Feature-Übersicht: Erweiterung des Berichtswesens

Lisa Slater Nichols, Milind Lele, Richard Stanton

Mit VFP 9 wurde das Berichtssystem von Visual FoxPro radikal geändert und es wurde eine komplexe Partnerschaft zwischen dem Produkt und seinen Xbase-Komponenten für das Design und das Generieren von Ausgaben eingeführt. In Sedna haben wir vor, verschiedene Aspekte dieser Partnerschaft zu überarbeiten, um die Möglichkeit des Anwenders zu verbessern, die Fähigkeiten der Xbase-Seite dieser Partnerschaft zu erweitern.

Einführung

Die Reaktion auf die Berichts-Features von VFP 9 war zufrieden stellend und die VFP-Anwender, die zunächst nur den verbesserten Basissatz einsetzten, haben begonnen, ihre eigenen Erweiterungen zu erstellen. Die FFC (FoxPro Foundation Classes) Basisklassen von VFP 9.0, die die Standardmechanismen für die Generierung aller unterstützten Ausgabearten enthalten, unterstützen nicht alle Features. Wäre dies in VFP 9 umgesetzt worden, hätte das für alle Berichte eine Verschlechterung der Performance bedeutet, während von den Vorteilen dieser Umsetzung die wenigsten Berichte profitiert hätten, da die alten Berichte keinen Zugriff auf die neuen Funktionalitäten haben.

Je vertrauter die Anwender mit den objektunterstützten Möglichkeiten des Berichtssystems von VFP wurden, desto mehr Berichtsfunktionalitäten erwarteten sie, die den Funktionalitäten in VFP-Formularen und Steuerelementen entsprechen, beispielsweise den dynamischen Wechsel zu unterschiedlichen Textstilen. Obwohl entsprechende Techniken als Beispielcode in der Hilfe von VFP 9.0 behandelt werden, ist es jetzt an der Zeit, sie direkt in den Standard-Basisklassen bereitzustellen.

Die Bereitstellung dieser Funktionalität hat kein direktes Gegenstück im „altmodischen" VFP-Berichtswesen und wird auch im FRX (Berichts-Metadaten)-Format nicht unterstützt. Aus diesem Grund wurde dies auch nicht in das angegebene Featureset der Xbase-Anwendung ReportBuilder von VFP aufgenommen, die die Aufgabe hat, während der Berichtsdesignsitzungen in das FRX-Format zu schreiben. Allerdings unterstützen die Memberdata von VFP 9.0 Erweiterungen an den Berichts-Metadaten für diesen und für ähnliche Zwecke.

In Sedna wird der ReportBuilder die Berichts-Metadaten umfassender unterstützen, so dass Berichtserweiterungen zur Laufzeit über die zusätzlichen Informationen verfügen, die sie benötigen. Die Methoden des ReportBuilders werden außerdem die grundlegende Erweiterbarkeit der Architektur als ein Modell für zusätzliche Features und Erweiterungen durch Dritthersteller sowie durch die FoxPro-Community bereitstellen.

Grundsätzlich war das Berichtssystem von VFP 9.0 eine Version 1.0, wenn sie auch ungewöhnlich vollständig war. In Sedna sollten wir die Features für erweiterte Ausgabeelemente abrunden und es wäre die Unterstützung alternativer Ausgabeformate möglich, wodurch die Grundlage für zukünftige Erweiterungen geschaffen würde.

Ziele

Hier eine Liste der allgemeinen Ziele:

- Minimale Produkterweiterungen, um eine Performancesteigerung zu ermöglichen, sowohl wenn dynamischer Xbase-Code eingesetzt wird, sowie auch, wenn er nicht eingesetzt wird.
- Bereitstellen des Codes der Basisklassen, um diese Erweiterungen zu veröffentlichen und zu unterstützen.
- Erweiterung der Xbase-Ausgabetypen (verschiedene XML-Ausgabedateitypen) in den Basisklassen, um das dynamische Verhalten angemessen zu nutzen.
- Hinzufügen von abwärtskompatiblen Erweiterungen zum VFP-RDL (Report Definition XML Schema), um sowohl das von uns hinzugefügte dynamische Verhalten zu fördern als auch nicht angegebene zusätzliche Erweiterungen zu verhindern.
- Bereitstellen von Berichts-Memberdata mittels dynamischer Laufzeiterweiterungen sowie mit Hilfe von Erweiterungen des ReportBuilders.
- Bereitstellen von Erweiterungsmechanismen des ReportBuilders.

Keine Ziele

Die folgende Liste enthält Features, die explizit keine Ziele darstellen

- Ein explizites UI des ReportBuilders für die Emulation eines Teils/aller Möglichkeiten des Klassen-Designers. So werden die Eigenschaften „DynamicFontBold“, „DynamicFontItalic“ uws. nicht korrekt dargestellt, obwohl es aufgrund der von uns bereitgestellten Features klar sein sollte, dass dies möglich ist.
- Eine explizite Demonstration aller Möglichkeiten der Verbesserungen der FFC und des VFP-RDL. Obwohl wir beispielsweise die Architektur für GDI+ in Berichten bereitstellen, werden wir keine Grafikklassen in das Produkt aufnehmen, die diese Architektur nutzen. Auch wenn wir die dynamische Unterstützung für die Verwendung der Methode AdjustObjectSize des ReportListeners in benutzerdefinierten Steuerelementen bereitstellen, werden wir ein solches Steuerelement nicht erstellen.
- Das Erstellen alternativer Oberflächen für die Berichtsvorschau. Der ausgelieferte Vorschaucontainer wurde erstellt, um den originalen VFP-Vorschaucontainer durch eine optisch und von der Arbeitsweise her Xbase-typische Oberfläche zu ersetzen. Obwohl viele Alternativen möglich sind, ist es deutlich, dass die Community mit dieser Aufgabe vertraut ist und keinen großen Anschub benötigt. Wir werden eine Erweiterung hinzufügen, um eine kleine Produktverbesserung bereitzustellen, ohne die der Druck aus der Originaloberfläche der Xbase-Vorschau nicht möglich ist. Diese Spezifikation umfasst keine Anregungen für alternative Vorschau-Oberflächen, die als Beispiel bereitgestellt werden, um den Einsatz von Features für die Interoperabilität zwischen Vorschau und den Druck von VFP-Berichten zu demonstrieren.

