".

Parameter:

hJob: List & Label Job-Handle

Rückgabewert:

Fehlercode

Beispiel:

Siehe Kapitel "DOM-Funktionen".

Siehe auch:

LlProjectSave, LlProjectOpen

LlProjectOpen

Syntax:

```
INT LlProjectOpen(HLLJOB hJob, UINT nObjType, LPCTSTR pszObjName,
   UINT nOpenMode);
```

Aufgabe:

Diese Funktion steht erst ab der Professional Edition zur Verfügung! Öffnet die angegebene Projektdatei. Um das DOM-Handle für das Projektobjekt zu erhalten rufen Sie anschließend LlDomGetProject() auf. Dieses Objekt ist die Basis für alle weiteren DOM-Funktionen. Ausführliche Anwendungsbeispiele finden Sie im Kapitel "DOM-Funktionen".

Parameter:

hJob: List & Label Job-Handle

nObjType:

Wert	Bedeutung
LL_PROJECT_LABEL	für Etiketten
LL_PROJECT_CARD	für Karteikarten
LL_PROJECT_LIST	für Listen

pszObjName: Projektdateiname mit Pfadangabe

nOpenMode: Kombination (Veroderung) jeweils eines Flags aus den folgenden drei Gruppen:

Wert	Bedeutung
LL_PRJOPEN_CD_OPEN_EXISTING	Datei muss bereits existieren, sonst wird Fehlercode zurückgeliefert.
LL_PRJOPEN_CD_CREATE_ALWAYS	Datei wird immer neu erzeugt. Wenn schon vorhanden wird der Inhalt gelöscht.
LL_PRJOPEN_CD_CREATE_NEW	Datei wird neu erzeugt, wenn nicht vorhanden. Wenn Datei bereits existiert wird

Wert	Bedeutung
	Fehlercode zurückgeliefert.
LL_PRJOPEN_CD_OPEN_ALWAYS	Wenn Datei vorhanden, wird der Inhalt verwendet, sonst wird Datei neu erzeugt.

Wert	Bedeutung
LL_PRJOPEN_AM_READWRITE	Datei wird für Lese/Schreibzugriff geöffnet.
LL_PRJOPEN_AM_READONLY	Datei wird nur für Lesezugriff geöffnet.

Wert	Bedeutung
LL_PRJOPEN_EM_IGNORE_FORMULAERRORS	Syntaxfehler werden ignoriert. Siehe Hinweise.

Rückgabewert:

Fehlercode

Hinweise:

Wenn das Flag *LL_PRJOPEN_EM_IGNORE_FORMULAERRORS* verwendet wird, werden Syntaxfehler im Projekt ignoriert. Dies hat den Vorteil, dass Projekte auch dann erfolgreich geöffnet und bearbeitet werden können, wenn die Datenstruktur nicht bekannt bzw. angemeldet ist. Da die Formeln im Projekt dann wie Platzhalter behandelt werden, kann die Sektion mit den verwendeten Variablen (siehe *LlGetUsedIdentifiers()*) nicht korrekt geschrieben werden, wenn Sie z.B. in einer Tabelle weitere Spalten anhängen. Der Inhalt dieser Sektion wird beim Speichern unverändert gelassen. Das gleiche gilt für den Fall, dass in einem Berichtscontainer eine neue Tabelle eingefügt wird, die bisher nicht verwendet wurde. Für solche Fälle darf daher *LL_PRJOPEN_EM_IGNORE_FORMULAERRORS* nicht gesetzt werden. Wenn das Flag nicht gesetzt wird, kann LL_NTFY_EXPRERROR verwendet werden um die Fehlermeldungen für die Anzeige zu sammeln.

Beispiel:

Siehe Kapitel "DOM-Funktionen".

Siehe auch:

LlProjectSave, LlProjectClose, LlDomGetProject

LlProjectSave

Syntax:

```
HLLDOMOBJ LlProjectSave(HLLJOB hJob, LPCTSTR pszObjName);
```

Aufgabe:

Diese Funktion steht erst ab der Professional Edition zur Verfügung! Speichert ein geöffnetes Projekt. Ausführliche Anwendungsbeispiele finden Sie im Kapitel "DOM-Funktionen".

Parameter:

hJob: List & Label Job-Handle

pszObjName: Projektdateiname mit Pfadangabe. Darf NULL sein (s. Hinweise)

Rückgabewert:

Fehlercode

Hinweise:

Wenn pszObjName NULL ist, wird die Datei unter dem gleichen Namen gespeichert, unter dem sie geöffnet wurde.

Beispiel:

Siehe Kapitel "DOM-Funktionen".

Siehe auch:

LlProjectOpen, LlProjectClose

LlRTFCopyToClipboard

Syntax:

```
INT LlRTFCopyToClipboard(HLLJOB hJob, HLLRTFOBJ hRTF);
```

Aufgabe:

Kopiert den Inhalt des RTF-Objektes in die Zwischenablage. Dabei werden verschiedene Clipboard Formate zur Verfügung gestellt (CF_TEXT, CF_TEXTW, CF_RTF)

Parameter

hJob: List & Label Job-Handle

hRTF: Handle auf RTF-Objekt

Rückgabewert:

Fehlercode

Siehe auch:

LlRTFCreateObject

LlRTFCreateObject

Syntax:

```
HLLRTFOBJ LlRTFCreateObject(HLLJOB hJob);
```

Aufgabe:

Erstellt eine Instanz des List & Label RTF-Objekts zum Aufruf unabhängig vom List & Label Designer.

Rückgabewert:

Handle auf RTF-Editorobjekt oder NULL, wenn Fehler

Parameter

hJob: List & Label Job-Handle

Siehe auch:

LlRTFGetText

LlRTFDeleteObject

Syntax:

```
INT LlRTFDeleteObject(HLLJOB hJob, HLLRTFOBJ hRTF);
```

Aufgabe:

Gibt das RTF-Objekt wieder frei.

Parameter

hJob: List & Label Job-Handle

hRTF: Handle auf RTF-Objekt

Rückgabewert:

Fehlercode

Siehe auch:

LlRTFCreateObject

LlRTFDisplay

Syntax:

```
INT LlRTFDisplay(HLLJOB hJob, HLLRTFOBJ hRTF, HDC hDC, _PRECT pRC,
   BOOL bRestart, LLPUINT pnState);
```

Aufgabe:

Gibt den Inhalt des RTF-Objektes in einem beliebigen Gerätekontext aus. Dies kann verwendet werden, um den Inhalt etwa zu drucken oder in einem eigenen Fenster darzustellen.

Parameter

hJob: List & Label Job-Handle

hRTF: Handle auf RTF-Editorobjekt

hDC: Gerätekontext für die Ausgabe. Kann auch NULL sein, in diesem Fall wird der Standard-Drucker DC verwendet.

pRC: Zeiger auf Ausgaberechteck. Kann auch NULL sein, in diesem Fall wird für einen Druckerkontext die ganze bedruckbare Seite verwendet. Andernfalls muss die Angabe in logischen Koordinaten erfolgen (mm/10, inch/100 etc.), sofern der DC kein Bildschirm-DC ist.

bRestart: Wenn TRUE, dann wird der Inhalt des Objektes (wieder) von Anfang an dargestellt, ansonsten wird der Text von der letzen Ausgabe fortgesetzt, um eine mehrseitige Ausgabe zu bekommen.

pnState: Ausgabestatus

Rückgabewert:

Fehlercode

Beispiel:

```
// Drucker-Devicecontext erzeugen
HDC   hDC = CreateDC(NULL,"\\\\prnsrv\\standard",NULL,NULL);
RECT  rc = {0,0,1000,1000};
BOOL bFinished = FALSE;
INT nPage = 0;
// Dokument initialisieren
StartDoc(hDC,NULL);
while (!bFinished)
   {
   nPage++;
   UINT     nState = 0;
   // Seite initialisieren
   StartPage(hDC);
   // DC vorbereiten
   SetMapMode(hDC,MM_ISOTROPIC);
   SetWindowOrgEx(hDC,rc.left,rc.top,NULL);
   SetWindowExtEx(hDC,rc.right-rc.left,rc.bottom-rc.top,NULL);
   SetViewportOrgEx(hDC,0,0,NULL);
   SetViewportExtEx(hDC,GetDeviceCaps(hDC,HORZRES),
```

```
            GetDeviceCaps(hDC,VERTRES),NULL);
        // RTF-Text auf Drucker ausgeben
        BOOL bFinished = (LlRTFDisplay(hJob,hRTF,hDC,&rc,nPage ==
                          1,&nState) == LL_WRN_PRINTFINISHED);
        // Seite bschliessen
        EndPage(hDC);
        }
EndDoc(hDC);
```

Siehe auch:

LIRTFCreateObject

LIRTFEditObject

Syntax:

```
INT LlRTFEditObject(HLLJOB hJob, HLLRTFOBJ hRTF, HWND hWnd,
   HDC hPrnDC, INT nProjectType, BOOL bModal);
```

Aufgabe:

Erzeugt einen RTF-Editor für die Bearbeitung des RTF-Objekts durch den Benutzer. Es stehen alle innerhalb des übergebenen List & Label-Jobs definierten Variablen und Felder (im Fall eines Listenprojekts) zur Verfügung.

Parameter

hJob: List & Label Job-Handle

hRTF: Handle auf RTF-Objekt

hWnd: Parent-Fensterhandle bzw. Handle des Controls, das für die Darstellung des Objektes verwendet werden soll (siehe bModal-Flag)

hPrnDC: Referenz-Gerätekontext (wichtig z.B. für die zur Verfügung stehenden Schriftarten). Kann auch NULL sein, in diesem Fall wird der Standard-Drucker DC verwendet.

nProjectType: Projekttyp

Wert	Bedeutung
LL_PROJECT_LABEL	für Etiketten
LL_PROJECT_CARD	für Karteikarten
LL_PROJECT_LIST	für Listen

bModal: bestimmt, ob das Fenster als modaler Dialog angezeigt werden soll (TRUE) oder ob das Control, dessen Fensterhandle in hWnd übergeben wurde, durch das RTF-Control ersetzt werden soll (FALSE). Beachten Sie, dass von Visual C++ erzeugte Fenster leider nicht als Control-Handle für die nicht-modale Variante geeignet sind. Wir empfehlen stattdessen die Verwendung des RTF-OCX-Controls (cmll20r.ocx).

Rückgabewert:

Fehlercode

Siehe auch:

LIRTFCreateObject

LIRTFEditorInvokeAction

Syntax:

```
INT LlRTFEditorInvokeAction(HLLJOB hJob, HLLRTFOBJ hRTF,
   INT nControlID);
```

Aufgabe:

Erlaubt es, eine Schaltfläche im RTF-Editor per Code zu aktivieren. Dies ist wichtig, wenn Sie den Editor inplace anzeigen (s. *LIRTFEditObject()*) und die umgebende Applikation ein zusätzliches Menü zur Verfügung stellen soll.

Parameter

hJob: List & Label Job-Handle

hRTF: Handle auf RTF-Objekt

nControlID: Hier kann die Control-ID der zu aktivierenden Schaltflächen angegeben werden. Die entsprechenden IDs finden Sie in der Datei MENUID.TXT in Ihrer List & Label Installation.

Rückgabewert:

Fehlercode

Siehe auch:

LIRTFCreateObject , LIRTFEditorProhibitAction, LIRTFEditObject

LIRTFEditorProhibitAction

Syntax:

```
INT LlRTFEditorProhibitAction(HLLJOB hJob, HLLRTFOBJ hRTF,
   INT nControlID);
```

Aufgabe:

Erlaubt es, einzelne Schaltflächen des Editors auszublenden.

Parameter

hJob: List & Label Job-Handle

hRTF: Handle auf RTF-Objekt

nControlID: Hier können die Control-IDs der auszublendenden Schaltflächen angegeben werden. Die entsprechenden IDs finden Sie in der Datei MENUID.TXT in Ihrer List & Label Installation.

Rückgabewert:

Fehlercode

Siehe auch:

LIRTFCreateObject , LIRTFEditorInvokeAction, LIRTFEditObject

LIRTFGetText

Syntax:

```
INT LlRTFGetText(HLLJOB hJob, HLLRTFOBJ hRTF, INT nFlags,
   LPTSTR lpszBuffer, UINT nBufferSize);
```

Aufgabe:

Fragt den Text des RTF-Objekts ab

Parameter

hJob: List & Label Job-Handle

hRTF: Handle auf RTF-Objekt

nFlags: Optionen (s. *LIRTFGetTextLength()*)

lpszBuffer: Puffer für die Rückgabe

nBufferSize: Puffergröße

Rückgabewert:

Fehlercode

Hinweise:

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Beispiel:

```
HLLRTFOBJ hRTF = LlRTFCreateObject(hJob);
if (LlRTFEditObject(hJob,hRTF,NULL,NULL,LL_PROJECT_LABEL) >= 0)
{
   INT nFlags = LL_RTFTEXTMODE_RTF|LL_RTFTEXTMODE_EVALUATED);
   INTnLen = LlRTFGetTextLength(hJob,hRTF,nFlags);
   TCHAR*pszText = new TCHAR[nLen+1];
   LlRTFGetText(hJob,hRTF,nFlags,pszText,nLen+1);
```

```
   printf("'%s'\n\n", pszText);
   delete[] pszText;
}
```

Siehe auch:

LIRTFCreateObject, LIRTFGetTextLength

LIRTFGetTextLength

Syntax:

```
INT LlRTFGetTextLength(HLLJOB hJob, HLLRTFOBJ hRTF, INT nFlags);
```

Aufgabe:

Liefert die Länge des Inhalts des RTF-Objektes zurück. Damit kann dann z.B. ein passender Puffer bereitgestellt werden.

Parameter

hJob: List & Label Job-Handle

hRTF: Handle auf RTF-Objekt

nFlags: jeweils eine Option der folgenden beiden Optionsgruppen muss angegeben sein:

Wert	Bedeutung
Optionen zur Wahl des Darstellungsmodus:	
LL_RTFTEXTMODE_RTF	Länge des RTF-formatierten Texts (incl. RTF-Steuerzeichen)
LL_RTFTEXTMODE_PLAIN	Länge des unformatierten Texts
Optionen zur Wahl des Inhalts:	
LL_RTFTEXTMODE_RAW	Text in unberechneter Form (ggf. mit Formeln)
LL_RTFTEXTMODE_EVALUATED	Text in ausgewerteter Form

Rückgabewert:

Länge des benötigten Puffers

Siehe auch:

LIRTFCreateObject, LIRTFGetText

LlRTFSetText

Syntax:

```
INT LlRTFSetText(HLLJOB hJob, HLLRTFOBJ hRTF, LPCTSTR lpszText);
```

Aufgabe:

Setzt den Inhalt des RTF-Objekts auf die übergebene Zeichenkette. Das Format (Plain oder RTF) wird automatisch erkannt.

Parameter

hJob: List & Label Job-Handle

hRTF: Handle auf RTF-Objekt

lpszText: Neuer Inhalt des Objekts

Rückgabewert:

Fehlercode

Siehe auch:

LlRTFCreateObject

LlSelectFileDlgTitleEx

Syntax:

```
INT LlSelectFileDlgTitleEx (HLLJOB hJob, HWND hWnd, LPCTSTR pszTitle,
   UINT nObjType, LPTSTR pszBuffer, UINT nBufLen, LPVOID pReserved);
```

Aufgabe:

Öffnet einen Dateiauswahl-Dialog mit integriertem Vorschau-Fenster.

Parameter:

hJob: List & List Job-Handle

hWnd: Fensterhandle des aufrufenden Programms

pszTitle: Fenstertitel des Dateiauswahl-Dialogs

nObjType:

Wert	Bedeutung
LL_PROJECT_LABEL	für Etiketten
LL_PROJECT_CARD	für Karteikarten
LL_PROJECT_LIST	für Listen

wenn *LL_FILE_ALSONEW* addiert wird, kann ein neuer Dateiname angegeben werden, ansonsten kann man nur eine existierende Datei wählen.

pszBuffer: initialisierter Puffer für Dateinamen mit Pfadangabe

nBufLen: Länge des Puffers

pReserved: reserviert, muss NULL oder leer ('') sein.

Rückgabewert:

Fehlercode

Hinweise:

Wichtig für Visual Basic bei direkter DLL-Ansteuerung (nicht OCX): Der Puffer muss Null-terminiert (chr$(0)) sein und bereits auf *nBufLen* voralloziert sein. Analoges gilt für die meisten Programmiersprachen.

Vorteile gegenüber normalem CommonDialog: Anzeige der Beschreibung, Preview-Skizze, Sprachkonsistenz innerhalb List & Label und die Dialogdesign-Anpassung.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Beispiel:

```
char  szFilename[255+1]={0};
if LlSelectFileDlgTitleEx(hJob,hWnd, "Report", LL_PROJECT_LIST,
   szFilename,sizeof(szFilename), NULL)==0;
   <ok, weiter>
```

Siehe auch:

LL_OPTION_OFNDIALOG_NOPLACESBAR, LL_OPTIONSTR_..._PRJDESCR

LlSetDebug

Syntax:

```
void LlSetDebug(INT nOnOff);
```

Aufgabe:

Schaltet den Debug-Modus ein oder aus.

Parameter:

nOnOff: Null, wenn Debug-Mode ausgeschaltet werden soll, sonst können folgende Werte verodert (addiert) übergeben werden:

Wert	Bedeutung
LL_DEBUG_CMBTLL	zum Einschalten der normalen Debugging-Info
LL_DEBUG_CMBTDWG	zum Einschalten der Debugging-Info für Grafikfunktionen
LL_DEBUG_CMBTLL_-	LL-Debugging, aber keine Callback-Info

NOCALLBACKS	
LL_DEBUG_CMBTLL_NOSTORAGE	LL-Debugging, aber keine StgAPI-Info
LL_DEBUG_CMBTLL_NOSYSINFO	Kein System-Informations-Dump beim Einschalten des Debugging-Modus
LL_DEBUG_CMBTLL_LOGTOFILE	Die Ausgaben werden auch in eine Log-Datei namens COMBIT.LOG im %APPDATA%-Verzeichnis geschrieben.

Hinweise:

Benutzen Sie das im Lieferumfang enthaltene Programm Debwin, um die Debug Ausgaben in einem Extra-Fenster oder auf einem Monochrom-Monitor auszugeben.

Wenn in List & Label über *LlSetDebug*(*LL_DEBUG_CMBTLL*) der Debug-Modus eingeschaltet wird, gibt die DLL jeden Funktionsaufruf mit den dazugehörigen Parametern und Ergebnissen aus. Den Funktionsnamen ist ein '@' vorgesetzt, damit man die Funktionsaufrufe leicht von anderen internen List & Label Debugging-Ausgaben, unterscheiden kann.

Die Ausgaben sind durch Einrückungen geschachtelt, falls eine DLL im Debugging-Modus andere Funktionen einer DLL (auch sich selbst), die sich auch im Debugging-Modus befindet, aufruft.

Weitere Informationen über das Debugtool Debwin finden Sie in Kapitel "Fehlersuche mit Debwin".

Beispiel:

```
HLLJOB hJob;
INT      v;

LlSetDebug(LL_DEBUG_CMBTLL);
hJob = LlJobOpen(0);
v = LlGetVersion(LL_VERSION_MAJOR);
LlJobClose(hJob);
```

gibt in etwa folgendes auf dem Debugging-Output aus:

```
@LlJobOpen(0)=1
@LlGetVersion(1)=6
@LlJobClose(1)
```

Siehe auch:

LlDebugOutput

LlSetDefaultProjectParameter

Syntax:

```
INT LlSetDefaultProjectParameter(HLLJOB hLlJob, LPCTSTR pszParameter,
   LPCTSTR pszValue, UINT nFlags)
```

Aufgabe:

Setzt den Default-Wert eines Projekt-Parameters (siehe auch Kapitel Projekt-Parameter)

Parameter:

hJob: List & Label Job-Handle

pszParameter: Name des Parameters. Wenn dieser Parameter NULL ist, werden alle USER-Parameter aus der internen Liste entfernt.

pszValue: Wert des Parameters

nFlags: Typ des Parameters, Werte siehe Kapitel Vordefinierte Projektparameter

Rückgabewert:

Fehlercode

Hinweise:

Diese Funktion sollte vor *LlDefineLayout()* und *LlPrint[WithBox]Start()* aufgerufen werden!

Siehe auch:

LlGetDefaultProjectParameter, LlPrintSetProjectParameter, LlPrintGetProject-Parameter

LlSetFileExtensions

Syntax:

```
INT LlSetFileExtensions (HLLJOB hJob, INT nObjType,
   LPCTSTR lpszProjectExt, LPCTSTR lpszPrintExt, LPCTSTR lpszSketchExt);
```

Aufgabe:

Einstellung von benutzerdefinierten Dateiendungen.

Parameter:

hJob: List & Label Job-Handle

nObjType:

Wert	Bedeutung
LL_PROJECT_LABEL	für Etiketten
LL_PROJECT_CARD	für Karteikarten
LL_PROJECT_LIST	für Listen

lpszProjectExt: Extension für Project-File. Voreinstellung:

Wert	Voreinstellung
LL_PROJECT_LABEL	"lbl"
LL_PROJECT_CARD	"crd"
LL_PROJECT_LIST	"lst"

lpszPrintExt: Extension für Druckerdefinitions-File. Voreinstellung:

Wert	Voreinstellung
LL_PROJECT_LABEL	"lbp"
LL_PROJECT_CARD	"crp"
LL_PROJECT_LIST	"lsp"

lpszSketchExt: Extension für Dateidialog-Skizze. Voreinstellung:

Wert	Voreinstellung
LL_PROJECT_LABEL	"lbv"
LL_PROJECT_CARD	"crv"
LL_PROJECT_LIST	"lsv"

Rückgabewert:

Fehlercode

Hinweise:

Es ist wichtig, dass alle 9 Datei-Erweiterungen verschieden sind!

Bitte rufen sie diese Funktion vor *LlDefineLayout()* und vor den *LlPrint...Start()*-Funktionen auf, am besten also direkt nach *LlJobOpen()*/*LlJobOpenLCID()*.

Die Dateierweiterungen können auch über *LlSetOptionString()* gesetzt werden.

Beispiel:

```
HLLJOB hJob;
INT      v;

hJob = LlJobOpen(0);
v = LlSetFileExtensions(hJob, LL_PROJECT_LIST, "rpt", "rptp", "reptv");
// ....
LlJobClose(hJob);
```

LlSetNotificationCallback

Syntax:

```
FARPROC LlSetNotificationCallback (HLLJOB hJob, FARPROC lpfnNotify);
```

Aufgabe:

Definition einer Prozedur, die bei Notifications aufgerufen werden soll.

Parameter:

hJob: List & Label Job-Handle

lpfnNotify: die Adresse einer Funktion (s.u.)

Rückgabewert:

Adresse der übergebenen Funktion (oder NULL, wenn Fehler)

Hinweise:

Die Callback-Funktion hat höhere Priorität als die Callback-Nachricht; wenn sie definiert ist, wird keine Nachricht gesendet, sondern die Callback-Funktion aufgerufen.

Diese Funktion darf nur dann genutzt werden, wenn Callbacks direkt verwendet werden, d.h. nicht bei .NET, OCX oder VCL, da dort die Callbacks auf Events abgebildet werden.

Die Callback-Funktion hat folgende Definition:

```
LPARAM WINAPI MyCallback(UINT nFunction, LPARAM lParam)
```

und muss eine exportierte Funktion sein.

Die Bedeutung der Parameter *nFunction* und *lParam* können Sie in dem Kapitel über die Callback-Objekte nachlesen.

Beispiel:

```
LPARAM WINAPI MyCallback(UINT nFunction, LPARAM lParam)
{ //…. }
HLLJOB hJob;
unsigned int wMsg;
hJob = LlJobOpen(0);
v = LlSetNotificationCallback(hJob, MyCB);
// ....
LlJobClose(hJob);
```

Siehe auch:

LlSetNotificationCallbackExt, NotificationMessage, LlSetNotificationMessage

LlSetNotificationCallbackExt

Syntax:

```
FARPROC LlSetNotificationCallbackExt (HLLJOB hJob, INT nEvent,
   FARPROC lpfnNotify);
```

Aufgabe:

Definition einer Prozedur, die bei Notifications des genannten Events aufgerufen werden soll.

Parameter:

hJob: List & Label Job-Handle

nEvent: Event-ID (*LL_CMND_xxx* oder *LL_NTFY_xxxx*)

lpfnNotify: die Adresse einer Funktion (s.u.)

Rückgabewert:

Adresse der übergebenen Funktion (oder NULL, wenn Fehler)

Hinweise:

Die "spezialisierte" Callback-Funktion hat höhere Priorität als die "generelle" Callback-Funktion oder eine Callback-Nachricht.

List & Label sucht demnach zuerst, ob es für den Event, den es auslösen möchte, einen über diese Funktion definierten "spezialisierten" Callback gibt. Wenn ja, wird dieser aufgerufen. Anderenfalls überprüft List & Label, ob ein über *LlSetNotificationCallback()* definierter "unspezifischer" Callback definiert ist und ruft dann diesen auf. Ansonsten überprüft List Label, ob eine Nachrichten-Nummer über *LlSetNotificationMessage()* definiert ist und sendet dann diese Nachricht gesendet.

Diese Funktion darf auch genutzt werden, wenn Callbacks direkt verwendet werden, d.h. bei OCX oder VCL. Nicht aber bei der .NET-Komponente, da diese diese Funktion schon für sich verwendet.

Die Callback-Funktion hat folgende Definition:

```
LPARAM WINAPI MyCallback(UINT nFunction, LPARAM lParam)
```

und muss eine exportierte Funktion sein.

Die Bedeutung der Parameter *nFunction* und *lParam* können Sie in dem Kapitel über die Callback-Objekte nachlesen.

Beispiel:

```
LPARAM WINAPI MyCallback(UINT nFunction, LPARAM lParam)
{ //....}

HLLJOB hJob;
unsigned int wMsg;
hJob = LlJobOpen(0);
v = LlSetNotificationCallbackExt(hJob, LL_CMND_CHANGE_DCPROPERTIES_DOC,
MyCB);
// ....
LlJobClose(hJob);
```

Siehe auch:

LlSetNotificationCallback

LlSetNotificationMessage

Syntax:

```
UINT LlSetNotificationMessage (HLLJOB hJob, UINT nMessage);
```

Aufgabe:

Definition einer von der Voreinstellung abweichenden Nachrichtennummer für Callback (USER-)-Objekte.

Parameter:

hJob: List & Label Job-Handle

nMessage: die neue Nachrichtennummer

Rückgabewert:

Fehlercode

Hinweise:

Die voreingestellte Nachrichtennummer hat den Wert der Funktion *RegisterWindowMessage("cmbtLLMessage");*

Höhere Priorität hat die Callback-Funktion; wenn diese definiert ist, wird keine Nachricht gesendet.

Die Bedeutung der Parameter der Nachricht können Sie in dem Kapitel über die Callback-Objekte nachsehen.

Diese Funktion darf nur dann genutzt werden, wenn Callbacks direkt verwendet werden, d.h. nicht bei OCX, oder VCL, da dort die Callbacks auf Events abgebildet werden.

Beispiel:

```
hJob     hJob;
unsigned int    wMsg;

LlSetDebug(TRUE);
hJob = LlJobOpen(0);
v = LlSetNotificationMessage(hJob,WM_USER+1);
// ....
LlJobClose(hJob);
```

Siehe auch:

LlGetNotificationMessage, LlSetNotificationCallback, LlSetNotificationCallbackExt

LlSetOption

Syntax:

```
INT LlSetOption (HLLJOB hJob, INT nMode, INT_PTR nValue);
```

Aufgabe:

Setzt diverse Einstellungen in List & Label.

Parameter:

hJob: List & Label Job-Handle

nMode: Optionsindex:

LL_OPTION_ADDVARSTOFIELDS

TRUE: Der Formelassistent in Tabellenobjekten bietet die Variablen zusätzlich zu den Feldern an.

FALSE: Variablen werden im Formelassistent nicht angeboten, nur die Felder (Voreinstellung).

Diese Option macht nur im Tabellenmodus (*LL_PROJECT_LIST*) Sinn.

LL_OPTION_ALLOW_COMBINED_COLLECTING_OF_DATA_FOR_-COLLECTIONCONTROLS

TRUE: Wenn im Berichtscontainer mehrere Elemente die gleiche Datenquelle verwenden (z.B. mehrere Charts hintereinander, Charts und Kreuztabellen etc.) werden die Daten nur einmal durchlaufen und an alle Elemente gleichzeitig weitergegeben. Dies kann – je nach Projekt – zu einem erheblichen Performancegewinn führen. Allerdings erhöht sich der Speicherverbrauch. Zudem ist es dann nicht mehr möglich, während des Drucks den Wert von Variablen, die das Aussehen oder den Inhalt der Elemente im Container beeinflussen, zu verändern (Voreinstellung).

FALSE: Die Datenversorgung wird für jedes Element getrennt vorgenommen.

LL_OPTION_ALLOW_LLX_EXPORTERS

TRUE: Export-Module, die in der Liste der zu ladenden Extension-DLLs (*LL_-OPTIONSTR_LLXPATHLIST*) gefunden werden, werden akzeptiert

FALSE: Es werden keine Export-Module akzeptiert

Diese Option muss vor dem Setzen der Option *LL_OPTIONSTR_LLXPATHLIST* eingestellt werden.

Voreinstellung: TRUE

LL_OPTION_CALCSUMVARSONINVISIBLELINES

Hiermit geben Sie die Voreinstellung für die Option an, ob auch bei unterdrückten Datenzeilen die Summenberechnung durchgeführt werden soll. Der eingestellte Wert wird dann in der Projektdatei gespeichert.

Voreinstellung: FALSE

LL_OPTION_CALC_SUMVARS_ON_PARTIAL_LINES

TRUE: Die Summenvariablen werden aktualisiert, sobald eine Datenzeile angedruckt wurde.

FALSE: Die Summenvariablen werden aktualisiert, sobald alle Datenzeilen komplett gedruckt werden konnten.

Voreinstellung: FALSE

LL_OPTION_CALLBACKMASK

Als Wert ist eine beliebige Kombination folgender Werte möglich: *LL_CB_PAGE, LL_CB_PROJECT, LL_CB_OBJECT, LL_CB_HELP, LL_CB_TABLELINE, LL_CB_TABLEFIELD*

Zur Bedeutung der Parameter lesen Sie bitte das Kapitel über Callbacks.

LL_OPTION_CALLBACKPARAMETER

Setzt einen Wert, der in der scCallback Struktur an alle Callbacks übergeben wird.

Zur Bedeutung des Parameters lesen Sie bitte das Kapitel über Callbacks.

LL_OPTION_CODEPAGE

Hierüber setzen Sie die Codepage, die für sämtliche Konvertierungen von SBCS/DBCS- und Unicode-Strings benutzt wird.

Betroffen sind die A-Funktionen der DLL sowie das Lesen/Schreiben einer Projektdatei.

Diese Einstellung ist global gültig, d.h für alle List & Label-Jobs innerhalb einer Task. Entsprechend wird der Wert in hJob auch ignoriert.

Die Codepage muss auf dem System installiert sein (Stichwort NLS[1]).

Voreinstellung: CP_ACP.

1 NLS = "National Language Support"

LL_OPTION_COMPRESSRTF

Wenn Sie einen festen Text in einem RTF-Control eingeben, wird der Text im Projektfile abgespeichert. Wenn diese Option auf TRUE gesetzt wird, wird der Text komprimiert.

Eigentlich ist es nur für Debugging-Zwecke sinnvoll, diese Option auszuschalten.

Voreinstellung: TRUE

LL_OPTION_COMPRESSSTORAGE

TRUE: Die Metafile-Preview-Dateien werden automatisch komprimiert (Voreinstellung).

FALSE: keine Kompression, diese Option hat normalerweise einen Geschwindigkeitsvorteil.

LL_OPTION_CONVERTCRLF

TRUE: List & Label übersetzt CR-LF-Kombinationen in Variablen- und Feldinhalten durch LF (und verhindert somit doppelte Zeilenumbrüche). (Voreinstellung).

FALSE: Der Inhalt bleibt unverändert.

LL_OPTION_DEFAULTDECSFORSTR

Mit dieser Option setzen Sie die Anzahl der Nachkommastellen, die die Designerfunktion Str$() verwendet, wenn die Anzahl durch den Anwender im Designer nicht explizit vorgegeben wurde.

Voreinstellung: 5

LL_OPTION_DEFDEFFONT

Mit dieser Option setzen Sie ein Font-Handle für die Schriftart, die als Voreinstellung für die Projekt-Schriftart verwendet wird. Es wird eine Kopie des Fonts angelegt, das Handle muss also nicht mehr nach dem Aufruf gültig bleiben.

Diese Schriftart kann auch über *LL_OPTIONSTR_DEFDEFFONT* gesetzt oder abgefragt werden.

Voreinstellung: *GetStockObject(ANSI_VAR_FONT)*

LL_OPTION_DELAYTABLEHEADER

Diese Option beschreibt, ob der Kopf einer Tabelle bei dem Aufruf von *LlPrint()*, oder erst bei dem ersten Drucken einer Datenzeile (*LlPrintFields()*) gedruckt werden soll:

TRUE: erst bei *LlPrintFields()* (Voreinstellung).

FALSE: schon bei *LlPrint()*. Falls Felder in der Kopfzeile verwendet werden können, muss der Inhalt schon vor dem 1. *LlPrint* übergeben worden sein.

LL_OPTION_ DESIGNEREXPORTPARAMETER

s. Kapitel Direkter Druck und Export aus dem Designer.

LL_OPTION_ DESIGNERPREVIEWPARAMETER

s. Kapitel Direkter Druck und Export aus dem Designer.

LL_OPTION_ DESIGNERPRINT_SINGLETHREADED

s. Kapitel Direkter Druck und Export aus dem Designer.

LL_OPTION_ ERR_ON_FILENOTFOUND

TRUE: Wenn eine Grafikdatei zur Druckzeit nicht gefunden wurde, wird *LL_ERR_DRAWINGNOTFOUND* ausgelöst.

FALSE: Der Fehler wird ignoriert und die Grafik ohne weitere Rückmeldung nicht ausgegeben (Voreinstellung).

LL_OPTION_ESC_CLOSES_PREVIEW

Diese Option bestimmt, ob ein Druck auf die Escape-Taste das Vorschaufenster schließt. Default: FALSE.

LL_OPTION_EXPRSEPREPRESENTATIONCODE

Diese Option bestimmt den Code des Zeichens, das verwendet wird, um in dem mehrzeiligen Editfeld des Formelassistenten die Zeilen zu trennen.

Dieser Wert muss u.U. für den Einsatz mit anderen Sprachen/ Zeichensätzen geändert werden, da der Default-Code in DBCS-Zeichensystemen evtl. verwendet wird.

LL_OPTION_FONTQUALITY

(Siehe auch *LL_OPTION_FONTPRECISION*)

Über diese Option kann man den Windows-Fontmapper beeinflussen, um die Wahrscheinlichkeit, dass z.B. eine Druckerschriftart gewählt wird, zu erhöhen.

Der hier angegebene Wert wird für das LOGFONT.lfQuality-Feld verwendet, um Schriftart-Instanzen zu erzeugen.

Eine Dokumentation der verfügbaren Werte finden Sie im MSDN. Nicht alle Druckertreiber unterstützen diese Option korrekt, hier müssen Sie im Zweifelsfall bei Ihrem Hersteller nachfragen.

Voreinstellung: DEFAULT_QUALITY.

LL_OPTION_FONTPRECISION

(Siehe auch *LL_OPTION_FONTQUALITY*)

Über diese Option kann man den Windows-Fontmapper beeinflussen, um die Wahrscheinlichkeit, dass z.B. eine Druckerschriftart gewählt wird, zu erhöhen.

Der hier angegebene Wert wird für das LOGFONT.lfOutPrecision-Feld verwendet, um Schriftart-Instanzen zu erzeugen.

Eine Dokumentation der verfügbaren Werte finden Sie im MSDN. Nicht alle Druckertreiber unterstützen diese Option korrekt, hier müssen Sie im Zweifelsfall bei Ihrem Hersteller nachfragen.

Voreinstellung: OUT_STRING_PRECIS.

LL_OPTION_FORCE_DEFAULT_PRINTER_IN_PREVIEW

TRUE: Die Druckernamenseinstellungen werden an den Preview nicht übergeben, so dass dieser immer den Standard-Drucker wählt.

FALSE: Die Druckernamenseinstellungen werden übergeben (Voreinstellung)

LL_OPTION_FORCEFONTCHARSET

Wählt aus, ob man alle auf dem System verfügbaren Fonts angezeigt bekommt (siehe auch *LL_OPTION_SCALABLEFONTSONLY*), oder nur die, die Zeichen in der eingestellten LCID (siehe *LL_OPTION_LCID*) anbieten.

Voreinstellung: FALSE

LL_OPTION_FORCEFIRSTGROUPHEADER

Über diese Option kann erzwungen werden, dass der erste Gruppenkopf auch ausgegeben wird, wenn das Ergebnis der Evaluation der "Gruppieren nach"-Eigenschaft leer ist.

Voreinstellung: FALSE

LL_OPTION_HELPAVAILABLE

TRUE: Hilfe-Buttons anzeigen (Voreinstellung)

FALSE: Hilfe-Buttons unterdrücken

LL_OPTION_IMMEDIATELASTPAGE

FALSE: das *LastPage()*-Flag wird erst gesetzt, wenn alle Objekte (bis zu einer Tabelle, wenn Tabellendruck) gedruckt wurden.

TRUE: wenn ein Objekt einen Seitenumbruch braucht, wird sofort *LastPage()* auf *FALSE* gesetzt und alle angehängten Objekte dieses Objekts werden neu berechnet.

Voreinstellung: TRUE

LL_OPTION_INCREMENTAL_PREVIEW

TRUE: die Vorschau wird sofort nach Erzeugung der ersten Seite angezeigt und weitere Seiten nach und nach der Anzeige hinzugefügt. Wenn der Anwender das Vorschaufenster während des Druckes schließt, erhalten Sie *LL_ERR_USER_ABORTED* von den Druckfunktionen zurück. Dieser Fehlercode muss also in jedem Falle verarbeitet werden. Wird zum Abschluss des Drucks *LlPreviewDisplay()* aufgerufen, so kehrt diese API erst dann zurück, wenn der Anwender das Vorschaufenster schließt.

FALSE: die Vorschau wird nicht sofort angezeigt, die Anwendung muss explizit *LlPreviewDisplay()* zur Anzeige aufrufen.

Voreinstellung: TRUE

LL_OPTION_INTERCHARSPACING

TRUE: Wenn Text im Blocksatz dargestellt werden soll, wird der leere Raum nicht nur auf die Leerzeichen, sondern auch zwischen den Zeichen verteilt.

FALSE: hier wird der Leerraum nur auf die Leerzeichen verteilt (Voreinstellung)

LL_OPTION_INCLUDEFONTDESCENT

TRUE: für die Berechnung von Zeilenabständen wird auch der zum Font gehörende Parameter *LOGFONT.lfDescent* verwendet. Dies führt zu etwas vergrößerten Zeilenabständen, verhindert jedoch das Abschneiden von extremen Unterlängen (Voreinstellung ab Version 13).

FALSE: kompatibler Modus

LL_OPTION_LCID

Über diese Option setzen Sie die Voreinstellungen für die locale-abhängigen Einträge (Einheit metrisch/Inch, Dezimalpunkt, Tausendertrennzeichen und Währungssymbol) und für Schriften (siehe *LL_OPTION_FORCEFONTCHARSET*).

Außerdem ist dies die Default-Locale für die Formelfunktionen *Loc...$()* und *Date$()*.

Voreinstellung: *LOCALE_USER_DEFAULT*.

LL_OPTION_LOCKNEXTCHARREPRESENTATIONCODE

Diese Option bestimmt den Code des "Umbruch-Sperrung"-Zeichens.

Dieser Wert muss u.U. für den Einsatz mit anderen Sprachen/ Zeichensätzen geändert werden, da der Default-Code in DBCS-Zeichensystemen evtl. verwendet wird. Anstelle dessen können Sie meist auch Code 160 (NO BREAK SPACE) verwenden.

LL_OPTION_MAXRTFVERSION

Unter Windows gibt es z.Z. 5 verschiedene RTF-Control Versionen (Versionsnummer 1, 2, 3, 4 und 4.1). Diese unterscheiden sich in dem Umfang der Möglichkeiten wie auch in der Art der Hintergrundzeichnung.

Sie können über diese Option angeben, welches RTF-Control maximal von List & Label verwendet werden soll. So bewirkt ein Setzen der Option auf 0x100, dass immer das Control Version 1 verwendet wird, ein Setzen auf 0x401 verwendet das Control in Version 4.1.

Wenn Sie diese Option mit 0 aufrufen, verhindert dies, dass das RTF Control geladen wird. Vorteil davon ist, dass List & Label etwas schneller startet und weniger Ressourcen verbraucht werden.

Diese Option muss immer mit Job Handle -1 übergeben werden, noch bevor der erste Job geöffnet wird.

LL_OPTION_METRIC

TRUE: Metrisches System wird eingestellt.

FALSE: Inch-System wird eingestellt.

Voreinstellung: Windowseinstellung. Siehe auch *LL_OPTION_UNITS*.

LL_OPTION_NOAUTOPROPERTYCORRECTION

FALSE: Zusammenhängende Eigenschaften werden automatisch gemeinsam gesetzt. (Voreinstellung)

TRUE: Zusammenhängende Eigenschaften werden nicht automatisch gemeinsam gesetzt.

Diese Option wird im Zusammenhang mit dem Objektmodell benötigt, wenn automatische Korrekturen an Eigenschaften vermieden werden sollen. So würde z.B. die Eigenschaft "Font.Charset" automatisch auf einen passenden Wert gesetzt, wenn z.B. ein chinesischer Font ausgewählt wird. Ist dies nicht gewünscht, so kann dies über diese Option gesteuert werden.

LL_OPTION_NOFAXVARS

FALSE: Die Variablen für den Faxversand sind im Designer sichtbar (Voreinstellung).

TRUE: Die Variablen für den Faxversand sind im Designer nicht sichtbar.

LL_OPTION_NOFILEVERSIONUPGRADEWARNING

Diese Option bestimmt das Verhalten des Designers beim Öffnen und Konvertieren von Dateien aus älteren Versionen.

TRUE: Die Konvertierung erfolgt still im Hintergrund, der Benutzer bemerkt nichts vom Vorgang.

FALSE: Es wird eine Warnmeldung angezeigt, dass das Projekt anschließend nicht mehr mit der älteren Version bearbeitet werden kann (Voreinstellung).

LL_OPTION_NOMAILVARS

FALSE: Die Variablen für den Mailversand sind im Designer sichtbar (Voreinstellung).

TRUE: Die Variablen für den Mailversand sind im Designer nicht sichtbar.

LL_OPTION_NONOTABLECHECK

TRUE: List & Label testet bei einem Listenprojekt nicht, ob mindestens eine Tabelle vorhanden ist (Voreinstellung).

FALSE: Der Check wird beim Laden eines Projekts durchgeführt und ggf. *LL_ERR_NO_TABLEOBJECT* zurückgegeben.

LL_OPTION_NOPARAMETERCHECK

TRUE: List & Label unterdrückt seinen internen Parametercheck bei den Übergabeparametern der API-Funktionen, so dass eine höhere Verarbeitungsgeschwindigkeit erreicht wird.

FALSE: Die Funktions-Parameter werden überprüft (Voreinstellung).

LL_OPTION_NOPRINTERPATHCHECK

TRUE: List & Label checkt nicht, ob die für das Projekt relevanten Drucker existieren. Sofern z.B. Netzwerkdrucker im Projekt eingestellt sind, diese aber im Moment des Gebrauchs nicht oder nur "langsam" verfügbar sind, können Wartezeiten entstehen.

FALSE: List & Label checkt, ob die für das Projekt relevanten Drucker existieren (Voreinstellung). Für den Fall, dass ein Drucker nicht angesprochen werden kann, erfolgt ein automatischer Fallback auf den Standarddrucker, in diesem Fall ist dies aber mit Wartezeit verbunden.

LL_OPTION_NOPRINTJOBSUPERVISION

Über diese Option kann die Druckjobüberwachung eingeschaltet werden (vgl. *LL_INFO_PRINTJOBSUPERVISION)*. Voreinstellung: *TRUE*.

LL_OPTION_NOTIFICATIONMESSAGEHWND

Hier geben Sie das Fenster explizit an, an das List & Label seine Callbacks sendet (falls Sie nicht die Callback-Prozedur verwendet haben).

Normalerweise werden Events an das in der Fensterhierarchie nächste Nicht-Child-Fenster von dem bei *LIDefineLayout()* oder *LIPrintWithBoxStart()* übergebenen Fenster gesendet, hier können Sie explizit ein Zielfenster angeben.

Voreinstellung: NULL

LL_OPTION_NULL_IS_NONDESTRUCTIVE

List & Label behandelt dort, wo möglich, NULL-Werte entsprechend der SQL-92 Spezifikation. Ein wichtiger Effekt dabei ist, dass Funktionen und Operatoren, die NULL-Werte als Parameter bzw. Operanden bekommen in der Regel als Ergebnis auch wieder NULL zurückliefern. Ein Beispiel dafür ist die folgende Designerformel:

Titel+" "+Vorname+" "+Nachname

Wenn der Titel mit NULL gefüllt ist, ist das Ergebnis dieser Formel entsprechend dem Standard ebenfalls NULL. Da es häufig gewünscht sein kann, in einem solchen Fall stattdessen eben (sinngemäß)

Vorname+" "+Nachname

zu erhalten, kann über die Option *LL_OPTION_NULL_IS_NONDESTRUCTIVE* erreicht werden, dass NULL-Werte entgegen der Spezifikation eben nicht den ganzen Ausdruck zu NULL machen, sondern je nach Datentyp zu "0", einem Leerstring oder einem ungültigen Datum werden. Die bessere Alternative ist aber, mit der *NULLSafe()*-Designerfunktion zu arbeiten, da dann der Ersetzungswert im NULL-Fall genau bestimmt werden kann.

TRUE: NULL-Werte werden durch Ersatzwerte dargestellt.

FALSE: NULL-Werte als Operanden oder Parameter ergeben NULL als Funktionswert (Voreinstellung).

LL_OPTION_PHANTOMSPACEREPRESENTATIONCODE

Diese Option bestimmt den Code des "Phantom Space"-Zeichens.

Dieser Wert muss u.U. für den Einsatz mit anderen Sprachen/ Zeichensätzen geändert werden, da der Default-Code in DBCS-Zeichensystemen evtl. verwendet wird.

LL_OPTION_PRINTERDCCACHE_TIMEOUT_SECONDS

Diese Option bestimmt, wie lange (in Sekunden) Druckergerätekontexte zwischengespeichert werden dürfen. Beachten Sie: manche Drucker starten einen Druckjob erst dann, wenn der Gerätekontext geschlossen wird. In diesen Fällen muss der Wert der Option auf "0" geändert werden. Die Voreinstellung ist "60".

LL_OPTION_PRINTERDEVICEOPTIMIZATION

TRUE: Drucker die bezogen auf das DEVMODE Ergebnis identisch sind werden "wegoptimiert" (Voreinstellung). Dies bedingt auch, dass Druckaufträge über Seiten hinweg zusammengefasst werden können, wenn an zwei getrennten Stellen im Bericht die gleiche Druckerkonfiguration verwendet wird. Um dies zu vermeiden sollten Sie die Option ggf. auf FALSE setzen.

FALSE: Alle Drucker werden angezeigt, Druckaufträge werden nicht zusammengefasst.

LL_OPTION_PROHIBIT_USERINTERACTION

TRUE: Hinweismeldungen und Dialoge werden nicht angezeigt. Hinweismeldungen werden automatisch den voreingestellten Wert zurückliefern. Diese Option wird üblicherweise automatisch in Webserver-Umgebungen gesetzt. Wenn aus irgendeinem Grund die Webserver-Erkennung fehlschlagen sollte, können Sie den nicht-UI Modus manuell über diese Option setzen.

FALSE: Hinweismeldungen und Dialoge werden angezeigt (Voreinstellung).

LL_OPTION_PROJECTBACKUP

TRUE: beim Bearbeiten eines Projektes im Designer wird eine Sicherungskopie angelegt (Voreinstellung).

FALSE: eine Sicherungskopie wird nicht angelegt.

LL_OPTION_PRVZOOM_LEFT, LL_OPTION_PRVZOOM_TOP, LL_-OPTION_PRVZOOM_WIDTH, LL_OPTION_PRVZOOM_HEIGHT

Die anfängliche Größe des Echtdatenpreview-Fensters in Prozent der Bildschirmfläche. Initiell werden die Positionen genommen, die das Fenster beim letzten Beenden hatte.

LL_OPTION_PRVRECT_LEFT, LL_OPTION_PRVRECT_TOP, LL_OPTION_PRV-RECT_WIDTH, LL_OPTION_PRVRECT_HEIGHT

Dasselbe in Bildschirmkoordinaten.

LL_OPTION_PRVZOOM_PERC

Der initielle Echtdatenpreview-Zoomfaktor in Prozent (Voreinstellung: 100). Ein Wert von -100 stellt die Vorschau in Seitenbreite dar.

