TEIL 3

Web-Dynpro-Context verwenden

Der Context ist die Schnittstelle für den Datenaustausch zwischen der Benutzeroberfläche und der Web-Dynpro-Component. Ihm kommt daher in Web Dynpro eine besonders große Bedeutung zu. In diesem Teil gebe ich Ihnen Tipps, wie Sie den Context möglichst effizient verwenden können.

› Tipps in diesem Teil

Tipp 25

Supply-Funktionen einsetzen

Supply-Funktionen ermöglichen die einfache Befüllung von Context-Knoten mit Daten, der optimale Aufrufzeitpunkt wird dabei vom Web-Dynpro-Framework bestimmt. In diesem Tipp erfahren Sie mehr über die Anwendungsmöglichkeiten von Supply-Funktionen.

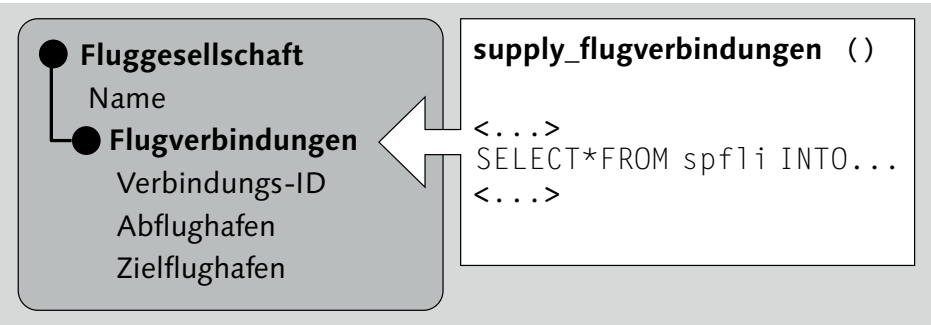
Supply-Funktionen sind Methoden, die zur automatischen Befüllung von Context-Knoten verwendet werden. Sie werden immer dann vom Web-Dynpro-Framework aufgerufen, wenn die Daten des ihnen zugeordneten Context-Knotens gebraucht werden. Dies ist z. B. dann der Fall, wenn die Daten eines Context-Knotens zum ersten Mal auf der Benutzeroberfläche angezeigt werden sollen. Durch diesen Mechanismus des Web-Dynpro-Frameworks sparen Sie in vielen Fällen wertvolle Zeit und Speicherplatz, da die Daten erst bei Bedarf geladen und in den seltensten Fällen alle Daten einer Anwendung benötigt werden.

Besonders im Zusammenhang mit Singleton-Knoten bietet sich die Verwendung von Supply-Funktionen an. Diese haben die Eigenschaft, ihre Kind-Elemente bei jeder Änderung der Lead Selection zu invalidieren, sodass erneut die Supply-Funktionen für die Kind-Elemente aufgerufen werden. Haben Sie also beispielsweise eine zweistufige Knotenhierarchie mit einer Liste von Fluggesellschaften auf der oberen Ebene und den Flugverbindungen der jeweiligen Gesellschaft eine Ebene tiefer, werden die Flugverbindungen immer dann über die Supply-Funktion nachgelesen, wenn Sie die Auswahl ändern.

Und so geht's

In diesem Tipp zeige ich Ihnen, wie Sie das beschriebene Beispiel mithilfe von Supply-Funktionen realisieren. Beginnen Sie mit dem Anlegen einer Test-Component und Testanwendung. Wechseln Sie anschließend in den View, und legen Sie auf der Registerkarte **Context** den Knoten `FLUGGESELL-`

SCHAFT an. Tragen Sie die Dictionary-Struktur SCARR ein, und wählen Sie die Kardinalität 1..n aus. Setzen Sie bei **Init. Lead-Selection** und bei **Singleton** den Wert Ja. Tragen Sie zuletzt in das Feld **Supply-Funktion** den Namen SUPPLY_FLUGGESELLESCHAFT der noch anzulegenden Supply-Funktion ein, und schließen Sie den Vorgang durch einen Klick auf **Attribute aus Struktur hinzufügen** und die Übernahme der Attribute CARRID und CARRNAME aus dem ABAP Dictionary in den Knoten ab.



Beispiel für Supply-Funktionen in Web Dynpro: Flugverbindungen

Legen Sie nun die Supply-Funktion für das Lesen der Fluggesellschaft an. Klicken Sie dazu in den Eigenschaften des Context-Knotens doppelt auf die Supply-Funktion SUPPLY_FLUGGESELLSCHAFT, und implementieren Sie das folgende Listing zur Initialisierung des Contexts.

```
DATA lt_fluggesellschaft TYPE wd_this->elements_fluggesellschaft.

SELECT * FROM scarr INTO TABLE lt_fluggesellschaft.

node->bind_table(
  new_items            =  lt_fluggesellschaft
  set_initial_elements = abap_true ).
```

Supply-Funktion »supply_fluggesellschaft( )«: Lesen der Fluggesellschaften

Gehen Sie nun eine Ebene tiefer, und legen Sie unterhalb der Fluggesellschaft den Context-Knoten FLUGVERBINDUNGEN an. Tragen Sie die Dictionary-Struktur SPFLI ein. Setzen Sie bei **Singleton** den Wert Ja. Tragen Sie zuletzt noch in das Feld **Supply-Funktion** den Namen SUPPLY_FLUGVERBINDUNGEN der Supply-Funktion ein. Schließen Sie den Vorgang durch einen Klick auf **Attribute aus Struktur hinzufügen** und die Übernahme der Attribute aus dem ABAP Dictionary in den Knoten ab.

Nun können Sie die Supply-Funktion für das Laden der Flugverbindungen nach der Auswahl der Fluggesellschaft anlegen. Die Supply-Funktion hierzu

muss die im darüberliegenden Context-Knoten ausgewählte Fluggesellschaft als Suchkriterium verwenden. Um diese auszulesen, lesen Sie den Importing-Parameter `PARENT_ELEMENT` aus. Dieser beinhaltet das die Lead Selection tragende Context-Element des Vater-Knotens mit der Fluggesellschaft. Verwenden Sie das folgende Listing zur Implementierung der Supply-Funktion.

```
DATA ls_fluggesellschaft TYPE wd_this->element_fluggesellschaft.
DATA lt_flugverbindungen TYPE wd_this->elements_flugverbindungen.

parent_element->get_static_attributes(
  IMPORTING static_attributes = ls_fluggesellschaft ).

SELECT * FROM spfli INTO TABLE lt_flugverbindungen
  WHERE carrid EQ ls_fluggesellschaft-carrid.

node->bind_table( new_items = lt_flugverbindungen ).
```

Supply-Funktion »supply_flugverbindungen( )«: Verbindungen von Gesellschaften

Damit haben Sie den Hauptteil der Übung abgeschlossen. Um die Supply-Funktionen testen zu können, müssen Sie zuletzt noch zwei Tabellen für die Darstellung der Fluggesellschaften und der Flugverbindungen anlegen. Am schnellsten können Sie die Tabellen mithilfe des Web-Dynpro-Code-Wizards anlegen, den ich Ihnen in Tipp 55, »Quellcode mit dem Code Wizard generieren«, vorstelle. Testen Sie anschließend die Anwendung. Nach der Auswahl einer Fluggesellschaft in der oberen Tabelle sollten sich die Flugverbindungen in der unteren Tabelle automatisch aktualisieren.

Fluggesellschaft	Fluggesellschaft
AC	Air Canada
AF	Air France
LH	Lufthansa

Fluggesellschaft	Flugnummer	Land	Abflugstadt	Startflugh.	Land	Ankunftstadt	Zielflugh.	Flugdauer	Abflug	Ankunftszeit
LH	0400		FRANKFURT	FRA		NEW YORK	JFK	504:00	10:10:00	11:34:00
LH	0402		FRANKFURT	FRA		NEW YORK	EWR	515:00	13:30:00	15:05:00
LH	0454		FRANKFURT	FRA		SAN FRANCISCO	SFO	740:00	10:10:00	12:30:00
LH	0455		SAN FRANCISCO	SFO		FRANKFURT	FRA	810:00	15:00:00	10:30:00
LH	2402		FRANKFURT	FRA		BERLIN	SXF	65:00	10:30:00	11:35:00

Test der Supply-Funktion: Alle LH-Verbindungen wurden geladen.