Szenarien

Viele der unten aufgeführten Szenarien bestehen aus zwei Teilen. Der erste Teil enthält ein oder mehrere Testszenarien, die mit dem Produkt ausgeliefert werden. Der zweite Teil behandelt Erweiterungsszenarien, die durch einen Tester, ein Mitglied der Community, oder einen Drittthersteller mit Hilfe der im Produkt enthaltenen Komponenten erstellt werden könnten.

Ausgabe eines Berichtsausdrucks in unterschiedlichen Farben, abhängig vom Wertebereich

Test-Szenario

Mary hat einen Bericht, der die Sollmenge für eine Inventur anzeigt. Jede Zeile, die einen Artikel beschreibt, enthält sowohl die Sollmenge als auch die aktuelle Menge in der Inventur. Sie möchte, dass die Istmenge in roter Farbe angezeigt wird, wenn sie kleiner ist als die Sollmenge, in blauer Farbe, wenn sie mit der Sollmenge übereinstimmt und in Orange, wenn sie höher ist als die Sollmenge.

Mary klickt doppelt auf das Element mit dem Berichtsausdruck, das die aktuelle Menge im Berichtslayout repräsentiert und wählt im Eigenschaften-Dialog den Tab Style. Sie entfernt die Markierung neben **use default pen color** und die Schaltfläche, die eine alternative Farbe bereitstellt, ist aktiviert. Sie wählt blau (nebenbei bemerkt: dies ist eine Standardfähigkeit von VFP 9.0).

Jetzt klickt sie auf den Tab **Dynamics**. Auf der linken Seite sieht sie die Liste **Conditions**. Standardmäßig enthält die Liste ein Element mit der Beschriftung „Default". Auf der rechten Seite sieht sie in einem Beispiel, wie ihr Text mit der Standardformatierung aussehen wird, etwa das gleiche, was sie als Beispiel auf der Seite Style gesehen hat. Die Seite enthält Schaltflächen für das Hinzufügen, Ändern und Löschen, sowie die Schaltfläche **Preview Script**.

Klickt Sie auf Add, gibt ihr ein modaler Dialog die Möglichkeit, die dynamischen Aspekte des Ausgabeausdrucks zu ändern, sowie eine logische **Condition**, die dem Steuerelement **Print Only When Expression ist True** im Tab **Print When** entspricht.

Da es sich um eine neue Bedingung handelt, entsprechen die Startwerte aller einstellbaren Elemente in diesem Dialog den Standardwerten für diesen Berichtsausdruck. So ist der Wert für Forecolor (Pen) blau und das Textelement ist „Inventory.CurrentLevel".

Nachdem Mary ihre Bedingung fertig gestellt und die zu ändernden Elemente ausgewählt hat, klickt sie in diesem Dialog auf OK. Sie wird aufgefordert, ein Label für die neue Bedingung zu erstellen. Das Label wird auf Eindeutigkeit geprüft (innerhalb der Liste der Bedingungen dieses Objekts; es muss nicht für mehrere Objekte eindeutig sein). Nach der Rückkehr in den Eigenschaften-Dialog zeigt der Tab Dynamics auf der linken Seite zwei Bedingungen. Wenn sie die beiden Bedingungen abwechselnd markiert, sieht sie im Beispiel auf der rechten Seite die Unterschiede.

Wenn sie nicht im PROTECTED (Endanwender)-Modus arbeitet und auf **Preview Script** klickt, sieht sie eine generierte Version des Skripts. Es enthält für jede Bedingung einen Kommentar. Will sie oder jemand anders das Skript direkt ändern, kann sie dies mit Hilfe des Features **Runtime extensions** im Tab **Other** des Dialogs tun. Sie kann auch die generierte Vorschauversion des Skripts an diese Stelle kopieren.

Beachten Sie bitte:
Ändert ein Anwender das Skript im nicht-PROTECTED-Modus, entfernt er es damit aus der Liste der Bedingungen. Es ist aber kein Nachteil, mit Ausnahme einer geringen Performanceeinbuße, die zusätzlichen CASE-Anweisungen im Skript zu belassen. Siehe nächste Notiz.

Wenn sie ihre Änderungen durch einen Klick auf **OK** bestätigt oder den Eigenschaften-Dialog schließt, speichert der Tab Dynamics die Änderungen als XML-Datensatz in den Metadaten des Layout-Steuerelements und verwendet für diese Datensätze ein spezielles reserviertes Namensattribut.

Beachten Sie bitte:

Ein explizites Skript in der reservierten ersten Metadaten-Zeile mit leerem Namensattribut kann zusammen mit dem vom Tab Dynamics generierten Skript ausgeführt werden. Das durch den Tab Dynamics generierte Skript wird als letztes ausgeführt. Dadurch hat der Endanwender bei kosmetischen Änderungen das letzte Wort.