LL_OPTION_REALTIME

TRUE: *Die Time()* und *Now()*-Funktionen berechnen immer wieder die Zeit neu.

FALSE: Die Zeit wird beim Laden des Projekts genommen und immer wieder verwendet (Voreinstellung).

LL_OPTION_RESETPROJECTSTATE_FORCES_NEW_DC

TRUE: Nach *LlPrintResetProjectState()* wird der Ausgabe-Gerätekontext neu erzeugt (Voreinstellung).

FALSE: Der Gerätekontext bleibt nach *LlPrintResetProjectState()* erhalten. Wird nicht von allen Druckern unterstützt, bringt aber Performancevorteile im Seriendruck mit sich.

LL_OPTION_RESETPROJECTSTATE_FORCES_NEW_PRINTJOB

TRUE: Nach *LlPrintResetProjectState()* wir immer ein neuer Druckjob forciert. Diese Option ist insbesondere dann von Nutzen, wenn das gleiche Projekt mehrfach hintereinander gedruckt wird und wichtig ist, dass jeder der Drucke im Spooler einen eigenen Job erzeugt.

FALSE: Mehrere Berichte können in einen Druckjob zusammengefasst werden. Ein neuer Druckjob wird nur dann erzeugt, wenn er aus Gründen des Duplexdrucks benötigt wird (Voreinstellung).

LL_OPTION_RETREPRESENTATIONCODE

Wert des neuen Zeichens, das einen Zeilenumbruch repräsentiert (Voreinstellung: '¶').

Dieser Wert muss u.U. für den Einsatz mit anderen Sprachen/ Zeichensätzen geändert werden, da der Default-Code in DBCS-Zeichensystemen evtl. verwendet wird.

LL_OPTION_RIBBON_DEFAULT_ENABLEDSTATE

TRUE: Ab Windows Vista verwendet der Designer und der Vorschaudruck das Windows Scenic Ribbon-Framework. Der Ribbon kann im Optionsdialog des Projekts ausgeschaltet werden (default).

FALSE: Die Verwendung des Ribbons muss im Projektoptionsdialog explizit eingeschaltet werden.

LL_OPTION_RTFHEIGHTSCALINGPERCENTAGE

Prozentwert, um der Platz um den RTF Text skaliert wird, so dass MS Word RTF Text komplett darstellt (Voreinstellung: 100).

LL_OPTION_SCALABLEFONTSONLY

Sollen in den Schriftartdialogen alle (*FALSE*) oder nur die Vektorschriften (*TRUE*) auswählbar sein?

Rasterschriften haben den Nachteil, nicht gut skalierbar zu sein und daher in der Vorschau seltsame Ausgaben zu produzieren.

Voreinstellung: TRUE

LL_OPTION_SETCREATIONINFO

TRUE: List & Label speichert diverse Benutzerdaten (User- und Rechnername sowie Datum/Uhrzeit für Erstellung und letzte Veränderung) in das Projekt und in eine Vorschaudatei. Dies kann für Fehlersuche interessant sein (Voreinstellung)

FALSE: keine Speicherung

LL_OPTION_SHOWPREDEFVARS

TRUE: Die von List & Label vordefinierten Variablen werden im Formelassistenten angeboten (Voreinstellung).

FALSE: Die von List & Label vordefinierten Variablen werden im Formelassistenten nicht angeboten.

LL_OPTION_SKETCH_COLORDEPTH

Über diese Option kann man festlegen, mit welcher Farbtiefe (Bits) die Sketch-Dateien für die Dateidialoge erzeugt bzw. gespeichert werden. Voreinstellung ist 8, also 256 Farben. 32 wäre beispielsweise True-Color.

LL_OPTION_SKIPRETURNATENDOFRTF

Am Ende von RTF-Texten können Zeilenumbrüche stehen.

TRUE: Sie werden entfernt (verhindert doppelten Abstand am Ende des RTF).

FALSE: belässt die Daten so, wie sie übergeben werden (Voreinstellung).

LL_OPTION_SORTVARIABLES

TRUE: Die Variablen und Felder werden in den Auswahldialogen alphabetisch sortiert.

FALSE: Die Variablen und Felder werden in den Auswahldialogen nicht sortiert (Voreinstellung), d.h. die Anmeldereihenfolge ist entscheidend.

LL_OPTION_SPACEOPTIMIZATION

TRUE: Die "Leerzeichenoptimierung" ist bei neuen Abschnitten in Textobjekten oder bei neuen Tabellenspalten eingeschaltet (Voreinstellung)

FALSE: Sie ist ausgeschaltet.

Dies gilt nicht für vorhandene Objekte!

LL_OPTION_SUPERVISOR

TRUE: es sind alle Menü-Optionen erlaubt, und auch gesperrte Objekte sind nicht gesperrt. Dieser Modus erlaubt es, ohne große zusätzliche Programmierung die dem "normalen" Benutzer unzugänglichen Teile zu benutzen.

FALSE: Eventuelle Einschränkungen sind wirksam. (Voreinstellung).

LL_OPTION_SUPPORT_HUGESTORAGEFS

TRUE: Vorschaudateien können beliebig groß werden. Die Vorschaudateien sind nur ab Windows 2000 lesbar.

FALSE: Die Größe der Vorschaudateien ist auf 2 GB beschränkt. Die Vorschaudateien sind in allen unterstützten Betriebssystemen lesbar (Voreinstellung).

LL_OPTION_SUPPORTS_PRNOPTSTR_EXPORT

TRUE: Der Benutzer kann im Designer ein Druckziel als "voreingestelltes Ausgabemedium " wählen, der dann im Druckoptionendialog voreingestellt ist.

FALSE: Normalerweise setzt die Applikation das voreingestellte Ausgabemedium.

Das voreingestellte Ausgabemedium wird im Projekt-File gespeichert.

Voreinstellung: FALSE

LL_OPTION_TABLE_COLORING:

LL_COLORING_LL: das Kolorieren von Listenobjekten wird nur von List & Label durchgeführt. (Voreinstellung)

LL_COLORING_PROGRAM: das Kolorieren von Listenobjekten wird nur über Notifications oder Callback durchgeführt (s. Kapitel "Notifications und Callbacks"), Farbeinstellung im Designer ist nicht möglich

LL_COLORING_DONTCARE: das Kolorieren wird erst vom Programm über Notification oder Callback durchgeführt, dann aber bei Bedarf noch von List & Label.

LL_OPTION_TABREPRESENTATIONCODE

Zeichencode für das Zeichen, das den Tabulator repräsentiert.

Dieser Wert muss u.U. für den Einsatz mit anderen Sprachen/ Zeichensätzen geändert werden, da der Default-Code in DBCS-Zeichensystemen evtl. verwendet wird.

LL_OPTION_TABSTOPS:

LL_TABS_DELETE: Tabulatoren (Zeichen-Code 0x09!) im Text von Textobjekten werden durch Leerzeichen ersetzt (Voreinstellung)

LL_TABS_EXPAND: Tabulatoren im Text werden wie üblich erweitert auf 8-Zeichen-Bündigkeit. Bei Proportionalschriften macht nur ein Tabulator am Zeilenanfang Sinn.

LL_OPTION_UISTYLE

Über diese Option kann das Look & Feel des Designers gesteuert werden.

LL_OPTION_UISTYLE_STANDARD: Office 97® Look & Feel.

LL_OPTION_UISTYLE_OFFICEXP: Office XP®/Visual Studio .NET® Look & Feel.

LL_OPTION_UISTYLE_OFFICE2003: Office 2003® Look & Feel (Voreinstellung).

LL_OPTION_UNITS

Werte und Beschreibung siehe *LL_PRNOPT_UNIT*

LL_OPTION_USEBARCODESIZES

TRUE: Manche Barcodes haben Standardgrößen oder einen Standardgrößenbereich. Wenn diese Option enabled ist (TRUE), kann der Benutzer im Barcodeobjekt die Einschränkung auf die Standardgrößen einschalten und über die Maus dann keine falschen Größen mehr einstellen.

FALSE: Es können alle Größen eingestellt werden (Voreinstellung).

LL_OPTION_USECHARTFIELDS

TRUE: nach Einschalten dieser Option müssen Charts von der Anwendung her über die Chart-APIs versorgt werden.

FALSE: kompatibler Modus, Charts werden über die Funktion *LlPrintFields()* mit Daten versorgt (Voreinstellung).

Bitte lesen Sie hierzu auch das entsprechende Kapitel in diesem Handbuch.

LL_OPTION_USEHOSTPRINTER

TRUE: Überlässt die Verwaltung des Druckers dem Anwenderprogramm, Hinweise zu den dann zu übernehmenden Aufgaben finden Sie in Kapitel "Referenz der Callback-Notifications".

FALSE: List & Label verwaltet den Drucker (Voreinstellung).

LL_OPTION_USE_JPEG_OPTIMIZATION

TRUE: List & Label bettet JPEG-Dateien als JPEG-Stream in die Vorschaudateien ein. Diese werden somit wesentlich kleiner, die in der Vorschau enthaltenen Metafiles können aber nur innerhalb von List & Label richtig angezeigt werden und eignen sich nicht zur Weiterverarbeitung z.B. in Bildeditoren (Voreinstellung).

FALSE: JPEG-Dateien werden als Bitmap-Records in die Metafiles eingefügt. Die Vorschaudateien werden somit wesentlich größer.

LL_OPTION_VARSCASESENSITIVE

TRUE: Die Variablen- und Feldnamen sind großschreibungssensitiv.

Vorsicht: Dennoch dürfen auch dann sich Variablen-/Feldnamen nicht nur in der Groß-/Kleinschreibung unterscheiden!

FALSE: Die Groß- und Kleinschreibung wird ignoriert: "Name" ist gleichwertig mit "NAME" (Voreinstellung).

LL_OPTION_XLATVARNAMES

TRUE: Sonderzeichen in Variablen- und Feldbezeichnern werden zu einem '_' umgewandelt (Voreinstellung).

FALSE: Variablen- und Feldbezeichner werden nicht modifiziert. Dies bedeutet - je nach Zahl der Variablen und Felder - eine nicht unerhebliche Geschwindigkeitssteigerung.

nValue: neuer Wert

Rückgabewert:

Fehlercode

Hinweise:

Bitte rufen Sie diese Funktion vor *LlDefineLayout()* und vor den *LlPrint...Start()*-Funktionen auf, am besten also direkt nach *LlJobOpen()*/ *LlJobOpenLCID()*.

Beispiel:

```
HLLJOB hJob;
BOOL     b;

hJob = LlJobOpen(0);
LlSetOption(hJob, LL_OPTION_NEWEXPRESSIONS, TRUE);
LlSetOption(hJob, LL_OPTION_VARSCASESENSITIVE, FALSE);
// ....
LlJobClose(hJob);
```

Siehe auch:

LlGetOption

LlSetOptionString

Syntax:

```
INT LlSetOptionString (HLLJOB hJob, INT nMode, LPCTSTR pszValue);
```

Aufgabe:

Setzt diverse Einstellungen in List & Label.

Parameter:

hJob: List & Label Job-Handle

nMode: Folgende Werte sind als Funktionsindex möglich:

LL_OPTIONSTR_CARD_PRJEXT

Die Dateinamenerweiterung ("Extension") eines Karteikartenprojekts. Voreinstellung:"crd".

LL_OPTIONSTR_CARD_PRVEXT

Die Dateinamenerweiterung ("Extension") der Bitmap eines Karteikartenprojekts, die im Datei-Lade-Dialog angezeigt wird. Voreinstellung: "crv".

LL_OPTIONSTR_CARD_PRNEXT

Die Dateinamenerweiterung ("Extension") der Druckereinstellungen eines Karteikartenprojekts. Voreinstellung: "crp".

LL_OPTIONSTR_CURRENCY

Die Zeichenkette, die als Währungssymbol ("€", "$", ...) zur Formatierung von Währungen bei der *FStr$()*-Funktion verwendet wird.

Wird automatisch auf die Benutzer-Einstellungen eingestellt bzw. auf die entsprechend lokalisierte Zeichenkette, wenn *LL_OPTION_LCID* eingestellt wird.

LL_OPTIONSTR_DECIMAL

Die Zeichenkette, die als Dezimalzeichen bei der *FStr$()*-Funktion verwendet wird.

Wird automatisch auf die Benutzer-Einstellungen eingestellt bzw. auf die entsprechend lokalisierte Zeichenkette, wenn *LL_OPTION_LCID* eingestellt wird.

LL_OPTIONSTR_DEFDEFFONT

Diese Option beschreibt die Schriftart, die als Voreinstellung für die Projekt-Schriftart verwendet wird.

Die Zeichenkette muss das Format haben:

"{(R,G,B),H, L}"
R = Rotanteil der Farbe, G = Grünanteil der Farbe, B = Blauanteil der Farbe, H = Höhe der Schriftart in Punkten, L = kommaseparierte Felder der LOGFONT-Struktur (siehe SDK)

Diese Schriftart kann auch über *LL_OPTION_DEFDEFFONT* gesetzt oder abgefragt werden.

LL_OPTIONSTR_EXPORTS_ALLOWED

Über diese Funktion können Sie die möglichen Ausgabemedien beschränken, die der Benutzer bei *LlPrintOptionsDialog()* oder *LlPrintOptionsDialogTitle()* angezeigt bekommt. Im Designer stehen nur die hier erlaubten Exportformate unter *Projekt > Seitenlayout > Ausgabemedien* zur Verfügung.

Übergeben Sie hier auch eine semikolonseparierte Liste mit Kürzeln der Ausgabemedien, siehe *LL_OPTIONSTR_EXPORTS_AVAILABLE*

LL_OPTIONSTR_EXPORTS_ALLOWED_IN_PREVIEW

Über diese Funktion können Sie die möglichen Ausgabe-Medien beschränken, die der Benutzer im Vorschaufenster angezeigt bekommt. Übergeben Sie hier auch eine semikolonseparierte Liste mit Kürzeln der Ausgabemedien, siehe *LL_OPTIONSTR_EXPORTS_AVAILABLE*

LL_OPTIONSTR_EXPORTS_AVAILABLE

Über diese Funktion können Sie die möglichen Ausgabe-Medien abfragen (read only)

Der Rückgabewert ist eine Zeichenkette, die aus den Kürzeln aller verfügbaren Ausgabe-Medien besteht, z.B.:

"PRN;PRV;FILE;HTML;RTF"

Folgende Kürzel sind definiert, sofern die entsprechenden Exportmodule vorhanden sind:

Wert	Bedeutung
PRN	Ausgabe direkt auf Drucker
PRV	Vorschau-(Datei-)Druck
FILE	Drucker-Ausgabe auf Datei
DOCX	Microsoft Word Format
HTML	HTML Format
MHTML	Multi-Mime HTML Format
JQM	HTML jQuery Mobile Format
PDF	Adobe PDF Format
PICTURE_BMP	Bitmap-Grafik
PICTURE_EMF	Metafile-Grafik (EMF)
PICTURE_JPEG	JPEG-Grafik
PICTURE_PNG	PNG-Grafik
PICTURE_MULTITIFF	Multi-TIFF-Grafik
PICTURE_TIFF	TIFF-Grafik
RTF	Rich Text Format (RTF)
SVG	SVG Format
TTY	Nadeldrucker (TTY)
TXT	Text (CSV) Format
TXT_LAYOUT	Text (Layout) Format
XLS	Microsoft Excel Format
XHTML	XHTML/CSS Format

Wert	Bedeutung
XML	XML Format
XPS	Microsoft XPS Format

Die weiteren Einträge hängen von der über *LlSetOptionString(LL_OPTIONSTR_-LLXPATHLIST*) gefundenen Liste von Exportmodulen ab.

Beispiel:

```
LlPrintStart(hJob,..., LL_PRINT_EXPORT,...);

// allow only printer and preview (EXPORT sets all bits)
LlSetOption(hJob, LL_OPTIONSTR_EXPORTS_ALLOWED, "PRN;PRV");
// Default should be preview!
LlPrintSetOptionString(hJob, LL_PRNOPTSTR_EXPORT, "PRV");

// printer dialog allows user to change
LlPrintOptionsDialog(hJob,....);
// get the final medium:
LlPrintGetOptionString(hJob, LL_PRNOPTSTR_-
EXPORT,sMedium,sizeof(sMedium));
// ...print job....
// finished
LlPrintEnd(hJob,0);

if (strcmp(sMedium, "PRV") == 0) ...
```

LL_OPTIONSTR_EXPORTFILELIST

Diese Option kann nur gelesen und nicht gesetzt werden.

Über diese Funktion können Sie nach dem Druck abfragen, in welche Dateien ausgegeben wurde. Die Dateinamen sind Semikolon-separiert.

Insbesondere bei den Exportmodulen gilt: reservieren Sie genug Platz, oder vergrößern Sie den Puffer solange, bis Sie von *LlGetOptionString()* keine Fehlermeldung mehr bekommen (*LL_ERR_BUFFERTOOSMALL*).

Die Liste ist erst nach *LlPrintEnd()* verfügbar.

LL_OPTIONSTR_HELPFILENAME

Über diese Funktion können Sie den Namen der Hilfedatei vorgeben, wenn Sie z.B. Ihre eigene Hilfedatei angezeigt bekommen möchten.

LL_OPTIONSTR_FAX_RECIPNAME,
LL_OPTIONSTR_FAX_RECIPNUMBER,
LL_OPTIONSTR_FAX_SENDERNAME,
LL_OPTIONSTR_FAX_SENDERCOMPANY ,
LL_OPTIONSTR_FAX_SENDERDEPT,
LL_OPTIONSTR_FAX_SENDERBILLINGCODE

Mit diesen Optionen können Default-Werte für die Einstellungen im Faxdialog (**Projekt > Fax-Variablen**) gesetzt werden. Wenn der Benutzer im Designer andere Formeln für die Einstellungen wählt bzw. im Projekt andere Einstellungen gespeichert wurden, so haben diese Vorrang. Wenn das Projekt auf das Fax-Exportmodul ausgedruckt wird, werden die hier bzw. im Projekt gewählten Einstellungen ausgewertet und das Projekt direkt als Fax verschickt. Alternativ bieten viele Faxprogramme die Möglichkeit, die Faxnummer in besonderer Formatierung zu übergeben. Details hierzu müssen Sie bei Ihrem Faxtreiber-Hersteller erfragen.

Wenn die Optionen nicht gesetzt werden und der Benutzer im Projekt keine Einstellungen gewählt hat, steht das Fax Export-Modul nicht zur Verfügung.

LL_OPTIONSTR_LABEL_PRJDESCR, LL_OPTIONSTR_CARD_PRJDESCR, LL_-OPTIONSTR_LIST_PRJDESCR

Mit diesen Parametern können Sie die Beschreibung des entsprechenden Projekttypen bestimmen. Diese Beschreibungen erscheint in der Dateityp- Combobox der Laden- und Speichern-Dialoge.

LL_OPTIONSTR_LABEL_PRJEXT

Die Dateinamenerweiterung ("Extension") eines Etikettenprojekts. Voreinstellung: "lbl".

LL_OPTIONSTR_LABEL_PRVEXT

Die Dateinamenerweiterung ("Extension") der Bitmap eines Etikettenprojekts, die im Datei-Lade-Dialog angezeigt wird. Voreinstellung: "lbv".

LL_OPTIONSTR_LABEL_PRNEXT

Die Dateinamenerweiterung ("Extension") der Druckereinstellungen eines Etikettenprojekts. Voreinstellung: "lbp".

LL_OPTIONSTR_LICENSINGINFO

Mit dieser Option wird der persönliche Lizensierungsschlüssel übergeben, der auch den Lizenzumfang bestimmt.

Diese Option muss *unbedingt* gesetzt werden, bevor ein Projekt distributiert wird. Weitere Informationen finden sich in der Datei "perslic.txt" sowie "re-

dist.txt" im List & Label-Installationsverzeichnis. In der Trialversion ist das Setzen dieser Option nicht notwendig.

LL_OPTIONSTR_ORIGINALPROJECTFILENAME

Wird benötigt, um Dateien mit Relativpfad korrekt in der Designer-Vorschau anzeigen zu können. Weitere Informationen im Kapitel "Direkter Druck und Export aus dem Designer".

LL_OPTIONSTR_LIST_PRJEXT

Die Dateinamenerweiterung ("Extension") eines Listenprojekts. Voreinstellung: "lst".

LL_OPTIONSTR_LIST_PRVEXT

Die Dateinamenerweiterung ("Extension") der Bitmap eines Listenprojekts, die im Datei-Lade-Dialog angezeigt wird. Voreinstellung: "lsv".

LL_OPTIONSTR_LIST_PRNEXT

Die Dateinamenerweiterung ("Extension") der Druckereinstellungen eines Listenprojekts. Voreinstellung: "lsp".

LL_OPTIONSTR_LLFILEDESCR

hierüber wird die Beschreibung für die List & Label Preview-Dateien übergeben, die in der Dateityp-Combobox des Speichern-Dialogs im Preview angezeigt wird.

LL_OPTIONSTR_LLXPATHLIST

Diese Option bestimmt die zu ladenden externen LLX-Module. Übergeben Sie hier eine semikolonseparierte Liste mit den Pfadnamen der Export-Module, die Sie in Ihrer Applikation eingebunden haben möchten.

Als Voreinstellung, also beim Öffnen eines Jobs bzw. bei jedem Aufruf dieser Option, werden folgende Erweiterungen automatisch geladen:

CMLL20PW.LLX, CMLL20HT.LLX, CMLL20EX.LLX, CMLL20OC.LLX

sowie in der Professional-/Enterprise-Edition zusätzlich

CMLL20BC.LLX

Als Pfad dieser Dateien wird der Pfad der List & Label-DLL verwendet.

Sie können Wildcards als Dateimasken verwenden, um mehrere Module auf einmal zu laden.

Um eine Erweiterung zu unterdrücken, geben Sie den Dateinamen mit einem vorangestellten Dach-Symbol an, also "^CMLL20PW.LLX". Um alle voreingestellten Erweiterungen zu unterdrücken, geben Sie "^*" als ersten 'LLX-Namen' an, beispielsweise um einen anderen Pfad zu verwenden.

Bei der Abfrage über *LlGetOptionString()* wird eine semikolonseparierte Liste der geladenen Module zurückgegeben.

Bei eingeschaltetem Debug-Modus gibt List & Label aus, welche Module auf Grund welcher Regeln geladen oder entladen werden.

LL_OPTIONSTR_MAILTO

Hier kann eine Adresse übergeben werden, die als voreingestellter Empfänger bei dem Senden aus der Vorschau in die eMail eingetragen wird. Mehrere Adressen können semikolonsepariert übergeben werden.

LL_OPTIONSTR_MAILTO_CC

Hier kann eine Adresse übergeben werden, die als voreingestellter Empfänger bei dem Senden aus der Vorschau in die eMail als CC (Carbon Copy) eingetragen wird. Mehrere Adressen können semikolonsepariert übergeben werden.

LL_OPTIONSTR_MAILTO_BCC

Hier kann eine Adresse übergeben werden, die als voreingestellter Empfänger bei dem Senden aus der Vorschau in die eMail als BCC (Blind Carbon Copy) eingetragen wird. Mehrere Adressen können semikolonsepariert übergeben werden.

LL_OPTIONSTR_MAILTO_SUBJECT

Hier kann ein Text übergeben werden, der bei dem Senden aus der Vorschau in die eMail als Betreff eingetragen wird.

LL_OPTIONSTR_NULLVALUE

Hier kann ein Text übergeben werden, der im Druck als Darstellung eines NULL-Wertes ausgegeben wird. Voreistellung: Leer ("").

LL_OPTIONSTR_PREVIEWFILENAME

Über diese Option kann der Name vorgegeben werden, der für die Erstellung von Preview-Dateien verwendet wird. Per Voreinstellung werden die Dateien im Verzeichnis des Projektes bzw. in einem alternativen Verzeichnis (s. *LlPreviewSetTempPath()*) mit dem Namen <Projektname>.LL angelegt. Mit dieser Option kann ein anderer Dateiname gewählt werden, der dann <Projektname> ersetzt.

LL_OPTIONSTR_PRINTERALIASLIST

Über diese Option kann eine Übersetzungstabelle für Drucker übergeben werden.

Um die Tabelle zu löschen, übergibt man NULL oder einen Leerstring.

Danach übergibt man für jeden Quelldrucker eine Übersetzungstabelle mit altem und einem oder mehreren neuen Druckern. Entweder geschieht das durch mehrere Aufrufe dieser Funktion hintereinander, oder man trennt die verschiedenen Tabellen mit '\n'.

Die Form einer Tabelle selbst ist:

"alter Drucker=neuer Drucker1[;neuer Drucker2[;...]]"

also beispielsweise

"\\server\eti=\\server\eti1;\\server_eti2"
"\\server\a4fast=\\server\standard"

Wenn nun der Drucker \\server\eti nicht benutzt werden kann, wird die Alias-Liste (in der definierten Reihenfolge) durchlaufen, bis einer der Drucker benutzt werden kann (oder die Liste zu Ende ist). Dabei bleibt das gewählte Seitenformat erhalten. Die Groß-/Kleinschreibung ist nicht signifikant.

LL_OPTIONSTR_PROJECTPASSWORD

Erlaubt es, Projekte zu verschlüsseln, um sie vor Verwendung in anderen Applikationen zu schützen. Das hier übergebene Passwort dient als Schlüssel, daher sollten Sie ein möglichst langes Passwort mit einer Vielzahl verschiedener Zeichen wählen, um die Verschlüsselung sicherer zu machen.

Im Designer ist es auch, beispielsweise zu Support-Zwecken, möglich, das Projekt unverschlüsselt zu speichern. Halten Sie hierzu die "SHIFT"-Taste gedrückt und speichern Sie das Projekt. Es erscheint ein Passwort-Dialog, in dem Sie das Passwort eingeben müssen.

Die Qualität der Verschlüsselung ist von der Passwort-Länge abhängig. Die Maximallänge sind 5 Zeichen, das entspricht 40 bit Verschlüsselung, vorausgesetzt dass die ASCII-Werte der Zeichen frei zwischen 1 und 255 verteilt sind. Das Passwort ist nicht (!) absolut sicher, da es per API übergeben wird. Die Hürde für den Projekt-Diebstahl ist aber sicherlich recht hoch.

LL_OPTIONSTR_SAVEAS_PATH

Der hierüber übergebene Parameter wird als voreingestellter Pfad für den "Speichern Als"-Vorgang im Preview benutzt. Der Pfad beinhaltet Verzeichnis und Dateiname.

LL_OPTIONSTR_SHORTDATEFORMAT

Die Zeichenkette, die zur automatischen Umwandlung eines Datums in eine Zeichenkette verwendet wird bei:

Date$(<Datum>, "%x")

automatischer Typkonvertierung (*LlExprEval(), Concat$()*)

Format und Voreinstellung: Siehe Windows API *GetLocaleInfo(LOCALE_USER_-DEFAULT, LOCALE_SSHORTDATE,...)*

LL_OPTIONSTR_THOUSAND

Die Zeichenkette, die als Tausenderzeichen bei der *fstr$()*-Funktion verwendet wird.

Wird automatisch auf die Benutzer-Einstellungen eingestellt bzw. auf die entsprechend lokalisierte Zeichenkette, wenn *LL_OPTION_LCID* eingestellt wird.

LL_OPTIONSTR_VARALIAS

Diese Option ermöglicht es, die in einem Projekt verwendeten Variablen- und Feldnamen zu lokalisieren, indem für den Namen ein Alias angegeben wird. Im Designer wird immer nur der Alias angezeigt, während im Projektfile immer nur der tatsächliche Name gespeichert wird. So kann durch Austausch des Alias' eine lokalisierte Version der Variablen- und Feldliste erreicht werden. Die Option muss für jede zu lokalisierende Variable einmal in der Form "<lokal>=<global>" gesetzt werden, also z.B.

```
LlSetOptionString(hJob, LL_OPTIONSTR_VARALIAS, "Vorname=FirstName");
LlDefineVariable(hJob, "FirstName", "John");
```

um eine Variable "FirstName" anzumelden, die im Designer als "Vorname" angezeigt wird.

Um alle angemeldeten Alias-Namen wieder zu löschen, rufen Sie einfach

```
LlSetOptionString(hJob, LL_OPTIONSTR_VARALIAS, "");
```

Auf.

Die .NET, OCX und VCL-Komponenten bieten eine eigene Dictionary-API an, mit der Ihnen das Setzen der Option komfortabel abgenommen wird. Details finden Sie in der Online-Hilfe der Komponenten.

pszValue: neuer Wert

Rückgabewert:

Fehlercode

Beispiel:

```
HLLJOB  hJob;

hJob = LlJobOpen(0);
LlSetOptionString(hJob, LL_OPTIONSTR_LIST_PRJEXT, "list");
// ....
LlJobClose(hJob);
```

Siehe auch:

LlGetOptionString

LlSetPrinterDefaultsDir

Syntax:

```
INT LlSetPrinterDefaultsDir (HLLJOB hJob, LPCTSTR pszDir);
```

Aufgabe:

Setzt den Pfad für die Druckerbeschreibungsdatei (P-Datei), um z.B. im Netzwerkbetrieb benutzerspezifische Druckereinstellungen speichern zu können.

Parameter:

hJob: List & Label Job-Handle

pszDir: Der Pfadname des gewünschten Verzeichnisses

Rückgabewert:

Fehlercode

Beispiel:

```
HLLJOB   hJob;
hJob = LlJobOpen(0);
LlSetPrinterDefaultsDir(hJob, "c:\\temp\\user");
if (LlPrintStart(hJob, LL_PROJECT_LIST, "c:\\test.lst",
         LL_PRINT_NORMAL) == 0)
{
   <... etc ...>
   LlPrintEnd(hJob);
}
else
   MessageBox(NULL, "Fehler", "List & Label",MB_OK);
LlJobClose(hJob);
```

Siehe auch:

LlSetPrinterToDefault, LlPrintStart, LlPrintWithBoxStart, LlPrintCopyPrinterConfiguration, LlSetPrinterInPrinterFile

LlSetPrinterInPrinterFile

Syntax:

```
INT LlSetPrinterInPrinterFile (HLLJOB hJob, UINT nObjType,
   LPCTSTR pszObjName, INT nPrinter, LPCTSTR pszPrinter,
   _PCDEVMODE pDM);
```

Aufgabe:

Ermöglicht die Veränderung von Druckerkonfigurationen in der Druckerkonfigurationsdatei.

Parameter:

hJob: List & Label Job-Handle

nObjType: *LL_PROJECT_LABEL, LL_PROJECT_LIST* oder *LL_PROJECT_CARD*

pszObjName: Dateiname des Projekts

nPrinter: Druckerindex (0: Bereich mit "Page()==1" [wird notfalls angelegt], 1: Standardbereich, -1: legt nur den Standardbereich an, ggf. existierende andere Bereiche werden gelöscht).

Alternativ können – wenn im Projekt mehrere Layout-Bereiche verwendet werden - auch Indizes ab 99 gesetzt werden. 99 setzt dabei den Drucker für alle Bereiche, 100 für den ersten, 101 für den zweiten usw.

pszPrinter: Druckername

pDM: Zeiger auf die neue *DEVMODE*-Struktur. Kann NULL sein, dann wird der Drucker mit seinen momentanen Voreinstellungen eingetragen.

Rückgabewert:

Fehlercode

Hinweise:

Diese Funktion ermöglicht es Ihnen, einen (oder beide) Drucker in der Druckerkonfigurationsdatei einzutragen. Wenn keine existiert, wird sie angelegt. Durch Verodern des Projekttyps mit LL_PRJTYPE_OPTION_-FORCEDEFAULTSETTINGS können Sie forcieren, dass die Standardeinstellungen des Druckers vorbelegt werden.

Da diese Datei von *LlPrintStart()* und *LlPrintWithBoxStart()* eingelesen wird, muss der Befehl VOR dem Aufruf dieser Funktionen durchgeführt werden.

Die *DEVMODE*-Struktur ist in der Windows-API definiert.

Durch die Möglichkeit, unterschiedliche Druck-Bereiche im Designer definieren zu können, ist die praktische Nutzbarkeit dieser Funktion sehr stark eingeschränkt, wenn Sie keine Kontrolle über das Layout haben. Wir empfehlen daher, in diesen Fällen über das LL Objektmodell gemäß Kapitel ("Verwendung der DOM-API (ab Professional Edition)" auf die Bereiche und die dort eingestellten Drucker zuzugreifen.

Siehe auch:

LlSetPrinterToDefault, LlPrintCopyPrinterConfiguration, LlSetPrinterDefaultsDir, LlGetPrinterFromPrinterFile

LlSetPrinterToDefault

Syntax:

```
INT LlSetPrinterToDefault (HLLJOB hJob, UINT nObjType,
   LPCTSTR lpszObjName);
```

Aufgabe:

Löscht die Druckerbeschreibungsdatei, so dass List & Label bei der nächsten Benutzung des Projekts den im Projekt voreingestellten Drucker verwendet.

Parameter:

hJob: List & Label Job-Handle

nObjType: *LL_PROJECT_LABEL, LL_PROJECT_LIST* oder *LL_PROJECT_CARD*

lpszObjName: Der Dateiname des Projekts

Rückgabewert:

Fehlercode

Beispiel:

```
HLLJOB hJob;

hJob = LlJobOpen(0);

LlSetPrinterToDefault(hJob, LL_PROJECT_LIST, "test.lst");
if (LlPrintStart(hJob, LL_PROJECT_LIST, "test", LL_PRINT_NORMAL) == 0)
{
   <... etc ...>
   LlPrintEnd(hJob);
}
else
   MessageBox(NULL, "Fehler", "List & Label", MB_OK);
LlJobClose(hJob);
```

Siehe auch:

LlSetPrinterDefaultsDir, LlPrintStart, LlPrintWithBoxStart, LlCopyPrinterConfiguration, LlSetPrinterInPrinterFile

LlViewerProhibitAction

Syntax:

```
INT LlViewerProhibitAction (HLLJOB hJob, INT nMenuID);
```

Aufgabe:

Entfernt Buttons aus dem Vorschaufenster.

Parameter:

hJob: List & Label Job-Handle

nMenuID: Menü-ID des Buttons, der entfernt werden soll. Die entsprechenden IDs finden Sie (für die englische Version) in der Datei MENUID.TXT in Ihrer List & Label Installation.

Rückgabewert:

Fehlercode

Hinweise:

Wird für *nMenuID* eine 0 übergeben, wird die Liste der zu unterdrückenden IDs gelöscht.

Siehe auch:

LlPreviewDisplay, LlPreviewDisplayEx

LlXGetParameter

Syntax:

```
INT LlXGetParameter (HLLJOB hJob, INT nExtensionType,
   LPCTSTR pszExtensionName, LPCTSTR pszKey, LPTSTR pszBuffer,
   UINT nBufSize);
```

Aufgabe:

Gibt Einstellungen des entsprechenden Erweiterungsmoduls zurück.

Parameter:

hJob: List & Label Job-handle

nExtensionType: Art des Erweiterungsmoduls:

Wert	Bedeutung
LL_LLX_EXTENSIONTYPE_EXPORT	Export Modul
LL_LLX_EXTENSIONTYPE_BARCODE	2D-Barcode Modul

pszExtensionName: Name des Moduls ("HTML", "RTF", "PDF417", ...)

pszKey: Der Name der Option (z.B. "Export.File")

pszBuffer: Zeiger auf einen Puffer

nBufSize: Größe des Puffers

Rückgabewert:

Fehlercode

Hinweise:

Die Namen der Optionen sind spezifisch für ein Erweiterungsmodul. Bitte benutzen Sie die Referenz des entsprechenden Moduls.

Bezüglich des Rückgabewerts im Puffer siehe Kapitel "Wichtiges zu den Funktionsparametern".

Siehe auch:

LlXSetParameter

LIXSetParameter

Syntax:

```
INT LlXSetParameter (HLLJOB hJob, INT nExtensionType,
   LPCTSTR pszExtensionName, LPCTSTR pszKey, LPCTSTR pszValue);
```

Aufgabe:

Setzt Parameter in einem Erweiterungsmodul.

Parameter:

hJob: List & Label Job-handle

nExtensionType: Art des Erweiterungsmoduls:

Wert	Bedeutung
LL_LLX_EXTENSIONTYPE_EXPORT	Export Modul
LL_LLX_EXTENSIONTYPE_BARCODE	2D-Barcode Modul

pszExtensionName: Name des Moduls ("HTML", "RTF", "PDF417", ...)

pszKey: Der Name der Option

pszValue: Der zuzuweisende Inhalt

Rückgabewert:

Fehlercode

Hinweise:

Mit dieser Funktion kann man Optionen setzen wie beispielsweise den Ausgabepfad für HTML oder RTF, Schwärzungsgrad von Barcodes etc.

Die Namen und die Bedeutung der Werte sind spezifisch für ein Erweiterungsmodul. Bitte benutzen Sie die Referenz des entsprechenden Moduls.

Siehe auch:

LIXGetParameter

6.2 Referenz der Callback-Notifications

LL_CMND_DRAW_USEROBJ

Aufgabe:

Fordert das Programm auf, das benutzerdefinierte Objekt zu zeichnen.

Aktivierung:

```
LlDefineVariableExt(hJob, <Name>, <Inhalt>, LL_DRAWING_USEROBJ,
   <Parameter>);
```

```
LlDefineFieldExt(hJob, <Name>, <Inhalt>, LL_DRAWING_USEROBJ,
   <Parameter>);
```

oder

```
LlDefineVariableExt(hJob, <Name>, <Inhalt>,
   LL_DRAWING_USEROBJ_DLG,<Parameter>);
```

Parameter:

lParam zeigt auf eine *scLIDrawUserObj*-Struktur:

_nSize: Größe der Struktur, sizeof(*scLIDrawUserObj*)

_lpszName: Name der Variablen, die dem Objekt zugeordnet ist

_lpszContents: Text-Inhalt der Variablen, die dem Objekt zugeordnet ist. Dieser Wert ist nur gültig, wenn die Variable über *LIDefineVariableExt()* definiert wurde, ansonsten ist der _hPara-Wert gültig.

_lPara: *lPara* der Variablen, die dem Objekt zugeordnet ist (*LL_DRAWING_USEROBJ* oder *LL_DRAWING_USEROBJ_DLG*). Entspricht dem 4. Parameter des *LIDefineVariableExt()*-Aufrufs.

_lpPtr: *lpPtr* der Variablen, die dem Objekt zugeordnet ist. Entspricht einem Zeiger auf die Struktur, die mit dem 5. Parameter des *LIDefineVariableExt()*-Aufrufs übergeben wurde.

_hPara: Handle-Inhalt der Variablen, die dem Objekt zugeordnet ist. Dieser Wert ist nur gültig, wenn die Variable über *LIDefineVariableExtHandle()* definiert wurde, ansonsten ist der *_lpszContents*-Wert gültig.

_bIsotropic: TRUE: Das Objekt soll unverzerrt gezeichnet werden, FALSE: Die Zeichnung soll in das Rechteck eingepasst werden

_lpszParameters: bei benutzerdefinierten Objekten als Tabellenfeld: NULL, bei *LL_DRAWING_USEROBJ*: Zeiger auf einen leeren String, bei *LL_DRAWING_USEROBJ_DLG*: Zeiger auf die Parameter-Zeichenkette, die bei *LL_CMND_EDIT_USEROBJ* zurückgegeben wurde.

_hPaintDC: Device Context für die Ausgaben

_hRefDC: Device Context für Referenzen

_rcPaint: Rechteck, in dem das Objekt gezeichnet werden soll. Der Mapping-Mode ist in den üblichen Zeichnungs-Einheiten, also mm/10, inch/100 oder inch/1000.

_nPaintMode: 1: auf Designer-Preview, 0: auf Drucker/Echtdatenpreview

Rückgabewert (_lResult):

0

Hinweise:

In diesem Callback darf keine List & Label-Funktion aufgerufen werden, die Ausgaben zur Folge hat (*LlPrint()*, etc.)! Funktionen wie *LlPrintGetCurrentPage()* oder *LlPrintGetOption()* oder auch *LlPrintEnableObject()* sind erlaubt.

Siehe: Hinweise zur Benutzung von GDI-Objekten

Beispiel:

```
case LL_CMND_DRAW_USEROBJ:
   pSCD = (PSCLLDRAWUSEROBJ)pSC->_lParam;
   FillRect(pSCD->_hPaintDC,pSCD->_rcPaint,
      GetStockObject(_ttoi(lpszContents)));
   break;
```

LL_CMND_EDIT_USEROBJ

Aufgabe:

Fordert das Programm auf, einen objektspezifischen Dialog zu starten, über den der Benutzer die zugehörigen Darstellungsparameter eingeben und ändern kann.

Aktivierung:

```
LlDefineVariableExt(hJob, <Name>, <Inhalt>, LL_DRAWING_USEROBJ_DLG,
   <Parameter>);
```

Parameter:

lParam zeigt auf eine *scLlEditUserObj*-Struktur:

_nSize: Größe der Struktur, sizeof(*scLlEditUserObj*)

_lpszName: Name der Variablen, die dem Objekt zugeordnet ist

_lPara: *lPara* der Variablen, die dem Objekt zugeordnet ist (*LL_DRAWING_USEROBJ* oder *LL_DRAWING_USEROBJ_DLG*). Entspricht dem 4. Parameter des *LlDefineVariableExt()*-Aufrufs.

_lpPtr: *lpPtr* der Variablen, die dem Objekt zugeordnet ist. Entspricht dem 5. Parameter des *LlDefineVariableExt()*-Aufrufs.

_hPara: Handle-Inhalt der Variablen, die dem Objekt zugeordnet ist. Dieser Wert ist nur gültig, wenn die Variable über *LlDefineVariableExtHandle()* definiert wurde, ansonsten ist der *_lpszContents*-Wert gültig.

_bIsotropic: TRUE: Das Objekt soll unverzerrt gezeichnet werden FALSE: Die Zeichnung soll in das Rechteck optimal eingepasst werden

_hWnd: Fenster-Handle des Dialogs. Dieses sollte als Parent-Handle Ihres Dialogs genommen werden.

_lpszParameters: Zeiger auf einen Puffer mit der maximalen Größe *_nParaBufSize*.

_nParaBufSize: Größe des internen Puffers.

Rückgabewert (_lResult):

0

Hinweise:

Dieser Callback kommt nur bei Objekten mit dem Variablentyp *LL_DRAWING_USEROBJ_DLG*!

In diesem Callback darf keine List & Label-Funktion aufgerufen werden, die Ausgaben zur Folge hat (*LlPrint()*, etc.)! Funktionen wie *LlPrintGetCurrentPage()*, *LlPrintGetOption()* oder *LlPrintEnableObject()* sind erlaubt.

Siehe: Hinweise zur Benutzung von GDI-Objekten.

Die Bearbeitung des *_bIsotropic*-Flags ist optional, da dieses schon vom aufrufenden Dialog eingestellt werden kann. Wenn Sie dieses Flag in der Struktur verändern, wird die Veränderung von List & Label übernommen.

_lpszParameters zeigt auf einen String, in dem die beim letzten Aufruf des Dialogs eingegebenen Werte stehen. Dieser Puffer ist 1024+1 Zeichen groß, so dass ein Parameterstring, der von Ihrem Dialog erstellt wird, in diesen Puffer geschrieben werden kann, wenn er kürzer ist als der Wert in *_nParaBufSize*. Ansonsten müssen Sie diesen Zeiger mit einem Zeiger auf Ihre gewünschten Daten überschreiben. Die Problematik eines längeren Parameterstrings ist, dass dieser von List & Label nicht freigegeben werden kann, falls es ein allozierter Speicherbereich ist.

Die in der Parameter-Zeichenkette erlaubten Zeichen sind alle druckbaren Zeichen, also Zeichen mit Codes >= 32 (' ').

Beispiel:

```
case LL_CMND_EDIT_USEROBJ:
   pSCE = (PSCLLEDITUSEROBJ)pSC->_lParam;

   lpszNewParas = MyDialog(pSCE->_hWnd,...,);

   if (_tcslen(lpszNewParams) < pSCE->_lpszParameters)
      _tcscpy(pSCE->_lpszParameters, lpszNewParas);
    else
      pSCE->_lpszParameters = lpszNewParas;
   break;
```

LL_CMND_ENABLEMENU

Aufgabe:

Callback, bei dem bestimmt werden kann, welche Menü-Einträge im Designer erlaubt sind etc.

Aktivierung:

Immer aktiviert

Parameter:

lParam: menu handle

Hinweise:

Diese Funktion wird aufgerufen, wenn List & Label das Menü ändert bzw. anpasst. Hier können z.B. die über *LL_CMND_MODIFYMENU* eingefügten Menüpunkte enabled oder disabled werden.

Beispiel:

```
case LL_CMND_ENABLEMENU:
   if (<whatever>)
      EnableMenuItem(hMenu,IDM_MYMENU,
         MF_ENABLED|MF_BYCOMMAND);
    else
      EnableMenuItem(hMenu,IDM_MYMENU,
         MF_DISABLED|MF_GRAYED|MF_BYCOMMAND);
   break;
```

LL_CMND_EVALUATE

Aufgabe:

Fragt das Benutzerprogramm nach der Interpretation des Inhalts der *External$()*-Funktion im Expression-Modus.

Aktivierung:

Eingabe einer *External$()* -Funktion in einen Ausdruck

Parameters:

lParam zeigt auf eine *scLIExtFct*-Struktur:

_nSize: Größe der Struktur, *sizeof(scLIExtFct)*

_lpszContents: Parameter der Funktion *External$().* Dieser wurde List & Label-seitig bereits entsprechend evaluiert, d.h. darin etwaige verwendete Formeln und Variablen wurden bereits aufgelöst.

_bEvaluate: TRUE, wenn der Inhalt ausgewertet werden soll, FALSE, wenn nur ein Syntax-Test durchgeführt werden soll.

_szNewValue: Array, worin das Ergebnis als Null-terminierte Zeichenkette abgelegt wird. Voreinstellung: leer.

_bError: TRUE: Fehler aufgetreten, FALSE: kein Fehler aufgetreten (Voreinstellung).

_szError: Array, worin eine eventuelle Fehlerbeschreibung abgelegt werden kann, die später mit *LlExprError()* abgefragt werden kann. **Dieser Text wird dem Benutzer auch im Designer von der automatischen Syntaxprüfung angezeigt.**

Rückgabewert (_lResult):

0

Hinweise:

Wichtig: die Rückgabe-Felder müssen NULL-terminiert sein und dürfen die maximale Länge (16385 Zeichen inkl. Terminierung bei dem Rückgabewert, 128 Zeichen inkl. Nullterminierung bei der Fehlerbeschreibung) nicht überschreiten.

LL_CMND_GETVIEWERBUTTONSTATE

Aufgabe:

Über diesen Callback fragt List & Label, welchen Status der entsprechende Toolbar-Button des Echtdatenpreviews haben kann.

Aktivierung:

Immer aktiviert

Parameter:

HIWORD(lParam): Toolbutton ID

LOWORD(lParam): Vom Viewer errechneter Status

Rückgabewert (_lResult):

Neuer Status	Bedeutung
0	keine Änderung
1	enabled
2	disabled
-1	unsichtbar

Hinweise:

Dieser Callback wird vom Echtdatenpreview aufgerufen. Die entsprechenden IDs finden Sie in der Datei MENUID.TXT in Ihrer List & Label Installation.

Beispiel:

```
case LL_CMND_GETVIEWERBUTTONSTATE:
   switch (HIWORD(lParam))
   {
      case 112:
         // don't allow one-page print
         return(-1);
   }
```

```
    break;
```

LL_CMND_HELP

Aufgabe:

Ermöglicht es, ein externes Hilfesystem statt dem List & Label-eigenen einzubinden.

Aktivierung:

```
LlSetOption(hJob, LL_OPTION_CALLBACKMASK, <andere Flags> | LL_CB_HELP);
```

Parameter:

HIWORD(lParam):

Wert	Bedeutung
HELP_CONTEXT	LOWORD(lParam) ist dann die Kontext-Nummer des Hilfe-Themas
HELP_INDEX	Index der Hilfedatei wurde angefordert
HELP_HELPONHELP	Der Benutzer will die Hilfe-Übersicht

Rückgabewert (_lResult):

Wert	Bedeutung
0	Aufforderung an List & Label, seine Hilfe anzuzeigen
1	Gibt an, dass List & Label nichts tun soll

Siehe auch:

LlExprError

LL_CMND_MODIFYMENU

Aufgabe:

Callback, über den das Menü das List & Label anzeigt, verändert werden kann. Dieser Callback ist nur aus Kompatibilitätsgründen noch enthalten, um den Designer zu erweitern verwenden Sie bevorzugt *LlDesignerAddAction()*.

Aktivierung:

Immer aktiviert

Parameter:

lParam: menu handle

Hinweise:

Diese Funktion wird aufgerufen, wenn List & Label das Menü anlegt, so dass die Menüstruktur angepasst werden kann, um neue, eigene Funktionen hinzuzufügen, andere zu löschen etc.

Die entsprechenden IDs finden Sie in der Datei MENUID.TXT in Ihrer List & Label Installation. Benutzerdefinierte Menü-IDs sollten im Bereich über 10100 angesiedelt werden.