Tipp 26

Context-Attributeigenschaften verwenden

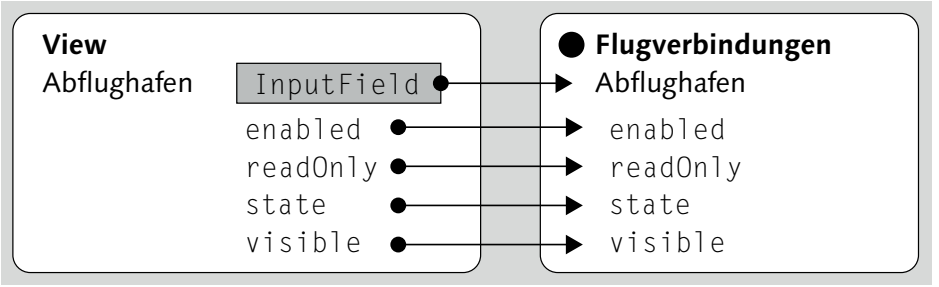
Viele UI-Elemente, wie beispielsweise das Eingabefeld, besitzen grundlegende Eigenschaften, die deren Aussehen steuern. Diese Eigenschaften können Sie direkt an ihre Context-Attribute binden, ohne zusätzliche Werte einzufügen. Wie dies funktioniert, erfahren Sie in diesem Tipp.

Bei der Verwendung von Eingabefeldern stehen Sie immer wieder vor der Aufgabe, diese in bestimmten Situationen auf nur Lesen zu setzen oder nicht anzuzeigen. Hierzu bietet Ihnen das `InputField` neben der primären Eigenschaft `value` für den eigentlichen Wert zusätzlich die UI-Element-Eigenschaften `readOnly` und `visible`. Diese Eigenschaften können Sie im View-Designer statisch setzen und später zur Laufzeit in der View-Methode `wddomodifyview( )` dynamisch über einen direkten Zugriff auf das UI-Element ändern.

Praktischer und übersichtlicher ist es, diese Eigenschaften eines Eingabefelds an ein Context-Attribut zu binden und über den Context zu steuern. Um hierfür keine zusätzlichen Context-Attribute anlegen zu müssen, besitzt jedes Context-Attribut mit `enabled`, `readOnly`, `required` und `visible` vier grundlegende Eigenschaften, die Sie für die Bindung der gleichnamigen UI-Element-Eigenschaften verwenden können.

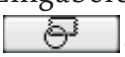
Und so geht's

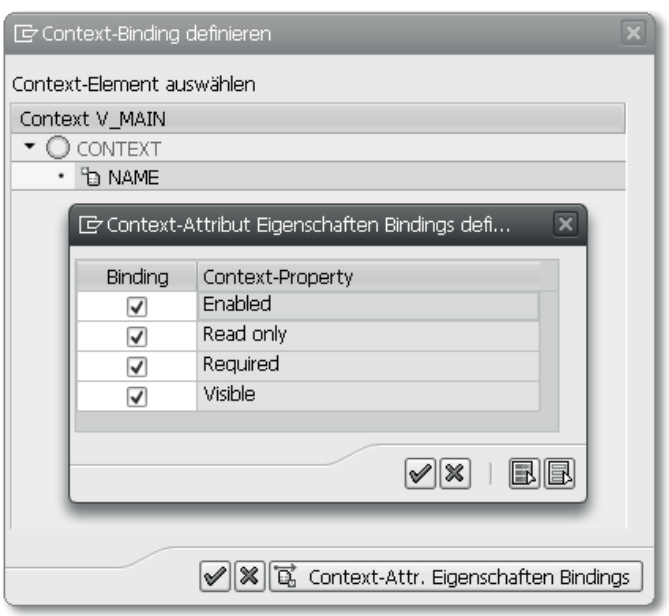
In diesem Tipp zeige ich Ihnen, wie Sie UI-Element-Eigenschaften an deren zugehörige Context-Attributeigenschaften binden und wie Sie diese programmatisch ändern können. Dazu werden Sie ein UI-Element vom Typ `InputField` anlegen und dieses mit seinen Eigenschaften gegen den Context binden. Über einen Button können Sie die Eigenschaften des Feldes steuern. Die folgende Abbildung zeigt Ihnen das Konzept von Context-Attributeigenschaften am Beispiel eines Eingabefeldes für einen Abflughafen.



View mit einem Abflughafen: Verwendung von Context-Attributeigenschaften

Beginnen Sie mit dem Anlegen einer Test-Component und einer Testanwendung. Wechseln Sie in den View der Component, und legen Sie auf der Registerkarte **Context** das Context-Attribut `NAME` vom Typ `STRING` an. Gehen Sie dann auf die Registerkarte **Layout**, und fügen Sie ein neues Eingabefeld (`InputField`) in die UI-Element-Hierarchie ein.

Wählen Sie das neue Eingabefeld aus, und klicken Sie bei der Eigenschaft `value` auf den Button  . Es öffnet sich das Pop-up **Context-Binding definieren**, in dem Sie das Eingabefeld gegen das Context-Attribut `NAME` binden können. Klicken Sie anschließend auf den Button **Context-Attr. Eigenschaften Bindings**. Im sich daraufhin öffnenden Pop-up-Fenster können Sie auswählen, welche Eigenschaften des Eingabefeldes Sie an das Context-Attribut binden möchten. Wählen Sie alle aus, und schließen Sie beide Pop-ups durch einen Klick auf den Button  . Sie gelangen nun zurück in den View-Designer. Wie Sie am Icon erkennen können, sind die UI-Element-Eigenschaften `Enabled`, `Read only`, `State` und `Visible` des `InputField` gegen das Context-Attribut `NAME` gebunden.



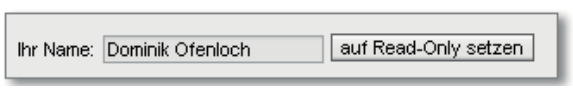
Bindung von Context-Attributeigenschaften

Möchten Sie nun die Eigenschaften eines Context-Attributs während der Laufzeit programmatisch ändern, können Sie hierzu im Interface `IF_WD_CONTEXT_ELEMENT` die Methode `set_attribute_property( )` verwenden. Über den Importing-Parameter `PROPERTY` können Sie angeben, welche der vier Context-Attributeigenschaften Sie verändern möchten. Das Interface bietet hierzu die Konstantenkomponente `E_PROPERTY` an. Möchten Sie z. B. das Context-Attribut `NAME`, und damit letztlich Ihr Eingabefeld, auf `Read only` setzen, könnte Ihr Listing hierzu wie folgt aussehen:

```
wd_context->set_attribute_property(
  EXPORTING
    attribute_name = 'NAME'
    property       = if_wd_context_element=>e_property-read_only
    value          = abap_true
    index          = wd_context->use_lead_selection
    all_elements   = abap_false ).
```

Context-Attributeigenschaften ändern: NAME wird auf Read only gesetzt.

Gehen Sie zurück zu unserer Beispiel-Component. Erweitern Sie diese um einen Button mit der Beschriftung **auf Read-Only setzen**, der direkt rechts neben dem im vorhergehenden Schritt angelegten Eingabefeld platziert wird. Legen Sie aus den Eigenschaften des Buttons heraus eine neue Aktion an, und implementieren Sie in deren Ereignisbehandler den im vorhergehenden Listing vorgestellten Methodenaufruf zum Setzen des Context-Attributs `NAME` auf `Read only`. Speichern und aktivieren Sie die Component. Wenn Sie die Testanwendung starten, können Sie durch Anklicken des Buttons das Eingabefeld auf `Read only` setzen.