Erweiterungsszenarien

- Mary liefert eine Anwendung aus, in der ein Bericht durch die Endanwender im PROTECTED Modus geändert werden kann. Bei einigen Objekten möchte sie zusätzlich zu dem von ihr angegebenen Verhalten kein dynamisches Verhalten zulassen, so dass der gesamte Tab Dynamics für diese Berichtsausdrücke nicht zur Verfügung steht. Mary leitet vom Tab Dynamics eine Klasse ab, um die Schutzeinstellungen eines Objekts zu prüfen. Sie ersetzt in ihrer ausgelieferten Version den Klassennamen dieses Tabs in der Registry des ReportBuilders.
- Michelle erstellt ein Add-On, das Text entsprechend angegebener Grad- oder Prozentwerte um die eigene Achse dreht. Sie implementiert dieses Verhalten als GFX-Erweiterungsobjekt (siehe: Xbase-Erweiterungsfeatures, FXListener, weiter unten). Sie stellt dafür ihre eigene Benutzeroberfläche als separaten Tab des ReportBuilders bereit. Michelles Add-On könnte ihre Informationen im Feld User als Wertepaare speichern oder in separaten Metadaten-Datensätzen mit dem von ihr angegebenen Wert im Namensattribut. Sie deklariert ihre Schnittstellenklasse für die Designzeit in der Registry des ReportBuilders und gibt an, dass ihr GFX-Objekt in der Registrierungstabelle des ReportListeners geladen werden soll. Das GFX-Objekt empfängt seine Informationen über den dynamisch erweiterten Text, konsultiert seine eigenen Anweisungen und rotiert das Ergebnis entsprechend.
- Michelle stellt ihre Klasse Mary zur Verfügung und dokumentiert das Laufzeitanweisungsformat ihrer Klasse. Mary kann wählen, ob sie den Tab mit Michelles Oberflächenerweiterung des ReportBuilders übernehmen will oder ob sie in ihren eigenen abgeleiteten Klassen den Tab Dynamics anpassen will, um Michelles Rotationsanweisungen zu schreiben.

Hinzufügen und Verwenden von Dokumenteneigenschaften

Testszenario

James möchte einem Bericht die Eigenschaft „Keywords“ hinzufügen. Er klickt mit der rechten Maustaste auf einen leeren Bereich seines Berichtslayouts und wählt im Kontextmenü **Eigenschaften**, um den Dialog **Berichtseigenschaften** anzuzeigen. Dort wählt er den Tab **Document**.

James gibt als Eintrag in Keywords **„Sales Despartment, Product Catalog, 2002“** ein. Beachten Sie, dass dies ein *Ausdruck* ist und dass Stringwerte daher die Anführungszeichen benötigen. James könnte hier auch den Namen einer Berichtsvariablen oder einer UDF() eingeben. James vergibt zusammen mit Keywords auch einen Eintrag für **Document Title**.

Beachten Sie bitte:

Wir verwenden hier eine Untermenge der Elemente des Tab Summary des Eigenschaften-Dialogs von Word. Hier besteht eventuell ein Potential für eine Erweiterung entsprechend Words Tab Benutzer im Dialog Dokumenteigenschaften. Dies wäre ein modaler Dialog, der das Hinzufügen, Ändern und Löschen ermöglicht. James kann den Namen seines neuen Dokumentenattributs angeben und dafür einen Ausdruck angeben, der in diesem Dialog ausgewertet wird.

Wenn James eine HTML-Ausgabe dieses Berichts erstellt, enthält das darunter liegende XML-Dokument ein entsprechendes Element, das in seinem Abschnitt <Run/> die Schlüsselworte in Form eines geprüften Ausdrucks enthält. Das HTML-Dokument wird diese Information in seinem Element <HEAD/> enthalten. Der Abschnitt <Run/> enthält außerdem den Knoten Title, so dass das HTML-Dokument statt der Standardwerte diese Information nutzt.

Verwendet James NOPAGEEJECT, um mehrere Berichte in ein einzelnes HTML-Dokument zu schreiben und setzt er für jeden dieser Berichte Schlüsselwortausdrücke ein, werden im Dokumentenelement <HEAD/> mehrere Sätze mit Schlüsselworten generiert. Der Knoten <TITLE/> des zusammengesetzten Dokuments wird wie die Druckseiteninformation und andere Aspekte der zusammengehängten Berichte durch den ersten Bericht des Satzes ausgewertet.

Erweiterungsszenario

Ein neuer Ausgabetyp (PDF oder DOC) kann die vorhandenen Dokumenteneigenschaften verwenden und ein oder zwei Methoden (über XML oder direkt) verwenden. Er kann auch weitere Elemente anlegen (neuer Tab, neuer Wert für Metadaten).

Hyperlinks und Verankerungen in HTML-Dokumente einfügen

Die Layoutsteuerung erhält den Tab HTML im Eigenschaften-Dialog.

Testszenarien

- Zwei HTML-Dokumente, eines mit Links, eines mit Verankerungen
- Grafiken erhalten im Attribut ALT andere als die standardmäßigen Werte
- Layoutelemente erhalten das Attribut TITLE

Erweiterungsszenario

RSS: RSS-Knoten enthalten Hyperlinks, um Verankerungen in HTML zu referenzieren.

Simultane dynamische Druck- und HTML-Ausgabe

In Sedna werden dynamische Inhalte Successoren zuarbeiten.

Testszenario

Mary verwendet den Bericht, den wir in unserem ersten Szenario erstellt haben. Sie instantiiert einen Listener für die Typen 0 und 5 und gibt den zweiten Listener als Successor des ersten an.

```
LOCAL loX, loY
STORE NULL TO loX, loY
DO (_REPORTOUTPUT) WITH 0, loX
DO (_REPORTOUTPUT) WITH 5, loY
loX.Successor = loY
REPORT FORM Inventory OBJECT loX
```

Sowohl das Druck- als auch das HTML-Ergebnis sollten Farbänderungen anzeigen.

Bereitstellen von grafischem und benutzerdefiniertem Verhalten der Steuerelemente

Erweiterungsszenario

Erweiterung: Hinzufügen des Tabs Dynamics, entsprechend der Berichtsausdrücke für Grafik- und Shape-Layoutsteuerelemente, aber das Bereitstellen verschiedener anpassbarer Attribute von AdjustObjectSize. Hinzufügen von MemberData und der Verwendung von gfx oder nicht.