Dieser Callback ist nur noch aus Kompatibilitätsgründen enthalten, um den Designer zu erweitern verwenden Sie bevorzugt *LlDesignerAddAction()*.

Beispiel:

```
case LL_CMND_MODIFYMENU:
   DeleteMenu(_hMenu, IDM_HELP_CONTENTS, MF_BYCOMMAND);
   DeleteMenu(_hMenu, IDM_HELP_INDEX, MF_BYCOMMAND);
   DeleteMenu(_hMenu, IDM_HELP_HELPONHELP, MF_BYCOMMAND);
   break;
```

LL_CMND_OBJECT

Aufgabe:

Ermöglicht, etwas vor und nach List & Label in oder neben das Objekt-Rechteck zu zeichnen, oder das Objekt während des Drucks zu verstecken und ermöglicht viele Modifikationen.

Aktivierung:

```
LlSetOption(hJob, LL_OPTION_CALLBACKMASK, <andere Flags> | LL_CB_-
OBJECT);
```

Parameter:

lParam zeigt auf eine *scLlObject*-Struktur:

_nSize: Größe der Struktur, *sizeof(scLlObject)*

nType*: Art des Objekts (LL*OBJ_... Konstante).

_lpszName: Name des Objektes in der Vorlage.

_bPreDraw: TRUE bei Aufruf, vor dem Zeichnen des Objekts, FALSE bei Aufruf nach dem Zeichnen des Objekts

_hPaintDC: Device Context für die Ausgaben

_hRefDC: Device Context für Referenzen

_rcPaint: Rechteck, in dem das Objekt gezeichnet wird. Der Mapping-Mode ist in der eingestellten Einheit, z.B. mm/100.

Rückgabewert (_lResult):

Wert von _bPreDraw	_lResult
TRUE	0: okay 1: Objekt soll nicht gezeichnet/versteckt werden
FALSE	immer 0

Hinweise:

In diesem Callback darf keine List & Label-Funktion aufgerufen werden, die Ausgaben zur Folge hat (*LlPrint()*, etc.)! Funktionen wie *LlPrintGetCurrentPage()* oder *LlPrintGetOption()* sind erlaubt.

Siehe: Hinweise zur Benutzung von GDI-Objekten.

Diese Funktion wird pro Objekt zwei Mal aufgerufen, einmal mit *_bPreDraw* = TRUE, dann mit *_bPreDraw* = FALSE.

_bPreDraw == TRUE:
Verwendung: Man kann z.B. einen eigenen Hintergrund zeichnen oder das Objekt verstecken.

Beachten Sie, dass die Objekte von List & Label möglicherweise kleiner gezeichnet werden als das Rechteck angibt, z.B. bei Listenobjekten, die keine fixe Größe besitzen. Wenn Sie beim Aufruf *_rcPaint* verändern, hat dies Auswirkungen auf die Größe des Objekts, denn das Objekt wird von List & Label in das hier angegebene Rechteck gezeichnet.

_bPreDraw == FALSE:
Verwendung: Man kann z.B. einen eigenen Hintergrund und/oder Schatten zeichnen, denn erst dann ist die wahre Größe des Objekts bekannt. Das Rechteck *_rcPaint* ist hier das korrekte Objekt-Rechteck. Wenn Sie beim Aufruf *_rcPaint* verändern, hat dies Auswirkungen auf angehängte Objekte, denn die Daten von _rcPaint werden als Objektrechteck verwendet, das wiederum die Koordinaten räumlich angehängter Objekte beeinflusst!

Beispiel:

```
case LL_CMND_OBJECT:
   pSCO = (PSCLLOBJECT)pSC->_lParam;
   if (pSCO->_nType == LL_OBJ_RECT && !pSCO->_bPreDraw)
   {
      FillRect(pSCO->_hPaintDC, pSCF->_rcPaint,
         GetStockObject(LTGRAY_BRUSH);
   }
   break;
```

LL_CMND_PAGE

Aufgabe:

Ermöglicht es, zusätzliche Zeichnungen auf der Seite unterzubringen. Interessant ist dies besonders beim Etikettendruck, da man so etikettenübergreifende Informationen auf eine Seite ausgeben kann.

Aktivierung:

```
LlSetOption(hJob, LL_OPTION_CALLBACKMASK, <andere Flags> | LL_CB_PAGE);
```

Parameter:

lParam zeigt auf eine *scLlPage*-Struktur:

_nSize: Größe der Struktur, *sizeof(scLlPage)*

_bDesignerPreview: TRUE, wenn der Aufruf vom Designer-Preview stattfindet, FALSE, wenn der Aufruf während des Echtdaten-Preview oder des Drucks stattfindet.

_bPreDraw: TRUE bei Aufruf, bevor List & Label die Seite zeichnet. FALSE bei Aufruf, nachdem List & Label die Seite gezeichnet hat.

_hPaintDC: Device Context für die Ausgaben

_hRefDC: Device Context für Referenzen

Rückgabewert (_lResult):

0

Hinweise:

In diesem Callback darf keine List & Label-Funktion aufgerufen werden, die Ausgaben zur Folge hat (*LlPrint()*, etc.)! Funktionen wie *LlPrintGetCurrentPage()* oder *LlPrintGetOption()* oder auch *LlPrintEnableObject()* sind erlaubt.

Siehe: Hinweise zur Benutzung von GDI-Objekten.

Diese Funktion wird pro Seite zwei Mal aufgerufen, einmal mit *_bPreDraw* = TRUE, dann mit *_bPreDraw* == FALSE.

Die Seitengröße kann über die *GetWindowExt()*-Funktion bestimmt werden. Benutzen Sie hier _hRefDC!

Wenn Sie bei *_bPreDraw* == TRUE den Fenster-Ursprung von *_hRefDC* mit *SetWindowOrg()* verändern, wirkt sich das auf die gesamte Seite aus. **Damit kann man z.B. einen Bundsteg für gerade/ungerade Seiten definieren**. Diese Ausgabeverschiebung wirkt sich nur auf Echtdatenpreview oder -Druck aus, nicht jedoch für den Designer-Preview.

Beispiel:

```
case LL_CMND_PAGE:
   pSCP = (PSCLLPAGE)pSC->_lParam;
```

```
if (pSCP->_bPreDraw && (LlPrintGetCurrentPage(hJob) % 2) == 1)
   SetWindowOrg(pSCP->_hPaintDC,-100,0);
break;
```

LL_CMND_PROJECT

Aufgabe:

Ermöglicht es, zusätzliche Zeichnungen im Etiketten- oder Karteikarten-Projekt auszugeben.

Dieser Callback wird nur bei Etiketten- und Karteikartenprojekten ausgelöst, bei Listenobjekten verwenden Sie anstattdessen *LL_CMND_PAGE*.

Aktivierung:

```
LlSetOption(hJob,LL_OPTION_CALLBACKMASK,<andere Flags> | LL_CB_PROJECT);
```

Parameter:

lParam zeigt auf eine *scLlProject*-Struktur:

_nSize: Größe der Struktur, *sizeof(scLlProbject)*

_bPreDraw: TRUE bei Aufruf, bevor List & Label die Seite zeichnet, FALSE bei Aufruf, nachdem List & Label die Seite gezeichnet hat.

_bDesignerPreview: TRUE, wenn der Aufruf vom Designer-Preview stattfindet, FALSE, wenn der Aufruf während des Echtdaten-Preview oder des Drucks stattfindet.

_hPaintDC: Device Context für die Ausgaben

_hRefDC: Device Context für Referenzen

_rcPaint: Rechteck, in dem das Projekt gezeichnet werden soll. Der Mapping-Mode ist in der eingestellten Einheit, z.B. mm/100.

Rückgabewert (_lResult):

0

Hinweise:

In diesem Callback darf keine List & Label-Funktion aufgerufen werden, die Ausgaben zur Folge hat (*LlPrint()*, etc.)! Funktionen wie *LlPrintGetCurrentPage()* oder *LlPrintGetOption()* oder auch *LlPrintEnableObject()* sind erlaubt.

Siehe: Hinweise zur Benutzung von GDI-Objekten.

Diese Funktion wird pro Seite zwei Mal aufgerufen, einmal mit *_bPreDraw* = TRUE, dann mit *_bPreDraw* = FALSE.

Beispiel:

```
case LL_CMND_PROJECT:
   pSCP = (PSCLLPROJECT)pSC->_lParam;
```

```
if (pSCP->_bPreDraw)
{
   FillRect(pSCL->_hPaintDC, pSCL->_rcPaint,
      GetStockObject(LTGRAY_BRUSH));
}
break;
```

LL_CMND_SAVEFILENAME

Aufgabe:

Informiert die Anwendung darüber, dass der Benutzer das Projekt im Designer gespeichert hat und liefert den Dateinamen.

Parameter:

lParam zeigt auf den nullterminierten Dateinamen.

Rückgabewert (_lResult):

0

Beispiel:

```
case LL_CMND_SAVEFILENAME:
   pszLastFilename = (LPCTSTR)lParam;
```

LL_CMND_SELECTMENU

Aufgabe:

Benachrichtigung, dass ein Menüpunkt/Toolbarbutton angewählt wurde.

Aktivierung:

Immer aktiviert

Parameter:

lParam ist die Menü-ID des Menüpunkts (negative ID, wenn es ein Toolbar-Button ist). Die entsprechenden IDs finden Sie in der Datei MENUID.TXT in Ihrer List & Label Installation.

Rückgabewert (_lResult):

TRUE, wenn der Menüeintrag bearbeitet wurde, FALSE sonst

Beispiel:

```
case LL_CMND_SELECTMENU:
   if (lParam == IDM_MYMENU)
   {
      // execute custom code
      return (TRUE);
   }
   break;
```

LL_CMND_TABLEFIELD

Aufgabe:

Ermöglicht es, die Farbgebung einzelner Tabellenfelder zu modifizieren.

Aktivierung:

```
LlSetOption(hJob, LL_OPTION_TABLE_COLORING, LL_COLORING_PROGRAM);
```

Dadurch wird die alleinige Kontrolle der Farbgebung in Tabellen Ihrem Programm überlassen (die entsprechenden Einstellungen im Tabellen-Eigenschaften-Dialog des Designers verschwinden dann).

```
LlSetOption(hJob, LL_OPTION_TABLE_COLORING, LL_COLORING_DONTCARE);
```

Mit dieser Option hingegen lässt List & Label erst Ihr Programm den Hintergrund zeichnen, dann zeichnet es den Hintergrund bei Bedarf noch einmal mit dem im Designer definierten Feldmuster. Dadurch ist eine Art Kooperation zwischen dem Programmierer und dem Benutzer möglich.

Parameter:

lParam zeigt auf eine *scLITableField*-Struktur:

_nSize: Größe der Struktur, *sizeof(scLITableField)*

_nType: Art des Feldes:

Wert	Bedeutung
LL_TABLE_FIELD_HEADER	Feld ist in Kopfzeile
LL_TABLE_FIELD_BODY	Feld ist in Datenzeile
LL_TABLE_FIELD_GROUP	Feld ist in Gruppenkopf
LL_TABLE_-FIELD_GROUPFOOTER	Feld ist in Gruppenfuß
LL_TABLE_FIELD_FILL	Feld ist die Füll-Fläche, die entsteht, wenn die Tabelle feste Größe besitzt und unter der letzten Datenzeile noch etwas Freiraum bleibt
LL_TABLE_FIELD_FOOTER	Feld ist in Fußzeile

_hPaintDC: Device Context für die Ausgaben

_hRefDC: Device Context für Referenzen

_rcPaint: Rechteck, in dem das Feld gezeichnet werden soll. Der Mapping-Mode ist in der eingestellten Einheit, z.B. mm/100.

_nLineDef: Die Nummer der Zeilendefinition, die ausgegeben wird.

_nIndex: Feldindex, 0-basiert (die erste Spalte hat als Index 0, die zweite 1, etc.)

_rcSpacing: Werte der Zellen-Abstände

_pszContents: Der Parameter liefert den (evaluierten) Inhalt der Zelle, die gerade ausgegeben wird, also z.B. "Müller" für eine Spalte, die Person.Nachname enthält. Diesen Parameter kann man verwenden, um z.B. bestimmte Werte im Event besonders zu behandeln.

Rückgabewert (_lResult):

0

Hinweise:

In diesem Callback darf keine List & Label-Funktion aufgerufen werden, die Ausgaben zur Folge hat (*LlPrint()*, etc.)!

Wenn Sie irgendein GDI-Objekt in diesen DC selektieren oder andere Änderungen vornehmen, z.B. des Mapping-Modes, sollten Sie die Änderungen vor der Beendigung der Routine wieder rückgängig machen. Tipp: die API-Funktionen *SaveDC()*, *RestoreDC()* können bei komplexen Veränderungen sehr helfen (verwendete Funktionen sind Windows-API-Funktionen).

Beispiel:

```
case LL_CMND_TABLEFIELD:
   pSCF = (PSCLLTABLEFIELD)pSC->_lParam;
   if (pSCF->_nIndex == 1)
   {
      FillRect(pSCF->_hPaintDC, pSCF->_rcPaint,
         GetStockObject(LTGRAY_BRUSH);
   }
   pSC._lResult = 0;
   break;
```

LL_CMND_TABLELINE

Aufgabe:

Ermöglicht es, die Farbgebung einzelner Tabellenzeilen zu modifizieren, z.B. um einen eigenen Zebramodus (jede zweite Zeile unterlegt) zu erzeugen.

Aktivierung:

```
LlSetOption(hJob, LL_OPTION_TABLE_COLORING, LL_COLORING_PROGRAM);
```

Dadurch wird die alleinige Kontrolle der Farbgebung in Tabellen Ihrem Programm überlassen! (die entsprechenden Einstellungsmöglichkeiten im Designer verschwinden dann)

oder

```
LlSetOption(hJob, LL_OPTION_TABLE_COLORING, LL_COLORING_DONTCARE);
```

Mit diesem Befehl lässt List & Label erst Ihr Programm den Hintergrund zeichnen, dann zeichnet es den Hintergrund bei Bedarf noch einmal mit dem im Designer definierten Feldhintergrund. Dadurch ist eine Art Kooperation zwischen Programmierer und Benutzer möglich.

Beachten Sie, dass in jedem Falle das Flag *LL_CB_TABLELINE* über *LL_OPTION_CALLBACKMASK* gesetzt werden muss.

Parameter:

lParam zeigt auf eine *scLITableLine*-Struktur:

_nSize: Größe der Struktur, *sizeof(scLITableLine)*

_nType: Art des Feldes:

Wert	Bedeutung
LL_TABLE_LINE_HEADER	Kopfzeile
LL_TABLE_LINE_BODY	Datenzeile
LL_TABLE_LINE_GROUP	Gruppenkopf
LL_TABLE_-LINE_GROUPFOOTER	Gruppenfuß
LL_TABLE_LINE_FILL	Füll-Fläche, die entsteht, wenn die Tabelle feste Größe besitzt und unter der letzten Datenzeile noch etwas Freiraum bleibt
LL_TABLE_LINE_FOOTER	Fußzeile

_hPaintDC: Device Context für die Ausgaben

_hRefDC: Device Context für Referenzen

_rcPaint: Rechteck, in dem die Zeile gezeichnet werden soll. Der Mapping-Mode ist in der eingestellten Einheit, z.B. mm/100.

_nPageLine: Zeilenindex. Bezeichnet die 0-basierte Zeilennummer der Zeile auf dieser Seite.

_nLine: Zeilenindex. Bezeichnet die 0-basierte Zeilennummer der Zeile im gesamten Ausdruck.

_nLineDef: Die Nummer der Zeilendefinition, die ausgegeben wird.

_bZebra: TRUE, wenn Benutzer Zebra-Modus gewählt hat.

_rcSpacing: Werte der Zellen-Abstände

Rückgabewert (_lResult):

0

Hinweise:

In diesem Callback darf keine List & Label-Funktion aufgerufen werden, die Ausgaben zur Folge hat (*LIPrint()*, etc.)!

Siehe: Hinweise zur Benutzung von GDI-Objekten

Beispiel:

```
case LL_CMND_TABLELINE:
   pSCL = (PSCLLTABLELINE)pSC->_lParam;
   if ((pSCL->_nPageLine % 2) == 1)
      FillRect(pSCL->_hPaintDC, pSCL->_rcPaint,
         GetStockObject(LTGRAY_BRUSH);
   pSC._lReply = 0;
   break;
```

LL_CMND_VARHELPTEXT

Aufgabe:

Ermöglicht eigene Beschreibungstexte zu Variablen und Feldern zu definieren, die im Formelassistenten angezeigt werden.

Parameter:

lParam zeigt auf den String, der den Namen der Variablen oder des Feldes enthält

Rückgabewert (_lResult):

_lReply muss auf den Erklärungstext zeigen

Beispiel:

```
case LL_CMND_VARHELPTEXT:
   sVariableDescr = (LPCSTR)pscCallback->_lParam;
   // Checkroutinen für Variable
   _tcscpy(szHelpText, T("Variable x für Aufgabe y"));
   pscCallback->_lreply = (LPARAM)szHelpText;
   break;
```

LL_NTFY_VIEWERBTNCLICKED

Aufgabe:

Benachrichtigung, dass ein Button im Echtdatenpreview gedrückt wurde.

Aktivierung:

Immer aktiviert

Parameter:

lParam: Toolbutton ID

Rückgabewert (_lResult):

Wert	Bedeutung
1	Button ignorieren
0	Default-Funktion ausführen

Hinweise:

Diese Funktion wird vom Echtdatenpreview von List & Label aufgerufen.

Die entsprechenden IDs finden Sie in der Datei MENUID.TXT in Ihrer List & Label Installation.

Beispiel:

```
switch (wParam)
{
   case LL_NTFY_VIEWERBTNCLICKED:
      switch (lParam)
      {
         case 112:
            // beep on one-page print and don't execute it!
            MessageBeep(-1);
            return(1);
      }
      break;
}
```

LL_INFO_METER

Aufgabe:

Benachrichtigung, dass eine möglicherweise länger dauernde Operation mit den Objekten durchgeführt wird.

Aktivierung:

Immer aktiviert

Parameter:

lParam zeigt auf eine *scLIMeterInfo*-Struktur:

_nSize: Größe der Struktur

_hWnd: Handle des List & Label Hauptfensters

_nTotal: Gesamtzahl der Objekte

_nCurrent: gegenwärtig bearbeitetes Objekt

_nJob: Aufgabe, mit der List & Label beschäftigt ist:

Aufgabe	Bedeutung
LL_METERJOB_SAVE	Speichern der Objekte
LL_METERJOB_LOAD	Laden der Objekte
LL_METERJOB_-CONSISTENCYCHECK	Konsistenzprüfung der Objektliste

Hinweise:

Durch diesen Callback kann z.B. ein Fortschrittsbalken angezeigt werden. Der Prozentwert einer Fortschrittsanzeige berechnet sich über MulDiv(100, *_nCurrent*, *_nTotal*).

Beispiel:

```
// Die Funktionen WaitDlg... müssen durch eigene Funktionen
// ersetzt werden

switch (wParam)
{
   case LL_INFO_METER:
   {
      scLlMeterInfo* pMI = (scLlMeterInfo*)lParam;
      static hJob  hMeterJob = 0;
      if (pMI->_nSize == sizeof(scLlMeterInfo)) // is actual version?
      {
         // do I have to do something?
         if (pMI->_nTotal > 0)
         {
            // get parent window handle for Dialog
            HWND  hWndParent = pMI->_hWnd ? pMI->_hWnd : hwndMyFrame;
            if (pMI->_nCurrent == 0)
            {
               // open meter bar with 0%!
               hMeterJob = WaitDlgStart(hWndParent, T("wait a moment"),
                  0);
            }
            else
            {
               // end:
               if (pMI->_nCurrent == pMI->_nTotal)
               {
                  // end meter bar!
                  WaitDlgEnd(hMeterJob);
               }
               else
               // somewhere in between 0 and 100
               {
                  // set meter value to MulDiv(100,_ nCurrent, _nTotal)
                  WaitDlgSetText(hMeterJob, T("still working..."),
                        MulDiv(100, pMI->_nCurrent, pMI->_nTotal));
               }
            }
         }
      }
   }
   break;
}
```

LL_INFO_PRINTJOBSUPERVISION

Aufgabe:

Überwachung des Druckjobs

Parameter:

lParam zeigt auf eine *scLIPrintJobInfo*-Struktur:

_nSize: Größe der Struktur

_hLIJob: Job-Handle des LL-Jobs, der den Druck auslöste

_szDevice: Name des Druckers

_dwJobID: Job-ID (nicht die Job-ID des Druckers, sondern eine globale, vergeben von List & Label)

_dwState: Kombination von Job-Zustand-Flags (JOB_STATUS_-Konstanten von WINSPOOL.H)

Hinweise:

Stellen Sie sicher, *LL_OPTION_NOPRINTJOBSUPERVISION* auf *FALSE* zu stellen, um diesen Callback zu erhalten.

Der Detail-Grad hängt vom Druckerspooler ab. Beachten Sie, dass alle Teile der Übertragungskette (also z.B. Druckserver und Client) NT-basierende Betriebssysteme benötigen (Windows NT4 und höher).

Die dwState-Flags sind wie folgt definiert:

```
#define JOB_STATUS_PAUSED              0x00000001
#define JOB_STATUS_ERROR               0x00000002
#define JOB_STATUS_DELETING            0x00000004
#define JOB_STATUS_SPOOLING            0x00000008
#define JOB_STATUS_PRINTING            0x00000010
#define JOB_STATUS_OFFLINE             0x00000020
#define JOB_STATUS_PAPEROUT            0x00000040
#define JOB_STATUS_PRINTED             0x00000080
#define JOB_STATUS_DELETED             0x00000100
#define JOB_STATUS_BLOCKED_DEVQ        0x00000200
#define JOB_STATUS_USER_INTERVENTION   0x00000400
#define JOB_STATUS_RESTART             0x00000800
```

LL_NTFY_DESIGNERPRINTJOB

Aufgabe:

Über den Callback *LL_NTFY_DESIGNERPRINTJOB* informiert List & Label Sie über die durchzuführende Aktion. Dieser Callback wird immer im Kontext des Designer Threads (dies ist der Thread, von dem aus Sie *LIDefineLayout()* aufgerufen haben) aufgerufen.

Aktivierung:

```
LlSetOption(hJob, LL_OPTION_ DESIGNERPREVIEWPARAMETER,
    (LPARAM)&oMyDesignerPreviewParameters);
```

sowie

```
LlSetOption(hJob, LL_OPTION_ DESIGNEREXPORTPARAMETER,
    (LPARAM)&oMyDesignerExportParameters);
```

Parameter:

lParam zeigt auf eine *scLIDesignerPrintJob*-Struktur:

_nUserParam: Wert, den Sie an *LL_OPTION_DESIGNERPREVIEWPARAMETER* oder *LL_OPTION_DESIGNEREXPORTPARAMETER* übergeben haben.

_pszProjectName: Name des auszugebenden Projekts. Dieser Parameter ist nur beim "START"-Kommando gültig, ansonsten NULL.

_pszOriginalProjectFileName: Name des Original-Projekts. Dieser Parameter ist nur beim "START"-Kommando gültig, ansonsten NULL. Er wird benötigt, damit List & Label relative Pfade und die *ProjectPath()*-Funktion korrekt auswerten kann.

_nPages: Maximalzahl der auszugebenden Seiten. Diese müssen Sie nach dem Druckstart über

```
LlPrintSetOption(hJob,LL_PRNOPT_LASTPAGE,_nPages);
```

dem Druckjob übergeben. Wenn _nPages den Wert Null hat, bedeutet dies, dass der Druck nicht eingeschränkt sein soll.

_nFunction: die durchzuführende Aufgabe. Es gibt vier verschiedene Aufgaben: Start, Abbruch, Finalisieren und Statusabfrage.

Da es zwei Aufgabengruppen gibt (EXPORT und PREVIEW), ergibt dies 8 Konstanten:

LL_DESIGNERPRINTCALLBACK_PREVIEW_START
LL_DESIGNERPRINTCALLBACK_PREVIEW_ABORT
LL_DESIGNERPRINTCALLBACK_PREVIEW_FINALIZE
LL_DESIGNERPRINTCALLBACK_PREVIEW_QUEST_JOBSTATE
LL_DESIGNERPRINTCALLBACK_EXPORT_START
LL_DESIGNERPRINTCALLBACK_EXPORT_ABORT
LL_DESIGNERPRINTCALLBACK_EXPORT_FINALIZE
LL_DESIGNERPRINTCALLBACK_EXPORT_QUEST_JOBSTATE

_hWnd: Fensterhandle. Die Bedeutung dieses Struktur Members wird weiter unten noch erklärt.

_hEvent: Eventhandle, dient zur Kommunikation und Synchronisation Ihrer Anwendung mit List & Label.

_pszExportFormat: Vorselektiertes Exportformat (nur im Ribbon-Modus benötigt), siehe Kapitel Direkter Druck und Export aus dem Designer.

_bWithoutDialog: Druck/Export ohne Dialog (nur im Ribbon-Modus benötigt), siehe Kapitel Direkter Druck und Export aus dem Designer.

Rückgabewert (_lResult):

Geben Sie *LL_DESIGNERPRINTTHREAD_STATE_RUNNING* zurück, wenn Ihr Thread arbeitet, ansonsten liefern Sie *LL_DESIGNERPRINTTHREAD_STATE_-STOPPED*.

Hinweise:

Siehe Kapitel Direkter Druck und Export aus dem Designer

LL_NTFY_EXPRERROR

Aufgabe:

Benachrichtigung der Applikation, dass ein Ausdruck einen Fehler hatte.

Aktivierung:

Immer aktiv

Parameter:

lParam zeigt auf den Fehlertext

Rückgabewert (_lResult):

0

Hinweise:

Da *LlExprError()* beim Laden eines Projekts (auch für den Druck) nur bedingt zum Erfolg führt (da der interne Formelparser evtl. beim Aufruf der Funktion schon eine völlig andere Formel geparst hat und somit evtl. gar keinen Fehler mehr "aktiv" hat), kann man über diesen Callback Fehlermeldungen sammeln und dann nach *LlPrintStart()* dem Benutzer melden.

LL_NTFY_FAILSFILTER

Aufgabe:

Benachrichtigung, dass der Datensatz, der gerade an List & Label übergeben wurde, nicht ausgedruckt wurde, da er der Filterbedingung nicht entsprach.

Aktivierung:

Immer aktiv

Parameter:

lParam hat keine Bedeutung.

Rückgabewert (_lResult):

0

Hinweise:

In diesem Callback darf keine List & Label-Funktion aufgerufen werden, die Ausgaben zur Folge hat (*LlPrint()*, etc.)!

Dient zum Setzen einer globalen Variable; kann aber auch über *LlPrintDidMatchFilter()* überflüssig gemacht werden.

Beispiel:

```
case LL_NTFY_FAILSFILTER:
   bFails = TRUE;
   break;
```

LL_NTFY_VIEWERDRILLDOWN

Aufgabe:

Benachrichtigung, dass eine Drilldown-Aktion durchgeführt werden soll.

Aktivierung:

```
LlSetOption(hJob, LL_OPTION_DRILLDOWNPARAMETER,
    (LPARAM)&oMyDrillDownParameters);
```

Parameter:

lParam zeigt auf eine *scLlDrillDownJob*-Struktur:

_nSize: Größe der Struktur

_nFunction: Die durchzuführende Aufgabe. Es gibt zwei verschiedene Aufgaben:

Aufgabe	Bedeutung
LL_DRILLDOWN_START	Start
LL_DRILLDOWN_FINALIZE	Finalisieren

_nUserParameter: Wert, der mit *LL_OPTION_DRILLDOWNPARAMETER* übergeben wurde

_pszTableID: Zeiger auf einen String, der den Namen der Eltern-Tabelle enthält.

_pszRelationID: Zeiger auf einen String, der den Namen der Relation zwischen Eltern- und Child-Tabelle enthält.

_pszSubreportTableID: Zeiger auf einen String, der den Namen der Child-Tabelle enthält.

_pszKeyField: Zeiger auf einen String, der den Namen des Schlüsselfeldes der Eltern-Tabelle enthält. Wenn die Relation mehrere Schlüsselfelder benötigt (geteilter Primärschlüssel) erhalten Sie diese hier Tab-getrennt. Beachten Sie auch die Dokumentation zu *LlDbAddTableRelationEx()*.

_pszSubreportKeyField: Zeiger auf einen String, der den Namen des Schlüsselfeldes der Child-Tabelle enthält. Wenn die Relation mehrere Schlüsselfelder benötigt (geteilter Primärschlüssel) erhalten Sie diese hier Tab-getrennt. Beachten Sie auch die Dokumentation zu *LlDbAddTableRelationEx()*.

_pszKeyValue: Zeiger auf einen String, der den Inhalt des Schlüsselfeldes der Eltern-Tabelle enthält. Wenn die Relation mehrere Schlüsselfelder benötigt (geteilter Primärschlüssel) erhalten Sie die Werte hier Tab-getrennt. Beachten Sie auch die Dokumentation zu *LlDbAddTableRelationEx()*.

_pszProjectFileName: Name des auszugebenden Projektes.

_pszPreviewFileName: Name der zu erstellenden Vorschaudatei.

_pszTooltipText: Zeiger auf einen String, der den Tooltip-Text enthält, wenn man mit der Maus über einen Eintrag in der Tabelle steht, welche einen Subreport (Drilldown) auslösen soll.

_pszTabText: Zeiger auf einen String, der auf der Lasche angezeigt wird, wenn der Benutzer den Drilldown-Bericht auf einer Lasche angezeigt haben möchte.

_hWnd: Fensterhandle, um eigene Dialoge anzeigen zu können (Fenster-Handle des Vorschau-Controls).

_nID: Enthält die eindeutige DrilldownJobID; nicht mit dem List & Label Druckjob zu verwechseln. Diese wird bei *FINALIZE* auf den Wert gesetzt, der bei *START* als Rückgabewert des Callbacks bestimmt wurde. Dies ermöglicht Ihrem Programm eine eindeutige Zuordnung der Drilldown-Jobs.

_hAttachInfo: Dieser Parameter wird für *LlAssociatePreviewControl()* benötigt, um den Viewer attachen zu können. Zusätzlich müssen die beiden Flags *LL_ASSOCIATEPREVIEWCONTROLFLAG_DELETE_ON_CLOSE* und *LL_ASSOCIATEPREVIEWCONTROLFLAG_HANDLE_IS_ATTACHINFO* übergeben werden. Weitere Hinweise dazu finden Sie in Kapitel Direkter Druck und Export aus dem Designer

Hinweise:

Dieser Callback wird immer im Kontext des Vorschau Threads aufgerufen, unabhängig davon, ob er aus der Echtdatenvorschau im Designer oder aus dem direkten Vorschaudruck heraus aufgerufen wurde.

Beispiele:

Siehe Kapitel Direkter Druck und Export aus dem Designer

LL_QUERY_DESIGNERACTIONSTATE

Aufgabe:

Über diesen Callback fragt List & Label den Zustand von Benutzerdefinierten Aktionen (vgl. *LlDesignerAddAction()*) ab. Sie können die Aktionen auf diese Weise je nach Notwendigkeit deaktivieren.

Aktivierung:

Immer aktiv

Parameter:

HIWORD(lParam): die (von Ihnen selbst vergebene) ID der Aktion

LOWORD(lParam): Vom Designer errechneter Status

Rückgabewert (_lResult):

Wert	Bedeutung
1	Aktion ist aktiv
2	Aktion ist inaktiv

Beispiel:

```
case LL_QUERY_DESIGNERACTIONSTATE:
    _lResult = (bEnabled? 1 : 2);
   break;
```

LL_QUERY_EXPR2HOSTEXPRESSION

Aufgabe:

Über diesen Callback weist List & Label den Host an, einen Filterausdruck (wie im Designer konfiguriert) in die native Syntax der Datenquelle zu übersetzen. Dieser Callback wird mehrere Male, für jeden Teil des Filterausdrucks, ausgelöst sobald die Anwendung *LlPrintDbGetCurrentTableFilter()* aufruft. Dieser Callback kann z.B. dafür verwendet werden, List & Label Filter in einer SQL-Abfrage zu übersetzen.

Aktivierung:

Immer aktiv, ausgelöst durch den Aufruf von *LlPrintDbGetCurrentTableFilter()*.

Parameter:

lParam zeigt auf eine *scLLEXPR2HOSTEXPR*-Struktur. Der Präfix "_p" bedeutet, dass es sich um einen Pointer auf das Argument handelt, in der Beschreibung wird es aber für die Lesbarkeit Argument genannt.

_nSize: Größe der Struktur

_pszTableID: Tabelle, zu der dieser Ausdruck gehört.

_nType: Art des Elements, das übersetzt werden soll. Für die meisten Operationen setzen Sie _pvRes auf das resultierende Statement für die Operation. Wenn _nType z.B. LLEXPR2HOSTEXPR_ARG_BINARY_OPERATOR_ADD ist, würde der typische Rückgabewert `_pvRes = _pv1 + _T("+") + _pv2` sein.

Wert	Bedeutung
LLEXPR2HOSTEXPR_ARG_BOOLEAN	Ein Boole'scher Wert.
LLEXPR2HOSTEXPR_ARG_TEXT	Ein Text-Wert. Wenn Sie Ihre Abfrage parametrisieren müssen um SQL Injection Angriffe zu vermeiden, setzen Sie den _pvName Member auf einen Parameternamen (eingangs enthält _pvName einen eindeutige, fortlaufende Integer Index, den Sie für den Namen verwenden können wenn nötig) und setzen Sie _pvRes auf den resultierende Text. Die Parameterwerte werden in die entsprechenden Variants zurückgeliefert, die an *LlPrintDbGetCurrentTableFilter()* übergeben wurden.
LLEXPR2HOSTEXPR_ARG_NUMBER	Ein numerischer Wert. _pvArg1->vt ist entweder VT_I4 für einen Integer- oder VT_R8 für einen Fließkomma-Wert.
LLEXPR2HOSTEXPR_ARG_DATE	Ein Datumswert
LLEXPR2HOSTEXPR_ARG_UNARY_OPERATOR_SIGN	Der Vorzeichen-Operator
LLEXPR2HOSTEXPR_ARG_UNARY_OPERATOR_NEGATION	Der Negierungs-Operator
LLEXPR2HOSTEXPR_ARG_BINARY_OPERATOR_ADD	Der "+" Operator
LLEXPR2HOSTEXPR_ARG_BINARY_OPERATOR_SUBTRACT	Der "-" Operator
LLEXPR2HOSTEXPR_ARG_BINARY_OPERATOR_MULTIPLY	Der "*" Operator
LLEXPR2HOSTEXPR_ARG_BINARY_OPERATOR_DIVIDE	Der "/" Operator
LLEXPR2HOSTEXPR_ARG_BINARY_OPERATOR_MODULO	Der "%" Operator

LLEXPR2HOSTEXPR_ARG_LOGICAL_OPERATOR_XOR	Der logische xor Operator
LLEXPR2HOSTEXPR_ARG_LOGICAL_OPERATOR _OR	Der logische or Operator
LLEXPR2HOSTEXPR_ARG_LOGICAL_OPERATOR_AND	Der logische and Operator
LLEXPR2HOSTEXPR_ARG_RELATION_EQUAL	Der "=" Operator
LLEXPR2HOSTEXPR_ARG_RELATION_NOTEQUAL	Der "<>" Operator
LLEXPR2HOSTEXPR_ARG_RELATION_LARGERTHAN	Der ">" Operator
LLEXPR2HOSTEXPR_ARG_RELATION_LARGEREQUAL	Der ">=" Operator
LLEXPR2HOSTEXPR_ARG_RELATION_SMALLERTHAN	Der "<" Operator
LLEXPR2HOSTEXPR_ARG_RELATION_SMALLEREQUAL	Der "<=" Operator
LLEXPR2HOSTEXPR_ARG_FUNCTION	Eine Designer-Funktion. _pvName enthält den Funktionsnamen, _pv1…_pv4 enthalten die Funktionsargumente.
LLEXPR2HOSTEXPR_ARG_FIELD	Ein Datenbankfeld. Abhängig von der Zielsyntax kann es notwendig sein, einen Identifier-Namen zu maskieren oder einzurahmen.

_pvRes: Ein VARIANT, der den resultierenden Ausdruck bekommt. Setzen Sie den Pointer auf NULL oder den VARIANT-Typ auf VT_EMPTY, wenn keine geeignete Übersetzung verfügbar ist. Der ganze (oder im Falle eines AND Operators der aktuelle Zweig des) Ausdruck(s) ist nicht übersetzt.

_nArgs: Anzahl Argumente. Dieser Member ist wichtig für Funktionen mit optionalen Argumenten.

_pvName: Name der zu übersetzenden Funktion. Dieser Member kann auch gesetzt werden um Abfrage-Parameter (siehe oben) zu behandeln.

_pv1: Abhängig von _nType (siehe oben), ist dies das erste Argument einer Funktion oder der linke Teil eines Operators.

_pv2: Abhängig von _nType (siehe oben), ist dies das zweite Argument einer Funktion oder der rechte Teil eines Operators.

_pv3: Das dritte Argument einer Funktion.

_pv4: Das vierte Argument einer Funktion.

Rückgabewert:

Wert	Bedeutung
0	Die Übersetzung wurde nicht verarbeitet oder konnte nicht verarbeitet werden. Der ganze (oder im Falle eines AND Operators der aktuelle Zweig des) Ausdruck(s) ist nicht übersetzt.
1	Die Übersetzung wurde genau verarbeitet.
2	Die Übersetzung wurde ungenau verarbeitet, das Ergebnis wird mehr Datensätze als passend enthalten. In diesem Fall wird List & Label zusätzlich seinen eigenen Filter anwenden um die überzähligen Datensätze zu filtern.

6.3 Verwaltung der Preview-Dateien

6.3.1 Überblick

Der in List & Label enthaltene Preview-Druck speichert das Ergebnis in einer Datei, wodurch eine Weiterverarbeitung möglich wird. Die Datei erhält die Namensendung ".ll".

6.3.2 Zugriffsfunktionen

Die Zugriffsfunktionen sind in der Datei CMLS20.DLL vorhanden. Diese DLL, die bei Internet-Anwendungen üblicherweise mitgeliefert werden sollte, ist möglichst klein gehalten. Wenn Sie Funktionen aus dieser DLL über eine Import-Bibliothek verwenden wollen, so müssen Sie die Datei CMLS20.LIB unter den Linker Einstellungen hinzufügen. In anderen Programmiersprachen reicht das Hinzufügen der entsprechenden Deklarationsdatei.

Nachfolgend die Auflistung der Funktionen:

LIStgsysAppend

Syntax:

```
INT LlStgsysAppend (HLLSTG hStg, HLLSTG hStgToAppend);
```

Aufgabe:

Hängt einen weiteren Druckjob an.

Parameter:

hStg: Das von *LIStgsysStorageOpen()* zurückgelieferte Handle

hStgToAppend: Handle der anzuhängenden Preview-Datei

Rückgabewert:

0: okay, <0: Fehlercode

Hinweise:

Diese Funktion setzt voraus, dass beide Dateien im *LL_STG_STORAGE*-Format erstellt wurden!

Wenn die hinzuzufügende Datei eine Seite für den Rückseitendruck enthält wird diese nur dann berücksichtigt, wenn in der Ursprungsdatei noch keine solche Seite enthalten ist.

Beispiel:

```
HLLSTGhStgOrg;
HLLSTGhStgAppend;

hStgOrg = LlStgsysStorageOpen("c:\\test\\label1.ll", "", FALSE, TRUE);
hStgAppend = LlStgsysStorageOpen("c:\\test\\label2.ll", "", FALSE,
TRUE);
LlStgsysAppend(hStgOrg,hStgAppend);
LlStgsysStorageClose(hStgOrg);
LlStgsysStorageClose(hStgAppend);
```

LlStgsysConvert

Syntax:

```
INT LlStgsysConvert (HLLSTG hStg, LPCTSTR pszDstFilename,
   LPCTSTR pszFormat);
```

Aufgabe:

Konvertiert eine Vorschaudatei in ein anderes Format.

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

pszDstFilename: Name der Zieldatei. Dieser kann auch den Formatierungsplatzhalter %d enthalten (z.B. "Seite %d"). Dies ist wichtig z.B. für JPEG-Konvertierung, da sonst nur eine einzige Seite erzeugt werden würde.

pszFormat: Gewünschtes Zielformat. Erlaubte Werte:

"TIFF" (auch als "PICTURE_MULTITIFF")
"JPEG" (auch als "PICTURE_JPEG")
"PNG" (auch als "PICTURE_PNG")
"EMF"
"TTY"
"PDF"
"XPS"
"PRN"
"TXT"

Über diesen Parameter kann eine semikolonseparierte Liste mit darin enthaltenen exporterspezifischen Optionen für die Konvertierung gesetzt werden. Die möglichen Parameter finden Sie im Kapitel über die Export-Module dokumentiert. Beachten Sie, dass nicht alle dort angegebenen Parameter berücksichtigt werden können. Ein Beispiel wäre die Übergabe von "PDF; PDF.Encryption.EncryptFile=1".

Zusätzlich zu den im o.g. Kapitel beschriebenen Parametern können die folgenden Parameter übergeben werden:

Wert	Bedeutung
PageIndexRange	Analog zum Druckdialog kann ein Bereich für die Seiten angegeben werden.
JobIndexRange	Analog zum Druckdialog kann ein Bereich für den Job angegeben werden.
IssueIndexRange	Analog zum Druckdialog kann ein Bereich für die Ausfertigungen angegeben werden.

Ein Beispiel hierfür wäre die Verwendung von "PDF;Export.PageIndexRange=2-3". Damit werden lediglich die Seiten 2 und 3 in das erzeugte PDF aufgenommen.

Rückgabewert:

0: okay, <0: Fehlercode

Hinweise:

-

Beispiel:

```
HLLSTG hStgOrg;

hStgOrg = LlStgsysStorageOpen("c:\\test\\label1.ll", "", FALSE, TRUE);
LlStgsysConvert(hStgOrg, "c:\\test\\label2.pdf", "PDF");
LlStgsysStorageClose(hStgOrg);
```

Siehe auch:

LlStgsysStorageOpen, LlStgsysStorageConvert

LlStgsysDeleteFiles

Syntax:

```
INT LlStgsysDeleteFiles (HLLSTG hStg);
```

Aufgabe:

Löschen der Preview-Datei(en).

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

Rückgabewert:

0: okay, <0: Fehlercode

Hinweise:

Diese Funktion löscht alle Preview-Datei(en). Sinnvollerweise sollte danach nur noch der Aufruf *LlStgsysStorageClose()* folgen.

Siehe auch:

LlStgsysStorageOpen, LlStgsysStorageClose

LlStgsysDestroyMetafile

Syntax:

```
INT LlStgsysDestroyMetafile (HANDLE hMF);
```

Aufgabe:

Gibt das Metafile wieder frei.

Parameter:

hMF: Das Metafile-Handle

Rückgabewert:

0: okay, <0: Fehlercode

Beispiel:

Siehe *LlStgsysGetPageMetafile()*

Siehe auch:

LlStgsysGetPageMetafile, LlStgsysGetPageMetafile16

LlStgsysDrawPage

Syntax:

```
INT LlStgsysDrawPage (HLLSTG hStg, HDC hDC, HDC hPrnDC, BOOL
bAskPrinter,
   PCRECT pRC, INT nPageIndex, BOOL bFit, LPVOID pReserved);
```

Aufgabe:

Ausgabe einer Seite auf Bildschirm oder Drucker.

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

hDC: DC, in den gezeichnet werden soll (meist Drucker- oder Bildschirm-DC). Darf NULL sein (s.u.)

hPrnDC: Referenz-DC, über den Seitenränder etc. bestimmt werden sollen. Bei Druck auf Drucker ist dies meist gleich *hDC*, bei Druck auf Bildschirm der Default-Drucker-DC. Darf NULL sein (s.u.)

bAskPrinter: Wenn *hPrnDC* NULL ist, entscheidet dieses Flag, ob der Benutzer gefragt wird, welchen Drucker er als Referenz verwenden will (TRUE), oder ob der Default-Drucker verwendet werden soll (FALSE).

pRC: Rechteck in Device-Koordinaten, in dem die Ausgabe erfolgen soll. Wenn dieser Wert NULL ist, wird bei Ausgabe auf Drucker dessen Druckbereich angenommen. Darf für Bildschirmausgabe nicht NULL sein!

nPageIndex: Index (1..) der gewünschten Seite

bFit: Soll die Ausgabe in das Rechteck von pRC optimal eingepasst werden (TRUE), oder soll die absolute Größe erhalten bleiben (FALSE)?

pReserved: NULL

Rückgabewert:

0: okay, <0: Fehlercode

Hinweise:

Wenn *hDC* NULL ist, wird *hDC* mit *hPrnDC* gleichgesetzt, nachdem der Referenzkontext erzeugt wurde.

Siehe auch:

LIStgsysPrint, LIStgsysStoragePrint

LIStgsysGetAPIVersion

Syntax:

```
INT LlStgsysGetAPIVersion (HLLSTG hStg);
```

Aufgabe:

Gibt die Version der Stgsys-Funktionsschnittstelle zurück.

Parameter:

hStg: Das von *LIStgsysStorageOpen()* zurückgelieferte Handle

Rückgabewert:

Die Versionsnummer der Stgsys-API in List & Label C?LL20.DLL und C?LS20.DLL.

Hinweise:

Die gegenwärtige Version ist 20 (Stand: 10/2014).

Diese Funktion sollte immer verwendet werden, um die Version der API vor dem Aufruf zu testen, da zukünftige APIs oft mehr Funktionalität zur Verfügung stellen.

Siehe auch:

LlStgsysGetFileVersion

LlStgsysGetFilename

Syntax:

```
INT LlStgsysGetFilename (HLLSTG hStg, INT nJob, INT nFile,
   LPTSTR pszBuffer, UINT nBufSize);
```

Aufgabe:

Gibt die Information über die Dateinamen der Preview-Datei(en).

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

nJob: Job-Nummer. 1: erster Job, ... (1..*LlStgsysGetJobCount()*)

nFile: Seitennummer für Dateinamen

Wert	Bedeutung
-1	Preview-Dateiname
0	Druckerkonfigurationsdateiname
>0	Seitennummer (1..*LlStgsysGetPageCount*())

lpszBuffer: Puffer für Dateiname

nBufSize: Größe des Puffers

Rückgabewert:

0: okay, <0: Fehlercode

Hinweise:

Durch den *nFile*-Parameter wird unterschieden, welcher Dateityp gemeint ist.

Bei Verwendung von *LL_STG_STORAGE* als Speichersystem liefern alle Aufrufe diesen Dateinamen zurück, da alle Informationen in dieser Datei gespeichert sind.

Beispiel:

```
CString  sFilename, sOutput;
LlStgsysGetFilename(hStg, 1, -1,
            sFilename.GetBuffer(_MAX_PATH), _MAX_PATH);
sFilename.ReleaseBuffer();
sOutput = CString(_T("Ansicht von ")) + sFilename;
```

Siehe auch:

LIStgsysGetJobCount

LIStgsysGetFileVersion

Syntax:

```
INT LlStgsysGetFileVersion (HLLSTG hStg);
```

Aufgabe:

Gibt die Versionsnummer der Vorschaudatei zurück.

Parameter:

hStg: Das von *LIStgsysStorageOpen()* zurückgelieferte Handle

Rückgabewert:

Versionsnummer und Typ der Preview-Datei:

Wert	Bedeutung
Bits 0..7	Versionsnummer, momentan 20 (Stand 10/2014)

Hinweise:

Dieser Aufruf ist ebenfalls wichtig, um mögliche Unterschiede der Preview-Datei-Versionen behandeln zu können.

Siehe auch:

LIStgsysGetAPIVersion

LIStgsysGetJobCount

Syntax:

```
INT LlStgsysGetJobCount (HLLSTG hStg);
```

Aufgabe:

Liefert die Zahl der in einer Vorschaudatei gespeicherten Druckjobs

Parameter:

hStg: Das von *LIStgsysStorageOpen()* zurückgelieferte Handle

Rückgabewert:

>0: Zahl der Druckjobs, <0: Fehlercode

Beispiel:

siehe *LIStgsysSetJob()*

Siehe auch:

LlStgsysStorageOpen, LlStgsysSetJob

LlStgsysGetJobOptionStringEx

Syntax:

```
INT LlStgsysGetJobOptionStringEx (HLLSTG hStg, LPCTSTR pszKey,
   LPTSTR pszBuffer, UINT nBufSize);
```

Aufgabe:

Abfrage diverser Zeichenketten.

Parameter:

hStg: von *LlStgsysStorageOpen()* geliefertes Handle

pszKey: Name der Option

pszBuffer: Adresse des Puffers für den gewünschten Wert

nBufSize: Größe des Puffers (inkl. 0-Terminierung)

Rückgabewert:

0: okay, <0: Fehlercode

Hinweise:

Die zur Verfügung stehenden Options-Werte sind die Parameter, die man über *LlPrintSetProjectParameter()* bzw. *LlSetDefaultProjectParameter()* als PUBLIC angemeldet hat. Beachten Sie, dass Sie zur Abfrage den Präfix "ProjectParameter." verwenden müssen. Lesen Sie hierzu auch das Kapitel über die Projekt-Parameter.