Steuerung der Eingabefeld-Eingabebereitschaft über den Context

Tipp 27

Context-Knoten zur Laufzeit anlegen

Alle Operationen, die Sie im View-Designer statisch während der Designzeit vornehmen, können Sie auch dynamisch während der Laufzeit durchführen. Am Beispiel des Context-Knotens zeige ich Ihnen in diesem Tipp, wie Sie Veränderungen der Context-Struktur dynamisch vornehmen können.

Immer wieder kann es notwendig sein, den Context einer Component dynamisch zu modifizieren. So können Sie über das Metadatenmodell des Contexts beispielsweise neue Attribute in einem Knoten ergänzen oder auch neue Knoten erstellen.

Ein Beispiel für die dynamische Erzeugung bzw. Veränderung von Context-Knoten könnte ein Tabellenbrowser sein, der auf Web Dynpro basiert und dem Data Browser (Transaktion SE16) ähnelt. Je nach zugrunde liegender Datenbanktabelle benötigen Sie eine andere Knotenstruktur, und damit müssen Sie einen neuen Knoten anlegen oder die Struktur eines existierenden Knotens ändern. Laden Sie z. B. die Datenbanktabelle T100 in den Context, benötigen Sie einen Context-Knoten mit der gleichnamigen Context-Struktur. Möchten Sie die Tabelle BUT000 anzeigen, benötigen Sie einen anderen Context-Knoten. Wie genau Sie einen Knoten dynamisch erzeugen können, zeige ich Ihnen in diesem Tipp am Beispiel der ABAP-Dictionary-Struktur und der Datenbanktabelle `SCARR` (Übersicht der Fluggesellschaften). Im Anschluss erzeugen Sie eine Tabelle zur Anzeige der Daten im View.

Und so geht's

Jeder Context-Knoten besitzt zur Laufzeit ein Metadatenobjekt, das über das Interface `IF_WD_CONTEXT_NODE_INFO` angesprochen werden kann. Jedes Metadatenobjekt beschreibt die Struktur und die Eigenschaften eines Knotens. Dazu bietet das Interface des Objekts eine Vielzahl von Methoden zur

Abfrage und Manipulation der Knoteneigenschaften und der Knotenstruktur. So können Sie beispielsweise über die Methode `add_new_child_node( )` neue Kind-Knoten in die Struktur unterhalb des Knotens einfügen.

Genug der Theorie. Beginnen Sie mit dem Anlegen einer neuen Test-Component und Anwendung. Öffnen Sie anschließend den View, und wechseln Sie in die Methode `wddomodifyview( )`. In dieser implementieren Sie das folgende Listing zur dynamischen Erzeugung des auf der Struktur der Datenbanktabelle `SCARR` basierenden Knotens.

```
* Rufe das folgende Listing von wddomodifyview( ) nur 1 x auf
CHECK first_time EQ abap_true.

DATA: lo_context_node_info TYPE REF TO if_wd_context_node_info,
      lo_node_scarr        TYPE REF TO if_wd_context_node.

* Hole Knoten-Info-Objekt des ROOT-Knotens
lo_context_node_info = wd_context->get_node_info( ).

* Erzeuge den Knoten dynamisch
lo_context_node_info->add_new_child_node(
  EXPORTING
  static_element_type = 'SCARR' " ABAP-Dictionary-Struktur des Knotens
    name                = 'SCARR' " Name des Knotens
    is_multiple       = 'X' " können mehrere Elemente existieren?
  ).

* Hole zuletzt noch den Knoten SCARR
lo_node_scarr = wd_context->get_child_node( 'SCARR' ).
```

Dynamische Erzeugung des Context-Knotens SCARR

Damit haben Sie den eigentlichen Gegenstand dieses Tipps abgeschlossen, das dynamische Anlegen von Context-Knoten zur Laufzeit. Jedoch ist dieser Knoten allein ziemlich nutzlos. Daher zeige ich Ihnen im folgenden Listing noch, wie Sie den Knoten mit Daten füllen und anschließend daraus eine Tabelle in das View-Layout generieren können.

```
DATA: lt_scarr     TYPE TABLE OF scarr,
      lo_container TYPE REF TO cl_wd_transparent_container.

* Lese die Datenbanktabelle SCARR aus
SELECT * FROM scarr INTO TABLE lt_scarr.

* Binde den Tabelleninhalt gegen den Context
lo_node_scarr->bind_table( new_items = lt_scarr ).
```

```
* Hole die UI-Wurzel
lo_container ?= view->get_element( 'ROOTUIELEMENTCONTAINER' ).

* Erzeuge dynamisch eine Client-Tabelle
cl_wd_dynamic_tool=>create_c_table_from_node(
  EXPORTING
    ui_parent = lo_container
    node      = lo_node_scarr ).
```

Füllung des SCARR-Knotens mit Daten und Erzeugung einer Tabelle

Aktivieren Sie die Component, und starten Sie die Testanwendung. Hat alles geklappt, sollte die Testanwendung die folgende Tabelle im Browser anzeigen.

Mandant	Fluggesellschaft	Fluggesellschaft	Währ. d. Flugg.
001	AC	Air Canada	CAD
001	AF	Air France	EUR
001	LH	Lufthansa	EUR

Fertige Testanwendung: Dynamisch erzeugte Tabelle und Knoten

Tipp 28

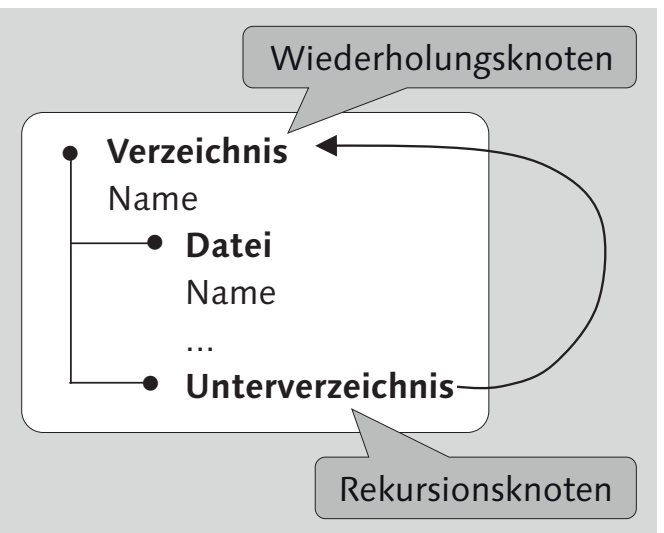
Rekursionsknoten anlegen

Hierarchien und Baumstrukturen finden Sie an vielen Stellen auf Ihrem Computer. Das bekannteste Beispiel ist die Verzeichnisstruktur einer Festplatte. Auch der Web-Dynpro-Context kann Hierarchien darstellen, was ich Ihnen in diesem Tipp zeigen möchte.

Mithilfe von sogenannten *Rekursionsknoten* können Sie im Context beliebig tief geschachtelte Hierarchien abbilden, wie z. B. Baum- oder Verzeichnisstrukturen. Ein Rekursionsknoten stellt dabei keinen Context-Knoten mit Eigenschaften und Attributen im klassischen Sinn dar, sondern er dient als Repeater eines darüberliegenden Knotens. Im folgenden Tipp zeige ich Ihnen, wie Sie einen solchen Rekursionsknoten anlegen.

Und so geht's

Zum Einstieg möchte ich Ihnen die Funktionsweise des Rekursionsknoten an einem Beispiel erläutern:



Rekursionsknoten schematisch dargestellt

Der Knoten **Verzeichnis** besitzt das Context-Attribut **Name**. Jedes Verzeichnis kann eine beliebige Anzahl von Dokumenten besitzen, die im Sub-Knoten **Datei** abgelegt werden. Der Sub-Knoten **Unterverzeichnis** ist ein Rekursionsknoten, der auf die darüberliegende Struktur des Knotens **Verzeichnis** verweist. Bei einem Wechsel in das **Unterverzeichnis** wiederholt sich die Struktur des Wiederholungsknotens, das heißt, die Attribute des Knotens **Verzeichnis** und dessen Sub-Knoten **Datei** und **Unterverzeichnis** werden in der Hierarchie eine Ebene tiefer angeboten.