Funktionale Beschreibung

Produkterweiterungen „unter der Oberfläche“

Das dynamische Verhalten in der Klasse ReportListener befindet sich im Ereignis EvaluateContents für Berichtsausdrücke in Layout-Steuerelementen, sowie für die Steuerelemente Shape und Image im Ereignis AdjustObjectSize. Derzeit sucht das Produkt in der Xbase-Klassenhierarchie nach dem Objekt, das als ReportListener dient, um zu prüfen, ob diese Ereignisse ausgelöst werden müssen, wodurch ein signifikanter Performanceeinbruch entsteht und kein nativer Dienst des Produkts ausgeführt wird. In Sedna ermöglichen es zwei neue Eigenschaften einer Xbase-Klasse, anzuzeigen, ob für sie beide Ereignisse ausgelöst werden müssen und ob in der Klassenhierarchie für jedes Code vorhanden ist oder nicht.

CallAdjustObjectSize

Eigenschaft vom Typ Integer. Beschreibung: *Zeigt an, ob ReportListener für alle Layout-Steuerelemente AdjustObjectSize aufrufen soll.*

Stellt ein Anwender einen anderen Wert als die unten aufgeführten ein, tritt ein Fehler auf (bestehende Fehlernummer, die zu passen scheint: 1469, *Eigenschaftswert ist außerhalb des gültigen Bereichs*).

Wert	Ergebnis
0 (Standard)	Das Verhalten von VFP 9.0: das Ereignis wird für alle Layout-Ereignisse ausgelöst, wenn sich in der Xbase-Klassenhierarchie ein Code für dieses Ereignis befindet.
1	Das Ereignis wird nicht ausgelöst, unabhängig davon, ob dafür Code vorhanden ist oder nicht; dynamisches Verhalten in AdjustObjectSize wird in diesem Fall nicht gewünscht oder ist für diesen Berichtslauf nicht erforderlich.
2	Das Ereignis wird immer ausgelöst, unabhängig davon, ob dafür Code vorhanden ist oder nicht. Dieser Wert dient primär den Anwendern, die BindEvents statt Code in dieser Methode einsetzen wollen, um sicherzustellen, dass das Verhalten nicht dadurch festgelegt wird, ob der ReportListener Code enthält oder nicht (da der Entwickler des gebundenen Objekts evtl. nicht weiß, welche ReportListener-Klasse verwendet wird). Dieses Verhalten ist auch für Hilfsobjekte sinnvoll, die den Prozess der Berichtserstellung unterstützen, da diese Funktionalität auch diesen Objekten zur Verfügung steht.

Der Wert dieser Eigenschaft wird durch die Basisklasse ReportListener am Beginn des Berichtslaufs *einmal* abgefragt, bevor das Ereignis BeforeReport aufgerufen wird. Dies ist die gleiche Stelle, an der ReportListener derzeit prüft, ob im Teil AdjustObjectSize seines FRX-Ladeprozesses benutzerdefinierter Code vorhanden ist; die Prüfung ändert sich nicht durch den Zeitpunkt, an dem die Prüfung durchgeführt wird. Beachten Sie, dass, wenn der Wert 1 oder 2 beträgt, das Produkt seine 9.0-Prüfung nicht durchlaufen muss, da diese Werte den Vorrang haben.

CallEvaluateContents

Vergleichen Sie CallAdjustObjectSize – funktioniert auf exakt die gleiche Weise, verwendet aber statt AdjustObjectSize das Ereignis EvaluateContents.

Zusätzlich zur Auswertung dieser Eigenschaften während der Initialisierung des Berichts werden wir prüfen, ob die Engine auch nach dem Start des Berichts die Werte dieser beiden Eigenschaften berücksichtigt. Eventuell ist es möglich, die Werte von CallAdjustObjectSitz und CallEvaluateContents im Ereignis BeforeBand zu prüfen, was bedeutet, dass alle Layout-Steuerelemente in einem Bereich berücksichtigt werden. Dadurch wäre eine signifikante Optimierung möglich, auch für Berichte, die ein dynamisches Verhalten benötigen.

Als Beispiel könnten Berichte mit Layoutflächen in Seitenköpfen, Detailbereichen und Seitenfußbereichen über eine Fläche in einem Gruppenkopf verfügen, der als Platzhalter für eine mit GDI+ erstellte grafische Zusammenfassung dient. Nur im Gruppenkopf muss CallAdjustObjectSize auf .T. gesetzt sein. Ebenso kann ein Berichtsausdruck in einem Seitenkopf für einige Präsentationstypen alternative Schriften enthalten, während andere Bereiche über eine Vielzahl Berichtsausdrücke verfügen könnten, die keine Anpassungen von EvaluateContents erfordern. In diesen Fällen kann der Anwender die Eigenschaften CallAdjustObjectSize und CallEvaluateContents anpassen.

Es ist für den Code des Anwenders nicht empfehlenswert und eventuell gar nicht machbar, Feineinstellungen für die Performance vorzunehmen, indem diese Eigenschaften für einzelne Objekte ein- und ausgeschaltet werden. Besonders bei AdjustObjectSize ist es im Hinblick auf Layoutsteuerelemente des gleichen Typs in einem gegebenen Bereich schwierig vorauszusagen, in welcher Reihenfolge oder auch wie häufig diese Ereignisse aufgerufen werden. Es gibt auch keine sinnvolle Stelle und kein sinnvolles Ereignis, an der diese Prüfung durchgeführt werden könnte, um die Eigenschaftenwerte für jedes Layout-Steuerelement zu ändern, bevor AdjustObjectSize und EvaluateContents für das jeweilige Layout-Steuerelement aufgerufen werden. Das Hinzufügen eines solchen Ereignisses (zusammen mit der Information in frxRecNo, um das Ereignis an das entsprechende Layout-Steuerelement zu binden) wäre nicht nur riskant, sondern würde auch den Performancegewinn schmälern.

Werte 91 bis 99 für SET("REPORTBEHAVIOR")

Der Befehl SET REPORTBEHAVIOR ermöglicht die Werte von 91 bis 99, ohne einen Fehler auszulösen und verhält sich trotzdem, als würde der Wert 90 betragen, wenn einer dieser Werte verwendet wird. Derzeit produzieren Werte niedriger als 80 und höher als 90 einen Fehler.