Siehe auch:

LlStgsysGetJobOptionValue

LlStgsysGetJobOptionValue

Syntax:

```
INT LlStgsysGetJobOptionValue (HLLSTG hStg, INT nOption);
```

Aufgabe:

Abfrage diverser numerische Einstellungen eines Jobs.

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

nOption: bestimmt die Art des Rückgabewerts

Wert	Bedeutung
LS_OPTION_BOXTYPE	Gibt den Stil der Wartebox an, der beim Preview-Druck verwendet wird (bzw. beim Druck auf Drucker im Preview). Dies ist eine der Konstanten *LL_BOXTYPE_xxx* (bei *LlPrintWithBoxStart*()) oder -1, wenn keine Box (*LLPrintStart*()).
LS_OPTION_UNITS	Gibt die Einheiten des Projekts zurück, siehe LL_PRNOPT_UNIT
LS_OPTION_PRINTERCOUNT	Anzahl der verwendeten Drucker.

Die Werte sind für eine selbstprogrammierte Ausgabe der Preview-Dateien auf vom Originaldrucker verschiedenen Druckern wichtig.

Rückgabewert:

>=0: Wert, <0: Fehlercode

Siehe auch:

LlStgsysGetJobOptionStringEx

LlStgsysGetLastError

Syntax:

```
INT LlStgsysGetLastError (HLLSTG hStg);
```

Aufgabe:

Gibt den Fehlercode des letzten Aufrufs der Storage-API zurück.

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

Rückgabewert:

<0. Fehlercode, 0: okay

Hinweise:

Kann für Funktionen verwendet werden, die NULL als Fehlerkennzeichnung zurückgeben (z.B. *LlStgsysGetPageMetafile()*)

Siehe auch:

LlStgsysGetPageMetafile

LlStgsysGetPageCount

Syntax:

```
INT LlStgsysGetPageCount (HLLSTG hStg);
```

Aufgabe:

Gibt die Zahl der in einem Job enthaltenen Seiten zurück.

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

Rückgabewert:

>0: Seitenzahl, <0: Fehlercode

Hinweise:

Dies gibt nur die Zahl der enthaltenen Seiten zurück. Die Seitennummern können durch Aufruf von *LlStgsysPageGetOptionValue*(*LS_OPTION_PAGENUMBER*)) bestimmt werden.

Beispiel:

Siehe *LlStgsysSetJob()*

Siehe auch:

LlStgsysSetJob, LlStgsysJobGetOptionValue

LlStgsysGetPageMetafile

Syntax:

```
HANDLE LlStgsysGetPageMetafile (HLLSTG hStg, INT nPageIndex);
```

Aufgabe:

Erzeugt ein (Enhanced-)Metafile-Handle, das dann zur Anzeige oder zum Druck verwendet werden kann.

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

nPageIndex: Der Seitenindex (1..*LlStgsysGetPageCount()*)

Rückgabewert:

NULL: Fehler, sonst: Metafile-Handle

Hinweise:

Der Rückgabewert ist ein Enhanced Metafile-Handle.

Das Handle muss nach Verwendung über *LlStgsysDestroyMetafile()* wieder freigegeben werden.

Beispiel:

```
HANDLE hMF;

hMF = LlStgsysGetPageMetafile(hStg, nPageIndex);
if (hMF == NULL)
```

```
{
   hMF = LlStgsysGetPageMetafile16(hStg, nPageIndex);
}
if (hMF == NULL)
   ret = LL_ERR_STG_CANNOTGETMETAFILE;
else
{
   POINT ptPixels;
   POINT ptPixelsOffset;
   POINT ptPixelsPhysical;
   POINT ptPixelsPerInch;

   ptPixels.x = LlStgsysGetPageOptionValue(hStg, nPageIndex,
         LS_OPTION_PRN_PIXELS_X);
   ptPixels.y = LlStgsysGetPageOptionValue(hStg, nPageIndex,
         LS_OPTION_PRN_PIXELS_Y);
   ptPixelsOffset.x = LlStgsysGetPageOptionValue(hStg, nPageIndex,
         LS_OPTION_PRN_PIXELSOFFSET_X);
   ptPixelsOffset.y = LlStgsysGetPageOptionValue(hStg, nPageIndex,
         LS_OPTION_PRN_PIXELSOFFSET_Y);
   ptPixelsPhysical.x = LlStgsysGetPageOptionValue(hStg, nPageIndex,
         LS_OPTION_PRN_PIXELSPHYSICAL_X);
   ptPixelsPhysical.y =LlStgsysGetPageOptionValue(hStg, nPageIndex,
         LS_OPTION_PRN_PIXELSPHYSICAL_Y);
   ptPixelsPerInch.x = LlStgsysGetPageOptionValue(hStg, nPageIndex,
         LS_OPTION_PRN_PIXELSPERINCH_X);
   ptPixelsPerInch.y = LlStgsysGetPageOptionValue(hStg, nPageIndex,
         LS_OPTION_PRN_PIXELSPERINCH_Y);
   <Paint Metafile>
   LlStgsysDestroyMetafile(hMF);
}
```

Siehe auch:

LlStgsysDestroyMetafile

LlStgsysGetPageOptionString

Syntax:

```
INT LlStgsysGetPageOptionString (HLLSTG hStg, INT nPageIndex,
   INT nOption, LPTSTR pszBuffer, UINT nBufSize);
```

Aufgabe:

Abfrage diverser Zeichenketten.

Parameter:

hStg: von *LlStgsysStorageOpen()* geliefertes Handle

nPageIndex: Seitenindex (1..*LlStgsysGetPageCount()*)

nOption: bestimmt die Art des Rückgabewerts:

Wert	Bedeutung
LS_OPTION_PROJECTNAME	Name der Projektdatei, von der die Seite stammt

Wert	Bedeutung
LS_OPTION_JOBNAME	Name des Jobs (siehe *LlPrintWithBoxStart*())
LS_OPTION_USER	benutzerspezifisch (siehe *LlStgsysPageSetOptionString*())
LS_OPTION_CREATION	Erstellungsdatum
LS_OPTION_CREATIONAPP	Erstellungs-Applikation
LS_OPTION_CREATIONDLL	Erstellungs-DLL
LS_OPTION_CREATIONUSER	Benutzer- und Computer-Name bei der Erstellung
LS_OPTION_PRINTERALIASLIST	Siehe *LL_OPTIONSTR_PRINTERALIASLIST*: Die zum Zeitpunkt der Erstellung der Vorschaudatei gültige Druckerersetzungstabelle (eine Zeichenkette; die einzelnen Zeilen sind durch Zeilenumbruch getrennt)
LS_OPTION_USED_PRTDEVICE	Devicename (z.B. "HP Laserjet 4L")

pszBuffer: Adresse des Puffers für den gewünschten Wert

nBufSize: Größe des Puffers (inkl. 0-Terminierung)

Rückgabewert:

0: okay, <0: Fehlercode

Siehe auch:

LlStgsysGetPageOptionValue, LlStgsysSetPageOptionString

LlStgsysGetPageOptionValue

Syntax:

```
INT LlStgsysGetPageOptionValue (HLLSTG hStg, INT nPageIndex,
   INT nOption);
```

Aufgabe:

Abfrage diverser numerischer Einstellungen.

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

nPageIndex: Seitenindex (1..*LlStgsysGetPageCount()*)

nOption: bestimmt die Art des Rückgabewerts:

Wert	Bedeutung
LS_OPTION_PAGENUMBER	Gibt die Seitennummer der ausgewählten

Wert	Bedeutung
	Seite zurück
LS_OPTION_COPIES	Gibt die Zahl der gewünschten Kopien für diese Seite zurück
LS_OPTION_PRN_ORIENTATION	Gibt die Orientierung der Seite zurück (*DMORIENT_PORTRAIT* oder *DMORIENT_LANDSCAPE*)
LS_OPTION_PHYSPAGE	Gibt die Information, ob die physikalische Seite bedruckt werden soll oder der bedruckbare Bereich
LS_OPTION_PRN_PIXELSOFFSET_X	Der horizontale Offset des bedruckbaren Bereichs relativ zum Seitenanfang (beim Originaldrucker)
LS_OPTION_PRN_PIXELSOFFSET_Y	Der vertikale Offset des bedruckbaren Bereichs relativ zum Seitenanfang (beim Originaldrucker)
LS_OPTION_PRN_PIXELS_X	Die horizontale Größe des bedruckbaren Bereichs (beim Originaldrucker)
LS_OPTION_PRN_PIXELS_Y	Die vertikale Größe des bedruckbaren Bereichs (beim Originaldrucker)
LS_OPTION_PRN_PIXELSPHYSICAL_X	Die horizontale Größe der Druckseite (beim Originaldrucker)
LS_OPTION_PRN_PIXELSPHYSICAL_Y	Die vertikale Größe der Druckseite (beim Originaldrucker)
LS_OPTION_PRN_PIXELSPERINCH_X	Die horizontale Drucker-Auflösung
LS_OPTION_PRN_PIXELSPERINCH_Y	Die vertikale Drucker-Auflösung
LS_OPTION_PRN_INDEX	Die Index des für diese Seite verwendeten Originaldruckers (0 für Erstseiten-, 1 für Folgeseitendrucker)
LS_OPTION_ISSUEINDEX	Gibt den Ausfertigungs-Index (1...) der Seite zurück.

Rückgabewert:

>=0: Wert, <0: Fehlercode

Hinweise:

"Drucker" bzw. "Originaldrucker" meint hier den zum Zeitpunkt der Erstellung der Druckdatei gewählten Drucker.

Die Werte sind für eine selbstprogrammierte Ausgabe der Preview-Dateien auf vom Originaldrucker verschiedenen Druckern unerlässlich.

Um den korrekten Wert zu erhalten, müssen Sie den gewünschten Job eingestellt haben!

Siehe auch:

LlStgsysGetPageOptionString

LlStgsysGetPagePrinter

Syntax:

```
INT LlStgsysGetPagePrinter (HLLSTG hStg, INT nPageIndex,
   LPTSTR pszDeviceName, UINT nDeviceNameSize, PHGLOBAL phDevmode);
```

Aufgabe:

Über diese Funktion kann man den Drucker und seine Einstellungen abfragen, der für die genannte Seite zuständig wäre.

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

nPageIndex: Der Seitenindex (1..*LlStgsysGetPageCount()*)

pszDeviceName: Zeiger auf Puffer für Device-Namen

nDeviceNameSize: Größe des Puffers

phDevmode: Zeiger auf globales Handle für die DEVMODE-Struktur. Kann NULL sein, wenn die DEVMODE-Struktur nicht abgefragt werden soll. Das Handle muss initialisiert sein (NULL oder ein gültiges Handle für einen globalen Block).

Rückgabewert:

Fehlercode

Hinweise:

Siehe auch:

LlGetPrinterFromPrinterFile, LlSetPrinterInPrinterFile

Beispiel:

```
HGLOBAL dev(NULL);
TCHAR*pszPrinter = new TCHAR[1024];
int iRet = LlStgsysGetPagePrinter(m_hStgOrg, 1, pszPrinter, 1096, &dev);
LPVOID pDevmode = GlobalLock(dev);
DEVMODE aDEVMODE = *((DEVMODE*)pDevmode);
…
```

```
// tidy-up
GlobalUnlock(dev);
GlobalFree(dev);
```

LlStgsysPrint

Syntax:

```
HLLSTG LlStgsysPrint (HLLSTG hStg, LPCTSTR pszPrinterName1,
   LPCTSTR pszPrinterName2, INT nStartPageIndex, INT nEndPageIndex,
   INT nCopies, UINT nFlags, LPCTSTR pszMessage, HWND hWndParent);
```

Aufgabe:

Druckt Seiten aus einer geöffneten Druckvorschaudatei

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

pszPrinterName1: Name des Druckers, der für die erste Seite verwendet wird (NULL -> s.u.)

pszPrinterName2: Name des Druckers, der für Folgeseiten verwendet wird (NULL -> s.u.)

nStartPageIndex: Index der ersten zu druckenden Seite

nEndPageIndex: Index der letzten zu druckenden Seite

nCopies: Zahl der Kopien

nFlags: Verknüpfung folgender Flags:

Wert	Bedeutung
LS_PRINTFLAG_FIT	Einpassen auf die maximale bedruckbare Fläche des Druckers
LS_PRINTFLAG_STACKEDCOPIES	Die Kopien werden pro Blatt, nicht pro Job ausgeführt (111222333 statt 123123123)
LS_PRINTFLAG_TRYPRINTERCOPIES	Kopien werden über Druckerfeature gedruckt, wenn vorhanden
LS_PRINTFLAG_METER	mit Fortschrittsdialog
LS_PRINTFLAG_ABORTABLEMETER	mit Fortschrittsdialog mit Abbrechen-Button
LS_PRINTFLAG_SHOWDIALOG	mit Druckerauswahldialog
LS_PRINTFLAG_FAX	Für Ausgabe auf Fax-Drucker benötigt

pszMessage: Wird im Titel eines optionalen Fortschrittsdialogs angezeigt und als Dokumentname für den Druck verwendet. Wenn *pszMessage* NULL oder auf eine leere Zeichenkette zeigt, wird der Eintrag aus der Vorschaudatei (Parameter von *LlPrintWithBoxStart()*) genommen

hWndParent: Fensterhandle, das als Parent für den optionalen Fortschrittsdialog genommen wird.

Rückgabewert:

0: okay, <0: Fehlercode

Hinweise:

Diese Routine druckt den gewählten Seitenbereich aus dem momentan eingestellten Storage-Job auf den/die angegebenen Drucker.

Wichtig: Der Job muss gültig sein, d.h. Sie müssen

```
LlStgsysSetJob(hStg,1);
```

(oder einen anderen Job, falls nötig) vor dem Aufruf durchführen.

Wenn ein Druckername NULL ist, versucht List & Label, den Drucker inklusive der zugehörigen Einstellungen aus der Preview-Datei zu lesen. Wenn dies nicht gelingt, d.h. der Drucker im System nicht existiert (der Device-Name ist der bestimmende Faktor), wird der im System eingestellte Default-Drucker genommen.

Siehe auch:

LlStgsysStoragePrint, LlStgsysSetJob

LlStgsysSetJob

Syntax:

```
INT LlStgsysSetJob (HLLSTG hStg, INT nJob);
```

Aufgabe:

Auswahl des Jobs für zukünftige Aufrufe

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

nJob: Die Nummer des Jobs (1..*LlStgsysGetJobCount()*)

Rückgabewert:

0: okay, <0: Fehlercode

Hinweise:

Nachfolgende Aufrufe, die jobabhängige Daten liefern, benutzen die Job-Nummer, die durch diese Funktion gesetzt wird.

Beispiel:

```
// berechnet die Gesamtzahl der enthaltenen Seiten
intnPages = 0;
intnJob;
for (nJob=1; nJob<LlStgsysGetJobCount(hStg); ++nJob)
{
   LlStgsysSetJob(hStg, nJob);
   nPages += LlStgsysGetPageCount(hStg);
}
```

Siehe auch:

LlStgsysGetJobCount

LlStgsysSetJobOptionStringEx

Syntax:

```
INT LlStgsysSetJobOptionStringEx (HLLSTG hStg, LPCTSTR pszKey,
   LPCTSTR pszValue);
```

Aufgabe:

Setzen diverser Zeichenketten.

Parameter:

hStg: von *LlStgsysStorageOpen()* geliefertes Handle

pszKey: Name der Option

pszValue: Wert

Rückgabewert:

0: okay, <0: Fehlercode

Hinweise:

Hierüber können Sie Werte in das LL-File schreiben (wenn es nicht READ-ONLY geöffnet wurde). Bitte vermeiden Sie Optionsnamen, die mit "LL." anfangen.

Siehe auch:

LlStgsysGetJobOptionStringEx

LlStgsysSetPageOptionString

Syntax:

```
INT LlStgsysSetPageOptionString (HLLSTG hStg, INT nPageIndex,
   INT nOption, LPCTSTR pszBuffer);
```

Aufgabe:

Setzen diverser Zeichenketten.

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

nPageIndex: Seitenindex (1..*LlStgsysGetPageCount()*)

nOption: bestimmt die Art des Inhalts:

Wert	Bedeutung
LS_OPTION_JOBNAME	Name des Jobs
LS_OPTION_USER	Ein benutzerspezifischer Wert, der dazu geeignet, eigene Informationen (z.B. Druckdatum, Benutzer o.ä.) in der Datei unterbringen zu können.

pszBuffer: Adresse des nullterminierten Puffers mit dem Inhalt

Rückgabewert:

0: okay, <0: Fehlercode

Hinweise:

Die Storage darf selbstverständlich nicht mit *bReadOnly*=TRUE geöffnet sein.

Beispiel:

```
LlStgsysSetJob(hStg,1);
LlStgsysSetPageOption(hStg,1, LS_OPTION_USER, "Buchstaben A-B");
```

Siehe auch:

LlStgsysGetPageOptionValue

LlStgsysStorageClose

Syntax:

```
void LlStgsysStorageClose (HLLSTG hStg);
```

Aufgabe:

Beendet den Zugriff auf die Druckvorschaudatei.

Parameter:

hStg: Das von *LlStgsysStorageOpen()* zurückgelieferte Handle

Siehe auch:

LlStgsysStorageOpen

LIStgsysStorageConvert

Syntax:

```
INT LlStgsysStorageConvert (LPCTSTR pszStgFilename,
   LPCTSTR pszDstFilename, LPCTSTR pszFormat);
```

Aufgabe:

Konvertiert eine Vorschaudatei in ein anderes Format.

Parameter:

pszStgFilename: Name der Ausgangsdatei

pszDstFilename: Name der Zieldatei

pszFormat: Gewünschtes Zielformat. Erlaubte Werte und erweiterte Optionen siehe *LIStgsysConvert()*.

Rückgabewert:

0: okay, <0: Fehlercode

Beispiel:

```
LlStgsysStorageConvert("c:\\test\\label2.ll",
            "c:\\test\\label2.pdf", "PDF");
```

Siehe auch:

LIStgsysStorageOpen, LIStgsysConvert

LIStgsysStorageOpen

Syntax:

```
HLLSTG LlStgsysStorageOpen (LPCTSTR lpszFilename, LPCTSTR pszTempPath,
   BOOL bReadOnly, BOOL bOneJobTranslation);
```

Aufgabe:

Öffnet eine Druckvorschaudatei

Parameter:

lpszFilename: Der Dateiname der Druckvorschaudatei oder der Projektdatei (List & Label setzt die Dateiextension immer auf .LL).

pszTempPath: Ein Dateipfad (kann NULL oder leer sein). Ist ein Pfad angegeben, wird eine evtl. Pfadangabe in *lpszFilename* überschrieben.

bReadOnly: *TRUE*: Die Datei wird im ReadOnly-Modus geöffnet, *FALSE*: Die Datei kann auch geschrieben werden

bJobTranslation: *TRUE*: Die Stgsys-Funktionen verwalten es transparent, wenn die Preview-Datei aus mehreren Jobs besteht, *FALSE*: Sie können (und müssen!) die Jobs getrennt verwalten

Rückgabewert:

Job-Handle für alle *LIStgsys()*-Funktionen, 0 für Fehler

Hinweise:

Wenn Sie einen Temporärpfad angeben, wird dieser als Pfad benutzt, ansonsten wird der Pfad der Projektdatei verwendet.

Diese Art des Aufrufs ist kompatibel mit dem Aufruf für die Preview-Anzeige *LIPreviewDisplay()*.

Die Dateiendung wird immer auf ".LL". gesetzt.

Es ist für die Funktionen *LIStgsysAppend()* und *LIStgsysPageSetOptionString()* wichtig, dass die Datei mit bReadOnly=FALSE geöffnet wird!

bJobTranslation ist sehr praktisch, wenn man die Verwaltung der Jobs der API überlassen möchte. Auf FALSE wird man es setzen, wenn die Anwender sehen sollen, dass die Datei möglicherweise aus mehreren Jobs besteht.

Siehe auch:

LIStgsysClose

LIStgsysStoragePrint

Syntax:

```
INT LlStgsysStoragePrint (LPCTSTR lpszFilename, LPCTSTR pszTempPath,
   LPCTSTR pszPrinterName1, LPCTSTR pszPrinterName2,
   INT nStartPageIndex, INT nEndPageIndex, INT nCopies, UINT nFlags,
   LPCTSTR pszMessage,HWND hWndParent);
```

Aufgabe:

Druckt eine Druckvorschaudatei

Parameter:

lpszFilename: Der Dateiname der Druckvorschaudatei oder der Projektdatei (List & Label setzt die Dateiextension immer auf .LL).

pszTempPath: Ein Temporärpfad (kann NULL oder leer sein)

pszPrinterName1: Name des Druckers, der für die erste Seite verwendet wird (NULL -> s.u.)

pszPrinterName2: Name des Druckers, der für Folgeseiten verwendet wird (NULL -> s.u.)

nStartPageIndex: Index der ersten zu druckenden Seite

nEndPageIndex: Index der letzten zu druckenden Seite

nCopies: Zahl der Kopien

nFlags: Verknüpfung folgender Flags:

Wert	Bedeutung
LS_PRINTFLAG_FIT	Einpassen auf die maximale bedruckbare Fläche des Druckers
LS_PRINTFLAG_-STACKEDCOPIES	Die Kopien werden pro Blatt, nicht pro Job ausgeführt (111222333 statt 123123123)
LS_PRINTFLAG_-TRYPRINTERCOPIES	Kopien werden über Druckerfeature gedruckt, wenn vorhanden
LS_PRINTFLAG_METER	mit Fortschrittsdialog
LS_PRINTFLAG_-ABORTABLEMETER	mit Fortschrittsdialog mit Abbruchbutton
LS_PRINTFLAG_SHOWDIALOG	mit Druckerauswahldialog
LS_PRINTFLAG_FAX	Für Ausgabe auf Fax-Drucker benötigt

pszMessage: Wird im Titel eines optionalen Fortschrittsdialogs angezeigt und als Dokumentname für den Druck verwendet. Wenn *pszMessage* NULL oder auf eine leere Zeichenkette zeigt, wird der Eintrag aus der Vorschaudatei genommen

hWndParent: Fensterhandle, das als Parent für den optionalen Fortschrittsdialog genommen wird.

Rückgabewert:

0: okay, <0: Fehlercode

Hinweise:

Diese Routine öffnet die Vorschaudatei und druckt den gewählten Bereich auf den/die angegebenen Drucker.

Wenn ein Druckername NULL ist, versucht List & Label, den Drucker inklusive der zugehörigen Einstellungen aus der Preview-Datei zu lesen. Wenn dies nicht gelingt, d.h. der Drucker im System nicht existiert (der Device-Name ist der bestimmende Faktor), wird der im System eingestellte Default-Drucker genommen.

Siehe auch:

LIStgsysPrint

LsMailConfigurationDialog

Syntax:

```
INT LsMailConfigurationDialog (HWND hWndParent, LPCTSTR pszSubkey,
   UINT nFlags, INT nLanguage);
```

Aufgabe:

Öffnet einen Dialog zur Konfiguration der Mailversandparameter, wenn der Mailversand über das Modul CMMX20.DLL abgewickelt werden soll (s. Kapitel "Exportdateien per eMail verschicken").

Die Einstellungen werden unter "HKEY_CURRENT_USER\software\combit\cmbtmx\<pszSubkey>\<User|Computer>" gespeichert.

Parameter:

hWndParent: Parent Fenster-Handle für den Dialog

pszSubkey: Unterschlüssel, der für das Speichern der Werte in der Registry verwendet wird. Hier sollte der Name der ausführbaren Datei (ohne Pfad und Dateierweiterung) der Applikation übergeben werden, dann werden die gewählten Werte bei einem Versand für diese Applikation automatisch gesetzt.

nFlags: eins oder mehrere der folgenden Flags

Wert	Bedeutung
LS_MAILCONFIG_USER	benutzerspezifische Daten
LS_MAILCONFIG_GLOBAL	computerspezifische Daten
LS_MAILCONFIG_PROVIDER	Auch Provider-Auswahl (SMAPI, SMTP, ...)

Alle Daten werden benutzerspezifisch gespeichert (auch die computerspezifischen Daten), die Flags definieren nur eine logische Trennung für den Dialog (Servereinstellungen und Benutzerdaten).

nLanguage: gewählte Sprache für den Dialog

Wert	Bedeutung
CMBTLANG_DEFAULT	im System voreingestellte Sprache
CMBTLANG_GERMAN	Deutsch
CMBTLANG_ENGLISH	Englisch

Weitere Konstanten in den Deklarationsdateien.

Rückgabewert:

0: okay, <0: Fehlercode

LsMailGetOptionString

Syntax:

```
INT LsMailGetOptionString (HLSMAILJOB hJob, LPCTSTR pszKey,
   LPTSTR pszBuffer, UINT nBufSize);
```

Aufgabe:

Liest die im Moment des Aufrufs gültigen eMail-Einstellungen für List & Label aus.

Parameter:

hJob: List & Label eMail-API Job-Handle

pszKey: Optionsname. Mögliche Werte siehe *LsMailSetOptionString()*.

lpszBuffer: Zeiger auf Puffer, in den der Inhalt gespeichert werden soll

nBufSize: Größe des Puffers

Rückgabewert:

0: okay, <0: Fehlercode

Siehe auch:

LsMailSetOptionString

LsMailJobClose

Syntax:

```
INT LsMailJobClose (HLSMAILJOB hJob);
```

Aufgabe:

Schließen des DLL-Jobs.

Parameter:

hJob: List & Label eMail-API Job-Handle

Rückgabewert:

0: okay, <0: Fehlercode

Hinweise:

Diese Funktion muss nach Benutzung der eMail-Funktionen der List & Label-DLL oder bei Beendigung Ihres Programms aufgerufen werden (paarweise mit *LsMailJobOpen()*.

Beispiel:

```
HLSMAILJOB  hMailJob;

hMailJob = LsMailJobOpen(CMBTLANG_DEFAULT);
...
LsMailJobClose(hMailJob)
```

Siehe auch:

LsMailJobOpen

LsMailJobOpen

Syntax:

```
HLSMAILJOB LsMailJobOpen (INT nLanguage);
```

Aufgabe:

Öffnet einen eMail-API Job.

Parameter:

nLanguage: gewählte Sprache für Benutzer-Interaktionen

Wert	Bedeutung
CMBTLANG_DEFAULT	im System voreingestellte Sprache
CMBTLANG_GERMAN	Deutsch
CMBTLANG_ENGLISH	Englisch

Weitere Konstanten in den Deklarationsdateien.

Rückgabewert:

Ein Handle, das bei den meisten Funktionen als Parameter benötigt wird, um auf die applikationsspezifischen Daten zugreifen zu können, 0 für Fehler.

Beispiel:

```
HLSMAILJOB   hMailJob;
hMailJob = LsMailJobOpen(0);
```

Siehe auch:

LsMailJobClose

LsMailSendFile

Syntax:

```
INT LsMailSendFile (HLSMAILJOB hJob, HWND hWndParent);
```

Aufgabe:

Versendet eine eMail mit den aktuellen Einstellungen.

Parameter:

hJob: List & Label eMail-API Job-Handle

hWndParent: Parent-Fensterhandle für den Maildialog. Wenn als Fensterhandle "0" angegeben wird, wird kein Versendedialog angezeigt und die eMail - wenn möglich - ohne Benutzerinteraktion versendet.

Rückgabewert:

0: okay, <0: Fehlercode

Beispiel:

```
HLSMAILJOB   hMailJob;

hMailJob = LsMailJobOpen(0);
LsMailSetOptionString(hMailJob, "Export.Mail.To", "test@domainname.de");
LsMailSetOptionString(hMailJob, "Export.Mail.Subject", "Test!");
```

```
LsMailSetOptionString(hMailJob, "Export.Mail.AttachmentList",
   "c:\\test.txt");
LsMailSendFile(hMailJob, 0);
LsMailJobClose(hMailJob)
```

Siehe auch:

LsMailSetOptionString

LsMailSetOptionString

Syntax:

```
INT LsMailSetOptionString (HLSMAILJOB hJob, LPCTSTR pszKey,
   LPCTSTR pszValue);
```

Aufgabe:

Setzt diverse eMail-Einstellungen in List & Label.

Parameter:

hJob: List & Label eMail-API Job-Handle

pszKey: Folgende Werte sind möglich:

Wert	Bedeutung
Export.Mail.To	Empfänger eMail Adressen, ggf. auch mehrere per Semikolon getrennt
Export.Mail.CC	CC Empfänger eMail Adressen, ggf. auch mehrere per Semikolon getrennt
Export.Mail.BCC	BCC Empfänger eMail Adressen, ggf. auch mehrere per Semikolon getrennt
Export.Mail.Subject	Betreffzeile der eMail
Export.Mail.Body	Nachrichtentext der eMail
Export.Mail.Body:text/html	HTML-Nachrichtentext der eMail. Angabe optional, ansonsten wird die Nachricht als reine Textnachricht mit dem unter *Export.Mail.Body* angegebenen Text versendet.
Export.Mail.AttachmentList	Tabulator-separierte Liste von Dateianhängen

pszValue: neuer Wert

Rückgabewert:

0: okay, <0: Fehlercode

Beispiel:

```
HLSMAILJOB  hMailJob;

hMailJob = LsMailJobOpen(0);
LsMailSetOptionString(hMailJob, "Export.Mail.To",
   "test@domainname.de ");
```

```
...
LsMailJobClose(hMailJob)
```

Siehe auch:

LsMailGetOptionString

LsSetDebug

Syntax:

```
void LsSetDebug (BOOL bOn);
```

Aufgabe:

Schaltet den Debug-Modus der LS-API ein oder aus.

Parameter:

bOn: gibt den gewünschten Zustand an

7. Die Export-Module

7.1 Übersicht

Neben der Ausgabe in die Vorschaudatei bietet List & Label weitere Zielformate, um das Druckergebnis weiterzuverwenden. Die Ausgabe in diese Zielformate geschieht über spezielle Export-Module, welche als List & Label Erweiterung in sog. "Erweiterungsdateien" mit der Endung .LLX (= List & Label Extension) vorliegen. Dabei können auch mehrere Export-Module in derselben Erweiterungsdatei untergebracht sein. Die von List & Label standardmäßig bereitgestellten Exportmodule unterstützen die Formate DOCX, HTML, MHTML, JQM, PDF, PICTURE_BMP, PICTURE_EMF, PICTURE_JPEG, PICTURE_PNG, PICTURE_MULTITIFF, PICTURE_TIFF, PPTX, PRES, RTF, SVG, TTY, TXT, TXT_LAYOUT, XLS, XHTML, XML und XPS.

Die Export-Module stehen neben den Standard-Ausgabemedien Drucker ("PRN"), Vorschau ("PRV") und Datei ("FILE") unter einem bestimmten, eindeutigen Namen ("DOCX", "HTML", "MHTML", "JQM", "PDF", "PICTURE_BMP", "PICTURE_EMF", "PICTURE_JPEG", "PICTURE_PNG", "PICTURE_MULTITIFF", "PICTURE_TIFF", "PPTX", "PRES", "RTF", "SVG", "TTY", "TXT", "TXT_LAYOUT", "XLS", "XHTML", "XML", "XPS") zur Verfügung. Der eigentliche Export in das jeweilige Format läuft aus Entwicklersicht daher auch analog zum sonstigen Druck.

Ob und wenn ja welche Export-Module überhaupt dem Endanwender angeboten werden sollen, oder ob ein Druck fest auf ein bestimmtes Export-Modul durchgeführt werden soll, kann ebenso programmiertechnisch festgelegt werden, wie einzelne exportformatspezifische Optionen.

7.2 Programmierschnittstelle

7.2.1 Export-Module global (de)aktivieren

Standardmäßig versucht List & Label, die Datei cmll20ex.llx aus dem Verzeichnis der DLL zu laden. Damit stehen Ihnen automatisch alle Exportformate zur Verfügung, die List & Label anbietet, sobald Sie als Druckziel bei *LlPrint(WithBox)Start LL_PRINT_EXPORT* angeben.

Möchten Sie die Export-Module deaktivieren, so können Sie dies über die Option *LL_OPTIONSTR_LLXPATHLIST* erreichen. Geben Sie dazu den Dateinamen mit einem Dach davor an, also "^CMLL20EX.LLX".

Um die Export-Module aus einem anderen Verzeichnis zu laden, verwenden Sie ebenfalls diese Option. Als Beispiel können Sie "c:\Programme\<Ihre Applikation>\cmll20ex.llx", angeben, um die Export-Module aus Ihrem Applikationsverzeichnis zu laden.

7.2.2 Einzelne Export-Module ein- und ausschalten

Eine semikolonseparierte Liste der verfügbaren Ausgabemedien erhalten Sie durch Abfrage der Option *LL_OPTIONSTR_EXPORTS_AVAILABLE*. Dies schließt die Standard-Ausgabemedien Drucker ("PRN"), Vorschau ("PRV") und Datei ("FILE") mit ein. Durch Setzen der Option *LL_OPTIONSTR_EXPORTS_ALLOWED* können die verfügbaren Ausgabemedien eingeschränkt werden. Dies betrifft dann die Auswahlmöglichkeit des Ausgabemediums für den Endanwender im *LlPrintOptionsDialog()*. Beachten Sie, dass die verfügbaren Ausgabemedien durch den Ausgabemedium-Parameter bei *LlPrint[WithBox]Start()* beeinflusst werden (wenn dort bspw. Druck auf Vorschau forciert wird), daher sollten Sie *LL_OPTIONSTR_EXPORTS_ALLOWED* am besten erst danach verwenden.

Beispiel zur Einschränkung der Export-Module:

```
LlPrintWithBoxStart(..., LL_PRINT_EXPORT, ...);
//Erlaube lediglich Druck auf Vorschau und HTML-Export:
LlSetOptionString(hJob, LL_OPTIONSTR_EXPORTS_ALLOWED, "PRV;HTML");
//...
LlPrintOptionsDialog(...);
```

Beispiel zum Ausschalten der Export-Module:

```
LlPrintWithBoxStart(..., LL_PRINT_EXPORT, ...);
//Verbiete alle Export-Module:
LlSetOptionString(hJob, LL_OPTIONSTR_EXPORTS_ALLOWED, "PRN;PRV;FILE");
//...
LlPrintOptionsDialog(...);
```

7.2.3 Ausgabemedium festlegen/abfragen

Die Festlegung/Abfrage des Ausgabemediums kann zum einen über einen Parameter beim Funktionsaufruf von *LlPrint[WithBox]Start()* erfolgen. Hier können verschiedene Werte übergeben werden:

Wert	Bedeutung
LL_PRINT_NORMAL	Ausgabemedium "Drucker" wird voreingestellt.
LL_PRINT_PREVIEW	Ausgabemedium "Vorschau" wird voreingestellt.
LL_PRINT_FILE	Ausgabemedium "Druckdatei" wird voreingestellt.
LL_PRINT_EXPORT	Als Ausgabemedium wird ein Export-Modul voreingestellt, welches anschließend über *LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT)* festgelegt werden kann.

Zum anderen kann über *LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT)* ein bestimmtes Ausgabemedium eingestellt werden, welches gleichzeitig im *LlPrintOptionsDialog()* als Auswahl voreingestellt wird.

Beispiel zur Festlegung des Ausgabemediums auf RTF-Export:

```
LlPrintWithBoxStart(..., LL_PRINT_EXPORT, ...);
LlPrintSetOptionString(hJob, LL_PRNOPTSTR_EXPORT, "RTF");
LlPrintOptionsDialog(...);
```

Falls der Endanwender kein anderes Ausgabemedium im *LlPrintOptionsDialog()* auswählen können soll, dann müssen diese über *LL_OPTIONSTR_EXPORTS_ALLOWED* vorher explizit deaktiviert werden. Dies geschieht, indem dort ausschließlich das gewünschte Ausgabemedium übergeben wird.

Der Endanwender kann allerdings auch im Designer über Projekt > Seitenlayout bei den Ausgabemedien ein Export-Modul voreinstellen. Das gewählte Export-Modul wird durch List & Label über die *LL_PRNOPTSTR_EXPORT* Option voreingestellt. Das Anwendungsprogramm sollte dies berücksichtigen und entweder selbst in diesem Falle keine Voreinstellung vornehmen oder die Möglichkeit der Voreinstellung im Designer verbieten. Ansonsten würde der Endanwender ziemlich verwirrt werden, wenn er im Designer explizit "RTF" als Standard-Exportmedium auswählt, das Anwendungsprogramm aber beim Druck fest "HTML" vorschlägt.

Beispiel in C++ zur Berücksichtigung eines evtl. bereits eingestellten Export-Mediums (wenn keine Voreinstellung seitens des Endanwenders im Designer, dann als Voreinstellung Druck auf Vorschau einstellen):

```
LlPrintGetOptionString(hJob, LL_PRNOPTSTR_EXPORT, sMedia.GetBuffer(256),
        256);
sMedia.ReleaseBuffer();
if (sMedia == "") //keine Voreinstellung vorhanden
{
        LlPrintSetOptionString(hJob, LL_PRNOPTSTR_EXPORT, TEXT("PRV"));
        LlPrintOptionsDialog(...);
}
```

Beispiel zum Ausschalten der Voreinstellungsmöglichkeit durch den Endanwender im Designer:

```
LlSetOption(hJob, LL_OPTION_SUPPORTS_PRNOPTSTR_EXPORT, FALSE);
//...
LlDefineLayout(...);
```

Die Abfrage des letztlich vom Endanwender gewählten Ausgabemediums geschieht nach erfolgtem *LlPrintOptionsDialog()* ebenfalls über diese Option.

Beispiel zur Abfrage des Ausgabemediums:

```
//...
LlPrintOptionsDialog(...);
LlPrintGetOptionString(hJob, LL_PRNOPTSTR_EXPORT, sMedia.GetBuffer(256), 256);
sMedia.ReleaseBuffer();
//...
if (sMedia == "PRV")
{
        LlPreviewDisplay(...);
```

```
        LlPreviewDeleteFiles(...); //optional
}
```

7.2.4 Export-spezifische Optionen setzen

Die Export-spezifischen Optionen werden über die Funktion *LlXSetParameter()* gesetzt und über *LlXGetParameter()* abgefragt. Die Optionen sind Export-Modul spezifisch, daher muss bei diesen Funktionen auch explizit der Name des Export-Moduls übergeben werden. Wo sinnvoll können die Optionen auch simultan für alle Export-Module gesetzt werden (z.B. "Export.ShowResult"), indem einfach ein Leerstring ("") als Name für das Export-Modul übergeben wird. Die jeweils unterstützten Optionen und ihre Bedeutung werden in den nachfolgenden Kapiteln bei den einzelnen Export-Modulen erläutert.

Ein Teil der Optionen kann auch durch den Endanwender interaktiv im Eigenschaftsdialog des jeweiligen Export-Moduls eingestellt werden (entweder im Designer oder im Druckoptionsdialog). Diese Optionen werden dann automatisch in der Druckerdefinitionsdatei (das "P-File") gespeichert und sind demnach Projektdatei-spezifisch.

Eine Sonderrolle kommt dem Exportformat "PRV" zu – über dieses kann ein Export auf das Vorschauformat durchgeführt werden. Hierfür stehen lediglich die Optionen Export.File, Export.Path, Export.Quiet und Export.ShowResult zur Verfügung.

7.2.5 Export ohne Benutzerinteraktion durchführen

Soll ein Export ohne Benutzerinteraktion durchgeführt werden, so kann dies durch den Einsatz der bisher besprochenen Funktionen realisiert werden.

Beispiel:

Angenommen, die Anwendung möchte ohne Benutzerinteraktion das Listenprojekt 'Artikel.lst' in das HTML-Dokument 'export.htm' im Verzeichnis 'c:\temp' exportieren:

```
//...
LlXSetParameter(hJob, LL_LLX_EXTENSIONTYPE_EXPORT, "HTML",
        "Export.File", "export.htm");
LlXSetParameter(hJob, LL_LLX_EXTENSIONTYPE_EXPORT, "HTML",
        "Export.Path", "c:\\temp\\");
LlXSetParameter(hJob, LL_LLX_EXTENSIONTYPE_EXPORT, "HTML",
        "Export.Quiet", "1");
LlPrintWithBoxStart(hJob, LL_PROJECT_LIST, "Artikel.lst",
        LL_PRINT_EXPORT, LL_BOXTYPE_BRIDGEMETER, hWnd,"HTML");
LlPrintSetOptionString(hJob, LL_PRNOPTSTR_EXPORT, "HTML");
//... Normale Druckschleife ...
```

Das war es schon! Die Bedeutung der HTML-Export-spezifischen Optionen entnehmen Sie bitte dem nachfolgenden Kapitel.

7.2.6 Export-Ergebnis abfragen

Um das Export-Ergebnis weiterzuverarbeiten, können über die Option *LL_OPTIONSTR_EXPORTFILELIST* alle durch den Druck/Export generierten Dateien ermit-

telt werden: Wenn diese Option per *LlGetOptionString()* nach *LlPrintEnd()*, also nach einem durchgeführten Druck, abgefragt wird, so erhält man eine Semikolon-separierte Liste aller Dateien (inkl. Verzeichnis), die durch den Druck generiert wurden. Bei einem HTML-Export wären dies bspw. alle erzeugten HTML- und JPEG-Dateien, bei einem Druck auf Vorschau lediglich die eine generierte LL-Vorschaudatei.

Die beim Export generierten Dateien werden nicht automatisch gelöscht, sondern müssen ggf. durch das Anwendungsprogramm entfernt werden.

7.3 Programmierreferenz

7.3.1 Excel Export-Modul

Übersicht

Das Excel-Exportmodul erzeugt Dokumente im Microsoft Excel® Format. Die Erzeugung läuft unabhängig von einer Installation dieses Produktes ab, es handelt sich also um eine native Unterstützung. Wahlweise kann ein voller Layout-erhaltender Export durchgeführt werden oder nur die Daten aus Tabellenobjekten unformatiert in die generierte Datei übernommen werden.

Einschränkungen

U.a. sind folgende Einschränkungen und Hinweise beim Excel Export-Modul zu beachten:

- Texte laufen unter Excel etwas höher als bei der Standardausgabe. Daher werden die Schriftarten um einen einstellbaren Faktor skaliert. Diesen Faktor können Sie über die Option XLS.FontScalingPercentage beeinflussen.
- Die Druckfläche kann unter Excel nicht auf den nicht-bedruckbaren Rand ausgeweitet werden, so dass die Projekte etwas breiter erscheinen. Dies kann durch einen Zoom beim Druck (vgl. XLS.PrintingZoom) ausgeglichen werden.
- RTF-Texte werden – bei entsprechend gesetzter Option - als JPEG-Dateien eingebettet. Dadurch wird der Exportvorgang sehr verlangsamt und die Dateien vergrößern sich rasch sehr stark. Wir empfehlen, weitestgehend auf die Verwendung von RTF-Text zu verzichten, bzw. ggf. die Auflösung der Bilddateien (s.u.) zurückzusetzen. Standardmäßig werden RTF-Texte ohne Formatierung exportiert. (siehe Verbosity.RTF).
- Tabulatoren in Textobjekten werden durch Leerzeichen ersetzt.
- Die Tabellen-Objekt Option "Separatoren Durchziehen" wird nicht unterstützt.
- Die Tabellenoption "fixe Größe" wird nicht unterstützt.
- Hintergrundmuster die in List & Label eingestellt werden können, werden nicht berücksichtigt.
- Das Chart- und HTML-Objekt werden als Bilder exportiert und können daher nicht transparent erscheinen.

- Druckreihenfolge Linie/Rechteck wird nicht berücksichtigt; Linien erscheinen immer im Vordergrund. Dieses gilt auch für Rechteckrahmen.
- Druckreihenfolge Text/Rechteck wird nicht berücksichtigt; Text erscheint immer im Vordergrund.
- Textobjekte, die nur halb in gefülltes Rechteck hineinragen werden nicht teilgefüllt.
- Sich überlappende Text- bzw. Bildobjekte werden ignoriert.
- Linien, die weder horizontal noch vertikal sind, werden ignoriert.
- Bildobjekte erhalten einen weißen Rahmen.
- Große gefüllte Bereiche in Projekten mit vielen verschiedenen Koordinaten können die Arbeitsgeschwindigkeit beeinträchtigen.
- Linienbreiten können nicht exportiert werden, Linien erscheinen immer mit Standardbreite.
- Rechteckschatten können nicht exportiert werden.
- Wenn Koordinaten von verschiedenen Objekten sehr dicht beieinander liegen, aber nicht identisch sind, können Rahmenlinien unsichtbar werden, da Excel diese nicht mehr darstellen kann.
- Gedrehte RTF-Objekte und Bilder werden nicht unterstützt.
- Um 180° gedrehte Texte werden nicht unterstützt und mit 0° Drehung dargestellt
- Gradientenfüllungen werden nicht unterstützt.
- Objekte die als Bild exportiert werden dürfen nicht aus ihrem Objektrahmen ragen. Daher müssen z.B. Barcodeobjekte mit fester Balkenbreite im Designer so gestaltet werden, dass der Inhalt in jedem Fall im Objektrechteck Platz findet.
- Selbst in einem Callback gezeichnete Ausgaben werden nicht exportiert.
- Zeilen- und Absatzabstände werden nicht exportiert.
- Negative Randabstände werden nicht unterstützt.
- Es werden maximal 256 Excel-Spalten unterstützt.
- Der Ausfertigungsdruck wird nicht unterstützt.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung aller vom XLS Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LlXSetParameter(..."XLS"...)* gesetzt und über *LlXGetParameter(..."XLS"...)* abgefragt werden.

Resolution: Definiert die Auflösung in dpi für Grafikgenerierung. Default: 300dpi.

Picture.BitsPerPixel: Gibt die Farbtiefe der generierten Grafiken an. Bitte beachten Sie, dass bei einer höheren Farbtiefe die Grafikdateien schnell sehr groß werden können.

Wert	Bedeutung
1	Schwarz-Weiß
24	24bit True Color
Default	24

Verbosity.Rectangle: Konfiguriert die Art und Weise, wie Rechteck-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Rechteck
2	Objekt als Grafik
Default	1

Verbosity.Barcode: Konfiguriert die Art und Weise, wie Barcode-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
Default	1

Verbosity.Drawing: Konfiguriert die Art und Weise, wie Grafik-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
Default	1

Verbosity.Ellipse: Konfiguriert die Art und Weise, wie Ellipsen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
Default	1

Verbosity.Line: Konfiguriert die Art und Weise, wie Linien-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Linie
2	Objekt als Grafik
Default	1

Verbosity.Text: Konfiguriert die Art und Weise, wie Text-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Textobjekt
2	Objekt als Grafik
Default	1

Verbosity.RTF: Konfiguriert die Art und Weise, wie RTF-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als normaler Text ohne Formatierungen
2	Objekt als Grafik
Default	1

Verbosity.Table: Konfiguriert die Art und Weise, wie Tabellen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als komplettes Tabellenobjekt
Default	1

Verbosity.LIXObject: Konfiguriert die Art und Weise, wie LLX-Objekte (OLE, HTML, Chart) exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als Grafik

Wert	Bedeutung
Default	1

XLS.FontScalingPercentage: Skalierungsfaktor, um den Schriftgrößen korrigiert werden. Dies ist notwendig, weil die Texte unter Excel etwas höher laufen als bei Normalausgabe. Default: 89 (=89% Schriftgröße)

XLS.PrintingZoom: Skalierungsfaktor, um den das Gesamtprojekt korrigiert wird. Dies ist notwendig, weil unter Excel immer der nichtbedruckbare Rand des Druckers freigehalten wird. Default: 88 (=88% Zoom)

XLS.IgnoreGroupLines: Erlaubt Gruppenkopf- und Fußzeilen zu ignorieren, wenn diese nicht in der resultierenden Excel-Datei erscheinen sollen. Wirkt sich nur aus, wenn Export.OnlyTabledata gesetzt ist.

Wert	Bedeutung
0	Gruppenzeilen werden exportiert
1	Gruppenzeilen werden ignoriert
Default	1

XLS.IgnoreHeaderFooterLines: Erlaubt Kopf- und Fußzeilen zu ignorieren, wenn diese nicht in der resultierenden Textdatei erscheinen sollen. Wirkt sich nur aus, wenn Export.OnlyTabledata gesetzt ist.

Wert	Bedeutung
0	Kopf- und Fußzeilen werden exportiert
1	Kopf- und Fußzeilen werden ignoriert
2	Kopf- und Fußzeilen werden genau einmal auf der ersten Seite exportiert. Möchten Sie die Fußzeile nur auf der letzten Seite exportieren, geben Sie der Fußzeile die Darstellungsbedingung *LastPage()*.
Default	1

XLS.IgnoreLinewrapForDataOnlyExport: Ermöglicht das Ignorieren von Zeilenumbrüchen. Wirkt sich nur aus, wenn Export.OnlyTabledata gesetzt ist.

Wert	Bedeutung
0	Zeilenumbrüche werden nach Excel übernommen
1	Zeilenumbrüche werden ignoriert
Default	1

XLS.ConvertNumeric: Hierüber kann die automatische Formatierung von Zahlenwerten in der erzeugten Excel-Datei ein- bzw. ausgeschaltet werden.