Rekursive Hierarchien können durch zwei UI-Elemente dargestellt werden. Für die Darstellung von Hierarchien in Tabellen können Sie das UI-Element `TreeByNestingTableColumn` verwenden, für die Darstellung außerhalb von Tabellen existiert das UI-Element `Tree`. Die Erstellung von rekursiven Knoten, und wie Sie diese im `Tree` verwenden können, zeige ich Ihnen am Beispiel einer Context-basierten Verzeichnisstruktur.

Legen Sie eine neue Test-Component und Testanwendung an. Wechseln Sie dazu in den View, und beginnen Sie mit dem Anlegen der zu wiederholenden rekursiven, Context-basierten Verzeichnisstruktur. Öffnen Sie die Registerkarte **Context**, und legen Sie den Knoten `VERZEICHNIS` mit der Kardinalität `0..n` an. Fügen Sie das Attribut `NAME` vom Typ `STRING` in den Knoten ein. Legen Sie nun eine Ebene tiefer den Knoten `DATEI` an, und fügen Sie die Attribute `NAME` und `TYP`, beide vom Datentyp `STRING`, in den Knoten ein.

Legen Sie nun den Rekursionsknoten an. Klicken Sie dazu mit der rechten Maustaste auf den Knoten `VERZEICHNIS`, und wählen Sie **Anlegen ▸ Rekursionsknoten** aus. Tragen Sie im folgenden Pop-up unter **Knotenname** den Namen des Rekursionsknotens `UNTERVERZEICHNIS` ein, und klicken Sie zur Auswahl des Wiederholungsknotens auf den Button **Auswählen**. Wählen Sie im folgenden Pop-up den Knoten `VERZEICHNIS` aus, und bestätigen Sie die Auswahl durch einen Klick auf ✔. Damit haben Sie den Rekursionsknoten angelegt. Im Anschluss sollte Ihre Context-Struktur aussehen, wie in der folgenden Abbildung dargestellt.

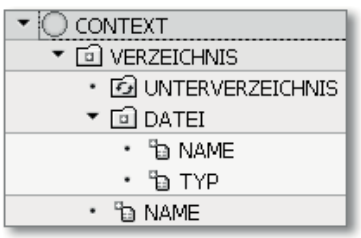


Abbildung von Verzeichnisstrukturen im Context mithilfe von Rekursionsknoten

Um nun die Verzeichnisstruktur als Baum auf der Benutzeroberfläche abbilden zu können, benötigen Sie jetzt noch das UI-Element `Tree`. Dieses besitzt, vergleichbar mit echten Bäumen im Wald, Blätter und Zweige:

- **Blätter (`TreeItemType`)**
  Blätter sind Endpunkte eines Baums. Im Beispiel der Verzeichnisstruktur können Blätter z. B. durch Dateien in einem Verzeichnis repräsentiert werden. Jeder Baum kann viele verschiedene Typen von Blättern besitzen, wobei jeder Typ in einem Baum durch das View-Element `TreeItemType` repräsentiert wird. Dies kann beispielsweise dann sinnvoll sein, wenn Sie mehrere verschiedene Dateiformate darstellen möchten.

- **Zweige (`TreeNodeType`)**
  Zweige stellen Knotenpunkte in Bäumen dar. Im Beispiel der Verzeichnisstruktur ist ein Zweig daher mit einem Verzeichnis gleichzusetzen. Auch Bäume können viele verschiedene Typen von Zweigen besitzen, diese werden durch View-Elemente vom Typ `TreeNodeType` repräsentiert. Durch die Verwendung verschiedener Zweigtypen können Sie z. B. unterschiedliche Icons für verschiedene Verzeichnistypen implementieren.

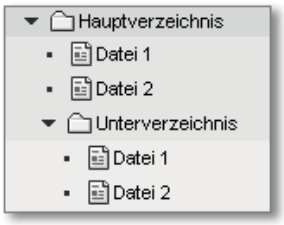
Wechseln Sie nun auf die Registerkarte **Layout**, und legen Sie den Baum an. Fügen Sie das UI-Element `Tree` in die View-Hierarchie ein. Deaktivieren Sie das Ankreuzfeld der Eigenschaft `rootVisible`, und binden Sie die Eigenschaft `dataSource` gegen den Knoten `VERZEICHNIS`. Durch diese Einstellungen setzen Sie den Knoten `VERZEICHNIS` direkt als Wurzel-Element des Baums fest.

Fügen Sie nun einen neuen Zweig in den Baum ein, indem Sie mit der rechten Maustaste auf den Baum klicken und im Kontextmenü den Eintrag **Knotentyp einfügen** auswählen. Geben Sie im folgenden Pop-up dem Zweig den Namen `VERZEICHNIS`, und wählen Sie den Knotentyp `TreeNodeType` aus. Nachdem Sie in den View-Designer zurückgekehrt sind, binden Sie dessen Eigenschaft `dataSource` gegen den Context-Knoten `VERZEICHNIS`. Binden Sie die Eigenschaft `text` gegen das Attribut `NAME` im Verzeichnis. Um dem Verzeichnis ein verzierendes Icon zu geben, empfehle ich Ihnen, als `iconSource` den Wert `~Icon/FolderFile` einzutragen.

Wiederholen Sie nun die letzten Schritte für die Blätter des Baums, das heißt für die Dateien. Wählen Sie erneut an der gewünschten Stelle in der Hierarchie den Eintrag **Knotentyp einfügen** im Kontextmenü aus, und fügen Sie ein Blatt vom Typ `TreeItemType` in den Baum ein. Binden Sie dessen Eigenschaft `dataSource` gegen den Knoten `DATEI`. Wiederholen Sie das Data

Binding anschließend für die Eigenschaft `text` mit dem Knotenattribut `NAME`. Als `iconSource` empfehle ich Ihnen das Attribut `~Icon/DocumentFile`.

Damit ist die Test-Component fast fertig: Sie haben eine rekursive Context-Struktur angelegt, die in Form eines Baums als Verzeichnisse mit Dateien angezeigt wird. Bislang ist der Context jedoch noch leer, Sie müssen noch Dateien und Verzeichnisse anlegen. Implementieren Sie hierzu die View-Methode `wddoinit( )`, und füllen Sie die Context-Struktur mit beliebigen Daten. Speichern Sie die Änderungen, und aktivieren Sie die Component.



Test der fertigen Testanwendung: Rekursiver Knoten

Tipp 29

Context nicht als Datenablage verwenden

Halten Sie Ihren Context sauber! Nicht selten werden im Context Daten abgelegt, die für das UI nicht relevant sind. Diese Zweckentfremdung führt später in vielen Fällen zu Problemen, die nur mit großem Aufwand behoben werden können.

Der Context ist die Datenaustauschschnittstelle zwischen dem Controller einer Component und den UI-Elementen im View bzw. im Browser. In der Praxis finden Sie aber auch häufig Knoten und Attribute im Context, die nicht für die Anzeige auf der Benutzeroberfläche gedacht sind. Welche Elemente Sie im Context ablegen sollten und welche nicht, erfahren Sie in diesem Tipp.

› Und so geht's

Nicht selten findet man im Context eine Mischung verschiedenster Daten. Zum Beispiel:

- das Attribut `NAME` zur Eingabe des Benutzernamens in einem Eingabefeld
- die Customizing-Tabelle `ZTC_BOOKING_TYPES` zur Zwischenspeicherung der verfügbaren Buchungstypen
- das Attribut `DATA_CHANGED`, als Indikator dafür, ob beim Verlassen des Views die Daten gespeichert werden müssen

Von diesen drei Beispielen gehört jedoch nur das Attribut `NAME` in den Context, da es einen direkten Bezug zu einem Eingabefeld und somit zur Benutzeroberfläche enthält. Daten, die nur in einer Component zwischengespeichert werden sollen, sei es die zwischengespeicherte Tabelle mit den

Buchungstypen oder das Attribut `DATA_CHANGED`, sollten Sie auf der Registerkarte **Attribute** oder in einer Assistance-Klasse ablegen.