SET("REPORTBEHAVIOR") wird den exakten Wert (beispielsweise 95) zurückgeben, den der Anwender angegeben hat.

Beachten Sie, dass die Werte zwischen 81 und 89 bereits auf diese Weise funktionieren: Die Engine verwendet SET REPORTBEHAVIOR 80 und SET("REPORTBEHAVIOR") liefert den genauen Wert zurück, den der Anwender angegeben hat:

```
set reportbehavior 85
? set("REPORTBEHAVIOR")
```

Signatur der Methode: keine Argumente, kein signifikanter Rückgabewert. Beschreibung: *Stellt die Möglichkeit bereit, Seiten im Cache zu drucken, nachdem ein Bericht mit dem ListenerType 1 oder 3 ausgeführt wurde.*

Diese Methode ermöglicht es einem Anwender, das Produkt anzuweisen, Seiten im Cache zu drucken, ohne die Berichtsvorschau zu beenden. Derzeit besteht die einzige Möglichkeit, dies zu bewerkstelligen, in einem Aufruf von ReportListener.OnPreviewClose(lPrint).

Wie OnPreviewClose achtet auch diese Methode auf die relevanten Elemente von ReportListener.Commandclauses (Prompt, PrintPageCurrent, PrintRangeFrom und PrintRangeTo). Intern könnte dies gelöst werden, indem die Arbeit, die aktuell ausgeführt wird, wenn OnPreviewClose aufgerufen wird, in zwei Teile aufgespalten wird, so dass das Verhalten beim Druck absolut konsistent ist.

Der Wert von ReportListener.CommandClauses.File in LoadReport ist nicht schreibgeschützt

Sinn von ReportListener.CommandClauses ist, dem Code des Anwenders mitzuteilen, welche Bedingungen und Klauseln im aktuellen Befehl REPORT FORM verwendet werden.

Der native Code füllt die Eigenschaften mit den CommandClauses eines Berichtslaufs, liest und verwendet diese aber nicht, außer unter bestimmten festgelegten Umständen. Als ein Beispiel für einen solchen Umstand kann der Code des Anwenders eine Änderung an ReportListener.CommandClauses.Prompt vornehmen. Diese Änderung wird beim Aufruf von ReportListener.PrintCachedPages() oder ReportListener.OnPreviewCloxe(.T.) berücksichtigt.

Die Spezialfälle, die in diese Eigenschaften schreiben oder aus ihnen lesen, sind im Eintrag CommandClauses der Hilfe dokumentiert. In VFP 9 RTM und SP1 gehört ReportListener.CommandClauses.File nicht zu den angegebenen Ausnahmen.

In Sedna wird der native Code den neuen Wert von ReportListener.CommandClauses nach LoadReport lesen, also an dem Punkt, an dem der native Code die FRX liest und zu verarbeiten beginnt. Der Wert dieser Eigenschaft, nicht der ursprüngliche Wert im Befehl REPORT FORM, legt fest, welche FRX der native Code liest und verarbeitet.

Diese Änderung ermöglicht es dem Code des Anwenders, die FRX auch in einer Mehrbenutzerumgebung auf eine sichere Weise vorzuverarbeiten, indem er eine private Kopie der FRX erstellt und die Engine angewiesen wird, diese Kopie für den aktuellen Durchlauf zu verwenden. Xbase-Code kann eine temporäre FRX-Kopie erstellen und (als Beispiel) Objekte löschen, die für den aktuellen Wert in OutputType nicht benötigt werden.

Obwohl es möglich ist, ähnliche Aufgaben während des Berichtslaufs mit Hilfe von Ereignissen zu erledigen, bietet diese Änderung eine bessere Performance für Änderungen, die während eines Berichtslaufs jede Instanz eines Layout-Steuerelements betreffen (oder die alle Layout-Steuerelemente löschen, oder wenn es sich um eine andere globale Änderung handelt, beispielsweise eine Änderung an den Elementen der Datenumgebung).

Es ist auch möglich, die gleiche Aufgabe (erneutes Verarbeiten der FRX-Tabelle, Erstellen einer temporären Kopie) zu erledigen, bevor der Befehl REPORT FORM aufgerufen wird, aber diese Lösung bindet das Verhalten bei der erneuten Verarbeitung nicht in das System des ReportListeners ein. Die Durchführung dieser Änderung ermöglicht es den Listenern der Basisklassen, vor der Verarbeitung einen Hook mit einer eingebauten Intelligenz einzufügen.

Ist der neue Wert unbrauchbar (die Datei kann nicht gefunden werden, es handelt sich um keine gültige FRX, sie ist in einer anderen Designer-Sitzung geöffnet usw.) wird der gleiche Fehler ausgelöst, als wäre der Befehl REPORT FORM ursprünglich mit einem falschen Dateinamen aufgerufen worden. Hier der Testcode:

```
LPARAMETERS WhichTest
#DEFINE OtherReport "c:\temp\some.frx"
#DEFINE ThisReport  "c:\temp\other.frx"
CLEAR
TRY
   ox = CREATEOBJECT("r")
   DO CASE
   CASE EMPTY(WhichTest)
      USE (OtherReport) EXCLU
   CASE WhichTest = 1
      MODI REPO (OtherReport) NOWAIT
   OTHERWISE
      * we will be setting to non-existant report name
      ox.fileNotExist = .T.
   ENDCASE
   REPORT FORM (ThisReport) object ox
CATCH TO err
   LIST OBJECTS LIKE err
   MESSAGEBOX(MESSAGE() + " " + MESSAGE(1))
ENDTRY
CLEAR ALL
CLOSE ALL
RETURN

DEFINE CLASS r as ReportListener
   listenertype = 1
   fileNotExist = .F.
   PROCEDURE loadreport
      IF THIS.fileNotExist
         THIS.CommandClauses.File = "somethingsomething.frx"
      ELSE
         THIS.CommandClauses.File = OtherReport
      ENDIF
   ENDPROC
ENDDEFINE
```

Aus dem Blickwinkel des Xbase-Codes des ReportListeners zusammen mit der Möglichkeit, diesen Fehler in einer TRY…CATCH-Struktur zu sehen, geschieht das gleiche, als wenn der Xbase-Code in der Methode LoadReport ein RETURN .F. absetzt: es wird kein Berichtslauf ausgeführt. Obwohl dieses Ergebnis bereits eine Möglichkeit darstellt, sollte es im Benutzercode in der Umgebung behandelt werden. Soll der Berichtslauf als Beispiel eine Datei erstellen, ist es möglich, dass die Ausgabedatei nach dem Berichtslauf nicht vorhanden ist.