Wert	Bedeutung
0	Es findet keine automatische Formatierung statt
1	Zahlenwerte werden nach der Einstellung im Designer unter Projekt > Optionen formatiert
2	Nur Spalten, die tatsächlich numerische Werte enthalten (also z.B. Preis) werden konvertiert. Wird eine numerische Spalte explizit innerhalb von List & Label formatiert (z.B. Str$(Preis,0,0)), so wird diese nicht konvertiert.
3	List & Label versucht, die im Designer gewählte Formatierung so exakt wie möglich in Excel wiederzugeben. Wenn die "Format"-Eigenschaft im Designer nicht verwendet wird, wird der Inhalt als Text an Excel übergeben.
Default	3

XLS.AllPagesOneSheet: Erlaubt es, in der erzeugten Excel-Datei pro Seite ein eigenes Worksheet anzulegen.

Wert	Bedeutung
0	Pro Seite wird ein eigenes Worksheet angelegt
1	Alle Seiten werden im gleichen Worksheet erzeugt
Default	1

XLS.FileFormat: Erlaubt es, das Dateiformat festzulegen.

Wert	Bedeutung
0	Format wird anhand der Dateiendung automatisch erkannt
1	Das Office XML(XLSX) Format wird verwendet
2	Das Excel (XLS) Format wird verwendet
Default	0

XLS.WorksheetName: Gibt den Namen für das bzw. die Worksheet(s) in der erzeugten Excel-Datei an. Sie können im Namen den Format-Identifier "%d" verwenden, dieser wird zur Laufzeit durch die Seitenzahl ersetzt (z.B. "Bericht Seite %d").

Export.File: Gibt den Dateinamen für das zu generierende XLS-Dokument an. Wenn leer, dann wird der Dateiauswahl-Dialog angezeigt.

Export.Path: Gibt den Pfad für das zu generierende XLS-Dokument an.

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt keinen Dateiauswahl-Dialog (sofern *Export.File* gesetzt) und keine "Überschreiben?" Rückfrage.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export, die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise Microsoft Excel® o.ä. gestartet werden sollte
Default	0

Export.ShowResultAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahldialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

Export.OnlyTableData: Ermöglicht, dass nur die Daten aus Tabellenzellen exportiert werden.

Wert	Bedeutung
0	Alle Objekte werden exportiert
1	Nur Tabellenzellen werden mit Ihren Daten exportiert. Die Schriftart-Eigenschaften "Fett", "Kursiv" und die horizontale Ausrichtung des Textes werden in der Ergebnisdatei verwendet. Andere Formatoptionen werden ignoriert um die bestmögliche Wiederverwendbarkeit des Ergebnisses in Excel sicherzustellen.
Default	0

7.3.2 Grafik Export-Modul

Übersicht

Das Grafik Export-Modul erzeugt für jede gedruckte Seite eine JPEG-, BMP-, EMF-, TIFF- oder PNG-Grafikdatei, welche die komplette Seite enthält bzw. eine Multi-TIFF-Datei. Die Dateinamen werden dabei fortlaufend durchnummeriert. Enthält der Dateiname der Startdatei den Format-Identifier "%d", so wird dieser durch die jeweilige Seitenzahl ersetzt.

Einschränkungen

Das Grafik-Export-Modul besitzt folgende Einschränkungen:

- Der Ausfertigungsdruck wird nicht unterstützt.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung aller vom Grafik Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LIXSetParameter(..."<Exportmodulname>"...)* gesetzt und über *LIXGetParameter(..."<Exportmodulname>"...)* abgefragt werden. *<Exportmodulname>* ist entweder "*PICTURE_JPEG*", "*PICTURE_BMP*", "*PICTURE_EMF*", "PICTURE_TIFF" oder "PICTURE_MULTITIFF", "PICTURE_PNG" je nach Zielformat.

Resolution: Definiert die Auflösung in dpi für die Grafikgenerierung. Default: 96dpi, Bildschirmauflösung.

Picture.JPEGQuality: Spezifiziert die Qualität und den damit abhängigen Kompressionsfaktor der generierten JPEG Grafiken. Der Wert liegt zwischen 0 und 100, wobei 100 der höchsten JPEG Qualität (und damit vergleichsweise schlechtesten Kompression) entspricht. Wirkt sich nur aus wenn die Quellgrafik nicht im JPEG-Format vorliegt, da eine Codierung von JPEG nach JPEG einen Qualitätsverlust mit sich bringen würde. Default: 75

Picture.BitsPerPixel: Gibt die Farbtiefe der generierten Grafiken an. Bitte beachten Sie, dass bei einer höheren Farbtiefe die Grafikdateien schnell sehr groß werden können. Nicht alle Grafikformate können alle Farbtiefen sinnvoll darstellen.

Wert	Bedeutung
1	Schwarz-Weiß
4	16 Farben
8	256 Farben
24	24bit True Color
Default	JPEG, PNG: 24, Andere: 8

Picture.CropFile: Schneidet überflüssigen weißen Rahmen ab. Unterstützt die folgenden Exportformate: PNG, JPEG und TIFF. Diese Option wird bei Verwendung in Services (wie z.B. IIS) nicht unterstützt, da dort GDI+ nicht zur Verfügung steht.

Wert	Bedeutung
0	Bild wird nicht zurecht geschnitten
1	Bild wird zurecht geschnitten
Default	0

Picture.CropFrameWidth: Bestimmt den Rand eines zugeschnittenen Bildes in Pixeln.

Export.Path: Definiert den Zielpfad für den Export. Ist er leer, so erscheint ein Zielpfad-Auswahldialog.

Export.File: Gibt den Dateinamen für die Dateien an. Wenn die Option gesetzt ist, müssen Sie im Dateinamen printf-Platzhalter wie z.B. "%d" verwenden (z.B. "Export Seite %d.htm"). In diesem Falle werden die erste Seite und die Folgeseiten durch Ersetzung des Platzhalters durch die entsprechend formatierte Seitenzahl benannt. Ansonsten erhalten die Seiten eine einfache Nummerierung.

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt kein Zielpfad-Auswahldialog (sofern *Export.Path* gesetzt) und keine "Überschreiben?" Rückfrage.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export, die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf die erste generierte Bilddatei aus, so dass üblicherweise ein Bildbearbeitungsprogramm o.ä. gestartet wird
Default	0

Export.ShowResultAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahldialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

TIFF.CompressionType: Legt den Kompressionstyp für die erzeugte TIFF-Datei fest. Beachten Sie, dass nicht alle Viewer Kompression unterstützen. Picture.BitsPerPixel muss für CCITTRLE, CCITT3 und CCITT4 auf 1 bzw. für JPEG auf 24 gesetzt werden.

Wert	Bedeutung
None	Keine Kompression
CCITTRLE	CCITT Modified Huffmann RLE
CCITT3	CCITT Group 3 Fax Codierung
CCITT4	CCITT Group 4 Fax Codierung
JPEG	JPEG DCT Kompression
ZIP	ZIP Kompression
LZW	LZW Kompression
Default	None

TIFF.CompressionQuality: Legt die Kompressionsqualität für die erzeugte TIFF-Datei fest. Wertebereich 0...100. Default: 75

7.3.3 HTML Export-Modul

Übersicht

Das HTML Export-Modul erzeugt (mit wenigen Einschränkungen, s.u.) HTML Code gemäß HTML 4.01 Spezifikation.

Das Export-Modul sammelt hierzu alle List & Label Objekte einer soeben gedruckten Seite zusammen und ordnet diese dann in einer großen HTML-Tabelle (dem sog. Layout-Grid) gemäß ihrer optischen Anordnung auf der Seite an. Die einzelnen Spaltenbreiten und Zeilenhöhen dieses Layout-Grids ergeben sich aus den gesamten X- und Y-Koordinaten aller Objektrechtecke.

Der Endanwender kann durch die HTML-Export Eigenschaften wählen, ob die Spaltenbreiten des Layout-Grids durch das Export-Modul prozentual (bezogen auf die aktuelle Browser-Fenstergröße) oder absolut (in Pixel) erfolgen soll. Eine absolute Anordnung hat den Vorteil einer optisch genaueren Umsetzung des Designer-Layouts in HTML, was bei prozentualem Layout nicht immer möglich ist. Eine prozentuale Anordnung hat den Vorteil, dass das Ergebnis hinterher vom Browser in der Regel besser ausdruckbar ist, da

hier der Browser den Inhalt in der Größe anpassen kann, um seine eigenen Kopf-/Fußzeilen u.ä. zu drucken, was bei einem absolutem Layout nicht möglich ist und woraus oftmals mehrere ungewollte Seiten resultieren.

Da jede unterschiedliche X- bzw. Y-Koordinate eine neue Spalte bzw. Zeile im Layout-Grid bewirkt, sollte man im Designer darauf achten, die Objekte möglichst an gleichen Kanten auszurichten. Dies resultiert dann zum einen in einem weniger komplexen (und damit auch vom Browser schneller darstellbaren) Layout-Grid, zum anderen (insbesondere bei der prozentualen Spaltenanordnung) verhindert es ggf. eine unvorhergesehene horizontale Anordnung von Objekten, da sich kleine Lücken zwischen Objekten prozentual unterschiedlich stark niederschlagen können (im Gegensatz zu absolutem, pixelgenauem Layout).

HTML unterstützt in der Version 4.01 keine überlappenden Objekte, so dass hier Einschränkungen beim Export gegeben sind: Wenn Objekte sich im Design überlappen, dann exportiert das HTML Export-Modul lediglich das Objekt, welches in der Objektanordnung "am tiefsten" liegt, also zuerst gedruckt wird. Die anderen Objekte, welche durch List & Label darüber gedruckt würden, werden ignoriert. Einzige Ausnahme: Gefüllte Rechteckobjekte im Hintergrund; diese werden durch "Einfärben" der Zelle des darüber liegenden Objektes realisiert.

Einschränkungen

Daneben gibt es natürlich diverse, durch das Zielformat bedingte Einschränkungen. Die wichtigsten werden nachfolgend genannt:

- Zeilen, die aneinander verankert sind, können nicht korrekt exportiert werden.
- Überlappende Objekte (abgesehen von Rechtecken) werden nicht unterstützt und ignoriert
- Rechtecke können keinen Rahmen haben und transparente Rechtecke (egal ob mit Rahmen oder ohne) werden ignoriert.
- Der Dezimaltabulator in Textobjekten und Tabellen wird auf 'rechtsbündig' umgesetzt.
- Tabulatoren und mehrere aufeinanderfolgende Leerzeichen werden nicht unterstützt.
- Zeilen- und Absatzabstände werden nicht unterstützt.
- Die Option 'Wortumbruch' in Textobjekten und Tabellenspalten ist bei HTML immer aktiv (auch wenn 'abschneiden' im Designer gewählt wurde)
- die Tabellen-Objekt Option "Separatoren Durchziehen" wird nicht unterstützt.
- In Tabellenzeilen wird der ggf. vorhandene Abstand von links für die 1. Spalte ignoriert.
- Die Tabellenoption "fixe Größe" wird nicht unterstützt.
- Das Chart -Objekt wird als Bild exportiert und kann daher nicht transparent erscheinen.

- Die Umsetzung von formatiertem RTF-Text in HTML-Code erfolgt über einen RTF-Parser, der die wichtigsten Absatz- und Zeichenformatierungen interpretiert und entsprechend umsetzt. Erweiterte Formatierungen, autom. Nummerierungen, sowie eingebettete Objekte und Grafiken werden ignoriert.
- Linien werden als Grafik realisiert. Dies geschieht allerdings lediglich für genau vertikale und horizontale Linien, alle diagonalen Linien werden ignoriert.
- Gradientenfüllungen werden nicht unterstützt.
- Gedrehte Texte werden nicht unterstützt.
- Gedrehter RTF-Text wird nicht unterstützt.
- Objekte die als Bild exportiert werden dürfen nicht aus ihrem Objektrahmen ragen. Daher müssen z.B. Barcodeobjekte mit fester Balkenbreite im Designer so gestaltet werden, dass der Inhalt in jedem Fall im Objektrechteck Platz findet.
- Selbst in einem Callback gezeichnete Ausgaben werden nicht exportiert.
- Abstände vor Tabellenzeilen werden nicht unterstützt.
- Rahmen von benachbarten Zellen werden nicht übereinander, sondern nebeneinander gemalt. Dadurch kann sich die Rahmendicke verdoppeln. Bitte berücksichtigen Sie dies bereits beim Layout.
- Die Funktion *TotalPages$()* kann nicht in gedrehten Textobjekten verwendet werden.
- Der Ausfertigungsdruck wird nicht unterstützt.

Folgende über den HTML 4.01 Standard hinaus verwendete Tags oder Attribute werden verwendet:

- Das Ausschalten des Seitenrandes für die HTML Seiten wird über jeweils zwei Browserspezifische Tags orientiert, um unter Netscape und Internet Explorer optimale Ergebnisse zu erzielen:

  <body TOPMARGIN=0 LEFTMARGIN=0 ist Internet Explorer spezifisch und wird ab der Internet Explorer Version 2.0 unterstützt.

  <body MARGINHEIGHT=0 MARGINWIDTH=0 ist Netscape spezifisch und wird ab der Netscape Version 3.0 unterstützt.
- Die Einstellung der Linienfarbe für das Tabellengitter (<table BORDER-COLOR="#ff0000">) ist Internet Explorer spezifisch ab der Internet Explorer Version 3.0.
- Die Einstellung der Linienfarbe für horizontale Tabellen-Linien (<hr COLOR="#ff0000">) ist Internet Explorer spezifisch ab der Internet Explorer Version 3.0.

Wenn das HTML Objekt nicht als Bild, sondern als HTML Text exportiert wird, dann wird der HTML Text des Objektes, der sich zwischen den <BODY> und </BODY> Tags

befindet, in das Exportresultat eingebettet. Damit ergeben sich zwangsläufig u.a. folgende Einschränkungen:

- Ggf. verwendete Cascading Stylesheets werden nicht alle unterstützt
- Seitenformatierungen, wie Ränder, Hintergrundfarbe u.a. gehen verloren
- HTML erlaubt keine Skalierung, daher kann sich das Layout des Exportergebnisses signifikant vom Layout im Designer unterscheiden. Insbesondere falls dort bspw. das HTML Objekt eine komplette HTML Seite enthält, aber von der Objektgröße her kleiner skaliert wurde.
- Auch wenn das HTML-Objekt einen Seitenumbruch auslöst, wird das exportierte Objekt auf einer Seite/in einer Datei ausgegeben. Ein Umbruch wird ignoriert.
- Eingebettete Scriptfunktionalitäten können verloren gehen

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung der vom HTML Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LIXSetParameter(..."HTML"...)* gesetzt und über *LIXGetParameter(..."HTML"...)* abgefragt werden.

Resolution: Definiert die Auflösung in dpi für Koordinatenumrechnung und Grafikgenerierung. Default: 96dpi, Bildschirmauflösung.

Picture.JPEGQuality*:* Spezifiziert die Qualität und den davon abhängigen Kompressionsfaktor der generierten JPEG Grafiken. Der Wert liegt zwischen 0..100, wobei 100 der höchsten JPEG Qualität (und damit vergleichsweise schlechtesten Kompression) entspricht. Wirkt sich nur aus wenn die Quellgrafik nicht im JPEG-Format vorliegt, da eine Codierung von JPEG nach JPEG einen Qualitätsverlust mit sich bringen würde. Default: 75

Picture.BitsPerPixel: Gibt die Farbtiefe der generierten Grafiken an. Bitte beachten Sie, dass bei einer höheren Farbtiefe die Grafikdateien schnell sehr groß werden können.

Wert	Bedeutung
1	Schwarz-Weiß
24	24bit True Color
Default	24

Verbosity.Rectangle: Konfiguriert die Art und Weise, wie Rechteck-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG (und zusätzlich als komplettes Rechteckobjekt, für farblich hinterlegte Objekte)

Wert	Bedeutung
Default	1

Verbosity.Barcode: Konfiguriert die Art und Weise, wie Barcode-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.Drawing: Konfiguriert die Art und Weise, wie Grafik-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.Ellipse: Konfiguriert die Art und Weise, wie Ellipsen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.Line: Konfiguriert die Art und Weise, wie Linien-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.Text: Konfiguriert die Art und Weise, wie Text-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als komplettes Textobjekt

Wert	Bedeutung
2	Objekt als JPEG
Default	1

Verbosity.Text.Frames: Konfiguriert die Art und Weise, wie Rahmen um Textobjekte exportiert werden sollen.

Wert	Bedeutung
0	einzelne Rahmen für oben, unten, links, rechts erzeugen (verwendet CSS)
1	kompletter Rahmen als Box
Default	0

Verbosity.RTF: Konfiguriert die Art und Weise, wie RTF-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als formatierter RTF-Text (wird interpretiert und nach HTML gewandelt)
2	als unformatierter Text (als Schriftart wird die beim Projekt eingestellte Default-Schriftart verwendet)
3	Objekt als JPEG
Default	1

Verbosity.RTF.Frames: Konfiguriert die Art und Weise, wie Rahmen um RTF-Objekte exportiert werden sollen.

Wert	Bedeutung
0	einzelne Rahmen für oben, unten, links, rechts erzeugen (verwendet CSS)
1	kompletter Rahmen als Box
Default	0

Verbosity.Table: Konfiguriert die Art und Weise, wie Tabellen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als komplettes Tabellenobjekt
Default	1

Verbosity.Table.Cell: Konfiguriert die Art und Weise, wie die Tabellen-Zellen exportiert werden sollen.

Wert	Bedeutung
0	Zellen ignorieren
1	als komplettes Zellenobjekt (gemäß den Verbosity-Einstellungen der jeweiligen Objekttypen)
2	Zellen als JPEG
Default	1

Verbosity.Table.Frames: Konfiguriert die Art und Weise, wie Tabellen-Rahmen exportiert werden sollen.

Wert	Bedeutung
0	keine Tabellenrahmen zeichnen
1	nur horizontalen Tabellenrahmen als horizontale Linie berücksichtigen
2	komplette Tabellenzeile mit allen Rahmen, sofern irgendein vertikaler Rahmen vorhanden, ansonsten wie 1
3	Zellenspezifische Rahmen zeichnen (verwendet CSS)
Default	3

Verbosity.LLXObject: Konfiguriert die Art und Weise, wie LLX-Objekte (z.B. Chart-Objekt) exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.LLXObject.HTMLObj: Konfiguriert die Art und Weise, wie das HTML Objekt exportiert werden soll.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
2	Objekt im HTML Format. Dabei werden nur die HTML text Anweisungen zwischen <BODY> und </BODY> exportiert. Bitte beachten Sie die o.g. Einschränkungen.
Default	2

HTML.Title: Spezifiziert den Titel des zu generierenden HTML-Dokuments. Default: Titel, der bei *LlPrintWithBoxStart()* dem Druckjob übergeben wird.

Layouter.Percentaged: Gibt an, ob das Layout absolut oder prozentual zur Seitenbreite erfolgen soll.

Wert	Bedeutung
0	Layout in X-Richtung absolut in Pixel
1	Layout in X-Richtung überall prozentual auf Seitenbreite
Default	0

Layouter.FixedPageHeight: Gibt an, ob der Layouter auf allen Seiten die Original-Seitenhöhe erzwingen soll.

Wert	Bedeutung
0	Layout kann z.B. auf letzter Seite zu kleinerer Seitenhöhe führen (wenn keine Objekte am Seitenfuß platziert sind)
1	Die Seitenhöhe wird exakt beachtet
Default	1

Export.Path: Definiert den Zielpfad für den Export. Ist er leer, so erfolgt in jedem Fall ein Zielpfad-Auswahldialog.

Export.File: Gibt den Dateinamen für die erste zu generierende HTML-Seite an. Default: "index.htm". Sie können im Dateinamen auch printf-Platzhalter wie z.B. "%d" verwenden (z.B. "Export Seite %d.htm"). In diesem Falle werden die erste Seite und die Folgeseiten durch Ersetzung des Platzhalters durch die entsprechend formatierte Seitenzahl benannt. Ansonsten erhalten die Seiten eine einfache Nummerierung.

Export.AllInOneFile: Konfiguriert das Export-Resultat.

Wert	Bedeutung
0	Das Ergebnis sind n verschiedene HTML-Dateien, für jede gedruckte Seite eine HMTL-Datei.
1	Das Ergebnis ist eine einzige HTML Datei (*Export.File)*, in der alle gedruckten Seiten aneinander hängen.
Default	1

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt kein Zielpfad-Auswahldialog (sofern *Export.Path* gesetzt ist) und keine "Überschreiben?" Rückfrage. Ebenso wird keine Zusammenfassung der überlappenden Objekte, die ignoriert wurden, angezeigt.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise ein Web-Browser gestartet wird
Default	0

Export.ShowResultAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahldialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

HTML.Form.Header: Definiert einen HTML-Formular-Tag der Art "<form method="POST" action=...". Wird dieser angegeben, dann werden die Objektnamen nach HTML Formularerweiterungen untersucht und es erfolgt eine HTML Formulargenerierung (siehe Abschnitt "HTML-Formular Generierung"). Default: Leer, also keine Formulargenerierung

HTML.Form.Footer: Definiert ein HTML-Formular-Tag für das Formularende. Default: "</form>"

Hyperlinks

Im Designer kann ein Hyperlink direkt über die Funktion *HyperLink$()* erzeugt und in Text-, RTF- und Tabellenobjekten eingebettet werden. So werden auch dynamische Hyperlinks inkl. Formeln möglich.

Eine andere Möglichkeit sind die Objektnamen, die im Designer für die Objekte vergeben werden können. Diese werden durch das HTML Export-Modul untersucht und ermöglichen eine erweiterte Exportfunktionalität:

Enthält der Objektname den Text "/LINK:<url>" so wird für Text- und Bildobjekte zusätzlich ein Hyperlink auf <url> generiert.

Beispiel:

Aus dem Grafikobjekt-Objektnamen "combit /LINK:http://www.combit.net" würde dann im HTML-Export eine Grafik mit dem Alternativ-Text "combit" und einem dahinterliegenden Hyperlink auf "www.combit.net".

HTML-Formular Generierung

Das HTML Export-Modul unterstützt einen speziellen Modus zur Generierung von HTML Formularen. Dabei werden die im Designer vergebenen Objektnamen auf bestimmte Namen untersucht, welche dann aus dem jeweiligen Objekt ein spezielles Eingabecontrol in einem HTML Formular werden lassen. Aktiviert wird die HTML-Formulargenerierung durch Angabe eines Formular-Tags über die Option *HTML.Form.Header*.

Beispiel:

```
LlXSetParameter(hJob, LL_LLX_EXTENSIONTYPE_EXPORT, "HTML",
        "HTML.Form.Header",
        "<form method="POST" action=http://www.xyz.de/x.cgi>");
```

Dadurch wird dieser HTML-Code in das Export-Ergebnis mit aufgenommen und alle Objektnamen auf folgende Werte untersucht:

Textobjekte

Wert	Bedeutung
@EDIT:<Value>	Erzeugt ein 1-zeiliges Eingabefeld. Der Inhalt des 1. Absatzes des Objekts ergibt die Vorbelegung.
@MULTIEDIT:<Value>	Erzeugt ein mehrzeiliges Eingabefeld. Der Inhalt der Absätze des Objekts ergibt die Vorbelegung.
@LISTBOX:<Value>	Erzeugt eine Listbox. Der Inhalt der Absätze des Objekts ergibt die einzelnen Einträge in der Liste.
@COMBOBOX:<Value>	Erzeugt eine Combobox. Der Inhalt der Absätze des Objekts ergibt die einzelnen Einträge in der Liste.
@RADIOBUTTON:<Groupname>,<Value>	Erzeugt einen Radiobutton. Ist der Inhalt des 1. Absatzes 'T', 'X' oder '1', dann wird der Radiobutton aktiviert.
@CHECKBOX:<Groupname>,<Vaue>	Erzeugt eine Checkbox. Ist der Inhalt des 1. Absatzes 'T', 'X' oder '1', dann wird die Checkbox aktiviert.

Wert	Bedeutung
@BUTTON:<Value>	Erzeugt einen Button. Der Inhalt des 1. Absatzes ergibt den Button-Text.
@SUBMIT:<Value>	Erzeugt einen Submit-Button. Der Inhalt des 1. Absatzes ergibt den Button-Text.
@RESET:<Value>	Erzeugt einen Reset-Button. Der Inhalt des 1. Absatzes ergibt den Button-Text.

<Value> ist der Name des Controls im HTML-Formular.

Bildobjekte

Wert	Bedeutung
@RADIOBUTTON:<Groupname>,<Value>	Erzeugt einen Radiobutton. Ist der Inhalt des 1. Absatzes 'T', 'X' oder '1', dann wird der Radiobutton aktiviert.
@CHECKBOX:<Groupname>,<Value>	Erzeugt eine Checkbox. Ist der Inhalt des 1. Absatzes 'T', 'X' oder '1', dann wird die Checkbox aktiviert.
@IMAGE:<Value>	Erzeugt ein Image-Control (welches dem Submit-Button entspricht)

Rechteckobjekte

Wert	Bedeutung
@RADIOBUTTON:<Groupname>,<Value>	Erzeugt einen Radiobutton. Ist der Inhalt des 1. Absatzes 'T', 'X' oder '1', dann wird der Radiobutton aktiviert.
@CHECKBOX:<Groupname>,<Value>	Erzeugt eine Checkbox. Ist der Inhalt des 1. Absatzes 'T', 'X' oder '1', dann wird die Checkbox aktiviert.

Für die betreffenden Objekte wird als Objektrechteck für das Layout die im Designer eingestellte Größe herangezogen, nicht wie bei allen anderen Objekten die sich durch die Daten erst ergebende Objektgröße. Diverse Eingabe-Controls erlauben keine pixelgenaue Größenangabe, sondern die Größe hängt bspw. von der Länge der Einträge ab. Eine genaue Abbildung 1:1 in HTML ist hier natürlich nicht möglich.

Generell ist zu beachten, dass nur die Einträge, welche durch List & Label auch infolge der Objektgröße im Designer gedruckt werden können, an das Export-Modul übergeben werden. Dies hat zu Folge, dass bei einer Combobox, das Textobjekt im Designer so groß sein muss, dass alle Absätze gedruckt werden können (und damit in die Combobox aufgenommen werden), auch wenn nachher im HTML Formular das Objekt nur 1-zeilig hoch ist.

7.3.4 JQM Export-Modul

Übersicht

Mit dem JQM (jQuery Mobile) Export-Modul kann der Report im HTML Format unter Verwendung des jQuery Mobile Frameworks und Javascript erzeugt werden. Es werden HTML-Dateien und Bilder erzeugt welche für die Darstellung auf mobilen Endgeräten optimiert sind. Die Informationen zum JQM finden Sie unter www.jquerymobile.com. Das Framework wird dabei über ein CDN (Content Delivery Network) geladen, weshalb zur Darstellung eine aktive Internet-Verbindung vorausgesetzt wird.

Einschränkungen

Es gibt einige durch das Zielformat bedingte Einschränkungen. Die wichtigsten werden nachfolgend genannt:

- Die erzeugten Seiten sind für eine Bedienung in Mobilen Endgeräten optimiert.
- Es werden nur "normale" Tabellen exportiert, und dadurch auch nur Listenprojekte unterstützt.
- Inhaltsverzeichnis und Index werden nicht unterstützt.
- Text, RTF Text und HTML-Text in Tabellenzellen werden speziell unterstützt. Alle anderen Objekte werden als Grafik exportiert.
- Nur Fußzeilen der letzten Seite bzw. die letzten einer Tabelle werden unterstützt, da dieser Export nicht seitenbasierend ist.
- Bei lokalem Zugriff müssen die entsprechenden Rechte (IE) vorhanden sein, damit die Seiten geladen werden können. Bei manchen Browsern kann der Zugriff auf file:// Probleme bereiten. Sobald die Seiten per http: zugegriffen werden, sollte das Problem nicht mehr auftauchen.
- Der Ausfertigungsdruck wird nicht unterstützt.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung aller vom JQM Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LIXSetParameter(..."JQM"...)* gesetzt und über *LIXGetParameter(..."JQM"...)* abgefragt werden.

***Picture.JPEGQuality**:* Spezifiziert die Qualität und den davon abhängigen Kompressionsfaktor der generierten JPEG Grafiken. Der Wert liegt zwischen 0..100, wobei 100 der höchsten JPEG Qualität (und damit vergleichsweise schlechtesten Kompression) entspricht. Wirkt sich nur aus wenn die Quellgrafik nicht im JPEG-Format vorliegt, da eine Codierung von JPEG nach JPEG einen Qualitätsverlust mit sich bringen würde. Default: 75

Resolution: Definiert die Auflösung in dpi für Koordinatenumrechnung und Grafikgenerierung. Default: 96dpi, Bildschirmauflösung.

Picture.BitsPerPixel: Gibt die Farbtiefe der generierten Grafiken an. Bitte beachten Sie, dass bei einer höheren Farbtiefe die Grafikdateien schnell sehr groß werden können.

Wert	Bedeutung
1	Schwarz-Weiß
24	24bit True Color
Default	32

Picture.Format: Gibt das Format der generierten Grafiken an.

Wert	Bedeutung
JPG	JPEG-Grafik
PNG	PNG-Grafik
Default	PNG

Export.Path: Definiert den Zielpfad für den Export mit abschließendem Backslash "\". Ist er leer, so erfolgt in jedem Fall ein Zielpfad-Auswahldialog.

Export.File: Gibt den Dateinamen für die erste zu generierende HTML-Seite an. Default: "index.htm".

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt kein Zielpfad-Auswahldialog (sofern *Export.Path* gesetzt ist) und keine "Überschreiben?" Rückfrage. Ebenso wird keine Zusammenfassung der überlappenden Objekte, die ignoriert wurden, angezeigt.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise ein Web-Browser gestartet wird

Wert	Bedeutung
Default	0

Verbosity.RTF: Konfiguriert die Art und Weise, wie RTF-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als formatierter RTF-Text (gewandelt in HTML)
2	als unformatierter Text
Default	1

Verbosity.LLXObject.HTMLObj: Konfiguriert die Art und Weise, wie das HTML Objekt exportiert werden soll.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt im HTML Format. Dabei werden nur die HTML-Anweisungen zwischen <BODY> und </BODY> exportiert. Bitte beachten Sie die o.g. Einschränkungen.
Default	1

JQM.CDN: CDN Anbieter der CSS und JS Dateien (Content Distribution Network).

Wert	Bedeutung
jQuery	http://code.jquery.com
Microsoft	http://ajax.aspnetcdn.com
Default	jQuery

JQM.Title: Titel der generierten HTML-Dateien. Default: "".

JQM.ListDataFilter: Spezifiziert, ob die Suchfilterbar im Ergebnis angezeigt werden soll.

Wert	Bedeutung
0	Keine Anzeige der Suchfilterbar
1	Stellt eine Suchfilterbar dar und filtert damit die Daten
Default	1

JQM.UseDividerLines: Konfiguriert die Verwendung von Trennzeilen.

Wert	Bedeutung
0	Alle Zeilen einer Tabelle werden als "normale" Datenzeile ausgegeben
1	Kopfzeilen, Fusszeilen und Gruppenzeilen werden als spezielle Trennzeilen mit eigenem Style ausgegeben
Default	1

JQM.BreakLines: Konfiguriert das Umbruchverhalten von Texten im Ergebnis.

Wert	Bedeutung
0	Texte werden nicht umgebrochen sondern über "..." am Ende gekennzeichnet, falls die Breite nicht ausreichend ist.
1	Texte werden automatisch umgebrochen.
Default	1

JQM.BaseTheme: Theme der Datenzeilen. Werte: "a", "b", "c", "d", "e". Siehe http://jquerymobile.com/test/docs/lists/lists-themes.html

Default: "d".

JQM.HeaderTheme: Theme der Headers (Zeile mit Navigation und Überschrift). Werte: "a", "b", "c", "d", "e". Siehe http://jquerymobile.com/test/docs/lists/lists-themes.html

Default: "a".

JQM.DividerTheme: Theme der Trennzeilen (siehe JQM.UseDividerLines). Werte: "a", "b", "c", "d", "e". Siehe http://jquerymobile.com/test/docs/lists/lists-themes.html

Default: "b".

7.3.5 MHTML Export-Modul

Übersicht

Das MHTML (Multi Mime HTML) Export-Modul funktioniert analog zum XHTML Export-Modul, mit dem Unterschied, dass Bilder direkt MIME codiert in die Exportdatei eingebettet werden und das Ergebnis somit nur aus einer einzigen (.MHT) Datei besteht. Dies ist bspw. nützlich, um die Datei per eMail zu versenden, da der Empfänger dann per Doppelklick den Report direkt öffnen und ansehen kann, ohne dass noch weitere (externe) Bilddateien notwendig wären.

Programmierschnittstelle

Es gelten die Optionen des XHTML Export-Moduls analog, als Export Modulname muss "MHTML" angegeben werden. Die Option Export.AllInOneFile wird ignoriert, da dieses Format immer nur eine Ergebnisdatei erzeugt.

7.3.6 PDF Export-Modul

Übersicht

Das PDF-Exportmodul erzeugt Dokumente im Portable Document Format. Dieses Format kann plattformunabhängig mit dem frei verfügbaren Adobe Acrobat Reader® angezeigt werden.

Einschränkungen

U.a. sind folgende Einschränkungen und Hinweise beim PDF Export-Modul zu beachten:

- Unicode/Multibyte-Zeichen werden über CID (Character Identifier) eingebettet. Achten Sie darauf, diesen Einbettungsmodus im Ausgabedialog auszuwählen. Japanische und Chinesische Zeichen setzen die entsprechenden Adobe FontPackages voraus. In RTF-Textobjekten verwendete Right-to-Left-Zeichensätze werden u.U. nicht korrekt exportiert. Schriftarten, die nicht chinesisch oder japanisch sind, werden u.U. nicht richtig eingebettet. Hinweis: Bei Einbettung der Schriftarten als CID wird der PDF/A-Standard für die Ausgabe nicht vollständig unterstützt.
- Gedrehte fette/kursive TrueType-Schriftarten können evtl. eine abweichende Laufweite haben.
- Fonts mit abweichender Laufweite werden nicht unterstützt.
- Bitmap-Fonts werden nicht unterstützt.
- Linienenden werden immer mit runden Enden dargestellt.
- Nicht alle EMF-Records können korrekt wiedergegeben werden – wenn Sie sehr komplexe EMFs verwenden, sollten diese ggf. z.B. als Bitmap übergeben werden bzw. im Designer die Option "Export als Bild" aktiviert werden.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung aller vom PDF Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LIXSetParameter(..."PDF"...)* gesetzt und über *LIXGetParameter(..."PDF"...)* abgefragt werden.

PDF.Title: Spezifiziert den Titel des zu generierenden PDF-Dokuments. Default: Titel, der bei *LIPrintWithBoxStart()* dem Druckjob übergeben wird.

PDF.Subject: Spezifiziert das Thema des zu generierenden PDF-Dokuments. Default: leer.

PDF.Keywords: Spezifiziert die Stichwörter des zu generierenden PDF-Dokuments. Default: leer.

PDF.Encryption.EncryptFile: Wenn dieser Parameter gesetzt ist, wird die Ergebnisdatei verschlüsselt. Dann stehen diverse weitere Optionen zur Verfügung, die im Folgenden erläutert werden.

Wert	Bedeutung
0	Datei nicht verschlüsseln
1	Datei verschlüsseln
Default	0

PDF.Encryption.EnablePrinting: Wenn dieser Parameter gesetzt ist, kann die Datei trotz Verschlüsselung gedruckt werden. Nur wirksam, wenn PDF.Encryption.EncryptFile auf "1" gesetzt wird.

Wert	Bedeutung
0	Drucken ist nicht möglich
1	Drucken ist möglich
Default	0

PDF.Encryption.EnableChanging: Wenn dieser Parameter gesetzt ist, kann die Datei trotz Verschlüsselung bearbeitet werden. Nur wirksam, wenn PDF.Encryption.EncryptFile auf "1" gesetzt wird.

Wert	Bedeutung
0	Bearbeiten ist nicht möglich
1	Bearbeiten ist möglich
Default	0

PDF.Encryption.EnableCopying: Wenn dieser Parameter gesetzt ist, können Teile der Datei trotz Verschlüsselung in die Zwischenablage übernommen werden. Nur wirksam, wenn PDF.Encryption.EncryptFile auf "1" gesetzt wird.

Wert	Bedeutung
0	Kopieren ist nicht möglich
1	Kopieren ist möglich
Default	0

PDF.Encryption.Level: Bestimmt die Verschlüsselungsstärke. Nur wirksam, wenn PDF.Encryption.EncryptFile auf "1" gesetzt wird.

Wert	Bedeutung
0	40 Bit Verschlüsselung
1	128 Bit Verschlüsselung (benötigt Acrobat Reader ab Version 5)
Default	0

PDF.OwnerPassword: Das Besitzerpasswort für die verschlüsselte Datei. Dieses wird benötigt, um die Datei bearbeiten zu können. Wenn kein Passwort angegeben wird, wird die Datei mit einem zufälligen Passwort verschlüsselt. Wir empfehlen, immer ein geeignetes Passwort explizit zu wählen.

PDF.UserPassword: Das Benutzerpasswort für die verschlüsselte Datei. Dieses wird benötigt, um auf eine verschlüsselte Datei zugreifen zu können. Wenn kein Passwort angegeben wird, ist der Zugriff ohne Passwort möglich (evtl. mit Einschränkungen, s.o.).

PDF.FontMode: Bestimmt, wie TrueType-Schriftarten behandelt werden.

Wert	Bedeutung
0	Die TrueType-Schriftarten des Zielrechners werden - wenn vorhanden - verwendet bzw. durch den Fontmapper ersetzt. Es findet keine Einbettung statt.
1	Alle TrueType-Schriftarten werden eingebettet.
2	Nur Symbol-TrueType-Schriftarten werden eingebettet.
3	Keine Einbettung von TrueType-Schriftarten. Es werden die Standard PostScript –Schriftarten verwendet.
4	Subset-Einbettung: Es werden nur die Zeichen der TrueType-Schriftarten eingebettet, die in der eingestellten Codepage vorhanden sind.
5	Subset-Einbettung: Es werden nur die Zeichen der TrueType-Schriftarten eingebettet, die tatsächlich verwendet werden.
6	CID-Schriftarten (character identifier font) werden verwendet (empfohlen wenn Unicode-Texte verwendet werden).
7	Schriften als Kurven umsetzen.
Default	5

PDF.ExcludedFonts: Bestimmt, welche Schriftarten nicht eingebettet werden sollen. Einige Schriftarten (z.B. Arial, Courier) können identisch durch PostScript-Schriftarten ersetzt werden. Durch diese Option können einzelne Schriftarten explizit von der Einbettung ausgenommen werden. Bsp. "Arial;Courier;...". Default: "Arial".

PDF.CompressStreamMethod: Bestimmt, inwieweit die Ergebnisdatei komprimiert wird.

Wert	Bedeutung
0	Keine Kompression.
1	Flate-Kompression
2	Run-Length-Kompression
3	FastFlate-Kompression
Default	1

Picture.JPEGQuality: Spezifiziert die Qualität und den davon abhängigen Kompressionsfaktor der generierten JPEG Grafiken. Der Wert liegt zwischen 0..100, wobei 100 der

höchsten JPEG Qualität (und damit vergleichsweise schlechtesten Kompression) entspricht. Wirkt sich nur aus wenn die Quellgrafik nicht im JPEG-Format vorliegt, da eine Codierung von JPEG nach JPEG einen Qualitätsverlust mit sich bringen würde. Default: 75

PDF.DontStackWorldModifications: Beeinflusst wie koordinatensystemverändernde EMF-Records behandelt werden. Über diese Option können u.U. Rundungsfehler bei der Ausgabe von komplexen eigenen EMF-Records umgangen werden, die auch dazu führen können, dass einzelne Objekte nicht sichtbar sind. Setzen Sie den Wert auf "1", um die Option zu aktivieren (Default: 0).

Export.File: Gibt den Dateinamen für das zu generierende PDF-Dokument an. Wenn leer, dann wird der Dateiauswahl-Dialog angezeigt.

Export.Path: Gibt den Pfad für das zu generierende PDF-Dokument an.

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt keinen Dateiauswahl-Dialog (sofern *Export.File* gesetzt) und keine "Überschreiben?" Rückfrage.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export, die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise der Acrobat Reader® o.ä. gestartet werden sollte
Default	0

Export.ShowResultAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahldialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

PDF.PDFAMode: Legt fest, ob ein PDF vom Typ PDF/A erzeugt wird.

Wert	Bedeutung
0	kein PDF/A erzeugen
1	PDF/A-1a erzeugen
Default	0

PDF.Author: Setzt das Author-Tag in der PDF-Datei. Default: leer.

7.3.7 Powerpoint Export-Modul

Übersicht

Das Powerpoint Export-Modul erzeugt Dokumente im Microsoft Powerpoint® Format. Die Erzeugung läuft unabhängig von einer Installation dieses Produktes ab, es handelt sich also um eine native Unterstützung. Es wird ein voller Layout-erhaltender Export durchgeführt.

Einschränkungen

U.a. sind folgende Einschränkungen und Hinweise beim Powerpoint Export-Modul zu beachten:

- Benötigt .NET Framework 3.5.
- Kompatibel mit Microsoft Powerpoint® 2007 und höher.
- Es wird empfohlen, dass die Breite aller Spalten einer Zeile der Gesamtbreite des Berichtscontainers entspricht. Versuchen Sie die Ränder verschiedener Zellen, welche in mehreren Tabellenabschnitten (Kopfzeile, Datenzeile etc.) oder mehreren Zeilendefinitionen vorkommen, immer bündig zu designen. Andernfalls kann es in Microsoft Word zu einem verfälschten Ergebnis kommen.
- Spalten können nicht kleiner als 0,54 cm (5,4mm) sein. Diese werden von Powerpoint automatisch auf diese Größe skaliert.
- Schriftarten werden um 1% verkleinert, da es sonst zu Darstellungsproblemen unter Powerpoint kommen kann.
- Tabellenzeilen, die ein Bild enthalten, werden mit einer festen Höhe exportiert.
- Es wird keine Mischung von verschiedenen Seitenformaten unterstützt. Um bspw. einen Export von Hoch- und Querformat zu realisieren, können die Seiten mit dem gleichen Format in jeweils ein separates Dokument exportiert werden.
- Aufgrund verschiedener Formateinschränkungen kann es notwendig sein das Layout des Berichts vor dem Export anzupassen. Wir schlagen daher vor, die Ausgabe gründlich zu testen bevor Sie Ihr Projekt redistributieren.
- Tabulatoren werden nicht unterstützt.
- Der Ausfertigungsdruck wird nicht unterstützt.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung aller vom PPTX Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LIXSetParameter(..."PPTX"...)* gesetzt und über *LIXGetParameter(..."PPTX"...)* abgefragt werden.

Resolution: Definiert die Auflösung in dpi für Grafikgenerierung. Default: 300dpi.

Picture.BitsPerPixel: Gibt die Farbtiefe der generierten Grafiken an. Bitte beachten Sie, dass bei einer höheren Farbtiefe die Grafikdateien schnell sehr groß werden können.

Wert	Bedeutung
1	Schwarz-Weiß
24	24bit True Color
Default	24

Verbosity.Rectangle: Konfiguriert die Art und Weise, wie Rechteck-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Rechteck
2	Objekt als Grafik
Default	1

Verbosity.Barcode: Konfiguriert die Art und Weise, wie Barcode-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
Default	1

Verbosity.Drawing: Konfiguriert die Art und Weise, wie Grafik-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
Default	1

Verbosity.Ellipse: Konfiguriert die Art und Weise, wie Ellipsen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Ellipse
2	Objekt als Grafik
Default	1

Verbosity.Line: Konfiguriert die Art und Weise, wie Linien-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Linie
2	Objekt als Grafik
Default	1

Verbosity.Text: Konfiguriert die Art und Weise, wie Text-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Textobjekt
2	Objekt als Grafik
Default	1

Verbosity.RTF: Konfiguriert die Art und Weise, wie RTF-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
Default	1

Verbosity.Table: Konfiguriert die Art und Weise, wie Tabellen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als komplettes Tabellenobjekt

Wert	Bedeutung
Default	1

Verbosity.LIXObject: Konfiguriert die Art und Weise, wie LLX-Objekte (OLE, HTML, Chart) exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als Grafik
Default	1

PPTX.FontScalingPercentage: Skalierungsfaktor, um den Schriftgrößen korrigiert werden. Default: 100 (=100% Schriftgröße)

PPTX.Animation: Legt die Animation, die in den einzelnen Folien verwendet wird, fest

Wert	Bedeutung
0	Keine Animation
1	Schnitt-Animation
2	Verblassen-Animation
3	Schieben-Animation
4	Bedecken-Animation
5	Wischen-Animation
Default	0

Export.File: Gibt den Dateinamen für das zu generierende Word-Dokument an. Wenn leer, dann wird der Dateiauswahl-Dialog angezeigt.

Export.Path: Gibt den Pfad für das zu generierende Word-Dokument an.

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt keinen Dateiauswahl-Dialog (sofern *Export.File* gesetzt) und keine "Überschreiben?" Rückfrage.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export, die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise Microsoft Powerpoint® o.ä. gestartet werden sollte
Default	0

7.3.8 RTF Export-Modul

Übersicht

Das RTF Export-Modul erzeugt Dokumente im Rich Text Format nach der Spezifikation Version 1.5/1.7 von Microsoft. Die Export-Ergebnisse wurden in erster Linie für Microsoft Word sowie Lotus Word Pro optimiert. Die Ergebnisse werden jedoch häufig von Textverarbeitung zu Textverarbeitung gewisse Unterschiede aufweisen.

Einschränkungen

U.a. sind folgende Einschränkungen und Hinweise beim RTF Export-Modul zu beachten:

- Zeilen, die aneinander verankert sind, können nicht korrekt exportiert werden.
- Es kann max. eine Farbtiefe von 24bit eingestellt werden.
- Bei Rechteck-Objekten werden keine Schatten unterstützt.
- Tabulatoren in Textobjekten werden durch Leerzeichen ersetzt.
- Objekte sollten nicht zu nahe zum Randbereich einer Seite platziert werden. Manche Textverarbeitungen führen ansonsten vor diesen Objekten automatische Seitenumbrüche ein. Diese Umbrüche bewirken dann, dass alle folgenden Objekte auch auf der nächsten Seite platziert werden.
- Die Tabellen-Objekt Option "Separatoren Durchziehen" wird nicht unterstützt.
- Die Tabellenoption "fixe Größe" wird nicht unterstützt.
- Nicht alle Hintergrundmuster die auch in List & Label eingestellt werden können, sind auch auf den RTF Text übertragbar, in RTF stehen weniger Muster zur Verfügung.
- Das Chart- und HTML-Objekt werden als Bilder exportiert und können daher nicht transparent erscheinen.
- Gedrehte RTF-Objekte und Bilder werden nicht unterstützt.
- Rahmen um Objekte werden nicht unterstützt.
- Gradientenfüllungen werden nicht unterstützt.
- Gedrehte Texte werden nicht unterstützt.
- Objekte die als Bild exportiert werden dürfen nicht aus ihrem Objektrahmen ragen. Daher müssen z.B. Barcodeobjekte mit fester Balkenbreite im Designer so gestaltet werden, dass der Inhalt in jedem Fall im Objektrechteck Platz findet.

- Selbst in einem Callback gezeichnete Ausgaben werden nicht exportiert.
- Die Funktion *TotalPages$()* kann nicht in gedrehten Textobjekten verwendet werden.
- Absatzabstände werden nicht unterstützt.
- Der Ausfertigungsdruck wird nicht unterstützt.
- Es wird keine Mischung von verschiedenen Seitenformaten unterstützt. Um bspw. einen Export von Hoch- und Querformat zu realisieren, können die Seiten mit dem gleichen Format in jeweils ein separates Dokument exportiert werden.

Bekannte Besonderheiten allgemein:

- Rahmen die kleiner als 1/2 Pt (ca. 0,4 mm in List & Label) sind, werden nicht richtig dargestellt.
- Positionsrahmen werden u.U. von Word unüblich zu anderen Textverarbeitungen behandelt. Trotz gleicher Kantenlänge kann es passieren, dass Kanten unterschiedlich lang erscheinen. Die Längenangaben bei Positionsrahmen könnten also u.U. falsch interpretiert werden.
- Bei schmalen Linienobjekten kann es passieren, dass diese scheinbar nicht sichtbar sind. Dieses Problem zeigt sich hauptsächlich bei horizontalen Linienobjekten. Der Positionsrahmen des Objektes wird zwar an der richtigen Position mit der richtigen Größe dargestellt, aber die enthaltene Bitmap bekommt einen Offset und liegt somit außerhalb des sichtbaren Bereichs des Positionsrahmens.
- Ab Word 2000 und dessen Folgeversionen werden Tabellenrahmen nicht immer korrekt dargestellt.
- Word 2000 kann das Inhaltsverzeichnis nicht korrekt darstellen.
- Abstände innerhalb von Zellen werden nicht unterstützt.
- Einige mit List & Label darstellbare Farben können zwar exportiert, aber in Word nicht eingestellt werden. Deshalb kann es sein, dass Word diese in eine andere Farbe konvertiert, z.B. Hellgelb wird zu Grau.
- Große Bilder in hohen Auflösungen werden von Word gelegentlich nicht richtig dargestellt, obwohl sie in der RTF-Datei korrekt enthalten sind.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung aller vom RTF Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LIXSetParameter(...\"RTF\...)* gesetzt und über *LIXGetParameter(...\"RTF\...)* abgefragt werden.