Warum aber werden so häufig Daten ohne direkten UI-Element-Bezug im Context abgelegt? Häufig liegt dies daran, dass die Entwickler von der Context-Mapping-Funktionalität Gebrauch machen möchten. Aus Entwicklersicht ist es häufig sehr bequem, in einer Component an einer zentralen Stelle (meist dem Component-Controller) einen `SETTINGS`-Knoten anzulegen und diesen dann in den Views der Component oder in verwendeten Components in den lokalen Context zu mappen.

Ein großer Nachteil des statischen Mappings ist es, dass Änderungen am Originalknoten nicht automatisch an die gemappten Knoten weitergereicht werden. Fügen Sie also z. B. im Component-Controller am `SETTINGS`-Knoten das Attribut `BUTTONS_ENABLED` nachträglich ein, müssen Sie diese Änderung in allen gemappten Knoten nachziehen, indem Sie **Aktualisieren Mapping** im Kontextmenü der jeweiligen Views in der entsprechenden Component auswählen. Das gilt auch für den Fall, dass Sie ein nicht mehr verwendetes Attribut im Originalknoten löschen.

Ein weiterer wichtiger Grund dafür, dass Sie den Context nicht als Datenablage für nicht UI-relevante Attribute verwenden sollten, ist der fehlende Verwendungsnachweis für Context-Attribute. Dieses Problem ist ein allgemeines Problem im Context, das für alle Context-Attribute gilt. Haben Sie einen Context-Knoten angelegt, sind für den Knoten zwar Konstanten verfügbar, nicht jedoch für die Attribute im Knoten. Für die Abfrage oder das Setzen eines Context-Attributwerts wird der Attributname dann häufig als Literal in einfachen Anführungszeichen geschrieben. Da auf Literale keine Verwendungsnachweise möglich sind, führt dies vor allem bei der Löschung von nicht mehr benötigten Attributen zu Problemen. Häufig kommt es vor, dass ein gelöschtes Attribut in irgendeiner Ecke Ihrer Component doch noch verwendet wird, was das System mit einem Shortdump quittiert.

Ein letzter Grund dafür, warum Sie den Context nicht als Datenablage verwenden sollten, ist schlicht und einfach die Performance. Jeder zusätzliche Knoten und jedes zusätzliche Context-Attribut bindet Systemressourcen, die Sie sparen können. So werden Context-Knoten und Context-Elemente je in eigenen Objekten abgebildet. Für jedes Context-Attribut werden darüber hinaus UI-Element-Eigenschaften verwaltet, die zusätzliche Performance benötigen. Daher empfehle ich Ihnen, den Context nur dann zu verwenden, wenn die darin unterzubringenden Daten einen direkten Bezug zu UI-Elementen in einem View haben.

Tipp 30

Mapping zwischen Components anlegen

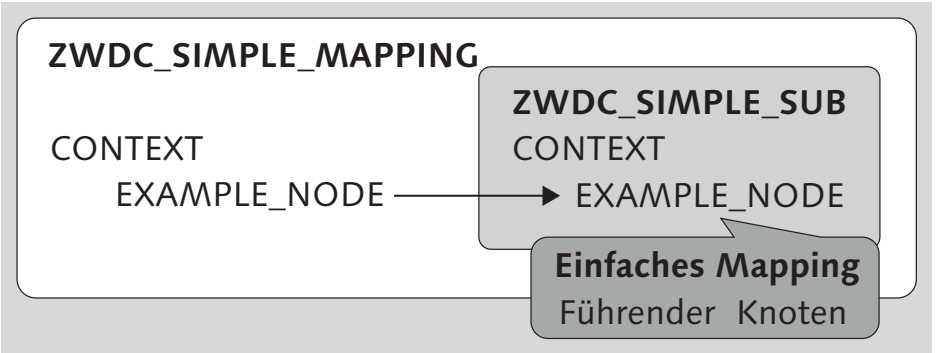
Durch das Component-übergreifende Mapping können Sie Context-Knoten über mehrere Components hinweg teilen, was Ihnen die Erstellung von Multi-Component-Architekturen erleichtert.

Wenn Sie schon einmal vor der Entwicklung einer umfangreichen Web-Dynpro-Anwendung gestanden haben, kommt Ihnen diese Fragestellung sicherlich bekannt vor: Soll ich eine große Component anlegen, oder soll die Anwendung in viele kleine Components aufgeteilt werden? In der Regel ist die Aufteilung in wenige, semantisch zusammengehörende Components die bessere Wahl. Mithilfe des Mappings können Sie in Context-Knoten abgelegte Daten über die Grenze eines Controllers hinweg austauschen. Wie das Component-übergreifende Mapping funktioniert und welche Besonderheiten es dabei gibt, erkläre ich in diesem Tipp.

Und so geht's

Beim Component-übergreifenden Mapping wird zwischen einfachem und externem Mapping unterschieden. Während beim einfachen Mapping der Informationsfluss vom Context der eingebetteten Component zum Context der verwendenden Component läuft, ist dies beim externen Mapping genau umgekehrt.

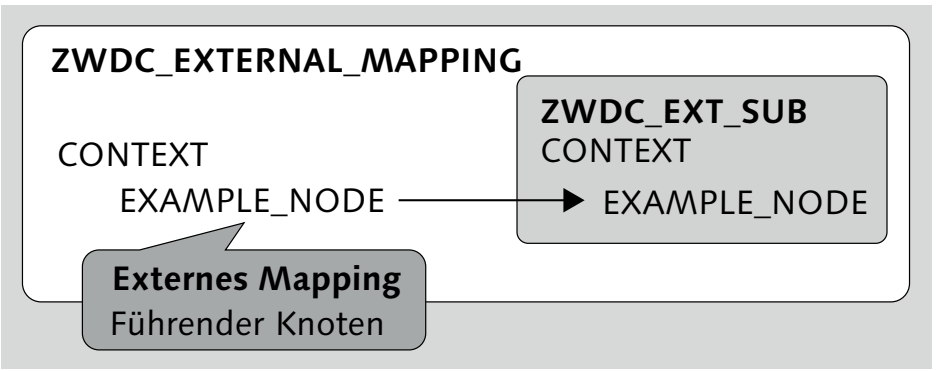
Betrachten wir zuerst das einfache Mapping. Bei diesem befindet sich der primäre Knoten in der eingebetteten Component (`ZWDC_SIMPLE_SUB`). Falls in dieser eine Supply-Funktion existiert, versorgt diese damit den gemappten Knoten in der umgebenden Component (`ZWDC_SIMPLE_MAPPING`). Bei der Erstellung des einfachen Mappings wird die Knotenstruktur von der eingebetteten Component in die umgebende Component übernommen.



Einfaches Mapping

Die Definition eines einfachen Mappings unterscheidet sich kaum von der Erstellung eines Controller-übergreifenden Mappings. Stellen Sie hierzu sicher, dass der zu mappende Knoten – im Component-Controller der eingebetteten Component – als Interface-Knoten definiert ist. Tragen Sie anschließend eine Component-Verwendung für die einzubettende Component in der umgebenden Component ein. Der anschließend durchzuführende Mapping-Vorgang unterscheidet sich nicht mehr vom normalen Controller-übergreifenden Mapping.

Beim externen Mapping verläuft der Informationsfluss im Vergleich zum einfachen Mapping umgekehrt: So ist nicht mehr der eingebettete (`ZWDC_EXT_SUB`), sondern der lokale Knoten (`ZWDC_EXTERNAL_MAPPING`) der umgebenden Component der primäre Knoten. Dieser kann dort mithilfe einer Supply-Funktion mit Daten gefüllt werden.



Externes Mapping

Darüber hinaus sind beim externen Mapping noch einige Besonderheiten zu beachten. So können Sie den extern zu mappenden Knoten der eingebetteten Component bei der Definition völlig untypisiert lassen. Der Knoten erhält in diesem Fall seine komplette Typisierung erst durch den Context-Knoten, für den ein Mapping auf den externen Knoten definiert wird. Gegen

einen so angelegten Knoten kann im zugehörigen Controller jedoch nur dynamisch programmiert werden, da seine Struktur von der umgebenden Component abhängt und somit zur Designzeit noch nicht bekannt ist. Das prominenteste Beispiel für eine dynamische Component ist die ALV-Component.