In der Regel übernimmt der Code des Benutzers, der eine Änderung vornimmt, die Verantwortung für die Sicherheit, löscht keine Elemente, die für Berechnungen des Berichts erforderlich sind, räumt hinterher die temporären Dateien wieder auf usw.

Wir werden empfehlen, dass CommandClauses.File durch den Code des Benutzers wiederhergestellt wird, so dass die Eigenschaft den gleichen Wert enthält, als wenn sie normal in UnloadReport enthalten wäre. Diese Aktion des Benutzers erfordert keine Behandlung oder Anpassung im nativen Code, ermöglicht es aber dem folgenden Code des Anwenders, den Wert nach dem Berichtslauf zu lesen, um den korrekten Wert zu ermitteln.

Beachten Sie, dass die Vorverarbeitung der FRX während des Designs innerhalb einer Sitzung des Berichts-Designers nicht auf genau die gleiche Weise möglich ist. Da der Berichts-Designer die FRX oder LBX in CommandClauses.File sperrt, kann der Code des Benutzers nicht einfach eine Kopie erstellen (lt. Design wird dem ReportListener der aktuelle FRX- oder LBX-Wert übergeben, nicht während der Design-Sitzung der temporäre FRX- oder LBX-Name. Dies werden wir in Sedna nicht ändern).

Wir werden empfehlen, dass der Xbase-Code ReportDesigner.CommandClauses.IsDesigner Loaded prüft, um festzustellen, welche Aktionen möglich sind. Unter manchen Umständen ist es dem Code des Benutzers möglich, dieses Szenario transparent zu behandeln. Unter anderen Umständen muss der Code des Anwenders anzeigen, dass ein bestimmtes Feature während der Designer-Sitzung nicht zur Verfügung steht. Wir werden mindestens ein Objekt bereitstellen, das das entsprechende Verhalten aufweist.

Beachten Sie, dass während einer Sitzung des Designers, wenn der Generator und die Ausgabe-Anwendung eng miteinander verwoben sind, es für den Generator vermutlich machbar ist, sich in das Ereignis Preview einzuhängen, auf der Festplatte eine Kopie der FRX in einer temporären Datei mit einem generierten Namen zu erstellen und den Namen der Kopie mit vollständigem Pfad irgendwo festzuhalten, wo ihn die vom ReportListener abgeleitete Klasse finden kann (beispielsweise in einer globalen Variablen). Der ReportListener könnte dann diese „sichere" Kopie anpassen und den Namen der Kopie in LoadReport an die Engine in CommandClauses.File übergeben. Wir werden keine Objekte erstellen, die sich auf diese Technik verlassen.

Erweiterungen der Features von Xbase

Erweiterung der ausgelieferten Vorschau: PrintCachedPages()

Die Schaltfläche Drucken in der mit VFP ausgelieferten Vorschau wird nach dem Drucken das Fenster nicht standardmäßig schließen. Dieses Verhalten ist mit dem Verhalten der Druckvorschau in anderen Microsoft-Werkzeugen konsistent.

Andere Vorschauwerkzeuge von Drittherstellen werden weiterhin wie bisher funktionieren (das Fenster wird nach dem Drucken geschlossen), da die alte API weiterhin funktionieren wird.

Erweiterung des ausgelieferten ReportBuilders und seiner Hilfsklassen (FFC)

Der Tab Dynamics im Eigenschaftendialog für Berichtsausdrücke

Neben der Schriftfarbe werden in diesem Dialog auch andere dynamisch einstellbare Aspekte der Berichtsausdrücke verfügbar sein (im Optimalfall alle Eigenschaften, die im Ereignis EvaluateContents gelesen und geschrieben werden können).

Verschiedene Eigenschaftendialoge erhalten den Tab HTML

Siehe das dritte Szenario weiter oben.

Beachten Sie auch die Erweiterungen des HTML Listener XSLT weiter unten.

Die Klasse _FRXCURSOR, um das Cursorhandling der Memberdata bereitzustellen

Wird von praktisch jedermann für dynamische Funktionen benötigt, und der Generator erledigt bereits einen Großteil dessen, was erforderlich ist.

Erweiterungen an der Berichtsausgabe-Anwendung sowie an den FFC Ausgabebasisklassen

Dieser Abschnitt verwendet die Begriffe „Basisset dynamischer Attribute“ und „unterstütztes Set dynamischer Attribute“. Das Wort „Attribute“ (jeglichen Typs) adressiert sowohl dynamisches Berichts- als auch Berichtssteuerungsverhalten, wie es durch den Code des Anwenders generiert wird, sowie den *Ausdruck* dieses Codes in der Berichtsausgabe. Beachten Sie bitte die folgenden Definitionen.

In diesem Dokument wird „Basisset dynamischer Attribute“ für Steuerattribute verwendet, die im Code von EvaluateContents und AdjustObjectSize geändert werden können. Diese Attribute wohnen der Stufe des Produkts inne und wir werden sie nicht im Xbase-Basisklassencode bereitstellen.

Das „unterstützte Set dynamischer Attribute“ umfasst dieses Basisset, sowie andere dynamische Attribute, für die wir Xbase-Code in das Produkt aufnehmen können.