Resolution: Definiert die Auflösung in dpi für Koordinatenumrechnung und Grafikgenerierung. Default: 96dpi, Bildschirmauflösung.

Picture.BitsPerPixel: Gibt die Farbtiefe der generierten Grafiken an. Bitte beachten Sie, dass bei einer höheren Farbtiefe die Grafikdateien schnell sehr groß werden können.

Wert	Bedeutung
1	Schwarz-Weiß
4	16 Farben
8	256 Farben
24	24bit True Color
Default	24

UsePosFrame: Beeinflusst die Positionierung von Texten.

Wert	Bedeutung
0	Es werden Textboxen zur Positionierung genutzt
1	Es werden Positionsrahmen zur Positionierung genutzt
Default	0

Verbosity.Rectangle: Konfiguriert die Art und Weise, wie Rechteck-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Positionsrahmen
Default	1

Verbosity.Barcode: Konfiguriert die Art und Weise, wie Barcode-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
Default	1

Verbosity.Drawing: Konfiguriert die Art und Weise, wie Grafik-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
Default	1

Verbosity.Ellipse: Konfiguriert die Art und Weise, wie Ellipsen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
2	Objekt als Shape-Objekt (ab Word 97)
Default	2

Verbosity.Line: Konfiguriert die Art und Weise, wie Linien-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
2	Objekt als Shape-Objekt (ab Word 97)
Default	2

Verbosity.Text: Konfiguriert die Art und Weise, wie Text-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als komplettes Textobjekt
2	Objekt als Grafik
Default	1

Verbosity.RTF: Konfiguriert die Art und Weise, wie RTF-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als formatierter RTF-Text
2	Objekt als Grafik
Default	1

Verbosity.Table: Konfiguriert die Art und Weise, wie Tabellen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als komplettes Tabellenobjekt
Default	1

Verbosity.LLXObject: Konfiguriert die Art und Weise, wie LLX-Objekte (z.B. HTML Objekt, Chart-Objekt) exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
Default	1

Export.Path: Definiert den Zielpfad für den Export.

Export.File: Gibt den Dateinamen für das zu generierende RTF-Dokument an. Wenn leer, dann wird der Dateiauswahl-Dialog angezeigt.

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt keinen Dateiauswahl-Dialog (sofern *Export.File* gesetzt) und keine "Überschreiben?" Rückfrage.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export, die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise eine Textverarbeitung o.ä. gestartet werden sollte
Default	0

Export.ShowResultAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahldialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

7.3.9 SVG Export-Modul

Übersicht

Das SVG Export-Modul erzeugt SVG Code gemäß Scalable Vector Graphics (SVG) 1.1 (Second Edition) Spezifikation.

Das Export-Modul sammelt dazu zuerst alle Objekte, die in dem Bericht vorkommen und platziert diese anhand ihrer Position und Ebene. Die Position eines Objekts ergibt sich aus zwei Werten: links und oben. Diese Werte geben den Abstand zum linken und oberen Rand der Seite an. Die Objekte werden absolut auf der Seite positioniert. Dies hat den Vorteil einer optisch genaueren Umsetzung.

Einschränkungen

Daneben gibt es natürlich diverse, durch das Zielformat bedingte Einschränkungen. Die wichtigsten werden nachfolgend genannt:

- Zeilen, die aneinander verankert sind, können nicht korrekt exportiert werden.
- Tabulatoren und mehrere aufeinanderfolgende Leerzeichen werden nicht unterstützt.
- Die Option 'Wortumbruch' in Textobjekten und Tabellenspalten ist bei SVG immer aktiv (auch wenn 'abschneiden' im Designer gewählt wurde)
- die Tabellen-Objekt Option "Separatoren Durchziehen" wird nicht unterstützt.
- Die Tabellenoption "fixe Größe" wird nicht unterstützt.
- Gedrehte Texte werden nicht unterstützt.
- RTF Texte werden als Bild exportiert.
- Objekte, die als Bild exportiert werden, dürfen nicht aus ihrem Objektrahmen ragen. Daher müssen z.B. Barcodeobjekte mit fester Balkenbreite im Designer so gestaltet werden, dass der Inhalt in jedem Fall im Objektrechteck Platz findet.
- Selbst in einem Callback gezeichnete Ausgaben werden nicht exportiert.
- Rahmen von benachbarten Zellen werden nicht übereinander, sondern nebeneinander gemalt. Dadurch kann sich die Rahmendicke verdoppeln. Bitte berücksichtigen Sie dies bereits beim Layout.
- Die Funktion *TotalPages$()* kann nicht in gedrehten Textobjekten verwendet werden.

- Auch wenn das HTML-Objekt einen Seitenumbruch auslöst, wird das exportierte Objekt auf einer Seite/in einer Datei ausgegeben. Ein Umbruch wird ignoriert.
- Eingebettete Scriptfunktionalitäten können verloren gehen.
- Schattenseiten werden nicht unterstützt.
- Es wird keine Mischung von verschiedenen Seitenformaten unterstützt. Um bspw. einen Export von Hoch- und Querformat zu realisieren, können die Seiten mit dem gleichen Format in jeweils ein separates Dokument exportiert werden.
- Der Ausfertigungsdruck wird nicht unterstützt.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung der vom SVG Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LIXSetParameter(..."SVG"...)* gesetzt und über *LIXGetParameter(..."SVG"...)* abgefragt werden.

Resolution: Definiert die Auflösung in dpi für Koordinatenumrechnung und Grafikgenerierung. Default: 96dpi, Bildschirmauflösung.

Picture.JPEGQuality: Spezifiziert die Qualität und den davon abhängigen Kompressionsfaktor der generierten JPEG Grafiken. Der Wert liegt zwischen 0…100, wobei 100 der höchsten JPEG Qualität (und damit vergleichsweise schlechtesten Kompression) entspricht. Wirkt sich nur aus wenn die Quellgrafik nicht im JPEG-Format vorliegt, da eine Codierung von JPEG nach JPEG einen Qualitätsverlust mit sich bringen würde. Default: 75

Picture.BitsPerPixel: Gibt die Farbtiefe der generierten Grafiken an. Bitte beachten Sie, dass bei einer höheren Farbtiefe die Grafikdateien schnell sehr groß werden können.

Wert	Bedeutung
1	Schwarz-Weiß
24	24bit True Color
Default	24

Verbosity.Rectangle: Konfiguriert die Art und Weise, wie Rechteck-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG (und zusätzlich als komplettes Rechteckobjekt, für farblich hinterlegte Objekte)
Default	1

Verbosity.Barcode: Konfiguriert die Art und Weise, wie Barcode-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.Drawing: Konfiguriert die Art und Weise, wie Grafik-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.Ellipse: Konfiguriert die Art und Weise, wie Ellipsen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.Line: Konfiguriert die Art und Weise, wie Linien-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als komplettes Linienobjekt
Default	1

Verbosity.Text: Konfiguriert die Art und Weise, wie Text-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als komplettes Textobjekt
2	Objekt als JPEG
Default	1

Verbosity.Text.Frames: Konfiguriert die Art und Weise, wie Rahmen um Textobjekte exportiert werden sollen.

Wert	Bedeutung
0	einzelne Rahmen für oben, unten, links, rechts erzeugen (verwendet CSS)
1	kompletter Rahmen als Box
Default	0

Verbosity.RTF: Konfiguriert die Art und Weise, wie RTF-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.Table: Konfiguriert die Art und Weise, wie Tabellen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als komplettes Tabellenobjekt
Default	1

Verbosity.Table.Cell: Konfiguriert die Art und Weise, wie die Tabellen-Zellen exportiert werden sollen.

Wert	Bedeutung
0	Zellen ignorieren
1	als komplettes Zellenobjekt (gemäß den Verbosity-Einstellungen der jeweiligen Objekttypen)
2	Zellen als JPEG
Default	1

Verbosity.Table.Frames: Konfiguriert die Art und Weise, wie Tabellen-Rahmen exportiert werden sollen.

Wert	Bedeutung
0	keine Tabellenrahmen zeichnen
1	nur horizontalen Tabellenrahmen als horizontale Linie berücksichtigen

Wert	Bedeutung
2	komplette Tabellenzeile mit allen Rahmen, sofern irgendein vertikaler Rahmen vorhanden, ansonsten wie 1
3	Zellenspezifische Rahmen zeichnen (verwendet Linienobjekte)
Default	3

Verbosity.LLXObject: Konfiguriert die Art und Weise, wie LLX-Objekte (z.B. Chart-Objekt) exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.LLXObject.HTMLObj: Konfiguriert die Art und Weise, wie das HTML Objekt exportiert werden soll.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

SVG.Title: Spezifiziert den Titel des zu generierenden SVG-Dokuments. Default: Titel, der bei *LlPrintWithBoxStart()* dem Druckjob übergeben wird.

Export.Path: Definiert den Zielpfad für den Export. Ist er leer, so erfolgt in jedem Fall ein Zielpfad-Auswahldialog.

Export.File: Gibt den Dateinamen für die erste zu generierende SVG-Seite an. Default: "index.svg". Sie können im Dateinamen auch printf-Platzhalter wie z.B. "%d" verwenden (z.B. "Export Seite %d.svg"). In diesem Falle werden die erste Seite und die Folgeseiten durch Ersetzung des Platzhalters durch die entsprechend formatierte Seitenzahl benannt. Ansonsten erhalten die Seiten eine einfache Nummerierung.

Export.AllInOneFile: Konfiguriert das Export-Resultat.

Wert	Bedeutung
0	Das Ergebnis sind n verschiedene SVG-Dateien, für jede gedruckte Seite eine SVG-Datei.
1	Das Ergebnis ist eine einzige SVG Datei (*Export.File)*, in der alle gedruckten Seiten aneinander hängen.
Default	1

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt kein Zielpfad-Auswahldialog (sofern *Export.Path* gesetzt ist) und keine "Überschreiben?" Rückfrage. Ebenso wird keine Zusammenfassung der überlappenden Objekte, die ignoriert wurden, angezeigt.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise ein Web-Browser gestartet wird
Default	0

Export.ShowResultAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahldialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

Hyperlinks

Im Designer kann ein Hyperlink direkt über die Funktion *HyperLink$()* erzeugt und in Text-, RTF- und Tabellenobjekten eingebettet werden. So werden auch dynamische Hyperlinks inkl. Formeln möglich.

Eine andere Möglichkeit sind die Objektnamen, die im Designer für die Objekte vergeben werden können. Diese werden durch das SVG Export-Modul untersucht und ermöglichen eine erweiterte Exportfunktionalität:

Enthält der Objektname den Text "/LINK:<url>" so wird für Text- und Bildobjekte zusätzlich ein Hyperlink auf <url> generiert.

Beispiel:

Aus dem Grafikobjekt-Objektnamen "combit /LINK:http://www.combit.net" würde dann im SVG-Export eine Grafik mit einem dahinterliegenden Hyperlink auf "www.combit.net".

7.3.10 Text (CSV) Export-Modul

Übersicht

Der CSV-Export liefert die Daten aus Tabellenobjekten in einem Textformat zurück. Dabei können Eigenschaften wie Spalteneinrahmung und Spaltentrennung frei bestimmt werden. Einzelne Datensätze werden durch einen Zeilenumbruch getrennt. Das Ergebnis ist eine einzelne Textdatei, die die Daten aus allen Tabellenobjekten enthält. Diese kann dann zur Weiterverarbeitung in anderen Applikationen verwendet werden. Beachten Sie bitte, dass in diesem Modus nur Daten aus Tabellen exportiert werden und keinerlei Layout-Informationen ausgewertet werden. Dies bedeutet auch, dass z.B. layoutbedingte Umbrüche aus dem exportierten Text gefiltert werden. Dieser Modus steht nur bei Tabellenprojekten zur Verfügung.

Einschränkungen

Das Text (CSV) Export-Modul besitzt folgende Einschränkungen:

- Der Ausfertigungsdruck wird nicht unterstützt.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung aller vom Text Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LIXSetParameter(..."TXT"...)* gesetzt und über *LIXGetParameter(...TXT"...)* abgefragt werden.

Export.Path: Definiert den Zielpfad für den Export. Ist er leer, so erfolgt in jedem Fall ein Zielpfad-Auswahldialog.

Export.File: Gibt den Dateinamen an. Default: "export.txt"

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt kein Zielpfad-Auswahldialog (sofern *Export.Path* gesetzt ist) und keine "Überschreiben?" Rückfrage.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise ein Text-Editor gestartet wird
Default	0

Export.ShowResultAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahldialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

TXT.FrameChar: Diese Zeichenkette spezifiziert das Spalteneinrahmungszeichen.

Wert	Bedeutung
NONE	Keine Spalteneinrahmung
"	" als Spalteneinrahmung
'	' als Spalteneinrahmung

TXT.SeparatorChar: Diese Zeichenkette spezifiziert das Spaltentrennzeichen.

Wert	Bedeutung
NONE	Keine Spaltentrennung
TAB	Tabulator als Spaltentrennung
BLANK	Leerzeichen als Spaltentrennung
,	, als Spalteneintrennung
;	; als Spaltentrennung

TXT.IgnoreGroupLines: Erlaubt Gruppenkopf- und Fußzeilen zu ignorieren, wenn diese nicht in der resultierenden Textdatei erscheinen sollen.

Wert	Bedeutung
0	Gruppenzeilen werden exportiert
1	Gruppenzeilen werden ignoriert
Default	1

TXT.IgnoreHeaderFooterLines: Erlaubt Kopf- und Fußzeilen zu ignorieren, wenn diese nicht in der resultierenden Textdatei erscheinen sollen.

Wert	Bedeutung
0	Kopf- und Fußzeilen werden exportiert
1	Kopf- und Fußzeilen werden ignoriert
2	Kopf- und Fußzeilen werden genau einmal auf der ersten Seite exportiert. Möchten Sie die Fußzeile nur auf der letzten Seite exportieren, geben Sie der Fußzeile die Darstellungsbedingung *LastPage()*.

Wert	Bedeutung
Default	1

TXT.Charset: Bestimmt den Zeichensatz der Ergebnisdatei. Hier muss zusätzlich die Codepage (z.B. 932 für Japanisch) per *LL_OPTION_CODEPAGE* gesetzt werden.

Wert	Bedeutung
ANSI	Ansi-Zeichensatz
ASCII	Ascii-Zeichensatz
UNICODE	Unicode-Zeichensatz
Default	UNICODE

7.3.11 Text (Layout) Export-Modul

Übersicht

Das Layout Export-Modul kann alternativ auch eine Textdatei erzeugen, die – soweit es das Format zulässt – die Formatierung des Originalprojektes widerspiegelt. Beachten Sie, dass die Schriftgröße so gewählt sein sollte, dass die einzelnen Zeilen im Textexport noch aufgelöst werden können. Zu kleine Schriftarten können zu überschriebenen Zeilen führen, d.h. es gehen Zeilen in der Ausgabedatei verloren. Eine Schriftgösse von mindestens 12 pt wird empfohlen.

Einschränkungen

Das Text (CSV) Export-Modul besitzt folgende Einschränkungen:

- Der Ausfertigungsdruck wird nicht unterstützt.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung aller vom Text Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LIXSetParameter(...\"TXT_LAYOUT\"...)* gesetzt und über *LIXGetParameter(...TXT_LAYOUT\"...)* abgefragt werden.

Verbosity.Text: Konfiguriert die Art und Weise, wie Text-Objekte in Tabellenspalten exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als komplettes Textobjekt
Default	1

Verbosity.RTF: Konfiguriert die Art und Weise, wie RTF-Objekte in Tabellenspalten exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als RTF-Stream
2	als unformatierter Text
Default	2

Verbosity.Table: Konfiguriert die Art und Weise, wie Tabellen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Als komplettes Tabellenobjekt
Default	1

Verbosity.Table.Cell: Konfiguriert die Art und Weise, wie die Tabellen-Zellen exportiert werden sollen.

Wert	Bedeutung
0	Zellen ignorieren
1	Als komplettes Zellenobjekt (gemäß den Verbosity-Einstellungen der jeweiligen Objekttypen)
Default	1

Export.Path: Definiert den Zielpfad für den Export. Ist er leer, so erfolgt in jedem Fall ein Zielpfad-Auswahldialog.

Export.File: Gibt den Dateinamen an. Default: "export.txt"

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt kein Zielpfad-Auswahldialog (sofern *Export.Path* gesetzt ist) und keine "Überschreiben?" Rückfrage.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise ein Text-Editor gestartet wird
Default	0

Export.ShowResultAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahldialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

Export.AllInOneFile: Konfiguriert das Export-Resultat.

Wert	Bedeutung
0	Das Ergebnis sind n verschiedene TXT-Dateien, für jede gedruckte Seite eine. Die Dateinamen werden (außer der *Export.File* Startdatei) fortlaufend durchnummeriert. Enthält der Dateiname der Startdatei den Formatidentifier "%d", so wird dieser durch die jeweilige Seitenzahl ersetzt.
1	Das Ergebnis ist eine einzige TXT Datei (*Export.File)*, in der alle gedruckten Seiten aneinander hängen.
Default	1

TXT.Charset: Bestimmt den Zeichensatz der Ergebnisdatei.

Wert	Bedeutung
ANSI	Ansi-Zeichensatz
ASCII	Ascii-Zeichensatz
UNICODE	Unicode-Zeichensatz
Default	UNICODE

7.3.12 TTY Export-Modul

Übersicht

Das TTY-Exportformat kann verwendet werden, um z.B. mit Nadeldruckern direkt zu kommunizieren. Die Umgehung des Windows-Treibers bringt große Performance-Vorteile mit sich, was gerade im Etikettendruck bei großen Stückzahlen wichtig ist.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung aller vom TTY Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LIXSetParameter(...*"TTY"*...)* gesetzt und über *LIXGetParameter(...*"TTY"*...)* abgefragt werden.

Export.File: Gibt den Dateinamen für die zu generierende PRN-Datei an. Wenn leer, dann wird der Dateiauswahl-Dialog angezeigt.

Export.Path: Gibt den Pfad für die zu generierende PRN-Datei an.

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt kein Zielpfad-Auswahldialog (sofern *Export.Path* gesetzt ist) und keine "Überschreiben?" Rückfrage.
Default	0

TTY.AdvanceAfterPrint: Bestimmt das Vorschub-Verhalten nach Druckende.

Wert	Bedeutung
FormFeed	Seitenvorschub nach Ausgabe
ToNextLabel	Vorschub auf Anfang des nächsten Etiketts
AfterNextLabel	Vorschub auf Anfang des übernächsten Etiketts (ein leeres Etikett als "Abtrennbereich")

TTY.Emulation: Bezeichnet die Emulation, die für den Export verwendet wird.

Wert	Bedeutung
ESC/P	ESC/P-Emulation
ESC/P 9Pin	ESC/P-Emulation für 9-Nadeldrucker
PlainTextANSI	Klartext ANSI Emulation
PlainTextASCII	Klartext ASCII Emulation
PlainTextUNICODE	Klartext Unicode Emulation
NEC Pinwriter	NEC Nadeldrucker Emulation
IBM Proprinter XL24	IBM Proprinter XL24 Emulation
PCL	PCL-Emulation

TTY.Destination: Ziel für den Export. Mögliche Werte sind z.B. "LPT1:", "LPT2:",..."FILE:" bzw. "FILE:<Filename>". Wenn "FILE:" verwendet wird, bekommt der Benutzer einen Dateiauswahl-Dialog angezeigt.

TTY.DefaultFilename: Vorschlagsname für diesen Dialog

7.3.13 Windows Fax Export-Modul

Sie können über dieses Exportmodul List & Label Dokumente direkt als Fax über den Windows Faxdienst verschicken. Der entsprechende Druckertreiber wird dann automatisch eingerichtet, wenn Sie ein faxtaugliches Modem an Ihrem Rechner installiert haben. Beim Versenden eines Faxes werden aber zusätzliche Informationen gebraucht, um das Fax adressieren zu können, damit kein extra Dialog angezeigt werden muss. Über LL_OPTIONSTR_FAX... können Sie diese aus Ihrer Applikation heraus vorgeben (s. *LlSetOptionString()*).

Beispiel:

```
HLLJOB hJob;
hJob = LlJobOpen(0);
LlSetOptionString(hJob, LL_OPTIONSTR_FAX_RECIPNAME, "combit");
LlSetOptionString(hJob, L_OPTIONSTR_FAX_RECIPNUMBER, "+497531906018");
LlSetOptionString(hJob, LL_OPTIONSTR_FAX_SENDERNAME, "Max Mustermann");
LlSetOptionString(hJob, LL_OPTIONSTR_FAX_SENDERCOMPANY, "Sunshine GmbH");
LlSetOptionString(hJob, LL_OPTIONSTR_FAX_SENDERDEPT, "Development");
LlSetOptionString(hJob, LL_OPTIONSTR_FAX_SENDERBILLINGCODE, "4711");
// ...
LlJobClose(hJob);
```

Wenn diese Optionen nicht gesetzt werden und der Benutzer im Projekt keine Einstellungen gewählt hat, steht das Fax Export-Modul nicht zur Verfügung.

Dieses Exportmodul bietet Ihnen keine weitergehende Programmierschnittstelle.

Viele herkömmliche Faxprogramme können von List & Label aus auch direkt über den zugehörigen Druck-/Fax-Treiber angesprochen werden. Sofern das entsprechende Faxprogramm Möglichkeiten vorsieht, dass die Faxnummer über das Dokument übergeben wird, kann in den meisten Fällen somit auch der Nummerneingabedialog unterdrückt werden. Um z.B. David der Firma Tobit anzusprechen, können Sie die sog. @@-Befehle verwenden. Platzieren Sie ein Textobjekt im Designer und fügen Sie die Zeile

```
"@@NUMMER "+<Faxnummer bzw. Feldname>+"@@"
```

als Inhalt ein. Der Faxtreiber erkennt diese Syntax und versendet den Druckauftrag ohne weitere Benutzerinteraktion an die angegebene Faxnummer. Andere Faxprogramme

bieten ähnliche Möglichkeiten – wir empfehlen Ihnen einen Blick in die Dokumentation Ihres Faxprogrammes.

7.3.14 Word Export-Modul

Übersicht

Das Word Export-Modul erzeugt Dokumente im Microsoft Word® Format. Die Erzeugung läuft unabhängig von einer Installation dieses Produktes ab, es handelt sich also um eine native Unterstützung. Es wird ein voller Layout-erhaltender Export durchgeführt. Tabellen werden Seiten-fortlaufend erzeugt um eine optimale nachträgliche Bearbeitung zu ermöglichen.

Einschränkungen

U.a. sind folgende Einschränkungen und Hinweise beim Word Export-Modul zu beachten:

- Benötigt .NET Framework 3.5.
- Kompatibel mit Microsoft Word® 2007 und höher.
- Es wird empfohlen, dass die Breite aller Spalten einer Zeile der Gesamtbreite des Berichtscontainers entspricht. Versuchen Sie die Ränder verschiedener Zellen, welche in mehreren Tabellenabschnitten (Kopfzeile, Datenzeile etc.) oder mehreren Zeilendefinitionen vorkommen, immer bündig zu designen. Andernfalls kann es in Microsoft Word zu einem verfälschten Ergebnis kommen.
- Tabellenzeilen, die ein Bild enthalten, werden mit einer festen Höhe exportiert.
- Es wird keine Mischung von verschiedenen Seitenformaten unterstützt. Um bspw. einen Export von Hoch- und Querformat zu realisieren, können die Seiten mit dem gleichen Format in jeweils ein separates Dokument exportiert werden.
- Aufgrund verschiedener Formateinschränkungen kann es notwendig sein das Layout des Berichts vor dem Export anzupassen. Wir schlagen daher vor, die Ausgabe gründlich zu testen bevor Sie Ihr Projekt redistributieren. Beachten Sie auch die Optionen DOCX.CellScalingPercentageHeight und DOCX.CellScalingPercentageWidth.
- Tabulatoren werden nicht unterstützt.
- Der Ausfertigungsdruck wird nicht unterstützt.
- Die Einpassen-Option "verschmälern" in den Eigenschaften einer Spalte wird nicht unterstützt.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung aller vom DOCX Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LlXSetParameter(..."DOCX"...)* gesetzt und über *LlXGetParameter(..."DOCX"...)* abgefragt werden.

Resolution: Definiert die Auflösung in dpi für Grafikgenerierung. Default: 300dpi.

Picture.BitsPerPixel: Gibt die Farbtiefe der generierten Grafiken an. Bitte beachten Sie, dass bei einer höheren Farbtiefe die Grafikdateien schnell sehr groß werden können.

Wert	Bedeutung
1	Schwarz-Weiß
24	24bit True Color
Default	24

Verbosity.Rectangle: Konfiguriert die Art und Weise, wie Rechteck-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Rechteck
2	Objekt als Grafik
Default	1

Verbosity.Barcode: Konfiguriert die Art und Weise, wie Barcode-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
Default	1

Verbosity.Drawing: Konfiguriert die Art und Weise, wie Grafik-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Grafik
Default	1

Verbosity.Ellipse: Konfiguriert die Art und Weise, wie Ellipsen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Ellipse
2	Objekt als Grafik

Wert	Bedeutung
Default	1

Verbosity.Line: Konfiguriert die Art und Weise, wie Linien-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Linie
2	Objekt als Grafik
Default	1

Verbosity.Text: Konfiguriert die Art und Weise, wie Text-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als Textobjekt
2	Objekt als Grafik
Default	1

Verbosity.RTF: Konfiguriert die Art und Weise, wie RTF-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als formatierter Text
2	Objekt als unformatierter Text
3	Objekt als Grafik
Default	1

Verbosity.Table: Konfiguriert die Art und Weise, wie Tabellen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als komplettes Tabellenobjekt
Default	1

Verbosity.LIXObject: Konfiguriert die Art und Weise, wie LLX-Objekte (OLE, HTML, Chart) exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als Grafik
Default	1

DOCX.FontScalingPercentage: Skalierungsfaktor, um den Schriftgrößen korrigiert werden. Default: 100 (=100% Schriftgröße)

DOCX.AllPagesOneFile: Erlaubt es, für jede Seite ein eigenes Word Dokument zu erzeugen.

Wert	Bedeutung
0	Pro Seite wird ein eigenes Word Dokument angelegt
1	Alle Seiten werden im gleichen Word Dokument erzeugt
Default	1

DOCX.CellScalingPercentageHeight: Skalierungsfaktor, um den Zellenhöhen korrigiert werden. Default: 100 (=100% Zellenhöhe)

DOCX.CellScalingPercentageWidth: Skalierungsfaktor, um den Zellenbreiten korrigiert werden. Default: 100 (=100% Zellenbreite)

DOCX.FloatingTableMode: Gibt an, ob Tabellen miteinander verknüpft werden. Bei einer größeren Anzahl von Seiten mit Tabellen muss diese Option auf '0' gesetzt werden, da Microsoft Office Word maximal 46 Tabellen miteinander verknüpfen kann.

Wert	Bedeutung
0	Tabellen werden nicht miteinander verknüpft
1	Tabellen werden miteinander verknüpft
Default	1

DOCX.IgnoreCellPadding: Gibt an, ob der Rahmenabstand einer Zelle ignoriert wird.

Wert	Bedeutung
0	Rahmenabstände werden nicht ignoriert
1	Rahmenabstände werden ignoriert
Default	0

Export.File: Gibt den Dateinamen für das zu generierende Word-Dokument an. Wenn leer, dann wird der Dateiauswahl-Dialog angezeigt.

Export.Path: Gibt den Pfad für das zu generierende Word-Dokument an.

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt keinen Dateiauswahl-Dialog (sofern *Export.File* gesetzt) und keine "Überschreiben?" Rückfrage.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export, die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise Microsoft Word® o.ä. gestartet werden sollte
Default	0

7.3.15 XHTML/CSS Export-Modul

Übersicht

Das XHTML/CSS Export-Modul erzeugt XHTML Code gemäß XHTML 1.0 Spezifikation und CSS Code gemäß CSS 2.1 Spezifikation.

Das Export-Modul sammelt dazu zuerst alle Objekte, die in dem Bericht vorkommen und ordnet diese dann gemäß ihrer Höhe, Breite und Position an. Die Position eines Objekts ergibt sich aus zwei Werten: links und oben. Diese Werte geben den Abstand zum linken und oberen Rand der Seite an. Die Objekte werden absolut auf der Seite positioniert. Dies hat den Vorteil einer optisch genaueren Umsetzung.

Einschränkungen

Daneben gibt es natürlich diverse, durch das Zielformat bedingte Einschränkungen. Die wichtigsten werden im nachfolgend genannt:

- Zeilen, die aneinander verankert sind, können nicht korrekt exportiert werden.
- Der Dezimaltabulator in Textobjekten und Tabellen wird auf 'rechtsbündig' umgesetzt.
- Tabulatoren und mehrere aufeinanderfolgende Leerzeichen werden nicht unterstützt.
- Die Option 'Wortumbruch' in Textobjekten und Tabellenspalten ist bei XHTML immer aktiv (auch wenn 'abschneiden' im Designer gewählt wurde).
- Die Tabellen-Objekt Option "Separatoren Durchziehen" wird nicht unterstützt.

- Das Chart-Objekt wird als Bild exportiert und kann daher nicht transparent erscheinen.
- Die Umsetzung von formatiertem RTF-Text in XHTML-Code erfolgt über einen RTF-Parser, der die wichtigsten Absatz- und Zeichenformatierungen interpretiert und entsprechend umsetzt. Erweiterte Formatierungen, autom. Nummerierungen, sowie eingebettete Objekte und Grafiken werden ignoriert.
- Diagonale Linien werden als Grafik realisiert.
- Gradientenfüllungen, mit mehr als drei Farben, werden nicht unterstützt.
- Objekte, die als Bild exportiert werden, dürfen nicht aus ihrem Objektrahmen ragen. Daher müssen z.B. Barcodeobjekte mit fester Balkenbreite im Designer so gestaltet werden, dass der Inhalt in jedem Fall im Objektrechteck Platz findet.
- Selbst in einem Callback gezeichnete Ausgaben müssen explizit als Bilder exportiert werden.
- Rahmen von benachbarten Zellen werden nicht übereinander, sondern nebeneinander gemalt. Dadurch kann sich die Rahmendicke verdoppeln. Bitte berücksichtigen Sie dies bereits beim Layout.
- Auch wenn das HTML-Objekt einen Seitenumbruch auslöst, wird das exportierte Objekt auf einer Seite/in einer Datei ausgegeben. Ein Umbruch wird ignoriert.
- Eingebettete Scriptfunktionalitäten können verloren gehen.
- Der Ausfertigungsdruck wird nicht unterstützt.
- Gedrehte Beschreibungen im Kreuztabellen-Objekt werden nicht unterstützt.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung der vom XHTML/CSS Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LlXSetParameter(..."XHTML"...)* gesetzt und über *LlXGetParameter(..."XHTML"...)* abgefragt werden.

Resolution: Definiert die Auflösung in dpi für Koordinatenumrechnung und Grafikgenerierung. Default: 96dpi, Bildschirmauflösung.

Picture.JPEGQuality*:* Spezifiziert die Qualität und den davon abhängigen Kompressionsfaktor der generierten JPEG Grafiken. Der Wert liegt zwischen 0..100, wobei 100 der höchsten JPEG Qualität (und damit vergleichsweise schlechtesten Kompression) entspricht. Wirkt sich nur aus wenn die Quellgrafik nicht im JPEG-Format vorliegt, da eine Codierung von JPEG nach JPEG einen Qualitätsverlust mit sich bringen würde. Default: 75

Picture.BitsPerPixel: Gibt die Farbtiefe der generierten Grafiken an. Bitte beachten Sie, dass bei einer höheren Farbtiefe die Grafikdateien schnell sehr groß werden können.

Wert	Bedeutung
1	Schwarz-Weiß

Wert	Bedeutung
24	24bit True Color
Default	24

Verbosity.Rectangle: Konfiguriert die Art und Weise, wie Rechteck-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG (und zusätzlich als komplettes Rechteckobjekt, für farblich hinterlegte Objekte)
Default	1

Verbosity.Barcode: Konfiguriert die Art und Weise, wie Barcode-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.Drawing: Konfiguriert die Art und Weise, wie Grafik-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.Ellipse: Konfiguriert die Art und Weise, wie Ellipsen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.Line: Konfiguriert die Art und Weise, wie Linien-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.Text: Konfiguriert die Art und Weise, wie Text-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als komplettes Textobjekt
2	Objekt als JPEG
Default	1

Verbosity.Text.Frames: Konfiguriert die Art und Weise, wie Rahmen um Textobjekte exportiert werden sollen.

Wert	Bedeutung
0	einzelne Rahmen für oben, unten, links, rechts erzeugen (verwendet CSS)
1	kompletter Rahmen als Box
Default	0

Verbosity.RTF: Konfiguriert die Art und Weise, wie RTF-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als formatierter RTF-Text (wird interpretiert, nach XHTML gewandelt und mit CSS formatiert)
2	als unformatierter Text (als Schriftart wird die beim Projekt eingestellte Default-Schriftart verwendet)
3	Objekt als JPEG
Default	1

Verbosity.RTF.Frames: Konfiguriert die Art und Weise, wie Rahmen um RTF-Objekte exportiert werden sollen.

Wert	Bedeutung
0	einzelne Rahmen für oben, unten, links, rechts erzeugen (verwendet CSS)
1	kompletter Rahmen als Box
Default	0

Verbosity.Table: Konfiguriert die Art und Weise, wie Tabellen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als komplettes Tabellenobjekt
Default	1

Verbosity.Table.Cell: Konfiguriert die Art und Weise, wie die Tabellen-Zellen exportiert werden sollen.

Wert	Bedeutung
0	Zellen ignorieren
1	als komplettes Zellenobjekt (gemäß den Verbosity-Einstellungen der jeweiligen Objekttypen)
2	Zellen als JPEG
Default	1

Verbosity.Table.Frames: Konfiguriert die Art und Weise, wie Tabellen-Rahmen exportiert werden sollen.

Wert	Bedeutung
0	keine Tabellenrahmen zeichnen
1	nur horizontalen Tabellenrahmen als horizontale Linie berücksichtigen
2	komplette Tabellenzeile mit allen Rahmen, sofern irgendein vertikaler Rahmen vorhanden, ansonsten wie 1
3	Zellenspezifische Rahmen zeichnen (verwendet CSS)
Default	3

Verbosity.LLXObject: Konfiguriert die Art und Weise, wie LLX-Objekte (z.B. Chart-Objekt) exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
Default	1

Verbosity.LLXObject.HTMLObj: Konfiguriert die Art und Weise, wie das HTML Objekt exportiert werden soll.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als JPEG
2	Objekt im HTML Format. Dabei werden nur die HTML text Anweisungen zwischen <BODY> und </BODY> exportiert. Bitte beachten Sie die o.g. Einschränkungen.
Default	2

XHTML.Title: Spezifiziert den Titel des zu generierenden XHTML-Dokuments. Default: Titel, der bei *LlPrintWithBoxStart()* dem Druckjob übergeben wird.

XHTML.UseAdvancedCSS: Gibt an, ob nicht standardisierte CSS Formatierungen verwendet werden.

Wert	Bedeutung
0	Es werden keine, nicht standardisierte, CSS Formatierungen verwendet.
1	Es werden, nicht standardisierte, CSS Formatierungen verwendet. Zum Beispiel um einen Farbverlauf darzustellen.
Default	1

XHTML.ToolbarType: Gibt an, ob eine Toolbar, mit erweiterten Funktionen, erzeugt werden soll.

Wert	Bedeutung
0	Es wird keine Toolbar erzeugt.
1	Es wird eine Toolbar im Farbschema *Skyblue* erzeugt.
2	Es wird eine Toolbar im Farbschema *Blue* erzeugt.
3	Es wird eine Toolbar im Farbschema *Black* erzeugt.
4	Es wird eine Toolbar im Farbschema *Web* erzeugt.
Default	1

XHTML.UseSeparateCSS: Gibt an, ob eine separate CSS Datei erzeugt werden soll.

Wert	Bedeutung
0	CSS wird in den HEAD-Bereich der XHTML Datei geschrieben.
1	CSS wird in eine separate Datei geschrieben.
Default	0

Layouter.Percentaged: Gibt an, ob das Layout absolut oder prozentual zur Seitenbreite erfolgen soll.

Wert	Bedeutung
0	Layout in X-Richtung absolut in Pixel
1	Layout in X-Richtung überall prozentual auf Seitenbreite
Default	0

Export.Path: Definiert den Zielpfad für den Export. Ist er leer, so erfolgt in jedem Fall ein Zielpfad-Auswahldialog.

Export.File: Gibt den Dateinamen für die erste zu generierende XHTML-Seite an. Default: "index.htm". Sie können im Dateinamen auch printf-Platzhalter wie z.B. "%d" verwenden (z.B. "Export Seite %d.htm"). In diesem Falle werden die erste Seite und die Folgeseiten durch Ersetzung des Platzhalters durch die entsprechend formatierte Seitenzahl benannt. Ansonsten erhalten die Seiten eine einfache Nummerierung.

Export.AllInOneFile: Konfiguriert das Export-Resultat.

Wert	Bedeutung
0	Das Ergebnis sind n verschiedene HTML-Dateien, für jede gedruckte Seite eine HTML-Datei.
1	Das Ergebnis ist eine einzige HTML Datei (*Export.File)*, in der alle gedruckten Seiten aneinander hängen.
Default	1

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt kein Zielpfad-Auswahldialog (sofern *Export.Path* gesetzt ist) und keine "Überschreiben?" Rückfrage. Ebenso wird keine Zusammenfassung der überlappenden Objekte, die ignoriert wurden, angezeigt.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise ein Web-Browser gestartet wird
Default	0

Export.ShowResultAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahldialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

Hyperlinks

Im Designer kann ein Hyperlink direkt über die Funktion *HyperLink$()* erzeugt und in Text-, RTF- und Tabellenobjekten eingebettet werden. So werden auch dynamische Hyperlinks inkl. Formeln möglich.

Eine andere Möglichkeit sind die Objektnamen, die im Designer für die Objekte vergeben werden können. Diese werden durch das XHTML/CSS Export-Modul untersucht und ermöglichen eine erweiterte Exportfunktionalität:

Enthält der Objektname den Text "/LINK:<url>" so wird für Text- und Bildobjekte zusätzlich ein Hyperlink auf <url> generiert.

Beispiel:

Aus dem Grafikobjekt-Objektnamen "combit /LINK:http://www.combit.net" würde dann im XHTML/CSS-Export eine Grafik mit dem Alternativ-Text "combit" und einem dahinterliegenden Hyperlink auf "www.combit.net".

7.3.16 XML Export-Modul

Übersicht

Mit dem XML Export-Modul kann der Report im XML Format erzeugt werden. Dies ermöglicht eine flexible Weiterverarbeitung durch andere Anwendungen. Sämtliche verfügbaren Objektinformationen werden dabei exportiert. Sind nur die Daten innerhalb einer Tabelle interessant, so kann der Export auf diese reduziert werden, so dass sämtliche Koordinatenangaben, Objekteigenschaften u.ä. entfallen.

Einschränkungen

Das XML Export-Modul besitzt folgende Einschränkungen:

- Der Ausfertigungsdruck wird nicht unterstützt.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung aller vom XML Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LIXSetParameter(..."XML"...)* gesetzt und über *LIXGetParameter(..."XML"...)* abgefragt werden.

Resolution: Definiert die Auflösung in dpi für Koordinatenumrechnung und Grafikgenerierung. Default: 96dpi, Bildschirmauflösung.

Picture.JPEGQuality: Spezifiziert die Qualität und den davon abhängigen Kompressionsfaktor der generierten JPEG Grafiken. Der Wert liegt zwischen 0..100, wobei 100 der höchsten JPEG Qualität (und damit vergleichsweise schlechtesten Kompression) entspricht. Wirkt sich nur aus wenn die Quellgrafik nicht im JPEG-Format vorliegt, da eine Codierung von JPEG nach JPEG einen Qualitätsverlust mit sich bringen würde. Default: 75

Picture.JPEGEncoding: Gibt an, wie die JPEG Bilder codiert werden sollen

Wert	Bedeutung
0	JPEG Bilder werden als (externe) Dateien gespeichert
1	JPEG Bilder werden Mime-encoded innerhalb der XML Datei gespeichert
2	JPEG Bilder werden gar nicht gespeichert
Default	0

Picture.BitsPerPixel: Gibt die Farbtiefe der generierten Grafiken an. Bitte beachten Sie, dass bei einer höheren Farbtiefe die Grafikdateien schnell sehr groß werden können.

Wert	Bedeutung
1	Schwarz-Weiß
24	24bit True Color
Default	24

Verbosity.Rectangle: Konfiguriert die Art und Weise, wie Rechteck-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Komplettinformation des Objekts
2	Objekt als JPEG

Wert	Bedeutung
Default	1

Verbosity.Barcode: Konfiguriert die Art und Weise, wie Barcode-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Komplettinformation des Objekts inkl. als JPEG
Default	1

Verbosity.Drawing: Konfiguriert die Art und Weise, wie Grafik-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Komplettinformation des Objekts inkl. als JPEG
Default	1

Verbosity.Ellipse: Konfiguriert die Art und Weise, wie Ellipsen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Komplettinformation des Objekts
2	Objekt als JPEG
Default	1

Verbosity.Line: Konfiguriert die Art und Weise, wie Linien-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Komplettinformation des Objekts
2	Objekt als JPEG
Default	1

Verbosity.Text: Konfiguriert die Art und Weise, wie Text-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Objekt als komplettes Textobjekt
2	Objekt als JPEG
Default	1

Verbosity.RTF: Konfiguriert die Art und Weise, wie RTF-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als RTF-Stream
2	als unformatierter Text
3	Objekt als JPEG
Default	1

Verbosity.Table: Konfiguriert die Art und Weise, wie Tabellen-Objekte exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	als komplettes Tabellenobjekt
Default	1

Verbosity.Table.Cell: Konfiguriert die Art und Weise, wie die Tabellen-Zellen exportiert werden sollen.

Wert	Bedeutung
0	Zellen ignorieren
1	als komplettes Zellenobjekt (gemäß den Verbosity-Einstellungen der jeweiligen Objekttypen)
2	Zellen als JPEG
Default	1

Verbosity.LLXObject: Konfiguriert die Art und Weise, wie LLX-Objekte (z.B. HTML Objekt, Chart-Objekt) exportiert werden sollen.

Wert	Bedeutung
0	Objekt ignorieren
1	Komplettinformation des Objekts inkl. als JPEG
Default	1

XML.Title: Spezifiziert den Titel des zu generierenden XML-Dokuments. Default: Titel, der bei *LlPrintWithBoxStart()* dem Druckjob übergeben wird.

Export.Path: Definiert den Zielpfad für den Export. Ist er leer, so erscheint ein Zielpfad-Auswahldialog.

Export.File: Gibt den Dateinamen für die erste zu generierende XML-Seite an. Default: "export.xml". Sie können im Dateinamen auch printf-Platzhalter wie z.B. "%d" verwenden (z.B. "Export Seite %d.xml"). In diesem Falle werden die erste Seite und die Folgeseiten durch Ersetzung des Platzhalters durch die entsprechend formatierte Seitenzahl benannt. Ansonsten erhalten die Seiten eine einfache Nummerierung.

Export.AllInOneFile: Konfiguriert das Export-Resultat.

Wert	Bedeutung
0	Das Ergebnis sind n verschiedene XML-Dateien, für jede gedruckte Seite eine XML-Datei. Die Dateinamen werden (außer der *Export.File* Startdatei) fortlaufend durchnummeriert. Enthält der Dateiname der Startdatei den Formatidentifier "%d", so wird dieser durch die jeweilige Seitenzahl ersetzt.
1	Das Ergebnis ist eine einzige XML Datei (*Export.File)*, in der alle gedruckten Seiten aneinander hängen.
Default	1

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt kein Zielpfad-Auswahldialog (sofern *Export.Path* gesetzt ist) und keine "Überschreiben?" Rückfrage.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise ein Web-Browser gestartet wird
Default	0

Export.ShowResultAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahldialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

Export.OnlyTableData: Ermöglicht, dass nur die Daten aus Tabellenzellen exportiert werden.

Wert	Bedeutung
0	Alle Objekte werden exportiert
1	Nur Tabellenzellen werden mit Ihren Daten exportiert
Default	0

7.3.17 XPS Export-Modul

Übersicht

Das XPS Exportformat ist ab Windows Vista verfügbar, oder sobald das .NET Framework 3.0 oder höher auf dem Rechner installiert wurde. Das Exportmodul benutzt den dadurch installierten XPS Druckertreiber von Microsoft für die Ausgabe.

Auch hier sind einige Einschränkungen zu beachten, u.a. unterstützt der Treiber nicht alle Clippingmöglichkeiten des Windows GDI. Dadurch kann es in der XPS-Datei zu Darstellungsfehlern beim Export von Charts und ganz allgemein abgeschnittenen/geclippten Objekten kommen. Wir empfehlen Ihnen sicherheitshalber, die XPS-Ausgabe vor der Auslieferung an Ihre Kunden sorgfältig zu prüfen.

Programmierschnittstelle

Nachfolgend finden Sie eine Beschreibung aller vom XPS Export-Modul unterstützten Optionen. Diese Optionen können durch das Anwendungsprogramm über die Funktion *LIXSetParameter(...*"XPS"*...)* gesetzt und über *LIXGetParameter(...*"XPS"*...)* abgefragt werden.

Export.File: Gibt den Dateinamen für die zu generierende PRN-Datei an. Wenn leer, dann wird der Dateiauswahl-Dialog angezeigt.

Export.Path: Gibt den Pfad für die zu generierende PRN-Datei an.

Export.Quiet: Gibt an, ob der Exportvorgang mit Benutzerinteraktion durchgeführt werden soll.

Wert	Bedeutung
0	Interaktivität/Dialoge erlaubt
1	Es erfolgt kein Zielpfad-Auswahldialog (sofern *Export.Path* gesetzt ist) und keine "Überschreiben?" Rückfrage.
Default	0

Export.ShowResult: Spezifiziert, ob im Anschluss an den Export, die mit der Dateiendung verknüpfte Anwendung automatisch gestartet werden soll.

Wert	Bedeutung
0	Keine Anzeige des Ergebnisses
1	Führt ein *ShellExecute()* auf *Export.File* aus, so dass üblicherweise der XPS-Viewer gestartet werden sollte
Default	0

Export.ShowResultAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahldialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

7.4 Exportdateien digital signieren

7.4.1 Übersicht

In Zusammenarbeit mit den Signaturprodukten digiSeal® office und digiSeal® server der secrypt GmbH, OpenLimit® CC Sign der OpenLimit® SignCubes GmbH oder esiCAPI® der e.siqia technologies GmbH ist es möglich, mit List & Label erstellte Exportdateien für die Formate PDF, TXT (wenn die Option "***Export.AllInOneFile***" gesetzt ist) und Multi-TIFF digital zu signieren. Sie benötigen dafür neben der Software einen Kartenleser und eine Karte mit einem digitalen Zertifikat. Details zur Hard- und Software finden Sie in der Dokumentation zum jeweiligen Signaturprovider.

Im Lieferumfang von digiSeal® office findet sich die Datei digiSealAPI.dll bzw. die Datei dsServerAPI.dll in digiSeal® server, welche zusätzlich zusammen mit den redistributier-

bare Dateien von List & Label ausgeliefert werden muss (mindestens Windows NT wird hierfür voraussetzt). Ggf. muss zusätzlich noch die zur DLL passende Signatur-Datei (*.signatur) vorhanden sein. Beachten Sie bei der Verwendung des digiSeal® server, dass Sie auch ein Softwarezertifikat (*.pfx-Datei) benötigen. Details dazu können Sie direkt bei der secrypt GmbH beziehen.

Für OpenLimit® CC Sign der OpenLimit® SignCubes GmbH wird ebenfalls eine zusätzliche Koppel-DLL benötigt, welche aber gesondert von der OpenLimit® SignCubes GmbH zur Verfügung gestellt wird.

Im Lieferumfang von esiCAPI® der e.siqia technologies GmbH befinden sich die Dateien esicAPI.dll und esicConfig.xml, welche zusätzlich ausgeliefert werden müssen. Die DLL für den jeweiligen Konnektor wird direkt von e.siqia technologies GmbH bereitgestellt.

7.4.2 Signaturvorgang starten

Aktivieren Sie die Checkbox "Exportdateien digital signieren" im Auswahldialog für das Exportziel. Beachten Sie, dass Ihnen diese nur dann zur Verfügung steht, wenn Sie auch die benötigte Signatursoftware installiert haben.