In vielen Fällen lässt sich ein gewünschtes Resultat sowohl über ein einfaches als auch über ein externes Mapping erreichen, wenn das Design der Anwendung entsprechend umgestellt wird. Haben Sie in Ihrer Architektur eine zentrale Component für den Datenaustausch vorgesehen, bietet sich meist das einfache Mapping der eingebetteten Component auf die umgebende Component an. Haben Sie jedoch eine generische Component für die Anzeige von Daten aus einer lokalen Component vorgesehen, ist das externe Mapping meist die bessere Wahl.

Das Anlegen eines externen Mappings unterscheidet sich deutlich vom Anlegen eines einfachen Mappings. Das externe Mapping wird nicht im Controller der verwendenden Component definiert, sondern in einer extra hierfür vorhandenen Benutzeroberfläche. Gehen Sie dazu wie folgt vor:

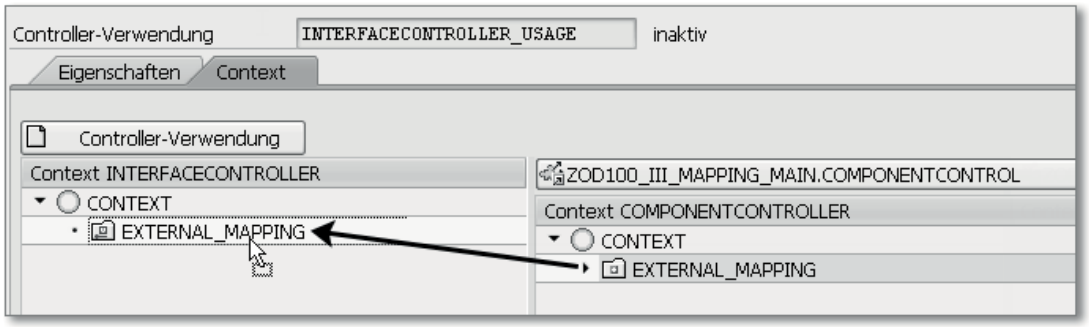
1. Stellen Sie sicher, dass der zu mappende Knoten in der eingebetteten Component als Interface-Knoten und als **Input-Element (ext.)** definiert ist. Öffnen Sie hierzu die Eigenschaften des Knotens auf der Registerkarte **Context** im Component-Controller der eingebetteten Component.

2. Tragen Sie eine Component-Verwendung für die einzubettende Component in der umgebenden Component ein.

3. Öffnen Sie den Pfad **Component-Verwendungen ▸ <Component-Verwendungsname>**, und wählen Sie dort im Kontextmenü **Controller-Verwendung anlegen** aus. Öffnen Sie anschließend den Sub-Knoten `INTERFACECONTROLLER_USAGE`. Definieren Sie im nun vorliegenden Dialog das externe Mapping:

   – Tragen Sie eine Component-Verwendung des lokalen Component-Controllers ein. Klicken Sie dazu auf der Registerkarte **Eigenschaften** auf das Symbol  . Wählen Sie den gewünschten lokalen Controller aus, und schließen Sie das Pop-up.

   – Wechseln Sie anschließend auf die Registerkarte **Context**, und legen Sie das externe Mapping an. Ziehen Sie den lokalen, zu mappenden Knoten von der rechten Bildhälfte per Drag & Drop zum Interface-Controller auf der linken Seite.



Externes Mapping anlegen

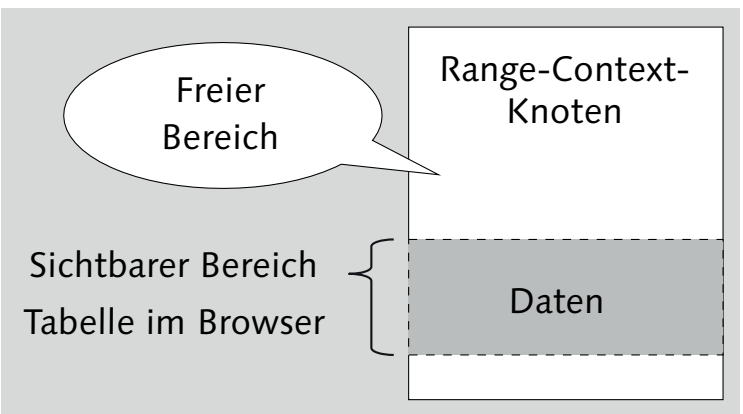
Vor allem die Möglichkeit, extern gemappte Knoten während der Designzeit untypisiert zu lassen, bietet Ihnen eine große Flexibilität beim Mapping unterschiedlicher Components. Von dieser Flexibilität machen daher vor allem generische Sub-Components Gebrauch. Für einfache und weniger komplexe Components reicht meistens das einfache Mapping aus.

Tipp 31

Range-Context-Knoten verwenden

Context-Knoten speichern häufig sehr große Tabellen mit mehreren Tausend Zeilen. Meist wird davon jedoch nur ein Bruchteil der Daten im Context benötigt, typischerweise ist dies der Ausschnitt der sichtbaren Tabellenzeilen. Was liegt daher näher, als nur den sichtbaren Ausschnitt der Daten im Context zur Speicheroptimierung abzulegen?

Mithilfe von Range-Context-Knoten können Sie große Knoten speichereffizient im Context ablegen. Das Grundprinzip ist dabei, dass nicht immer der gesamte Datenbestand eines Knotens im Context liegen muss, da meist immer nur ein kleiner Ausschnitt der Daten gebraucht wird. Dies ist typischerweise bei in Tabellen dargestellten Context-Knoten der Fall, da hier letztlich meist nur ungefähr zehn Zeilen einer Tabelle zum gleichen Zeitpunkt angezeigt werden. Der nicht angezeigte Bereich des Context-Range-Knotens wird dann erst gar nicht in den Speicher geladen.



Grundprinzip von Range-Context-Knoten

In diesem Tipp zeige ich Ihnen am Beispiel von Flugverbindungen aus der Datenbanktabelle `SFLIGHT`, wie Sie Range-Context-Knoten anlegen und verwenden können.

Und so geht's

Im Unterschied zu normalen Context-Knoten können Sie Range-Context-Knoten nur dynamisch zur Laufzeit erzeugen. Dazu verwenden Sie die Methode `add_new_child_node( )` des Interface `IF_WD_CONTEXT_NODE_INFO`. Durch das Setzen des Parameters `IS_RANGE_NODE` auf `X` legen Sie fest, dass der zu erzeugende Knoten ein Range-Context-Knoten sein soll. Über die Parameter `SUPPLY_METHOD` und `SUPPLY_OBJECT` legen Sie fest, welche Range-Supply-Methode verwendet werden soll.

Beginnen Sie mit dem Anlegen des Flugverbindungsbeispiels. Legen Sie eine neue Test-Component und Testanwendung an. Wechseln Sie in den View der neuen Component, und implementieren Sie das folgende Listing zur Erzeugung eines Range-Context-Knotens und einer einfachen Tabelle in der Methode `wddomodifyview( )`. Der Knoten wird über eine im folgenden Schritt anzulegende Supply-Methode `supply_sflight( )` mit Daten versorgt. Nach der Erzeugung der Range-Tabelle ermitteln Sie den Umfang der verfügbaren Flugverbindungen und übergeben die Zahl über `set_max_element_count( )` an den Range-Context-Knoten.

```
CHECK first_time EQ abap_true.

DATA: lo_nd_sflight_rng TYPE REF TO if_wd_context_node_range,
      lo_container      TYPE REF TO cl_wd_transparent_container,
      lv_sflight_count  TYPE i.

* Erzeuge einen Range-Context-Knoten
wd_context->get_node_info( )->add_new_child_node(
  EXPORTING
    supply_method       = 'SUPPLY_SFLIGHT'
    supply_object       = me
    static_element_type = 'SFLIGHT'
    name                = 'SFLIGHT'
    is_range_node       = abap_true ).

lo_nd_sflight_rng ?= wd_context->get_child_node( 'SFLIGHT' ).

* Ermittle Anzahl der insgesamt verfügbaren Elemente
SELECT COUNT(*) FROM sflight INTO lv_sflight_count.
lo_nd_sflight_range->set_max_element_count( lv_sflight_count ).

* Generiere eine dynamische Tabelle
lo_container ?= view->get_root_element( ).
cl_wd_dynamic_tool=>create_c_table_from_node(
  EXPORTING
```

```
    ui_parent = lo_container
    node      = lo_nd_sflight_range ).
```