„Erweiterte“ oder „benutzergesteuerte dynamische Attribute“ sind ebenfalls möglich, auch wenn diese hier nicht behandelt werden. Sie repräsentieren zusätzliches Verhalten, das nicht von Haus aus im FRX-Format, dem Basisprodukt, oder in der Auslieferung vorhanden ist.

Unterstützung von _ReportListener (FFC Xbase-Basisklasse)

_ReportListener wird jetzt Code enthalten, um während dynamischer Methoden Successoren aufzurufen. Am Beginn eines Berichtslaufs ruft _ReportListener alle Successoren auf und sucht nach entsprechenden Werten in CallEvaluateContents und CallAdjustObjectSize, um diese Werte zu prüfen.

Entsprechende Änderungen und Hinzufügungen in ReportListener.VCX

In die Hierarchie, abgeleitet von _ReportListener, werden FXListener integriert, die FX- und GFX-Decorator Collections unterstützen. FXListener pollt seine Collections, um zusätzliche Eingaben für die endgültigen Werte in CallEvaluateContents und CallAdjustObjectSize zu suchen.

Die Decoration-Fähigkeiten von FX und GFX sind nur vorhanden, wenn FXListener als führender Listener fungiert, in Successoren werden sie ignoriert. Nur der führende Listener hat die Möglichkeit, direkt mit der Engine zu kommunizieren, um die Aufrufe von EvaluateContents und AdjustObjectSize zu ändern. Es macht auch Sinn, alle Einstellungen am Anfang vorzunehmen, bevor zusätzliche Formen der Ausgabe generiert werden. Dieses Vorgehen wird in der Dokumentation bereits empfohlen und illustriert. Ein praktisches Beispiel finden Sie im vierten Szenario weiter oben.

Aus diesem Grund werden die FX- und GFX-Collections für CallEvaluateContents und CallAdjustObjectSize nur anfangs gepollt.

FXListener wird UpdateListener als Standardlistener für ListenerType 0 (Druck) und 1 (Vorschau) ersetzen. Diese Änderung ermöglicht es der Successorkette, sich wie im vierten Szenario beschrieben zu verhalten.

FXListener liefert Eigenschaften, die es den Anwendern ermöglichen, eine Feedback-Klasse, eine Klassenbibliothek, sowie ein Containermodul als Decoratorobjekt für das Feedback während eines Berichtslaufs anzugeben. Das Objekt FXTherm bietet das gleiche Feedback-Verhalten wie der UpdateListener.

Beachten Sie, dass der UpdateListener weiterhin in der Klassenbibliothek vorhanden ist, und dass, wenn am Verhalten von FXTherm Verbesserungen vorgenommen werden, diese Verbesserungen auch auf einfache Weise zurückportiert werden können. Wir werden die beiden Klassen gemeinsam verwalten. Dies ist wichtig, um sicherzustellen, dass den Anwendern, die explizit den UpdateListener einsetzen, die Verbesserungen nicht vorenthalten werden. Auch Menschen, die (aus irgendeinem Grund) ein Feedback des Anwenders aus einem folgenden Bericht und nicht aus dem führenden Listener benötigen, werden in der Lage sein, hierfür den UpdateListener zu verwenden.

UtilityReportListener wird von FXListener, nicht direkt von _ReportListener, abgeleitet. UtilityReportListener enthält die Dateibehandlung, sowie die Fähigkeit, eine Konfigurationstabelle zu lesen und zu verwalten, so dass sie die Klasse ist, von der alle unterstützten Xbase-Erweiterungs-Ausgabetypen (Debug, XML usw.) abgeleitet werden. Im Ergebnis stehen die Decoration-Fähigkeiten in all diesen Klassen zur Verfügung (HTMLListener wird die Fähigkeit beinhalten, die im vierten Szenario beschriebenen Farbänderungen vorzunehmen, wenn er kein Successor ist).

XMLListener und seine Ableitungen (XMLDisplayListener und HTMLListener) haben den Zugriff auf die dynamisch geänderten Attribute und stellen alle unterstützten Basissets dynamischer Attribute der Ausgabe bereit. Abgeleitete Klassen können ebenfalls dynamische Attribute entsprechend ihrer speziellen Form der Ausgabe unterstützen. Als Beispiel kann HTMLListener einen Ausdruck unterstützen, der geprüft werden soll, um den Namen eines CSS-Klassenattributs bereitzustellen und wird in der Regel das Hinzufügen von Ausdrücken unterstützen, um den Layout-Steuerelementen, die dies ebenfalls unterstützen, die Attribute href und title hinzuzufügen.

XMLListener wird einen Mechanismus enthalten, um einen neuen optionalen Abschnitt Run aus der VFP-RDL (siehe unten) zu erstellen. Abgeleitete Klassen werden die erforderlichen Möglichkeiten haben, um die Inhalte dieses Abschnitts zu ändern und zu verschönern. Als Beispiel kann HTMLListener ein oder mehrere untergeordnete Elemente einfügen, um die erforderlichen Referenzen auf externe CSS Stylesheets einzufügen.

XMLListener bietet im Abschnitt VFPDataSet auch einen Mechanismus für die Generierung von Knoten für Memberdata der VFP-RDL (siehe unten). Dafür wird das Objekt _FRXCURSOR. VCX eingesetzt. Die zusätzlichen Knoten werden das Schema verwenden, das bereits in der aktuellen Memberdata-Dokumentation aufgeführt ist.

Durch das „Herausbrechen“ des eingebetteten XML-Dokuments Memberdata, das in den Datensätzen der FRX enthalten ist, wird es für die Konsumenten des XML einfacher, diese Information zu verwenden, wenn sie benötigt wird. Beispielsweise wird ein RSSListener wissen, welche Berichtselemente für die Aufnahme in die RSS-Ausgabe markiert sind, ohne die Memberdata direkt parsen zu müssen.