Nach Erstellung der Ergebnisdatei wird der Signaturvorgang wie gewohnt gestartet. Bitte beachten Sie, dass sich durch die Signatur die Dateiendung der Ergebnisdatei ändern kann. Wenn der Signaturvorgang abgebrochen wird oder nicht erfolgreich durchgeführt werden kann, wird der Export (ggf. nach einer Anzeige des Fehlergrundes) ohne digitale Signatur fortgesetzt.

Aus rechtlichen Gründen ist eine Signatur im "Quiet"-Mode nicht möglich, d.h. es muss die PIN immer interaktiv eingegeben werden. Lediglich beim Produkt digiSeal® server 2 ist dies möglich, da es sich hierbei um eine Server-Komponente handelt und eine Massensignaturkarte vorausgesetzt wird.

7.4.3 Programmierschnittstelle

Der Signaturvorgang kann über eine Reihe von Parametern gesteuert werden.

Export.SignResult: Aktiviert die digitale Signatur der Exportdateien. Diese Option entspricht der Checkbox für den Endanwender "Exportdateien digital signieren". Der Wert wird nur dann ausgewertet, wenn eines der unterstützten Produkte auf dem Zielrechner installiert ist.

Wert	Bedeutung
0	Es erfolgt keine Signatur
1	Die Exportdateien werden digital signiert
Default	1

Export.SignResultAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahl-Dialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

Export.SignatureProvider: Hiermit kann die gewünschte Software für die digitale Signatur ausgewählt werden, falls mehrere Produkte auf einem System installiert sind.

Wert	Bedeutung
0	Default, d.h. keine explizite Auswahl des Produkts
1	Signatur über secrypt digiSeal® office
2	Signatur über OpenLimit® CC Sign Software
3	Signatur über esiCAPI® V 1.1 (vgl Export.SignatureProvider.Option)
4	Signatur über secrypt digiSeal® server 2
Default	0

Export.SignatureProvider.Option: Zusätzliche Option für den selektierten Signaturprovider unter Export.SignatureProvider.

Optionen für den Signaturprovider "esiCAPI":

Hiermit kann man den Konnektor bestimmen der verwendet wird. Informationen zu den einzelnen Konnektoren bekommen Sie direkt von e.siqia.

Wert	Bedeutung
0	Default, d.h. der erste verfügbare Konnektor wird verwendet
1	Signatur über SignLive! CC® 4.1.2
2	Signatur über SecSigner® 3.5.X
Default	0

Optionen für den Signaturprovider "digiSeal® server 2":

Diese Option hat lediglich einen Wert und enthält die Verbindungsdaten für den digiSeal® server 2. Die einzelnen Werte sind jeweils mit einem Pipe-Zeichen getrennt. Dabei gilt der folgende Aufbau:

<ServerHost>:<ServerPort>|<Dateipfad zum Softwarezertifikat für Identifizierung und Authentifizierung>|<Passwort für das Softwarezertifikat>

Beispiel:

localhost:2001|secrypt_Testzertifikat_D-TRUST_test.pfx|test

.NET-Komponente:

```
LL.ExportOptions.Add(LlExportOption.ExportSignatureProvider, "4");
LL.ExportOptions.Add(LlExportOption.ExportSignatureProviderOption,
"localhost:2001| secrypt_Testzertifikat_D-TRUST_test.pfx|test");
```

C++:

```
LlXSetParameter(hJob, LL_LLX_EXTENSIONTYPE_EXPORT, _T("PDF"),
_T("Export.SignaturProvider "), _T("4"));
LlXSetParameter(hJob, LL_LLX_EXTENSIONTYPE_EXPORT, _T("PDF"),
_T("Export.SignaturProvider.Option"),
_T("localhost:2001|secrypt_Testzertifikat_D-TRUST_test.pfx|test"));
```

Export.SignatureFormat: Hiermit kann das Signaturformat gewählt werden. Die verfügbaren Werte hängen vom Exportformat ab:

Wert	Bedeutung
pk7	Signatur im pk7-Format (Containerformat, das den Signaturgegenstand und die Signatur in einer Datei kapselt). Verfügbar für Multi-TIFF, TXT und PDF (letzteres nur für SignCubes). Die Ergebnisdatei hat nach der Signatur die Endung "pk7".
p7s	Signatur im p7s-Format. Dabei wird neben der Exportdatei eine zweite Datei *.p7s erstellt, die die Signatur enthält. Verfügbar für Multi-TIFF, TXT und PDF (letzteres nur für SignCubes). Bei einem Mailversand des Exportergebnisses werden beide Dateien angehängt.
p7m	Signatur im p7m-Format (Containerformat, das den Signaturgegenstand und die Signatur in einer Datei kapselt). Verfügbar für Multi-TIFF, TXT und PDF (letzteres nur für SignCubes). Die Ergebnisdatei hat nach der Signatur die Endung "p7m".
PDF	PDF-Signatur. Verfügbar für PDF und Multi-TIFF (nur für digiSeal® office und Schwarz-Weiß-TIFFs). Ein Multi-TIFF wird hierbei in eine PDF-Datei überführt und mit einem speziellen Barcode signiert, der eine Prüfung des Dokuments auch nach dem Ausdruck erlaubt.
Default	p7s für TXT und Multi-TIFF, PDF für PDF.

7.5 Exportdateien per eMail verschicken

7.5.1 Übersicht

Die durch einen Exportvorgang generierten Dateien können automatisch per eMail über die MAPI Schnittstelle, über Extended MAPI oder auch direkt über SMTP verschickt werden. Diese Funktion steht für alle Export-Module, außer für TTY- und Fax-Export zur Verfügung.

7.5.2 eMail Parameter per Programm setzen

Analog zu den übrigen Export-Optionen können auch die Parameter für das Verschicken per eMail gesetzt werden. Im Dialog unter Projekt > Einstellungen können vom Benutzer bereits einige Vorgaben gemacht werden, die automatisch berücksichtigt werden. Weitere Informationen hierzu finden Sie im Kapitel "Projekt-Parameter". Einige weitere Optionen können Sie über *LIXSetParameter(..."<Exportmodulname>"...)* / *LIXGetParameter(..."<Exportmodulname>"...)* setzen bzw. auslesen. Bitte beachten Sie, dass *<Exportmodulname>* auch ein Leerstring sein darf. In diesem Falle werden die Optionen an alle Exportmodule weitergereicht.

Export.SendAsMail: Aktiviert das Versenden der Exportdateien per eMail. Diese Option entspricht der Checkbox für den Endanwender "Exportdateien per eMail versenden".

Wert	Bedeutung
0	Es erfolgt kein Mailversand
1	Die Exportdateien werden per eMail verschickt
Default	0

Export.SendAsMailAvailable: Hiermit kann die entsprechende Auswahlbox im Dateiauswahl-Dialog versteckt werden.

Wert	Bedeutung
0	Auswahlbox versteckt
1	Benutzerauswahl möglich
Default	1

Export.Mail.Provider: Mit dieser Option kann der eMail-Provider für den Versand bestimmt werden. Für alle Versandarten außer Simple MAPI wird hierfür die Datei CMMX20.DLL benötigt.

Wert	Bedeutung
SMAPI	Simple MAPI
XMAPI	Extended MAPI
SMTP	SMTP
MSMAPI	Simple MAPI über Standard-MAPI-Client
Default	Der Default-Wert bestimmt sich aus den Systemeinstellungen bzw. den applikationsspezifischen Einstellungen (s.u.)

Wenn die DLL nicht gefunden werden kann, wird die eMail per Simple MAPI (MSMAPI) versandt.

Der Provider wird bestimmt, indem Sie entweder explizit die Option "Export.Mail.Provider" auf einen der obigen Typen setzen, oder indem Sie den Benutzer über *LsMailConfigurationDialog()* diese selbst bestimmen lassen.

Beim Versand wird erst versucht, die unter dem Applikationsnamen in der Registry gespeicherten Parameter zu finden. Diese können über *LsMailConfigurationDialog()* gewählt werden. Wenn Ihre Anwendung also den Mailversand unterstützt, sollten Sie Ihren Anwendern die Konfiguration durch einen Menüpunkt o.ä. ermöglichen, hinter dem Sie *LsMailConfigurationDialog()* aufrufen.

Export.Mail.To: Adressat der eMail.

Export.Mail.CC: CC-Adressat der eMail.

Export.Mail.BCC: BCC-Adressat der eMail.

Export.Mail.From: Absender der eMail.

Export.Mail.ReplyTo: Ziel für Antwort eMail, falls unterschiedlich zu "From" (nur bei SMTP).

Export.Mail.Subject: Betreff der eMail.

Export.Mail.Body: Nachrichtentext der eMail.

Export.Mail.Body:text/plain: Nachrichtentext im Plain-Text-Format der eMail. Identisch zu "Export.Mail.Body".

Export.Mail.Body:text/html: Nachrichtentext im HTML-Format der eMail.

Export.Mail.Body:application/RTF: Nachrichtentext im RTF-Format der eMail (nur bei XMAPI).

Export.Mail.AttachmentList: Ggf. neben dem Exportergebnis zusätzlich anzuhängende Dateien, durch Tabulator ("\t", ASCII-Code 9) getrennt.

Export.Mail.ShowDialog: Gibt an, ob ein Maildialog angezeigt werden soll.

Wert	Bedeutung
0	Die eMail wird direkt verschickt, ohne weitere Benutzer-Interaktion (mindestens ein Empfänger muss angegeben sein). Ist kein Empfänger angegeben, wird der Dialog immer angezeigt.
1	Es wird der entsprechend den anderen eMail-Optionen bereits ausgefüllte MAPI Standard-Maildialog angezeigt, der Benutzer muss das Versenden über den Dialog selbst auslösen
Default	0

Export.Mail.Format: Setzt die Voreinstellung für den Formatauswahl-Dialog in der Vorschau beim Versenden einer eMail. Mögliche Werte sind: "TIFF", "MULTITIFF", "LL", "XML", "XFDF", "XPS", "PDF", "JPEG", "TTY:<emulation>", "EMF",

Export.Mail.SendResultAs: Über diese Option können Sie beim HTML-Export bestimmen, dass das Exportresultat in den HTML-Text der eMail übernommen wird.

Wert	Bedeutung
text/html	Wenn SMTP als Versandprovider eingestellt wurde, wird das Exportresultat als HTML-Text der eMail verwendet. Wenn ein anderer Versandprovider gewählt wurde, wird diese Option ignoriert und das Exportergebnis als Dateianhang versendet.
Sonst/Leer	Das Exportergebnis wird als Dateianhang versendet.
Default	Leer

Export.Mail.SignResult:

Wert	Bedeutung
0	Die eMail wird nicht signiert.
1	Die eMail wird signiert.
Default	0

Speziell beim Versand per SMTP stehen noch weitere Optionen zur Verfügung. Diese brauchen in der Regel nicht explizit gesetzt werden, sondern können per Dialog applikationsspezifisch (s. *LsMailConfigurationDialog()*) bzw. global mit Hilfe des Systemsteuerungs-Applets vom Benutzer gewählt werden.

Export.Mail.SMTP.SocketTimeout: Socket-Timeout, in Millisekunden, Voreinstellung 1000.

Export.Mail.SMTP.LogonName: Anmeldename am Server, Voreinstellung: Computername (meist unwichtig).

Export.Mail.SMTP.SecureConnection: Verbindungssicherheit.

Wert	Bedeutung
-1	Automatisch (TLS nutzen, wenn der Server es anbietet)
0	TLS ausschalten (auch wenn es vom Server unterstützt wird)
1	SSL erzwingen (Abbruch, wenn der Server kein SSL anbietet)
2	TLS erzwingen (Abbruch, wenn der Server kein TLS anbietet)
Default	-1

Export.Mail.SMTP.ServerAddress: IP-Adresse oder URL des SMTP-Servers

Export.Mail.SMTP.ServerPort: Port des SMTP-Servers, Voreinstellung 25.

Export.Mail.SMTP.ServerUser: Benutzerkennung für die Authentifizierung am SMTP-Server (falls nötig)

Export.Mail.SMTP.ServerPassword: Passwort für die Authentifizierung am SMTP-Server (falls nötig)

Export.Mail.SMTP.ProxyType: Typ des Proxy (0=keiner, 1=Socks4, 2=Socks5)

Export.Mail.SMTP.ProxyAddress: IP-Adresse oder URL des Proxy

Export.Mail.SMTP.ProxyPort: Port des Proxy, Voreinstellung 1080

Export.Mail.SMTP.ProxyUser: Benutzerkennung für die Authentifizierung am Proxy (nur Socks5)

Export.Mail.SMTP.ProxyPassword: Passwort für die Authentifizierung am Proxy (nur Socks5)

Export.Mail.SMTP.POPBeforeSMTP: Manche SMTP Server erfordern eine vorherige Anmeldung per POP, damit die Verbindung nicht abgebrochen wird (0=keine POP Verbindung wird vorher aufgebaut, 1=POP Verbindung wird vorher aufgebaut)

Export.Mail.SMTP.SenderAddress: Adresse des eMailversenders (xyz@abc.def) – wird auch für das SMTP-Protokoll verwendet

Export.Mail.SMTP.SenderName: Klartext-Name des eMailabsenders

Export.Mail.SMTP.ReplyTo: Rücksendeadresse (optional)

Export.Mail.SMTP.From: ersetzt die Absendeadresse (aus "Export.Mail.SMTP.SenderName" und "Export.Mail.SMTP.SenderAddress" gebildet) in der eMail durch diesen Parameter. "Export.Mail.SMTP.SenderAddress" wird aber weiterhin für das SMTP-Protokoll verwendet.

Export.POP3.SocketTimeout: Timeout für Socket-Connection in ms, Voreinstellung: 10000

Export.POP3.SecureConnection: Verbindungssicherheit.

Wert	Bedeutung
-1	Automatisch (TLS nutzen, wenn der Server es anbietet)
0	TLS ausschalten (auch wenn es vom Server unterstützt wird)
1	SSL erzwingen (Abbruch, wenn der Server kein SSL anbietet)
2	TLS erzwingen (Abbruch, wenn der Server kein TLS anbietet)
Default	-1

Export.POP3.SenderDomain: Anmeldedomain am Server, Voreinstellung: Computername

Export.POP3.ServerPort: Voreinstellung: 110

Export.POP3.ServerAddress: URL/IP-Adresse des POP3-Servers, Voreinstellung: "localhost"

Export.POP3.ServerUser: Benutzername zur Authentifizierung

Export.POP3.ServerPassword: Passwort zur Authentifizierung

Export.POP3.ProxyAddress: Proxy-Server-Adresse

Export.POP3.ProxyPort: Proxy-Server Port, Voreinstellung 1080

Export.POP3.ProxyUser: Proxy-Server Benutzername

Export.POP3.ProxyPassword: Proxy-Server Passwort

Export.XMAPI.ServerUser: Profilname zur Authentifizierung

Export.XMAPI.ServerPassword: Passwort zur Authentifizierung

Export.XMAPI.SuppressLogonFailure: "0" / "1" keinen Dialog anzeigen bei Anmeldefehler

Export.XMAPI.DeleteAfterSend: "0" / "1" Mail nach Versand löschen

Beispiel:

```
LlXSetParameter(hJob,LL_LLX_EXTENSIONTYPE_EXPORT,"","Export.SendAsMail","1");
```

Dadurch wird das Exportergebnis automatisch per eMail an den unter Projekt > Einstellungen gewählten Empfänger versendet. Dafür werden die global vorgenommenen Maileinstellungen verwendet. Wenn Sie dem Benutzer einen Wert vorgeben möchten, so können Sie diesen z.B. über

```
LlSetDefaultProjectParameter(hJob,"LL.Mail.To", "EMAIL", 0);
```

vorgeben, wenn Ihr Datenbankfeld mit der eMail-Adresse den Namen EMAIL trägt. Wollen Sie eine konkrete Adresse vorgeben, so beachten Sie bitte, dass Sie diese mit Anführungszeichen umgeben müssen, da der an List & Label übergebene Wert als Formel interpretiert wird:

```
LlSetDefaultProjectParameter(hJob,"LL.Mail.To", "'abc@xyz.de'", 0);
```

7.6 Exportdateien in ZIP-Archiv komprimieren

Sollen z.B. die Ergebnisse eines Bild- oder HTML-Exports per Mail verschickt werden, so ist es häufig praktischer, das gesamte Ergebnis des Exports als ZIP-Archiv zu versenden. Alle Exportformate unterstützen eine Programmierschnittstelle zu diesem Zweck. Die Kompression kann entweder interaktiv durch den Benutzer im Dialog aktiviert werden, indem er aus der Liste der verfügbaren Dateifilter den Filter "ZIP-Archiv (*.zip)" auswählt.

Alternativ kann die Ausgabe natürlich auch vollständig per Code gesteuert werden. Folgende Optionen stehen zur Verfügung:

Export.SaveAsZIP: Aktiviert das Komprimieren der Exportdateien. Wenn diese Option gesetzt ist, wird der ZIP-Filter im Dialog vorausgewählt.

Wert	Bedeutung
0	Es erfolgt keine Kompression
1	Die Exportdateien werden in ein ZIP-Archiv komprimiert
Default	0

Beachten Sie, dass der Benutzer die hier voreingestellte Auswahl im Dialog wieder verändern kann. Wenn Sie dies nicht wünschen, setzen Sie die Option "Export.Quiet" auf "1".

Export.SaveAsZIPAvailable: Hiermit kann der ZIP-Archiv-Filter im Dateiauswahl-Dialog versteckt werden.

Wert	Bedeutung
0	Filter versteckt
1	Benutzerauswahl möglich
Default	1

Export.ZIPFile: (Default-)Name der zu erstellenden ZIP-Datei, z.B. "export.zip". Für den Namen der Dateien im ZIP-Archiv gelten folgende Regeln:

- wenn per "Export.File" kein Name vorgegeben wurde, wird der Name des ZIP-Archivs mit angepasster Endung verwendet (z.B. "export.htm")
- wenn per "Export.File" ein Name vergeben wurde, so wird dieser verwendet. Dabei kann bei den Formaten, die pro Seite eine eigene Datei erzeugen, auch der Platzhalter "%d" für die Seitenzahl verwendet werden, z.B. "Rechnung Seite %d.bmp" im Bitmapexporter

Export.ZIPPath: Pfad der zu erstellenden ZIP-Datei

8. Sonstiges zur Programmierung

8.1 Übergabe von NULL-Werten

NULL-Werte sind Daten, für die keine Inhalte existieren, etwa ein Lieferdatum für eine Lieferung, die noch nicht erfolgt ist. Die meisten Datenbanktreiber erlauben die Abfrage des Feldinhaltes auf NULL.

List & Label behandelt standardmäßig NULL-Werte entsprechend der SQL-92 Spezifikation. Eine wichtige Konsequenz daraus, dass Funktionen und Operatoren, die NULL-Werte als Parameter bzw. Operanden bekommen in der Regel als Ergebnis auch wieder NULL zurückliefern. Ein Beispiel dafür ist die folgende Designerformel:

```
Titel+" "+Vorname+" "+Nachname
```

Wenn der Titel mit NULL gefüllt ist, ist das Ergebnis dieser Formel entsprechend dem Standard ebenfalls NULL. Falls dies nicht gewünscht ist, verwenden Sie die Option *LL_OPTION_NULL_IS_NONDESTRUCTIVE* (siehe dort).

Grundsätzlich handhaben die List & Label Komponenten NULL-Werte aus Datenbanken automatisch, Sie können aber auch jederzeit NULL-Werte explizit an List & Label übergeben. Übergeben Sie in diesem Falle als Inhalt des Feldes bzw. der Variablen die Zeichenkette "(NULL)" an List & Label. Diese Möglichkeit steht Ihnen für alle Datentypen zur Verfügung.

8.2 Rundung

Bitte beachten Sie folgenden wichtigen Hinweis zur Rundung bei List & Label, damit Sie keine Inkonsistenzen zu Ihren Daten aus der Datenbank bekommen: Summenvariablen werden nicht gerundet, d.h. bei Verwendung von Nachkommastellen (z.B. bei Rechnungen) empfehlen wir die Verwendung einer Rundungsfunktion, oder am besten sollten keine Multiplikationen für die Erstellung von Summenvariablen verwendet werden.

8.3 Geschwindigkeitsoptimierung

Die Standardeinstellungen von List & Label stellen einen sinnvollen Kompromiss zwischen Dateigrößen und Performance dar. Sie können aber durch Veränderung der folgenden Optionen bzw. Beachten der folgenden Hinweise Performancesteigerungen in kritischen Anwendungen erreichen:

- Sorgen Sie dafür, dass immer mindestens ein Job geöffnet bleibt. Somit ist sichergestellt, dass nicht immer wieder alle DLLs in den Speicher geladen und anschließend entladen werden.

- Bei Druck auf die Vorschau: schalten Sie die Komprimierung aus (s. *LL_OPTION_COMPRESSSTORAGE*). Dann werden die Vorschaudateien allerdings unter Umständen deutlich größer.
- Aktivieren Sie die Option *LL_OPTION_VARSCASESENSITIVE*. Beachten Sie, dass dann die Groß-/Kleinschreibung der Variablen und Felder in Ihren Projekten eingehalten werden muss. Dies kann bestehende Projekte unbenutzbar machen!
- Verzichten Sie wo möglich auf die Verwendung von RTF- und HTML-Texten und benutzen Sie stattdessen das "normale" Textobjekt.

8.4 Projekt-Parameter

List & Label ermöglicht es, projektspezifische Parameter zu setzen, den Benutzer gegebenenfalls einstellen zu lassen und diese beim Druck wieder abzufragen. Ein Beispiel wäre z.B., dem Anwender die Erstellung eines SELECT-Statements innerhalb des Designers zu ermöglichen, so dass eine Filterung auf SQL-Basis direkt im Designer vorgenommen werden kann.

List & Label selbst verwendet diese Parameter, um die wichtigen Einstellungen für Fax und Mails zu setzen. Genauso kann aber ein Benutzerprogramm dadurch Informationen in einem Projekt speichern, um diese später wieder abfragen zu können – bei Bedarf sogar berechnet.

8.4.1 Parametertypen

Es gibt verschiedene Typen von Parametern, die sich durch den Übergabeparameter nFlags bei *LlSetDefaultProjectParameter()* unterscheiden. Jeweils eins der nachfolgenden drei Flag-Alternativen muss angegeben werden:

LL_PARAMETERFLAG_FORMULA	Der Parameter ist eine Formel, die beim Druck ausgewertet wird. Über *LlPrintGetProjectParameter()* bekommt man den berechneten Wert zurück (Voreinstellung).
LL_PARAMETERFLAG_VALUE	Der Parameter ist ein fester Wert. Über *LlPrintGetProjectParameter()* bekommt man diesen Wert unverändert zurück (Voreinstellung).
LL_PARAMETERFLAG_PUBLIC	Der Parameter erscheint im Designer, wenn kein Objekt selektiert ist. Der Benutzer kann die Formel bzw. den Wert ändern (Voreinstellung).
LL_PARAMETERFLAG_PRIVATE	Der Parameter kann durch den Benutzer (d.h. im Designer) nicht geändert werden.
LL_PARAMETERFLAG_GLOBAL	Der Parameter wird in die Druck-Projekt-Parameterliste übernommen und ggf. in der Vorschaudatei gespeichert und kann daraus wieder

	abgerufen werden (*LlStgsysGetJobOptionStringEx()*) (Voreinstellung).
LL_PARAMETERFLAG_LOCAL	Der Parameter wird nicht in die Druck-Projekt-Parameterliste übernommen und in der Vorschaudatei gespeichert, da er nur für den lokalen Benutzer oder Rechner gültig ist. Diese Werte existieren nur in der DefaultProjectParameter-Liste, sind also eine Art lokale Variable (die aber auch an die über *LlPreviewDisplay()* angezeigte Vorschau übergeben werden, da diese als "lokal" angesehen werden kann) (Voreinstellung).

Wenn die Parameter über *LlSetDefaultProjectParameter()* definiert werden, sind das die Voreinstellungen, die der Benutzer gegebenenfalls (wenn *LL_PARAMETERFLAG_PUBLIC* gesetzt ist) seinen Bedürfnissen entsprechend einstellen kann.

Wird das Projekt dann geladen (*LlDefineLayout()*, *LlPrint[WithBox]Start()*), werden die Parameter durch die im Projekt gespeicherten Formeln oder Werte ersetzt, d.h. überschrieben. Um dies zu verdeutlichen, heißt die API "DefaultParameters". Ungeänderte Parameter werden im Projektlayoutfile nicht gespeichert, so dass spätere Änderungen der Default-Parameter für den Druck übernommen werden. Ist dies nicht gewünscht, so können Sie zusätzlich das Flag *LL_PARAMETERFLAG_SAVEDEFAULT* mit angeben, damit der Default-Wert im Projektfile gespeichert wird. Dies ist insbesondere nützlich, um vom Benutzer im Designer gesetzte Projektparameter vor dem Druck per *LlGetProjectParameter()* auszulesen.

Es darf nicht mehrere Parameter mit gleichem Namen, aber verschiedenem Typ geben! Der Typ wird durch *LlSetDefaultProjectParameter()* festgelegt.

8.4.2 Parameterabfrage während des Druckvorgangs

Nach dem Start des Drucks über *LlPrint[WithBox]Start()* sind die für den Druck verbindlichen Werte über *LlPrintGetProjectParameter()* abfragbar, entweder als Formel oder als berechneter Wert.

Man kann auch über *LlPrintSetProjectParameter()* den Wert noch einmal ändern oder sogar neue Parameter hinzufügen. Dies macht dann Sinn, wenn man die Werte später noch braucht, denn sie werden – nach einer eventuellen Formelevaluierung - in der Vorschaudatei gespeichert und können über *LlStgsysGetJobOptionStringEx()* abgefragt werden. Somit kann man eigene Parameter dauerhaft darin speichern. Dort beginnen die Parameter mit "ProjectParameter." vor dem definierten Namen.

8.4.3 Vordefinierte Projektparameter

List & Label verwendet Projektparameter zum Versand per Fax oder eMail. Der Benutzer kann die Parameter übernehmen oder abändern, z.B. den Fax- oder eMail-Empfänger über eine Formel, die von der Applikation angebotene Variablen enthält, übernehmen.

Da List & Label bei der Übergabe der Fax- bzw. Mail-Parameter eine Formel erwartet, kann es notwendig sein, den übergebenen Wert als Zeichenkette zu maskieren (dann, wenn ein fester Wert übergeben werden soll).

Beispiel:

```
LlPrintSetProjectParameter(hJob, "LL.FAX.RecipNumber","\"+497531906018\"",0);
```

LL.FAX.Queue	LOCAL, PRIVATE
LL.FAX.RecipNumber	GLOBAL, PUBLIC [LL_OPTIONSTR_FAX_RECIPNUMBER]
LL.FAX.RecipName	GLOBAL, PUBLIC [LL_OPTIONSTR_FAX_RECIPNAME]
LL.FAX.SenderName	GLOBAL, PRIVATE [LL_OPTIONSTR_FAX_SENDERNAME]
LL.FAX.SenderCompany	GLOBAL, PRIVATE [LL_OPTIONSTR_FAX_SENDERCOMPANY]
LL.FAX.SenderDepartment	GLOBAL, PRIVATE [LL_OPTIONSTR_FAX_SENDERDEPT]
LL.FAX.SenderBillingCode	GLOBAL, PRIVATE [LL_OPTIONSTR_FAX_SENDERBILLINGCODE]
LL.MinPageCount	GLOBAL, FORMULA, PUBLIC
LL.ProjectDescription	GLOBAL, VALUE, PUBLIC
LL.IssueCount	GLOBAL, FORMULA, PUBLIC
LL.PageCondition	GLOBAL, FORMULA, PUBLIC
LL.PrintJobLCID	GLOBAL, FORMULA, PUBLIC

Eine Beschreibung der Parameter finden Sie im Designerhandbuch.

Analog zu den LL.FAX Parametern existieren auch LL.MAIL Parameter, vgl. Kapitel "eMail Parameter per Programm setzen".

Parameter, die von der Applikation vor *LlDefineLayout()* nicht definiert werden, oder mit dem LOCAL-Flag definiert wurden, sind nicht vom Benutzer einstellbar.

Beispielsweise macht es bei Adressdaten Sinn, wenn die Applikation "LL.MAIL.To" über die eMail-Adressen-Variable (hier "EMAIL") aus der Datenbank definiert:

```
LlSetDefaultProjectParameter(hJob,"LL.MAIL.To","EMAIL",0);
```

Der Benutzer kann dann über die (für dieses Beispiel angenommenen) Felder "VORNAME" und "NAME" den Empfänger abändern zu

VORNAME + " " + NAME + " <" + EMAIL + ">"

um die Adresse zu "verschönern".

Das Vorschau-Control übernimmt automatisch die Werte aus LL.FAX.* und LL.MAIL.*. Zudem werden die Werte auch an die Export-Module weitergegeben, so dass auch diese die gewählten Einstellungen berücksichtigen.

8.4.4 Automatische Formularspeicherung

Über die Projektparameter ist es auch möglich bei Verwendung von Formularelementen (siehe entsprechendes Kapitel im Designer Handbuch) eine automatische Speicherung bei Beenden der Vorschau zu realisieren. Neben der automatischen Formularspeicherung können diese Parameter auch verwendet werden, um den Dateinamen für den eMail-Versand aus der Vorschau vorzugeben und die Standardeinstellungen für das Speichern aus der Vorschau vorzugeben. Hierzu können Sie folgende Projektparameter verwenden:

SaveAs.Format	Gewünschtes Ausgabeformat, z.B. "XML" Unterstützt werden die Formate "TTY", "PDF", "EMF", "XPS", "PRN", "TIFF" bzw. "PICTURE_MULTITIFF", "JPEG" bzw. "PICTURE_JPEG", "PNG" bzw. "PICTURE_PNG", "LL" bzw. "PRV", "XML" bzw. "XFDF". Weitere Hinweise zu diesen Formaten finden Sie im Kapitel "Die Export-Module".
SaveAs.Filename	Ausgabedateiname, z.B. "test.xml". Die Dateiendung ist irrelevant und wird automatisch durch das SaveAs.Format bestimmt.
SaveAs.ShowDialog	Hierüber kann der Speichern-Dialog ein- ("1") bzw. ausgeschaltet werden ("0").
SaveAs.NoSaveQuery	Unterdrückt die Abfrage, ob die Datei beim Schließen gespeichert werden soll oder nicht.

Beispiel:

```
LlPrintSetProjectParameter(hJob,"SaveAs.Format","XML",
        LL_PARAMETERFLAG_VALUE);
LlPrintSetProjectParameter(hJob,"SaveAs.Filename","test.xml",
        LL_PARAMETERFLAG_VALUE);
LlPrintSetProjectParameter(hJob,"SaveAs.ShowDialog","0",
        LL_PARAMETERFLAG_VALUE);
LlPrintSetProjectParameter(hJob,"SaveAs.NoSaveQuery","1",
        LL_PARAMETERFLAG_VALUE);
```

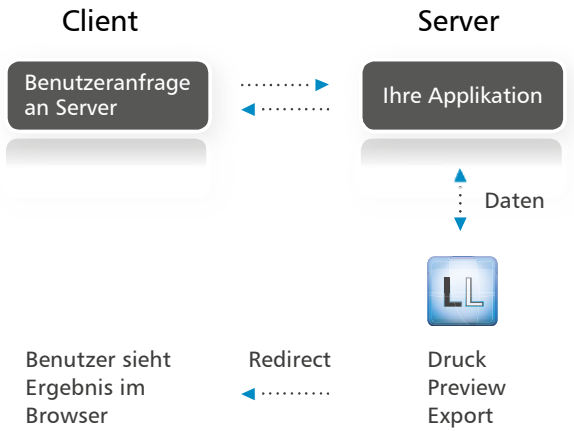
8.5 Webreporting

Die Verwendung von List & Label unter ASP.NET ist ausführlich im Kapitel "Webreporting" beschrieben. Natürlich können Sie auch andere Programmiersprachen für Webreporting verwenden.

Ganz allgemein gelten hierfür die folgenden Voraussetzungen:

- Der Server muss ein Windows-System sein, List & Label kann nur auf Windows-Plattformen eingesetzt werden. Diese Einschränkung gilt natürlich nicht für die Clients.
- Der Benutzer-Account, unter dem die Webanwendung läuft, benötigt Zugriff auf einen Druckertreiber. Der zugehörige Drucker muss nicht physikalisch auf dem System vorhanden sein, es reicht die bloße Treiberinstallation. Zudem muss sichergestellt sein, dass der verwendete Benutzer-Account die List & Label-DLLs laden kann, d.h. dass Rechte für den Pfad der DLLs vergeben wurden.
- Die eigentliche Webanwendung führt dann auf Benutzeraktion hin einen stummen Export durch (siehe Kapitel "Export ohne Benutzerinteraktion durchführen") und lenkt den Client z.B. durch einen Redirect auf die erzeugte Exportdatei.

Folgende Grafik veranschaulicht das Funktionsprinzip:



Beachten Sie bitte insbesondere auch für den Betrieb von List & Label auf einem Webserver die Lizenzbedingungen.

8.6 Hinweise zur Verwendung in mehreren Threads (Multithreading)

List & Label kann in mehreren Threads gleichzeitig verwendet werden. Auf diese Weise können umfangreichere Druck-Aufgaben z.B. auf mehrere Prozessoren/Cores verteilt werden. Intern wird von diesen Möglichkeiten rege Gebrauch gemacht, z.B. bei der Druckvorschau im Designer oder beim DrillDown.

Bei der Verwendung in Multithread-Umgebungen sind einige Punkte zu berücksichtigen:

- Achten Sie darauf, dass ein List & Label Job (bzw. eine Komponenteninstanz) immer nur innerhalb des gleichen Threads verwendet wird. Die Erzeugung, Verwendung und Zerstörung des Jobs / der Komponente muss also im gleichen Thread stattfinden. Wenn Sie mehrere parallele Druckthreads starten möchten, muss also jeder dieser Druckthreads seinen eigenen Job öffnen und schließen. Hintergrund: Windows-GDI-Ressourcen wie Fensterhandles, Drucker-Devicekontexte etc. können nicht aus verschiedenen Threads verwendet werden.

- Stellen Sie sicher, dass Sie vor dem Start des ersten Threads einen sogenannten "Schutzjob" öffnen (bzw. eine Komponenteninstanz erzeugen) und diesen erst nach dem Beenden des letzten Threads wieder schließen. Typischerweise wird Ihre Applikation beim Start diesen Job erzeugen und beim Beenden wieder zerstören. Hintergrund: der erste Job erzeugt einige Hilfsobjekte, die im gleichen Thread wieder zerstört werden müssen. Zudem kann die Performance auf diese Weise deutlich verbessert werden, da ein häufiges Laden und Entladen der List & Label DLLs vermieden wird.

- Threads, die den Designer anzeigen, müssen das Single Threaded Apartment-Modell (STA) verwenden. Es dürfen also keine .NET Worker Threads aus dem Threadpool sein, da diese das Multi Threaded Apartment-Modell (MTA) verwenden. Hintergrund: für die Drag & Drop-Unterstützung im Designer muss OleInitialize() aufgerufen werden, was STA zwingend benötigt.

9. Fehlercodes

9.1 Allgemeine Fehlercodes

Anbei finden Sie die verwendeten Fehlercode-Konstanten. Die Konstanten beginnen alle mit *LL_ERR_*, z.B. entspricht der Tabelleneintrag *BAD_JOBHANDLE* der Konstante *LL_ERR_BAD_JOBHANDLE*. Die Werte in Klammern sind die Dezimalangaben, die auch bei Debug-Ausgaben erscheinen.

Wert	Bedeutung
BAD_JOBHANDLE (-1)	Es wurde eine Funktion mit einem Jobhandle als Parameter aufgerufen, das nicht mit *LlJobOpen*() erzeugt wurde, bzw. der Job wurde schon geschlossen.
TASK_ACTIVE (-2)	Pro Applikation darf nur ein Designerfenster geöffnet sein, Sie haben versucht, ein zweites zu öffnen
BAD_OBJECTTYPE (-3)	Einer Funktion, die den Objekttyp als Parameter benötigt, wurde ein ungültiger Typ übergeben. Gültige Typen: *LL_PROJECT_LABEL, LL_PROJECT_LIST, LL_PROJECT_CARD*
PRINTING_JOB (-4)	Es wurde eine Druckfunktion aufgerufen, obwohl noch kein Druckjob gestartet wurde.
NO_BOX (-5)	*LlPrintSetBoxText()* wurde aufgerufen, obwohl der Druckjob nicht mit *LlPrintWithBoxStart()* geöffnet wurde.
ALREADY_PRINTING (-6)	Die derzeitige Operation kann nicht durchgeführt werden, solange ein Druckjob offen ist.
NOT_YET_PRINTING (-7)	*LlPrint[G|S]etOption[String](), LlPrintResetProjectState()*. Der Druckjob ist noch nicht gestartet
NO_PROJECT (-10)	*LlPrint[WithBox]Start()*: Es existiert kein Objekt mit dem angegebenen Dateinamen. Identisch mit *LL_ERR_NO_OBJECT*
NO_PRINTER (-11)	*LlPrint[WithBox]Start()*: Druckjob konnte nicht gestartet werden, da kein Drucker-Device geöffnet werden konnte. Sie müssen zumindest einen Druckertreiber auf dem System installieren.
PRINTING (-12)	Während des Druckens trat ein Fehler auf. Häufigste Ursache: Druckspooler voll, bzw. der vom Druckspooler benötigte Platz ist auf dem Laufwerk auf das TEMP zeigt nicht mehr vorhanden (Pro Seite kann je nach Druckauflösung und verwendeter Grafik ein Platzbedarf von einigen MB entstehen. Abhilfe schafft meist auch die Einstellung des Direktdrucks ohne Spooler). Mögliche Ursache bei Direktdruck: allg. Druckerfehler, Papierstau, etc.

EXPORTING (-13)	Beim Exportieren ist ein Fehler aufgetreten (z.B. keine Zugriffsrechte auf Zielpfad, zu exportierende Datei schon vorhanden und schreibgeschützt,...)
NEEDS_VB (-14)	Diese DLL-Version benötigt Visual Basic.
BAD_PRINTER (-15)	Bei Druckoptionen: kein Drucker verfügbar.
NO_PREVIEWMODE (-16)	Preview-Funktionen: bei *LIPrint[WithBox]Start()* wurde kein Preview-Mode eingestellt.
NO_PREVIEWFILES (-17)	*LIPreviewDisplay()*: Keine Preview-Dateien gefunden.
PARAMETER (-18)	NULL Zeiger als Parameter ist hier nicht gestattet, möglicherweise auch andere Parameter-Fehler. Bitte benutzen Sie den Debug-Modus zur Bestimmung des Fehlers.
BAD_EXPRESSION (-19)	Neuer Expression-Modus: Ein Ausdruck in *LIExprEvaluate()* konnte nicht interpretiert werden.
BAD_EXPRMODE (-20)	Unbekannter Ausdrucks-Modus in *LISetOption()*.
CFGNOTFOUND (-22)	*LIPrint[WithBox]Start()*: Projektdatei wurde nicht gefunden.
EXPRESSION (-23)	*LIPrint[WithBox]Start()/LIDefineLayout()*: Einer der verwendeten Ausdrücke hat einen Fehler. Beim Designstart werden die Fehler interaktiv angezeigt, beim Druckstart finden sich weitere Informationen im Debug-Protokoll. *LIExprEval():* Verwenden Sie *LIExprError()*, um den Fehler zu finden.
CFGBADFILE (-24)	*LIPrint[WithBox]Start()*: Projektdatei hat falsches Format oder ist defekt.
BADOBJNAME (-25)	*LIPrintEnableObject()*: Der Objektname ist nicht korrekt.
UNKNOWNOBJECT (-27)	*LIPrintEnableObject()*: Es existiert kein Objekt mit diesem Objektnamen.
NO_TABLEOBJECT (-28)	wird von *LIPrintStart()* und *LIPrintWithBoxStart()* zurückgegeben, wenn ein Listen-Projekt gestartet werden soll, das kein Listenobjekt enthält. Wird nur im neuen Expression-Modus zurückgegeben.
NO_OBJECT (-29)	*LIPrint[WithBox]Start()*: Das Projekt besitzt keine Objekte, und leere Seiten kann man auch anders drucken!
NO_TEXTOBJECT (-30)	*LIPrintGetTextCharsPrinted()*: Kein Textobjekt in diesem Projekt.
UNKNOWN (-31)	*LIPrintIsVariableUsed(), LIPrintIsFieldUsed()*: Die angegebene Variable gibt es nicht. *LIGetUsedIdentifiers()*: Das Projekt wurde noch nicht mit List & Label 11 oder neuer gespeichert und enthält daher keine Informationen über verwendete Variablen und Felder.

BAD_MODE (-32)	Feld-Funktionen wurden benutzt, obgleich das Projekt kein Tabellenprojekt ist.
CFGBADMODE (-33)	*LIPrint[WithBox]Start(), LIDefineLayout()*: Der Ausdruck-Modus der Projektdatei ist der neue Modus, eingestellt ist jedoch der alte Modus (siehe *LISetOption()*).
ONLYWITHONETABLE (-34)	Funktion ist nur anwendbar, wenn der OneTable-Modus gewählt wurde (*LL_OPTION_ ONLYONETABLE*) (siehe *LISetOption()*).
UNKNOWNVARIABLE (-35)	Die bei *LIGetVariableType()* oder *LIGetVariableContents()* angegebene Variable wurde nicht definiert.
UNKNOWNFIELD (-36)	Das bei *LIGetFieldType()* oder *LIGetFieldContents()* angegebene Feld wurde nicht definiert.
UNKNOWNSORTORDER (-37)	Die über die ID bei den Gruppierungs-Funktionen angegebene Sortierreihenfolge wurde nicht definiert.
NOPRINTERCFG (-38)	*LIPrintCopyPrinterConfiguration()*: Datei wurde nicht gefunden oder hat falsches Format
SAVEPRINTERCFG (-39)	*LIPrintCopyPrinterConfiguration()*: Datei konnte nicht geschrieben werden: Problem mit Festplattenplatz oder Zugriffsrechten
RESERVED (-40)	Reserviert
NOVALIDPAGES (-41)	Die Storage-Datei enthält keine gültigen Seiten
NOTINHOSTPRINTERMODE (-42)	Dieser Befehl kann nicht im *HOSTPRINTER*-Modus aufgerufen werden (z.B. *LISetPrinterInPrinterFile()*)
NOTFINISHED(-43)	Ein oder mehrere Objekte sind noch nicht fertig gedruckt
BUFFERTOOSMALL(-44)	*LI[G|S]etOptionString(), LIPrint[G|S]etOptionString()*, ...: Ein übergebener Puffer ist nicht groß genug für die darin zu speichernden Daten.
BADCODEPAGE (-45)	*LL_OPTION_CODEPAGE*: Die Codepage ist nicht gültig (NLS nicht auf dem System installiert).
CANNOTCREATETEMPFILE(-46)	Eine Temporärdatei konnte nicht erzeugt werden (falscher Temp-Pfad!)
NODESTINATION(-47)	List & Label hat kein gültiges Ausgabemedium beim Start des Drucks (siehe *LL_OPTIONSTRING_EXPORTS_ALLOWED*)
NOCHART(-48)	*LIPrintDeclareChartRow()*: Kein Chartobjekt im Projekt vorhanden.
TOO_MANY_CONCURRENT_PRINTJOBS (-49)	Kann nur in Server/Webserverapplikationen auftreten. Die Anzahl der Benutzer übersteigt die lizensierte Benutzeranzahl.
BAD_WEBSERVER_LICENSE (-50)	Kann nur in Server/Webserverapplikationen auftreten. Die Webserver-Lizenzdatei (*.wsl) ist ungültig oder beschädigt.

NO_WEBSERVER_LICENSE (-51)	Kann nur in Server/Webserverapplikationen auftreten. Die Webserver-Lizenzdatei (*.wsl) wurde nicht gefunden.
ERR_INVALIDDATE (-52)	*LISystemTimeFromLocaleString():* Ein ungültiges Datumsformat wurde verwendet
DRAWINGNOTFOUND(-53)	Eine benötigte Grafikdatei wurde nicht gefunden. S. LL_OPTION_ERR_ON_FILENOTFOUND
ERR_NOUSERINTERACTION (-54)	Ein Aufruf würde eine Benutzerinteraktion benötigen, List & Label läuft aber auf einem Webserver
ERR_BADDATABASESTRUCTURE (-55)	Die Datenbankstruktur, die für das Design verwendet wurde stimmt nicht mit der zur Druckzeit überein
ERR_UNKNOWNPROPERTY (-56)	Die Eigenschaft ist für das angegebene Objekt nicht verfügbar
ERR_CFGFOUND (-59)	*LIProjectOpen():* Das gewählte Projekt ist bereits vorhanden oder ist schreibgeschützt.
ERR_SAVECFG (-60)	*LIProjectSave():*Fehler beim Speichern der Projektdatei
USER_ABORTED (-99)	Der Benutzer unterbrach den Ausdruck
BAD_DLLS (-100)	Die von List & Label benötigten DLLs sind nicht auf dem benötigten Stand.
NO_LANG_DLL (-101)	Die benötigte Sprach-DLL wurde nicht gefunden, und auch CMLL20@@.LNG ist nicht vorhanden.
NO_MEMORY (-102)	Zu wenig freier Speicher
EXCEPTION (-104)	Ein GPF ist innerhalb der Funktion aufgetreten. List & Label könnte bei der weiteren Ausführungen instabil sein
LICENSEVIOLATION (-105)	Es wurde versucht, eine Funktion aufzurufen, die durch den Lizenzumfang nicht gedeckt ist oder aber eine falsche Lizenzinformation wurde mit der Option *LL_OPTIONSTR_LICENSEINFO* übergeben.
LL_WRN_TABLECHANGE (-996)	In einem hierarchischen Layout ändert sich der Tabellenname. siehe Kapitel "Drucken relationaler Daten"
LL_WRN_PRINTFINISHED (-997)	Rückgabewert bei *LIRTFDisplay()*: keine weiteren Daten mehr zu drucken
LL_WRN_REPEAT_DATA (-998)	Dies ist "nur" ein Hinweis: momentaner Datensatz passte nicht mehr auf die Seite. Dieser Rückgabewert wird benötigt, um zu melden, dass die Seitenzahl auf den neuesten Stand gebracht werden muss und der momentane Datensatz erneut geschickt werden muss.

9.2 Zusätzliche Fehlercodes der Storage-API

Wert	Bedeutung
STG_NOSTORAGE (-1000)	Die Datei ist keine List & Label-Preview-Datei
STG_BADVERSION (-1001)	Die Preview-Datei hat eine inkompatible Versionsnummer
STG_READ (-1002)	Fehler beim Lesen der Preview-Datei
STG_WRITE (-1003)	Fehler beim Schreiben der Preview-Datei
STG_UNKNOWNSYSTEM (-1004)	Unbekanntes Dateiformat für das Storage System
STG_BADHANDLE (-1005)	Falscher Parameter (Metafile-Handle ist ungültig)
STG_ENDOFLIST (-1006)	*LIStgsysGetFilename()*: Seite nicht vorhanden
STG_BADJOB (-1007)	*LIStgsysXxxx()*: Ungültiger Job-Index
STG_ACCESSDENIED (-1008)	Storage ist mit ReadOnly-Flag geöffnet worden und kann keinen Schreibzugriff zulassen
STG_BADSTORAGE (-1009)	Interner Fehler des Preview-Files bzw. leeres File
STG_CANNOTGETMETAFILE (-1010)	*LIStgsysDrawPage()*: Metafile konnte nicht erzeugt werden (z.B. fehlerhafte Datei)
STG_OUTOFMEMORY (-1011)	Speicheranforderung schlug fehl
STG_SEND_FAILED (-1012)	Fehler beim Mailversand. Weitere Informationen zum Problem finden Sie in der Regel im Debug-Protokoll
STG_DOWNLOAD_PENDING (-1013)	Eine Aktion konnte nicht durchgeführt werden, da die zu betrachtende Datei noch nicht fertig geladen wurde
STG_DOWNLOAD_FAILED (-1014)	Eine Aktion konnte nicht durchgeführt werden, da der Download der zu betrachtenden Datei fehlgeschlagen ist
STG_WRITE_FAILED (-1015)	Schreib-/Berechtigungsfehler beim Speichern
STG_UNEXPECTED (-1016)	Unerwarteter Fehler. Wird bei Auftreten dem Benutzer mit weiteren Informationen gemeldet
STG_CANNOTCREATEFILE (-1017)	Schreib-/Berechtigungsfehler beim Speichern
STG_INET_ERROR (-1019)	Unerwarteter Fehler. Wird bei Auftreten dem Benutzer mit weiteren Informationen gemeldet
WRN_STG_UNFAXED_PAGES (-1100)	*LIStgsysPrint() bzw. LIStgsysStoragePrint()* (nur bei Druck auf Fax-Device): Einige Seiten enthielten keine Faxnummern-Informationen und konnten deswegen nicht gefaxt werden.

10. Fehlersuche mit Debwin

Debwin ist ein Tool für vielfältige Debugging Aufgaben.