Implementierung von Methode »wddomodifyview( )«

Legen Sie nun die Supply-Funktion `supply_sflight( )` für den Range-Context-Knoten an. Diese Supply-Funktion muss dabei, im Unterschied zu klassischen Supply-Funktionen, nicht vom **Methoden-Typ** `Supply-Funktion`, sondern vom Typ `Methode` sein. Ein weiterer Unterschied der Range-Supply-Methode ist, dass sie mit `FROM_INDEX` und `TO_INDEX` zusätzliche Parameter für die Eingrenzung des angefragten Range-Bereichs besitzt. Definieren Sie für die Supply-Funktion daher manuell die folgenden Importing-Parameter:

Importing-Parameter	Referenz- bzw. Datentyp
NODE	IF_WD_CONTEXT_NODE_RANGE
PARENT_ELEMENT	IF_WD_CONTEXT_ELEMENT
FROM_INDEX	I
TO_INDEX	I

Importing-Parameter der Supply-Funktion

Implementieren Sie nun die Supply-Funktion. Im ersten Schritt müssen Sie die Flugverbindungen des angeforderten Ranges von der Datenbank lesen. Hierzu geben Sie beim `SELECT` mit der Ergänzung `UP TO to_index ROWS` den oberen angefragten Indexwert der Supply-Methode an. Da Sie beim `SELECT` leider keine untere Lesegrenze angeben können, müssen Sie die Werte unterhalb des Parameterwerts von `FROM_INDEX` im nächsten Schritt in einem `LOOP` verwerfen. Damit der Range-Context-Knoten den zu übergebenden Ausschnitt in den Gesamtbereich einsortieren kann, muss außerdem jede Tabellenzeile mit einem Zeilenindex versehen werden. Zuletzt können Sie den angeforderten Tabellenbereich an den Range-Context-Knoten über die Methode `set_table_range( )` übergeben. Das folgende Listing zeigt Ihnen das gesamte Listing der Range-Supply-Methode `supply_sflight( )`.

```
TYPES:
  BEGIN OF ty_s_sflight_index,
    index   TYPE int4.
    INCLUDE TYPE sflight.
TYPES: END OF ty_s_sflight_index.

DATA: lt_sflight_range TYPE TABLE OF ty_s_sflight_index,
```

```
      ls_sflight_range TYPE ty_s_sflight_index,
      lt_sflight       TYPE SORTED TABLE OF sflight
                       WITH UNIQUE KEY carrid connid fldate.

FIELD-SYMBOLS: <sflight> LIKE LINE OF lt_sflight.

SELECT * FROM sflight INTO CORRESPONDING FIELDS OF TABLE
  lt_sflight UP TO to_index ROWS ORDER BY carrid connid fldate.

* Konvertiere die SFLIGHT-Tabelle in eine Range-Context-Zeilen-
* Struktur mit vorangestelltem Zeilenindex
LOOP AT lt_sflight ASSIGNING <sflight> FROM from_index.
  MOVE-CORRESPONDING <sflight> TO ls_sflight_range.
  ls_sflight_range-index = sy-tabix.
  APPEND ls_sflight_range TO lt_sflight_range.
ENDLOOP.

* Übergebe den angeforderten Range an den Context
node->set_table_range(
   new_items              = lt_sflight_range
   index                  = from_index
   invalidate_child_nodes = abap_false ).
```

Supply-Methode »supply_sflight( )«

Damit ist das Beispiel mit dem Range-Supply-Knoten fertig. Aktivieren Sie die Component. Um das Beispiel besser nachvollziehen zu können, empfiehlt es sich, zu Beginn von `wddomodifyview( )` und `supply_sflight( )` je einen Breakpoint zu setzen. Starten Sie anschließend die Testanwendung, und testen Sie die Tabelle.

Fluggesellschaft	Flugnummer	Flugdatum	Flugpreis	Währ. d. Flugg.
AC	0820	20.12.2002	1.222,00	CAD
AF	0820	23.12.2002	2.222,00	EUR
LH	0400	28.02.1995	899,00	DEM
LH	0454	17.11.1995	1.499,00	DEM
LH	0455	06.06.1995	1.090,00	USD

Test der auf einem Range-Context-Knoten basierenden Tabelle mit Flugverbindungen

Tipp 32

Context-Change-Log verwenden

Ein Benutzer ändert in einer Tabelle mit mehreren Hundert Zeilen einige Datensätze. Mithilfe des Context-Change-Logs können Sie diese Änderungen sehr elegant speichern.

Beim Context-Change-Log handelt es sich um ein Protokoll, das Änderungen am Context durch den Benutzer überwacht und diese in einer Tabelle aufzeichnet. Mithilfe des hierbei entstehenden Protokolls können Sie anschließend gezielt Benutzereingaben weiterverarbeiten und dabei z. B. nur geänderte Daten auf der Datenbank speichern. Wie genau dies funktioniert, erfahren Sie am Beispiel von Flugbuchungen in diesem Tipp.

› Und so geht's

Legen Sie zur Vorbereitung des Change-Log-Beispiels eine neue Test-Component und Testanwendung an. Wechseln Sie in den View, und legen Sie den Context-Knoten `SBOOK` mit der Kardinalität `0..n` und der zugrunde liegenden ABAP-Dictionary-Struktur `SBOOK` an. Übernehmen Sie eine kleine Auswahl der Datenbankfelder als Attribute in den Context-Knoten. Generieren Sie anschließend mit dem Code Wizard eine auf der Knotenstruktur basierende Tabelle, wobei Sie als **Standard Cell-Editor** das UI-Element `TextEdit` auswählen. Fügen Sie nun eine Toolbar in die Tabelle ein, und legen Sie in dieser einen Button mit der Beschriftung `Neue Zeile` zum Hinzufügen eines neuen Context-Elements in den Knoten an. Legen Sie aus den Eigenschaften des Buttons heraus eine neue Aktion für das Ereignis `onAction` an, und implementieren Sie das folgende Listing zur Erzeugung von neuen Context-Elementen im `SBOOK`-Knoten.

```
DATA: lo_nd_sbook TYPE REF TO if_wd_context_node,
      lo_el_sbook TYPE REF TO if_wd_context_element.

lo_nd_sbook = wd_context->get_child_node( wd_this->wdctx_sbook ).
lo_el_sbook = lo_nd_sbook->create_element( ).

lo_nd_sbook->bind_element(
  new_item             = lo_el_sbook
  set_initial_elements = abap_false  ).
```

Aktionsimplementierung für den Button »Neue Zeile«

Wenden wir uns nun dem Context-Change-Log zu. Standardmäßig ist das Change-Log in jedem Controller deaktiviert. Bei Bedarf muss es daher im jeweiligen Context der Component aktiviert werden. Dies geschieht über das Interface `IF_WD_CONTEXT`. Sie erhalten die Referenz auf den Context durch den Aufruf der Controller-Methode `wd_context->get_context( )`. Anschließend können Sie das Change-Log durch einen Aufruf der Methode `enable_context_change_log( )` aktivieren. Öffnen Sie daher im View die Methode `wddoinit( )`, und implementieren Sie das folgende Listing.

```
DATA lo_context TYPE REF TO if_wd_context.
lo_context = wd_context->get_context( ).
lo_context->enable_context_change_log( ).
```

Methode »wddoinit( )«: Aktivierung des Context-Change-Logs

Wenn Sie nun die Component aktivieren und die Anwendung starten, können Sie durch den Klick auf den Button **Neue Zeile** eine neue Flugbuchung in die `SBOOK`-Tabelle einfügen und anschließend Buchungen eingeben. Um nun die Änderungen des Change-Logs auszulesen, müssen Sie die Änderungen vom Context abholen. Hierzu können Sie die Methode `get_context_change_log( )` des Interface `IF_WD_CONTEXT` verwenden.