Entsprechende Änderungen am VFP-RDL Schema

Lesen Sie bitte den Eintrag **Using VFP Report Output XML** in der Hilfedatei von VFP 9, der das vollständige VFP-RDL Schema enthält.

In Sedna wird das VFP-RDL Schema explizit die Verfügbarkeit grundlegender dynamischer Attribute herausstellen und wird aus Gründen der Lesbarkeit stark überarbeitet. Beachten Sie, dass durch die Überarbeitung des Schemas die unter der Version 9.0 erstellten Dokumente nicht ungültig werden. Beachten Sie auch, dass das XSD keine weiteren dynamischen Attribute unterstützen wird; bereits jetzt ist es möglich, verschiedenen Attributen ANY hinzuzufügen (mit <xs:anyAttribute processContents=“lax“/>).

Das VFP-RDL Schema wird einen optionalen neuen Abschnitt Run enthalten, der es ermöglicht, auf der Stufe des Berichts zusätzliche benutzergesteuerte dynamische Attribute aufzunehmen. In VFP 9.0 können Sie Attribute auf der Stufe des Objekts hinzufügen, aber es gibt keine offensichtliche Stelle, um neue globale Attribute anzulegen. Hier ein Ausschnitt aus der überarbeiteten XSD:

```
<xs:element name="VFP-Report">
   <xs:annotation>
      <xs:documentation>
         Contents of VFP-Report element are determined by
         XMLListener.XMLMode property
      </xs:documentation>
   </xs:annotation>
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="VFP-RDL" minOccurs="0"/>
         <xs:element ref="Data" minOccurs="0"/>
         <xs:element ref="Run" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>

<xs:element name="Run">
   <xs:annotation>
      <xs:documentation>
         Allows runtime data-dependent document attributes, such as
         contents of report variables or other accumulated data elements
         that do not occur in the layout itself, to be added at the
         conclusion of a report run.
         Content type is set at xs:any deliberately to allow extensions
         such as cursor-shaped XML for rows, etc.
      </xs:documentation>
   </xs:annotation>
   <xs:complexType mixed="true">
      <xs:sequence maxOccurs="unbounded">
         <xs:element name="property" type="VFP-Property"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:complexType name="VFP-Property" mixed="true">
   <xs:sequence>
      <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:string" use="required"/>
   <xs:anyAttribute processContents="lax"/>
</xs:complexType>
```

Wir werden die Verwendung dieses Abschnitts demonstrieren, indem wir RUN XML-Elemente mit den Dokumenteneigenschaften und den HTML-weiten Elementen generieren, die wir in den unterstützten Dokumentenattributen bereitstellen, beispielsweise mit den Dokumentenschlüsselworten oder den Referenzen auf das CSS Stylesheet.

Der Abschnitt des VFP-RDL Schemas, das die FRX beschreibt, wird überarbeitet, so dass er auch Informationen über die Memberdata aufnimmt. Unser VFP-RDL Schema wird jetzt die Typen referenzieren, die im VFP Memberdata Schema als xs:include definiert sind. Vergleichen Sie den Eintrag **Memberdata Extensibility** in der VFP-Dokumentation, der dieses einfache Schema vollständig enthält (und das wir in Sedna nicht ändern werden). Hier ein Auszug aus dem überarbeiteten VFP-RDL Schema mit dem neuen Element und dessen Definition, die direkt vom Memberdata Schema abgeleitet ist:

```
   <xs:element name="VFP-RDL">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="VFPDataSet" type="VFPDataSet"/>
         </xs:sequence>
         <xs:attribute name="id" type="xs:string" use="required"/>
      </xs:complexType>
   </xs:element>

<xs:complexType name="VFPDataSet">
      <xs:sequence>
         <xs:element name="VFPFRXLayoutObject" type="VFPFRXLayoutObject"
minOccurs="1" maxOccurs="unbounded"/>
         <xs:element name="VFPFRXLayoutNode" type="VFPFRXLayoutNode" minOccurs="1"
maxOccurs="unbounded"/>
         <xs:element name="VFPDataSource" type="VFPDataSource" minOccurs="0"
maxOccurs="unbounded"/>
         <xs:element name="VFPFRXCommand" type="VFPFRXCommand" minOccurs="0"
maxOccurs="1"/>
         <xs:element name="VFPFRXPrintJob" type="VFPFRXPrintJob" minOccurs="0"
maxOccurs="1"/>
         <xs:element name="VFPFRXMemberData" type="VFPFRXMemberData" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>

<xs:complexType name="VFPFRXMemberData">
      <xs:annotation>
         <xs:documentation>Provides information contained in the FRX's embedded
MemberData XML documents for various objects.</xs:documentation>
      </xs:annotation>
       <xs:attributeGroup ref="Common"/>
       <xs:attributeGroup ref="ReportTemplate"/>
       <xs:anyAttribute processContents="lax"/>
   </xs:complexType>
```

Überarbeitungen von HTML XSLT

Die XSLT, die dem HTMLListener zugeordnet ist, wird mit allen unterstützten dynamischen Attributen umgehen können, die im Zusammenhang mit HTML stehen.

Das Basis-XSLT enthält außerdem die Möglichkeit, Attribute im Zusammenhang mit HTML zu erkennen und zu nutzen, die nicht im Basisprodukt enthalten sind. Dies galt bereits für VFP 9.0, aber wir werden diese Fähigkeiten um so viele HTML-Fähigkeiten wie möglich erweitern.

Als Beispiel erkennt die Basis-XSLT die Attribute href in Berichtsausdruckknoten und erstellt die entsprechenden HTML-Ergebnisse. Obwohl VFP im Urzustand diese Funktionalität nicht nutzt, ist das Hinzufügen der Attribute href in das Quell-XML (a) durch die VFP-RDL möglich und (b) in einer von HTMLListener abgeleiteten Klasse einfach einzufügen, je nachdem, welche Regeln Sie wählen, um einen Berichtsausdruck als Hyperlink zu markieren. In Sedna könnten wir eine Standardmethode für das Generieren von href-Attributen verwenden.