Wenn der Debug-Modus über *LlSetDebug()* eingeschaltet ist, gibt List & Label auf Debwin Status-Informationen aus, sobald diese Applikation gestartet wurde. Um möglichst alle Ausgaben zu erhalten, empfehlen wir Debwin vor der zu debuggenden Applikation zu starten. Sobald Ihre Applikation gestartet ist, werden dann die Debug-Ausgaben von Debwin verwertet und ausgegeben. Alternativ kann in Debwin über **Logging** > **Force Debug Mode** die Protokollierung erzwungen werden.

Neben den Fehlercodes (s. Kapitel "Fehlercodes") erhalten Sie meist weitere Informationen, so dass Sie häufig auf diese Weise die Ursache für ein unerwünschtes Verhalten finden können.

Eine Ausgabe in einem typischen Debug-Log sieht wie folgt aus:

```
CMLL20 : 09:05:01.266 00000ee4 [lldemo32.exe] LlSelectFileDlgTitleEx
(1,0x00010a3a,'(NULL)',0x00008001,0x0012F86C,129,0x00000000)
CMLL20 : 09:05:19.726 00000ee4 ->'c:\vorlagen\artikel.lbl'=0
```

Man erkennt das aufgerufene Modul (CMLL20), Timing-Informationen, die Thread-ID, den Aufrufer (lldemo32.exe), die aufgerufene Funktion mit Parametern sowie – in der Folgezeile – den Rückgabewert der Funktion. Eine vollständige Log-Datei, wie sie auch unser Support-Team zur Problemanalyse benötigt, enthält eine große Vielzahl solcher Ausgaben. Ist dies bei Ihnen nicht der Fall, so haben Sie in der Regel entweder

- den Debug-Modus nicht mit *LlSetDebug()* eingeschaltet
- das Logging in Debwin nicht aktiviert

Debwin besitzt einen Zeilen-Ringpuffer, in dem nach Belieben vor- und zurückgeblättert werden kann. Dies geschieht durch die üblichen Cursor-Tasten oder den Scrollbalken.

Lesen Sie dazu auch die Datei "Selbsthilfe.pdf" im List & Label Dokumentationsverzeichnis oder im entsprechenden Knowledgebase-Artikel "Log-Datei erstellen und analysieren".

Wir leisten keinen Support für die nicht im Zusammenhang mit der Debug-Ausgabe stehenden Features dieses Werkzeuges.

11. Redistribution Ihrer Anwendung

Die List & Label-DLL und ihre benutzten Module können unter Windows zur Side-by-Side-Benutzung bei Ihrer Applikation installiert werden.

Die zur Weitergabe benötigten Dateien entnehmen Sie bitte der Datei Dokumentation\Files\REDIST.TXT unterhalb Ihres List & Label Installationsverzeichnisses.

11.1 Systemvoraussetzung

Die Systemvoraussetzungen für die Verteilung von List & Label mit Ihrer Anwendung sind dieselben wie für die Installation von List & Label auf Ihrem Entwicklungssystem. Diese finden Sie im Kapitel "Systemvoraussetzung".

11.2 64 Bit Module

Informationen zu den 64 Bit Modulen entnehmen Sie bitte der Datei REDIST.TXT in Ihrem List & Label Installationsverzeichnis.

Folgende Einschränkungen (Stand 10/2013) sind bei der Nutzung der 64 Bit Module in Betracht zu ziehen:

- kein OLE Container Objekt
- kein Projektassistent
- keine direkte digitale Signaturunterstützung (Einschränkung von OpenLimit/secrypt)

11.3 Die eigenständige Viewer-Applikation

11.3.1 Aufgabe

Der Viewer LLVIEW20.EXE ist eine eigenständige Applikation, die zur Anzeige der List & Label-Preview-Dateien dient.

Wenn der Viewer einmal registriert ist, wird die Dateiendung ".ll" mit dieser Applikation verknüpft, so dass über alle Links, die diese Endung enthalten (im Explorer, in Mails, in Internetseiten, ...) automatisch der Viewer gestartet wird.

11.3.2 Kommandozeilenoptionen

LLVIEW20 <Dateiname>

Lädt die angegebene Datei.

Eine URL als Parameter ist nicht möglich.

LLVIEW20 /p <Dateiname>

Druckt die angegebene Datei (auf Default-Drucker)

LLVIEW20 /pt <Dateiname> <Drucker-Name>

Druckt die angegebene Datei (auf den gewünschten Drucker). Falls der Druckername Leerzeichen enthält, muss er in Anführungszeichen gesetzt werden.

11.3.3 Registrierung

Rufen Sie den Viewer erstmals mit dem Parameter "/regserver" auf, um ihn zu registrieren - er beendet sich dann nach der Registrierung wieder. Zur De-Registrierung nutzen Sie den Parameter "/unregserver".

11.3.4 Benötigte Dateien

LLVIEW20.EXE benötigt zusätzlich die Dateien CMLL20.DLL, CMDW20.DLL, CMCT20.DLL, CMBR20.DLL, CMLS20.DLL und CMUT20.DLL. Sie benötigt außerdem mindestens eine Sprach-Ressourcen-Datei (z.B. CMLL2000.LNG). Abhängig von den benötigten direkten Export-Funktionalitäten, benötigen Sie außerdem:

- CMLL20XL.DLL wenn die direkte PDF Export-Funktionalität unterstützt werden soll

11.4 Die List & Label Dateien

List & Label speichert die Definition von Druckformularen in jeweils einzelnen Dateien ab. Zusätzlich zu dieser grundlegenden Definition werden spezielle Einstellungen wie Zieldruckername und -konfiguration, die im Designer bzw. Druckoptionsdialog angegeben werden können, in einer gesonderten Datei gespeichert (die sog. "P-Datei"). Ebenso liegt die kleine Skizze des Formulars, welche im Dateiauswahl-Dialog angezeigt wird, in einer eigenen Datei (die sog. "V-Datei").

Dateiextension:	Formular	Drucker-Definition	Skizze für Dialog
Etikettenprojekt	.lbl	.lbp	.lbv
Karteikartenprojekt	.crd	.crp	.crv
Listenprojekt	.lst	.lsp	.lsv

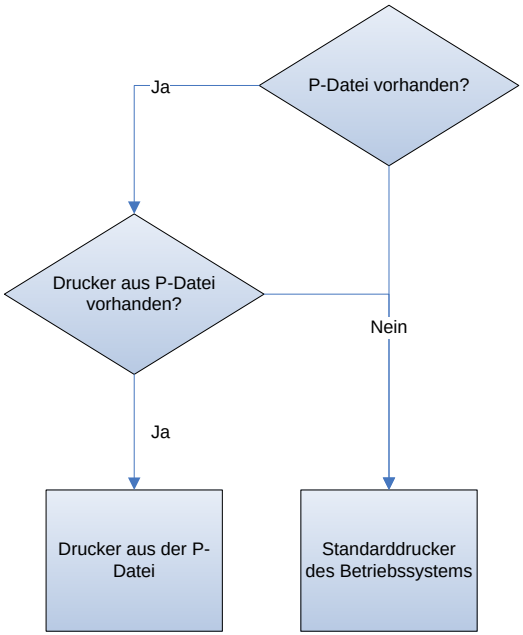
Diese Datei-Erweiterungen sind nicht verbindlich und können über *LISetFileExtensions()* oder *LISetOptionString()* geändert werden.

Die "entscheidende" Datei ist also lediglich die Formulardatei, welche die eigentliche Formulardefinition enthält.

Die Skizze für den Datei-Dialog ("V-Datei") kann jederzeit per *LICreateSketch()* generiert werden.

In der Druckerdefinitionsdatei ("P-Datei") werden neben den Druckereinstellungen auch die Einstellungen für die verschiedenen Exportmodule gespeichert. Diese Datei wird gewöhnlich zur Laufzeit vom Endanwender erstellt und sollte nicht von Ihnen redistribuiert werden – der Anwender hat mit hoher Wahrscheinlichkeit andere Drucker als Sie in seinem System.

Ist die P-Datei nicht vorhanden, so nimmt List & Label automatisch den aktuell eingestellten Windows-Standard-Drucker und erstellt für diesen auch die P-Datei. Der Pfad, an welchem die P-Datei gesucht bzw. erstellt wird, lässt sich per *LlSetPrinterDefaultsDir()* spezifizieren und könnte vor allem in einer Netzwerk-Anwendung relevant sein. Die Logik der Druckauswahl wird im folgenden Schaubild dargestellt:



Siehe hierzu auch auch *LlSetPrinterInPrinterFile()*.

Beim Druck auf Preview generiert List & Label eine Datei, welche alle gedruckten Vorschau-Seiten als Grafik enthält. Diese Datei hat die feste Endung .LL und kann auch jederzeit mit dem eigenständigen LLVIEWER-Programm angesehen werden. Der Pfad, in dem die LL-Datei erzeugt wird, lässt sich mittels *LlPreviewSetTempPath()* angeben.

Bei Druck auf eines der Exportmodule werden Dateien in einem vom Programm oder dem Benutzer anzugebenden Pfad angelegt. Bitte informieren Sie sich in der Dokumentation des Exportmodules über dessen Eigenschaften. Abfragen kann man eine Liste der erstellten Dateien mit *LL_OPTIONSTR_EXPORTFILELIST*.

Wenn eine Projektdatei gespeichert wird, wird eine Sicherungskopie angelegt. Der Name der Sicherungskopie wird erstellt, indem eine Tilde ("~") vor die Projekt-Endung gesetzt wird, beispielsweise wird aus der Endung ".LST" die Endung ".~LST".

Wenn das Laufwerk, auf der die Projektdatei liegt, keine langen Dateinamen unterstützt, wird eine solchermaßen berechnete Endung auf 3 Zeichen gekürzt, hier also ".~LS".

11.5 Sonstige Konfigurationseinstellungen

Folgende Daten werden abhängig von dem Namen Ihres Programms, in das Sie List & Label einbinden, verwaltet:

- Sprache
- Dialogdesign
- Dialogpositionen
- Designer-Voreinstellungen (Farben, Schriftart)

Dies bewirkt, dass die Einstellungen, die Ihr Programm bezüglich Sprache und Dialogdesign vornimmt, oder die Dialogpositionen und die Designer-Voreinstellungen, die vom Benutzer vorgenommen werden, nur für Ihre Applikation gültig sind. Somit braucht sich Ihre Anwendung nicht darum zu kümmern, welche Einstellungen andere Applikationen vornehmen.

List & Label speichert diese Einstellungen in der Registry unter "HKEY_CURRENT_USER/Software/combit/CMBTLL/<Applikationsname>".

12. Update-Hinweise für Version 20

Update-Hinweise für ältere Versionen finden Sie in der 'Dokumentationen' Programmgruppe als separates PDF-Dokument.

12.1 Überblick über Änderungen

12.1.1 Allgemein/API

- Neuer Callback "LL_QUERY_EXPR2HOSTEXPRESSION"
- Neue API *LlPrintDbGetCurrentTableFilter()*
- Neues Flag LL_ADDTABLEOPT_SUPPORTSADVANCEDFILTERING für *LlDbAddTable()*

12.1.2 Neue Features

- Parametrisierte Unterberichte (in Feldern, Tabellenzeilen, Kreuztabellen-Zellen, Torte/Trichter Diagramm, Balkendiagramm)
- Optionale Unterstützung von nativen Datenbankfiltern (siehe Allgemein/API)
- Mehrere Berichtscontainer (nur .NET)
- Echtdatenvorschau: Export nach Word, XHTML und Excel möglich wenn der Host den Drilldown Callback unterstützt.
- Office 2013 Style für die Menüband-/Ribbon-Oberfläche, allgemeiner Look & Feel Facelift
- Format übertragen für (die meisten) Objekte und Tabellenzellen hinzugefügt. Im Menüband/Ribbon und im 'klassischen' Menü.
- Der OLE-Container kann nun Formeln für seinen Inhalt verwenden.
- Checkbox Objekt
- PPTX-Export
- Das "Objekte" Toolfenster enthält nun Informationen des "Berichtsstruktur" Toolfensters.
- Die Position von eingefügten Objekten ist jetzt dort wo der Cursor ist, nicht die Originalposition des kopierten Objekts.
- Das eigenständige Vorschaufenster hat jetzt auch ein Zoomregler-Control.
- Tabellen mit "freiem Inhalt" werden jetzt auch als Untertabellen unterstützt.
- Neue Schaltflächen um die Auswahl im Funktionsassistenten ein- und auszukommentieren.
- Verschieben von Objekten aus dem Arbeitsbereich heraus scrollt nun den Arbeitsbereich in die Verschiebungsrichtung.

- Objektdimensionen werden bei Größenänderung angezeigt, der Hintergrund wird dabei ausgeblendet.
- Drücken der ALT-Taste beim Ändern der Größe: die Breite wird gerundet auf die Koordinate (mm/in)
- Neue Kontext-Schaltfläche im Menüband/Ribbon für Tabellenspalten Größenänderungsmodus
- Editor-Schriftart kann nun ausgewählt werden und wird in die Registrierung serialisiert. "Text in Layout-Vorschau anzeigen" wurde entfernt und ist jetzt immer Wahr.
- Beim Einfügen einer Tabellenzeile in eine Zeilendefinitionsgruppe, die nur eine leere (Standard) Zeilendefinition enthielt, wird diese leere Definition gelöscht.
- LocNumber$ hat nun einen optionalen dritten Parameter für Nachkommastellen.
- Ausklappbare Bereiche können in der Vorschau komplett geöffnet/geschlossen werden.
- Formel-Assistent Verbesserungen: zuverlässigere Autovervollständigung, Strg+Leertaste löst automatisch Autovervollständigung aus, verbessertes Verhalten der Autovervollständigung Listbox
- Neuer Fehlercode LL_ERR_NO_SUCH_INFORMATION wenn LlGetUsedIdentifiers[Ex] für eine Datei ohne solche Informationen verwendet wird
- Drag & Drop unterstützt nun auch Fußzeilenfelder mit automatischen Summen. Verwenden Sie STRG als Modifizierer um nur eine Kopfzeile hinzuzufügen.
- Fortschrittsanzeigen wurden für Export und Druck angeglichen.
- Neue Schaltfläche im Designer Menüband/Ribbon, die vertikale und horizontale Größenänderung in einem Schritt erlaubt.
- DWG (und somit die Bilder innerhalb LL) unterstützt nun das temporäre Herunterladen von Bildern
- XHTML unterstützt nun gedrehte Texte (alleinstehend & in Tabellenzellen). Umfangreiche Verbesserungen am Look & Feels des Export-Ergebnisses.
- SetVar hat nun einen neuen optionalen dritten Parameter um zu entscheiden, ob der gesetzte Wert ebenfalls mit ausgegeben werden soll.

12.2 Umstellung auf List & Label 20

12.2.1 Allgemein

Achten Sie darauf Ihren persönlichen Lizenzschlüssel zu aktualisieren, da dieser versions- und benutzerspezifisch ist.

Wie bei jedem Update einer Software empfehlen wir Ihnen auch bei einem List & Label Update alle Vorlagen und Projekte sorgfältig zu prüfen, da Verbesserungen zum Teil auch bedeuten, dass bestimmte Verfahren auf einem anderen Weg umgesetzt worden sind und dann nur eine hohe Annäherung aber keine 100%ige Identität erreicht werden kann.

12.2.2 Umstellung von .NET-Projekten

In der Regel genügt es, den Verweis auf die combit.ListLabel19.dll durch einen Verweis auf die combit.ListLabel20.dll auszutauschen und die Namespace-Verweise zu aktualisieren. Sie sollten zusätzlich die alten Komponenten aus der Toolbox entfernen und durch die neuen Komponenten ersetzen.

12.2.3 Erweiterungen gegenüber der Vorgängerversion

- Unterstützung für native Datenbankfilter für viele Datenprovider.
- Neuer Datenbindungsmodus zur Unterstützung von mehreren Berichtscontainern. Setzen Sie die Eigenschaft "NewDataBindingMode" auf "false" um das alte Verhalten wiederherzustellen.
- Neue Klasse AccessDataProvider
- Neue Klasse Google BigQuery DataProvider
- Neue Klasse InMemoryDataProvider
- XlsDataProvider unterstützt Excel 2013 Dateien
- DbConnectionDataProvider: 1:1 Relationen verwenden nun JOIN Ausdrücke für SQL Provider -> deutlich verbesserte Performance
- Verbesserte Unterstützung für In-Memory Datenbanken (SQLite); Filtern, Sortieren und das Auslesen der Datensatzzahl werden korrekt unterstützt.

12.2.4 Änderungen gegenüber der Vorgängerversion

- Der DbConnectionDataProvider hat nun einen abstrakten Member SupportsAdvancedFiltering. Wenn Ihr Provider Standard-SQL (Microsoft SQL) unterstützt, können Sie hier "true" zurückgeben. Anderenfalls ist die Rückgabe von "false" sicherer um SQL Syntaxfehler bei der Datenfilterung zu vermeiden. Siehe auch die Dokumentation hierzu in der .NET Hilfe.
- Die SqlConnectionDataProvider Klasse wurde in die Haupt-Assembly integriert, die separate Provider-Assembly wird nicht länger benötigt und ist auch nicht mehr verfügbar.
- LlGetOption liefert jetzt IntPtr statt int zurück. Bitte verwenden Sie einen expliziten Cast wo notwendig.

12.2.5 Umstellung von Projekten mit OCX (z.B. Visual Basic)

Sie können bestehende Visual Basic-Projekte folgendermaßen auf die aktuelle Version umstellen:

- Laden Sie die Visual-Basic Projektdatei (*.vbp bzw. *.mak) in einen Texteditor. Ersetzen Sie die Zeile

  ```
  Object="{2213E183-16BC-101D-AFD4-040224009C13}#19.0#0";"CMLL19O.OCX"
  ```

  durch folgende Zeile

  ```
  Object="{2213E183-16BC-101D-AFD4-040224009C14}#20.0#0";"CMLL20O.OCX"
  ```

und die Zeile

```
Module= CMLL19; CMLL19.BAS
```

durch die Zeile

```
Module=CMLL20; CMLL20.BAS
```

- Nach Speichern Ihrer Änderungen laden Sie die Form (*.frm) in den Texteditor, die das List & Label-OCX beinhaltet. Ersetzen Sie die Zeile

```
Object="{2213E183-16BC-101D-AFD4-040224009C13}#19.0#0";"CMLL19O.OCX"
```

durch folgende Zeile

```
Object="{2213E183-16BC-101D-AFD4-040224009C14}#20.0#0";"CMLL20O.OCX"
```

- Falls Sie ältere List & Label Versionen umstellen wollen, ändern Sie die entsprechenden Einträge analog ab. Bei Verwendung des Unicode-OCX-Controls passen Sie die ID ebenfalls entsprechend an. Die neue GUID des Unicode-Controls lautet {2213E280-16BC-101D-AFD4-040224009DF3}.
- Sie können nun Ihre Projekte in Visual Basic laden. Der Quellcode muss je nach Ausgangsversion geringfügig angepasst werden
- Da die List & Label-Konstanten im OCX-Control enthalten sind, ist ab VB 5 das .BAS-Deklarationsfile normalerweise nicht nötig.
- Beachten Sie, dass es nicht möglich ist, unterschiedliche List & Label-OCX-Versionen (also z.B. Version 19 und 20) im gleichen Projekt zu verwenden.

12.2.6 Umstellung von Projekten mit VCL (z.B. Delphi)

Beachten Sie hierzu die Hinweise in der Onlinehilfe für Delphi.

12.2.7 Umstellung bei API-Programmierung (z.B. C/C++)

- Passen Sie die Referenz auf die Deklarationsdatei auf die aktuelle Version an (z.B. bei C/C++ #include "cmbtll19.h" auf #include "cmbtll20.h")
- Passen Sie die Referenz auf die entsprechende Import-Bibliothek analog an (z.B. bei C/C++ in den Linker-Einstellungen cmbtll19.lib auf cmbtll20.lib)

13. Hilfe und Support

Viele Tipps und Tricks finden Sie in unserer Online Knowledgebase unter http://support.combit.net. Die Knowledgebase wird regelmäßig erweitert und um weitere Artikel ergänzt – reinschauen lohnt sich also!

Hinweise zum Supportkonzept finden Sie auf dem bei dem Produkt befindlichen Informationsblatt oder im Internet unter http://support.combit.net.

Voraussetzungen:

Bevor Sie uns kontaktieren, überprüfen Sie bitte folgende Punkte, bzw. verschaffen Sie sich die benötigten Informationen:

- Lesen Sie auf jeden Fall die neuesten Hinweise in der Datei readme_rel.pdf im Service Pack Downloadbereich im Internet unter http://support.combit.net.
- Lesen Sie dazu auch die Datei "Selbsthilfe.pdf" im List & Label Dokumentationsverzeichnis oder im entsprechenden Knowledgebase-Artikel "Log-Datei erstellen und analysieren".
- Verwenden Sie bei schriftlichen Anfragen das Online Support Formular.

14. Index

.

.NET 14
.NET Client Profile 19

0

0.0.46. 47

1

1:1 Relationen 102, 111
1:n Relationen 102, 108

A

Abbruch-Box 236
Abbruch-Dialogbox 228, 234
Ablauf 96
Access 31
AdoDataProvider 27
Alias 277
angehängte Objekte 101
API Referenz 62, 137
Aufbau 11
Ausfertigungen 37
Ausgabeformat vorgeben 25
Ausgabe-Medien 271
Ausklappbarer Bereich 126
AutoDefineField 35
AutoDefineNewLine 35, 50
AutoDefineNewPage 35, 50
AutoDefineVariable 35
AutoDestination 25
AutoFileAlsoNew 25
AutoMasterMode 25, 33
AutoProjectFile 25
AutoProjectType 25, 36
AutoShowPrintOptions 25
AutoShowSelectFile 25

B

Barcode 34
Barcodegröße 268
Barcodes 268
Barcodevariablen 92
Barcodevariablen definieren 157
Beispiele 45
benötigte Module 427, 428
Benutzerdaten 266
Benutzervariable 193
Berichtscontainer 41, 102
Berichtsparameter 125
Bild 34
Bitmaps 114
BMP 345

C

C/C++ 15
C++ Builder 15
Callbackroutine 113, 114
Callbacks 112, 113
Chart Objekte
  Ansteuern 127
Charts 49
Client Profile 19
Codepage 256
CSS 392

D

DataBinding 21
DataMember 33
DataProviderCollection 27
DataSet 27
DataSource 18, 28
DataTable 27
DataView 27
DataViewManager 27
Dateiauswahldialog 25
Dateiendungen 22, 250
Dateiextension 250, 429
Dateitypen 22
Daten unterdrücken 51
datenbankunabhängig 49
Datenprovider 26
Datenquelle 21
Datentyp
  Barcode 34
  Datum 34
  Grafik 34
  HTML 35
  Logisch 34
  RTF 34
  Text 34
  Zahl 34
Datentypen 32, 88
Datenübergabe 21
Datenversorgung 96
DateTime 34
Datumsformat 276
Datumsvariablen 90

DB2 52
DbCommandSetDataProvider 29
Debugging 16, 44, 146, 248, 427
Debug-Modus 248, 249
Debwin 249
Debwin3 44
Default-Drucker 187
Deklarationsdateien 15
Delphi 14, 86
Design 21
Designer 11, 93, 95
  anpassen 38
  erweitern 38, 54
DesignerControl 18
DesignerFunction 54
Device Context 116
Dezimalzeichen 270
Diagramme 127
Dialogstile 137, 138
Digitale Signatur 405
DLL 84
DOM 42, 129, 189, 279
  API 129, 189, 279
  Beispiele 133
  Einheiten 132
  Funktionen 130
DrawObject 35
DrawPage 35
DrawTableField 35
DrawTableLine 35
Drilldown-Berichte 120
Druck 22
  Netzwerk 58
Druckdaten 308
Drucker 37
Druckerauswahlfenster 213
Druckerbeschreibungsdatei 206, 278, 279
Druckereinstellungen 22
Druckerkonfiguration 206
Druckerschriftarten 265
Druckertreiber 26
Druckjob 87, 232, 422
Druckjobnummer 219
Druckoptionen 226, 229
Druckoptionsdialog 25
Druckschleife 96
Druckvorgang 97
Druckvorschau 96, 202, 203
Druckziel 96
Dynamic Link Library 84

E

Echtdatenvorschau 202
Echtdatenvorschau im Designer 117
Einbindung der Routinen 85
eMail 408, 413
eMail-Versand 56
EMF 345
Entity Framework 29
EntityCollection 29
Ereignisse 35
Etikett 45
Etiketten 13, 36
Etikettendruck 97
Excel 338, 366, 388
Export 23, 52
  Formate 23
  Formate einschränken 53
Export-Module 274, 334
  Digitale Signatur 405
  Excel 338, 366, 388
  Fax 387
  Grafik 345
  HTML 347
  In Zip archivieren 413
  JQM 358
  MHTML 361
  PDF 362
  RTF 370
  SVG 375
  Text (CSV) 381
  Text (Layout) 383
  TTY 385
  Versand per Mail 408, 413
  XHTML 392
  XML 399
  XPS 404
Exportoptionen 282
Extended MAPI 409
External$ 286

F

Faxversand 273, 387
Fehlercodes 422
Felder 13, 15, 25, 32, 36, 88
Feldpuffer 152
FileExtensions 22
Filter 211, 234
Filterbedingung 217, 302
Formelassistent 177
Formelparser 177
Fortschrittsanzeige 228, 236
Freier Inhalt 104, 107
Funktionen sperren 38
Fußzeile 294

G

GDI-Objekte 116
Geschwindigkeit 102
gesperrte Objekte 266
Grafik 92
Grafikdatei 92

Grafikformate
  Einbindung 92
Gruppenbereich 150
Gruppenfußbereich 150
Gruppenkopf Option 259

H

Hilfedatei 187
Hilfesystem 288
Hochformat 37
Hotline 436
HTML 35, 347
HTML Variablen 92
Hyperlinks 355, 380, 399

I

IDbCommand 29
IEnumerable<T> 29
IListSource 29
Import-Libraries 85
Instanzierung 19
Integration 19
Interaktive Sortierung 126
ITypedList 29

J

Java 15
Job Handle 196, 331, 422
JPEG 345
JQM 358
jQuery Mobile 358
Julianisches Datum 90

K

Karteikarten 13, 37
Karteikartendruck 97
Komponente
  Eigenschaften 22, 23, 24
Komponenten 18
Konfigurationseinstellungen 431
Konzept 87
Konzepte 26
Kopfzeile 294
Kopien 37, 101, 219, 229
Kreuztabellen 49

L

LastPage() 259
Layoutbereiche 37
Leerzeichenoptimierung 266
Linker 85
LINQ 29
List<T> 29
Liste 46
Listen 13, 36
Listendruck 98
Listenfuß 150
Listenheader 150
ListLabel 18
ListLabelDocument 18
ListLabelPreviewControl 18
ListLabelRTFControl 18
ListLabelWebViewer 19
Lizenzierung 9, 20
LL_DESIGNEROPTSTR _...
  PROJECTDESCRIPTION 168
  PROJECTFILENAME 168
  WORKSPACETITLE 168
LL_BARCODE 92
LL_BARCODE_...-Konstanten 93
LL_BOOLEAN 91
LL_CHAR_LOCK 89
LL_CHAR_NEWLINE 88
LL_CHAR_PHANTOMSPACE 88
LL_CMND_...
  DRAW_USEROBJ 113, 282
  EDIT_USEROBJ 284
  ENABLEMENU 285
  EVALUATE 286
  GETVIEWERBUTTONSTATE 287
  HELP 288
  MODIFYMENU 288
  OBJECT 289
  PAGE 291
  PROJECT 292
  SAVEFILENAME 293
  SELECTMENU 293
  TABLEFIELD 294
  TABLELINE 295
  VARHELPTEXT 297
LL_DATE 90
LL_DATE_DELPHI 90
LL_DATE_DELPHI_1 90
LL_DATE_DMY 90
LL_DATE_MDY 90
LL_DATE_MS 90
LL_DATE_OLE 90
LL_DATE_VFOXPRO 90
LL_DATE_YMD 90
LL_DATE_YYYYMMDD 90
LL_DRAWING 92
LL_DRAWING_HBITMAP 92
LL_DRAWING_HEMETA 92
LL_DRAWING_HICON 92
LL_DRAWING_HMETA 92
LL_DRAWING_USEROBJ 93
LL_DRAWING_USEROBJ_DLG 93
LL_ERR_...
  ALREADY_PRINTING (-6) 422

BAD_DLLS (-100) 425
BAD_EXPRESSION (-19) 423
BAD_EXPRMODE (-20) 423
BAD_JOBHANDLE (-1) 422
BAD_MODE (-32) 424
BAD_OBJECTTYPE (-3) 422
BAD_PRINTER (-15) 423
BAD_WEBSERVER_LICENSE (-50) 424
BADCODEPAGE (-45) 424
BADOBJNAME (-25) 423
BUFFERTOOSMALL (-44) 272, 424
CANNOTCREATETEMPFILE (-46) 424
CFGBADFILE (-24) 423
CFGBADMODE (-33) 424
CFGNOTFOUND (-22) 423
CONCURRENT_PRINTJOBS (-49) 424
DRAWINGNOTFOUND (-53) 425
EXCEPTION (-104) 425
EXPRESSION (-23) 423
LICENSEVIOLATION (-105) 425
NEEDS_VB (-14) 423
NO_BOX (-5) 422
NO_LANG_DLL (-101) 425
NO_MEMORY (-102) 425
NO_OBJECT (-29) 423
NO_PREVIEWFILES (-17) 423
NO_PREVIEWMODE (-16) 423
NO_PRINTER (-11) 422
NO_PROJECT (-10) 422
NO_TABLEOBJECT (-28) 423
NO_TEXTOBJECT (-30) 423
NO_WEBSERVER_LICENSE (-51) 425
NOCHART (-48) 424
NODESTINATION (-47) 424
NOPRINTERCFG (-38) 424
NOT_YET_PRINTING (-7) 422
NOTFINISHED (-43) 424
NOTINHOSTPRINTERMODE (-42) 424
NOVALIDPAGES (-41) 424
ONLY_ONE_JOB (-13) 423
ONLYWITHONETABLE (-34) 424
PARAMETER (-18) 423
PRINTFINISHED (-997) 425
PRINTING (-12) 422
PRINTING_JOB (-4) 422
SAVEPRINTERCFG (-39) 424
TASK_ACTIVE (-2) 422
UNKNOWN (-31) 423
UNKNOWNFIELD (-36) 424
UNKNOWNOBJECT (-27) 423
UNKNOWNSORTORDER (-37) 424
UNKNOWNVARIABLE (-35) 424
USER_ABORTED (-99) 233, 425
LL_INFO_METER 298
LL_INFO_PRINTJOBSUPERVISION 299
LL_NTFY_...
VIEWERBTNCLICKED 297
LL_NTFY_DESIGNERPRINTJOB 300
LL_NTFY_EXPRERROR 302
LL_NTFY_FAILS_FILTER 302
LL_NTFY_VIEWERDRILLDOWN 303
LL_NUMERIC 89
LL_NUMERIC_INTEGER 89
LL_NUMERIC_LOCALIZED 89
LL_OPTION_...
ADDVARSTOFIELDS 255
ALLOW_COMBINED_COLLECTING_OF_DATA_F OR_COLLECTIONCONTROLS 255
ALLOW_LLX_EXPORTERS 255
CALC_SUMVARS_ON_PARTIAL_LINES 256
CALCSUMVARSONINVISIBLELINES 256
CALLBACKMASK 256
CALLBACKPARAMETER 256
CODEPAGE 256
COMPRESSRTF 257
COMPRESSSTORAGE 257
CONVERTCRLF 257
DEFAULTDECSFORSTR 257
DEFDEFFONT 257, 270
DEFPRINTERINSTALLED 187
DELAYTABLEHEADER 101, 257
DESIGNEREXPORTPARAMETER 258
DESIGNERPREVIEWPARAMETER 258
DESIGNERPRINT_SINGLETHREADED 258
ERR_ON_FILENOTFOUND 258
ESC_CLOSES_PREVIEW 258
EXPRSEPREPRESENTATIONCODE 258
FONTPRECISION 258
FONTQUALITY 258
FORCE_DEFAULT_PRINTER_IN_PREVIEW259
FORCEFIRSTGROUPHEADER 259
FORCEFONTCHARSET 259
HELPAVAILABLE 187, 259
IMMEDIATELASTPAGE 259
INCLUDEFONTDESCENT 260
INCREMENTAL_PREVIEW 260
INTERCHARSPACING 260
LANGUAGE 187
LCID 277
LCID 260
LOCKNEXTCHARREPRESENTATIONCODE260
MAXRTFVERSION 261
METRIC 261
NOAUTOPROPERTYCORRECTION 261
NOFAXVARS 261
NOFILEVERSIONUPGRADEWARNING 261
NOMAILVARS 262
NONOTABLECHECK 262
NOPARAMETERCHECK 262
NOPRINTERPATHCHECK 262
NOPRINTJOBSUPERVISION 262
NOTIFICATIONMESSAGEHWND 262
NULL_IS_NONDESTRUCTIVE 263
PHANTOMSPACEREPRESENTATIONCODE263
PRINTERDCCACHE_TIMEOUT_SECONDS263
PRINTERDEVICEOPTIMIZATION 264
PROHIBIT_USERINTERACTION 264
PROJECTBACKUP 264

PRVRECT_HEIGHT 264
PRVRECT_LEFT 264
PRVRECT_TOP 264
PRVRECT_WIDTH 264
PRVZOOM_HEIGHT 264
PRVZOOM_LEFT 264
PRVZOOM_PERC 264
PRVZOOM_TOP 264
PRVZOOM_WIDTH 264
REALTIME 264
RESETPROJECTSTATE_FORCES_NEW_DC 265
RESETPROJECTSTATE_FORCES_NEW_PRINTJOB 265
RETREPRESENTATIONCODE 265
RIBBON_DEFAULT_ENABLEDSTATE 265
RTFHEIGHTSCALINGPERCENTAGE 265
SCALABLEFONTSONLY 265
SETCREATIONINFO 266
SHOWPREDEFVARS 266
SKETCH_COLORDEPTH 266
SKIPRETURNATENDOFRTF 266
SORTVARIABLES 266
SPACEOPTIMIZATION 266
SUPERVISOR 266
SUPPORT_HUGESTORAGEFS 267
SUPPORTS_PRNOPTSTR_EXPORT 267
TABLE_COLORING 267
TABREPRESENTATIONCODE 267
TABSTOPS 267
UISTYLE 267
UNITS 268
USE_JPEG_OPTIMIZATION 268
USEBARCODESIZES 268
USECHARTFIELDS 268
USEHOSTPRINTER 268
VARSCASESENSITIVE 268
XLATVARNAMES 269
LL_OPTIONSTR_...
CARD_PRJDESCR 273
CARD_PRJEXT 269
CARD_PRNEXT 270
CARD_PRVEXT 270
CURRENCY 270
DECIMAL 270
DEFDEFFONT 257, 270
EXPORTFILELIST 272
EXPORTS_ALLOWED 270
EXPORTS_ALLOWED_IN_PREVIEW 271
EXPORTS_AVAILABLE 271
FAX... 273
HELPFILENAME 272
LABEL_PRJDESCR 273
LABEL_PRJEXT 273
LABEL_PRNEXT 273
LABEL_PRVEXT 273
LICENSINGINFO 273
LIST_PRJDESCR 273
LIST_PRJEXT 274
LIST_PRNEXT 274
LIST_PRVEXT 274
LLFILEDESCR 274
LLXPATHLIST 255, 274
MAILTO 275
MAILTO_BCC 275
MAILTO_CC 275
MAILTO_SUBJECT 275
NULLVALUE 275
ORIGINALPROJECTFILENAME 274
PREVIEWFILENAME 275
PRINTERALIASLIST 275
PROJECTPASSWORD 276
SAVEAS_PATH 276
SHORTDATEFORMAT 276
THOUSAND 277
VARALIAS 277
LL_PRINT_MULTIPLE_JOBS 230
LL_PRINT_USERSELECT 96
LL_PRNOPT_...
COPIES 229
COPIES_SUPPORTED 218
DEFPRINTERINSTALLED 219
FIRSTPAGE 229
JOBID 219
JOBPAGES 230
LASTPAGE 229
OFFSET 228, 229
PAGE 229
PRINTDLG_ONLYPRINTERCOPIES 230
PRINTORDER 219
UNIT 219
UNITS 230
USE2PASS 220
LL_PRNOPTSTR_...
EXPORT 230
ISSUERANGES 230
PAGERANGES 231
PRINTDST_FILENAME 231
PRINTJOBNAME 231
LL_PROJECT_CARD 97
LL_PROJECT_LABEL 97
LL_PROJECT_LIST 98
LL_QUERY_EXPR2HOSTEXPRESSION 305
LL_QUERY_DESIGNERACTIONSTATE 305
LL_RTF 91
LL_TEXT 88
LL_WRN_REPEAT_DATA (-998) 101, 425
LlAddCtlSupport 137
LlAssociatePreviewControl 138
LlCreateSketch 139
LlDbAddTable 103, 139
LlDbAddTableEx 140
LlDbAddTableRelation 103, 141
LlDbAddTableRelationEx 142
LlDbAddTableSortOrder 103, 144
LlDbAddTableSortOrderEx 145
LlDbSetMasterTable 146
LlDebugOutput 146
LlDefineChartFieldExt 147

LIDefineChartFieldStart 223
LIDefineField 148
LIDefineFieldExt 149
LIDefineFieldExtHandle 151, 174
LIDefineFieldStart 152, 224
LIDefineLayout 95, 113, 153
LIDefineSumVariable 154
LIDefineVariable 155
LIDefineVariableExt 95, 156
LIDefineVariableExtHandle 157, 174
LIDefineVariableStart 102, 158, 225
LIDesignerAddAction 159
LIDesignerFileOpen 161
LIDesignerFileSave 162
LIDesignerGetOptionString 163
LIDesignerInvokeAction 163
LIDesignerProhibitAction 95, 164
LIDesignerProhibitEditingObject 165
LIDesignerProhibitFunction 166
LIDesignerRefreshWorkspace 167
LIDesignerSetOptionString 163, 167
LIDlgEditLineEx 168
LIDomCreateSubobject 130, 169
LIDomDeleteSubobject 131, 169
LIDomGetObject 130, 170
LIDomGetProject 170
LIDomGetProperty 132, 171
LIDomGetSubobject 130, 172
LIDomGetSubobjectCount 130, 172
LIDomSetProperty 131, 173
LIEnum...
GetEntry 173
GetFirstChartField 174
GetFirstField 175
GetFirstVar 175
GetNextEntry 176
LIExportOption 24
LIExprError 177, 302
LIExprEval 276
LIExprEvaluate 177, 423
LIExprFree 179
LIExprGetUsedVars 179
LIExprGetUsedVarsEx 180
LIExprParse 180
LIExprType 182
LIGetChartFieldContents 182
LIGetDefaultPrinter 183
LIGetDefaultProjectParameter 184, 416
LIGetErrortext 184
LIGetFieldContents 185
LIGetFieldType 186, 193
LIGetNotificationMessage 115, 186
LIGetOption 187
LIGetOptionString 188, 275
LIGetPrinterFromPrinterFile 188
LIGetProjectParameter 190, 416
LIGetSumVariableContents 191
LIGetUsedIdentifiers 152, 191, 223
LIGetUsedIdentifiersEx 192
LIGetUserVariableContents 193
LIGetVariableContents 194
LIGetVariableType 193, 194
LIGetVersion 195
LIJobClose 94, 195
LIJobOpen 87, 94, 96, 187, 196, 251
LIJobOpenLCID 87, 187, 197, 251
LIJobStateRestore 198
LIJobStateSave 198
LILocAddDesignLCID 199
LILocAddDictionaryEntry 200
LIPreviewDeleteFiles 201
LIPreviewDisplay 202, 423
LIPreviewDisplayEx 202, 203
LIPreviewSetTempPath 204
LIPrint 101, 204, 211, 234
LIPrintAbort 205, 233
LIPrintCopyPrinterConfiguration 206
LIPrintDbGetCurrentRelation 108
LIPrintDbGetCurrentTable 105, 207
LIPrintDbGetCurrentTableFilter 208
LIPrintDbGetCurrentTableRelation 209
LIPrintDbGetCurrentTableSortOrder 107, 209
LIPrintDbGetRootTableCount 207
LIPrintDeclareChartRow 210
LIPrintDidMatchFilter 211
LIPrintEnableObject 113, 211, 423
LIPrintEnd 212, 272
LIPrinterSetup 213
LIPrintFields 101, 211, 214, 234
LIPrintFieldsEnd 215
LIPrintGetChartObjectCount 216
LIPrintGetCurrentPage 216
LIPrintGetFilterExpression 217
LIPrintGetItemsPerPage 218
LIPrintGetOption 218
LIPrintGetOptionString 220
LIPrintGetPrinterInfo 220
LIPrintGetProjectParameter 221, 416
LIPrintIsChartFieldUsed 222
LIPrintIsFieldUsed 152, 223, 423
LIPrintIsVariableUsed 159, 224, 423
LIPrintOptionsDialog 214, 225, 270
LIPrintOptionsDialogTitle 226, 270
LIPrintResetProjectState 226, 422
LIPrintSelectOffset 227
LIPrintSelectOffsetEx 227
LIPrintSetBoxText 228, 234, 422
LIPrintSetOption 228, 229, 232, 233
LIPrintSetOptionString 230
LIPrintSetProjectParameter 231, 416
LIPrintStart 232
LIPrintUpdateBox 234
LIPrintWillMatchFilter 234
LIPrintWithBoxStart 97, 233, 235
LIProjectClose 237
LIProjectOpen 238
LIProjectSave 239
LIRTFCopyToClipboard 240

LIRTFCreateObject 241
LIRTFDeleteObject 241
LIRTFDisplay 241
LIRTFEditObject 243
LIRTFEditorInvokeAction 244
LIRTFEditorProhibitAction 244
LIRTFGetText 245
LIRTFGetTextLength 246
LIRTFSetText 247
LISelectFileDlgTitleEx 100, 247
LISetDebug 85, 248, 427
LISetDefaultProjectParameter 249, 416
LISetFileExtensions 250
LISetNotificationCallback 113, 251
LISetNotificationCallbackExt 252
LISetNotificationMessage 115, 254
LISetOption 96, 100, 255, 423, 424
LISetOptionString 100, 188, 251, 269
LISetPrinterDefaultsDir 278
LISetPrinterInPrinterFile 278
LISetPrinterToDefault 279
LLStaticTable 104, 107
LIStgsys...
Append 308, 327
Convert 309
DeleteFiles 310
DestroyMetafile 311
DrawPage 311
GetAPIVersion 312
GetFilename 313
GetFileVersion 314
GetJobCount 314
GetJobOptionStringEx 315
GetJobOptionValue 315
GetLastError 316
GetPageCount 316
GetPageMetafile 317
GetPageOptionString 318
GetPageOptionValue 319
GetPagePagePrinter 321
PageGetOptionValue 317
PageSetOptionString 319, 327
Print 322
SetJob 323
SetJobOptionStringEx 324
SetPageOptionString 324
StorageClose 325
StorageConvert 326
StorageOpen 326
StoragePrint 327
LLVIEW??.EXE 428
LIViewerProhibitAction 280
LIXGetParameter 281
LIXSetParameter 282
LoadFinished 67
Logische Variablen 91
Lokalisierung von Projekten 104, 277
LS_OPTION_...
PRINTERCOUNT 316
UNIT 316
LsMailConfigurationDialog 328
LsMailGetOptionString 329
LsMailJobClose 330
LsMailJobOpen 330
LsMailSendFile 331
LsMailSetOptionString 332
LsSetDebug 333

M

Mail 408, 413
MAPI 409
Mehrere Tabellen 102
Menü 184, 221, 231, 280
MENUID.TXT68, 70, 165, 244, 245, 280, 287, 289, 293, 298
Menü-IDs 289
Menüpunkte 159, 164
Menüpunkte sperren 38
MHTML 361
Multi Mime HTML 361
Multitabellen 102
Multithreading 421
MULTITIFF 345
MySQL 52

N

Nachrichten 113, 115
Nachrichtenschleife 233
Neues Projekt anlegen 25
Notifications 113
NULL-Werte 275, 415
Numerische Variablen 89

O

ObjectDataProvider 29
ObjectReportContainer 42
Objects 42
ObjectText 42
Objekt-Container 113
Objekte 39
Barcode 40
Bild 40
HTML 41
RTF-Text 40
Text 39
Objekte sperren 38
Objektmodell 42
OCX-Komponente
Datenübergabe 61
Designer-Funktionen 64
Designer-Objekte 66
Einbindung 60

Ereignisse 62
Print- und Design-Methode 60
Sprachwahl 62
Vorschaucontrol 63
Vorschau-Dateien 63
OCX-Viewer-Control 67, 71
OleDbConnectionDataProvider 31
Onlinehilfe 288
Optionen 255
Oracle 52
OracleConnection 31
OracleConnectionDataProvider 31

P

Parameter 423
Passwort 276
P-Datei 22
PDF 362
Pfadangaben 86
PNG 345
PostgreSQL 52
Preview 55, 423
Preview-Dateien 308
Preview-Fenster 95, 202
PreviewFile 55
Preview-Skizze 248
Progress 15
ProhibitedActions 38
ProhibitedFunctions 38
ProjectCard 42
ProjectLabel 42
ProjectList 42
Projektdatei vorgeben 25
Projektdateien
in Datenbank 22, 57, 59
Projektparameter 416
Projekttyp 25
Projekttypen 13, 36
Etiketten 36
Karteikarten 37
Listen 36
Property 'Pages' 71
Protokolldatei anfertigen 44

Q

Querformat 37

R

ReadOnlyObjects 38
Rechnung 33
Redistribution 11, 427, 428
Regions 42
Registrierung des Viewers 67, 72, 429
RTF 91, 257, 261, 370
RTF Variablen 91
RTF-Editor aufrufen 241
Rückgabewerte 87
Rundung 415

S

Sammelrechnung 47
Schriftart 257, 259, 265, 270
scLICallback 116
Seitenumbruch 101, 204, 214
Seitenvorschub 204
Seitenzahl 229
Serienbriefe 227
Seriendruck 33
Signatur 405
Skizze 429
SMTP 409
Spoolers 219
Sprache 187
Sprachen 13
SqlConnection 31
SqlConnectionDataProvider 31
SQLite 52
START-Event 122
Startseite 229
SubItemTable 43
Subreport 102
Subreports 49
Summenberechnung 256
Summenvariablen 155, 191, 415
Support 436
SVG 375
Systemvoraussetzung 9
Systemvorraussetzung 428

T

Tabellen 13
Tabellen, mehrere 102
Tausenderzeichen 277
Temporärpfad 204
Text (CSV) 381
Text (Layout) 383
Textbaustein 47
Textblöcke 32
Textvariablen 88
Threading 421
TIFF 345
Toolbar-Button Designer 293
Toolbar-Button Preview 287, 297
Translate$ 200
TTY 385

U

Übersetzung 277
ungebundene Daten 49
Unterbericht 102
Unterberichte 49
Update 434
  Erweiterungen 434
User-Objekte 113

V

VariableHelpText 35
Variablen 13, 15, 25, 32, 36
Variablenpuffer 94, 159
VCL-Komponente
  Datenbindung 74
  Datenübergabe 78
  Designer-Objekte 82
  Einbindung 74
  Ereignisse 79
  Print- und Design-Methode 77
  Relationale Verknüpfungen 75
  Sprachwahl 79
  Vorschaucontrol 79
  Vorschau-Dateien 79
VDF 15
Verschlüsselung 276
Versionsnummer 195
Viewer Applikation 428
Viewer OCX 67, 71
  'AsyncDownload' 67, 72, 73
  'BackColor' 67
  'CanClose' 69
  'CurrentPage' 68
  'Enabled' 67
  'FileURL' 68
  'GetOptionString' 69
  'GotoFirst' 69
  'GotoLast' 69
  'GotoNext' 69
  'GotoPrev' 69
  'PageChanged' 70
  'Pages' 68
  'PrintAllPages' 69
  'PrintCurrentPage' 69
  'PrintPage' 69
  'RefreshToolbar' 69
  'SaveAs' 69, 73
  'SaveAsFilePath' 69
  'SendTo' 69
  'SetOptionString' 69
  'SetZoom' 69
  'ShowThumbnails' 68
  'ToolbarButtons' 68
  'ToolbarEnabled' 68
  'Version' 69
  'ZoomReset' 69
  'ZoomRevert' 69
  'ZoomTimes2' 69
Viewer OCX Control 67, 71
Visual Basic 14, 85, 86
Visual C++ 15
Visual dBase 15
Visual FoxPro 15
Visual Studio 17
Vorschau 202
Vorschaudatei 204
Vorschaudateien
  konvertieren 55
  zusammenfügen 55
Vorzeitiger Seitenumbruch 101

W

Währungssymbol 260, 270
Webreporting 25, 420
Weitergabe 11
weitergebbare Dateien 427, 428
WIN.INI 279
Windows-Callback 114

X

Xbase++ 15
XHTML 392
XLS 338, 366, 388
XMAPI 409
XML 32, 399
XmlDataProvider 32
XP Look & Feel 267
XPS 404

Z

Zeichenketten 86
Zeichnungsvariablen 92
ZIP 413
Zoomfaktor 264
Zusatzdaten übergeben 35