Fügen Sie hierzu in der Toolbar einen zweiten Button mit der Beschriftung `Änderungen auslesen` ein, und legen Sie aus dem Button heraus eine neue Aktion für das Ereignis `onAction` an. Zum Auslesen und Ausgeben des Context-Change-Logs, das in der Praxis typischerweise beim Drücken der Taste ↵ oder vor dem Speichervorgang stattfinden würde, implementieren Sie das folgende Listing.

```
DATA: lo_msg_mgr TYPE REF TO if_wd_message_manager,
      lv_text    TYPE string,
      lv_index   TYPE string,
      lo_context TYPE REF TO if_wd_context,
```

```
      lt_changes TYPE wdr_context_change_list,
      ls_change  TYPE wdr_context_change.
FIELD-SYMBOLS: <ls_new_value> TYPE any.

* Vorbereitung
lo_msg_mgr =
    wd_comp_controller->wd_get_api( )->get_message_manager( ).

* Lese das Change-Log aus
lo_context = wd_context->get_context( ).
lt_changes =
  lo_context->get_context_change_log( and_reset = abap_true ).

* Gebe die einzelnen Attributänderungen in einer Nachricht aus
LOOP AT lt_changes INTO ls_change
  WHERE change_kind EQ 'A'. " Nur Attributänderungen anzeigen

  ASSIGN ls_change-new_value->* TO <ls_new_value>.
  lv_index = ls_change-element_index.

  CONCATENATE 'Änderung in Zeile ' lv_index ': Feld '
    ls_change-attribute_name ' Neuer Wert: ' <ls_new_value>
    INTO lv_text RESPECTING BLANKS.

  lo_msg_mgr->report_message(
      message_text = lv_text
      message_type = if_wd_message_manager=>co_type_info ).
ENDLOOP.
```

Änderungen aus dem Change-Log auslesen und als Nachricht ausgeben

Aktivieren Sie nun die Component, und starten Sie die Anwendung. Fügen Sie eine neue Zeile in die Tabelle ein, und tragen Sie diese Werte ein. Klicken Sie anschließend auf den Toolbar-Button **Änderungen auslesen**, um die Änderungen vom Context-Change-Log abzuholen. Über den Message-Manager werden die attributbezogenen Änderungen (vergleiche mit `WHERE`-Bedingung `CHANGE_KIND EQ 'A'`) des Change-Logs im Bild ausgegeben.

Ausgabe von Benutzereingaben im Context-Change-Log

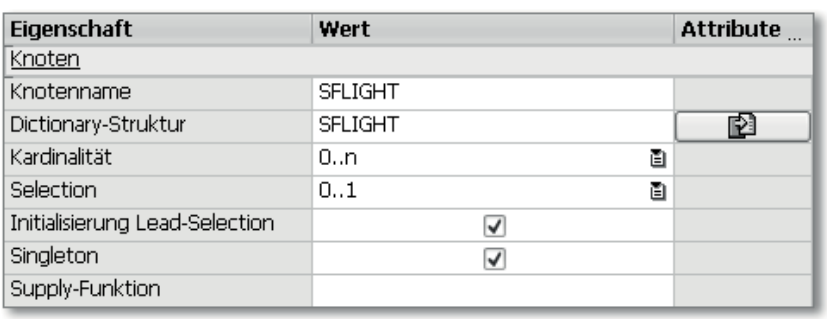
Beachten Sie, dass das Context-Change-Log keine programmatischen Änderungen am Context aufzeichnet. Ändern Sie also beispielsweise über die Methode `set_attribute( )` an einem Context-Element einen Attributwert, wird diese Änderung nicht automatisch vom Change-Log aufgezeichnet. Möchten Sie programmatische Änderungen am Context aufzeichnen, können Sie hierzu die Methode `add_context_attribute_change( )` des Interface `IF_WD_CONTEXT` verwenden.

Tipp 33

Singleton-Eigenschaft verwenden

Sicherlich haben Sie schon einmal vom Singleton-Entwurfsmuster gehört. Dieses stellt die Instanzierung eines einzigen Objekts pro Instanz sicher. Auch Knoten können als Singleton-Knoten eingestellt werden. Doch wieso sollte man nur einen Knoten instanzieren? Die Antwort finden Sie in diesem Tipp.

Jeder Context-Knoten besitzt die Eigenschaft `Singleton`. Nach der Aktivierung der Eigenschaft `Singleton` wird von dem jeweiligen Context-Knoten immer nur eine einzelne Instanz erzeugt, unabhängig von der Anzahl der Context-Elemente des darüberliegenden Knotens. Die Verwendung von Singleton-Knoten ist daher insbesondere bei mehrstufigen Knotenhierarchien mit großen Datenvolumen sinnvoll, da sonst eine vollständige Instanzierung einer umfangreichen Knotenhierarchie entsprechend viel Zeit und ebenso viel Speicherplatz benötigt.



Eigenschaft	Wert	Attribute ...
Knoten		
Knotenname	SFLIGHT	
Dictionary-Struktur	SFLIGHT	
Kardinalität	0..n	
Selection	0..1	
Initialisierung Lead-Selection	☑	
Singleton	☑	
Supply-Funktion		

Knoteneigenschaften: Knoten SFLIGHT mit aktivem Singleton-Kennzeichen

Und so geht's

Um Ihnen die Verwendung der `Singleton`-Eigenschaft näher zu erläutern, habe ich ein Beispiel mit einer zweistufigen Knotenhierarchie aufgebaut. Die Hierarchie beginnt bei dem Knoten `SCARR`, der auf der gleichnamigen Daten-

banktabelle basiert und dank Kardinalität `0..n` beliebig viele Fluggesellschaften beinhalten kann. Der Knoten `SCARR` liegt direkt unterhalb der Wurzel des Controller-Contexts. Da die Kardinalität jeder Context-Wurzel immer `1..1` ist, sind auch alle direkt unterhalb der Wurzel liegenden Context-Knoten und Context-Elemente per Definition immer Singleton-Knoten, ohne dass Sie die Eigenschaft `Singleton` explizit aktivieren müssen.

Eine Hierarchiestufe unterhalb von `SCARR` liegt der Knoten `SFLIGHT`. Dieser beinhaltet alle Flugverbindungen der jeweils darüberliegenden Fluggesellschaft. Für jede Fluggesellschaft im Knoten `SCARR` existiert ein eigenes Context-Element, das wiederum seine eigene Knoteninstanz des Sub-Knotens `SFLIGHT` anlegt. Befinden sich in Knoten `SCARR` also 100 Fluggesellschaften, werden hierfür 100 `SCARR`-Context-Elemente und 100 Knoteninstanzen des Sub-Knotens `SFLIGHT` angelegt. Der Knoten `SFLIGHT` wiederum beinhaltet so viele Context-Elemente, wie die jeweilige Fluggesellschaft an Flugverbindungen hat. Hat jede der 100 Fluggesellschaften im Jahr 1.000 Flugverbindungen, würden für die vollständige Abbildung der gesamten Hierarchie insgesamt 100.000 Context-Elemente benötigt.

Knotenstruktur zur Designzeit

```
CONTEXT
   SCARR (Fluggesellschaft)
   CARRID
   CARRNAME
      SFLIGHT (Flugverbindungen)
      CONNID
      FLDATE
```

Daten während der Laufzeit

```
CONTEXT
   SCARR (1)
   CARRID:   LH
   CARRNAME: Lufthansa
      SFLIGHT (1.1)
      CONNID:   LH280
      FLDATE:   24.12.2013
      SFLIGHT (1.2)
      CONNID:   LH4711
      FLDATE:   31.12.2013
   SCARR (2)
   CARRID:   AA
   CARRNAME: American Airlines
      SFLIGHT (2.1)
      CONNID:   ...
      FLDATE:   ...
```

Zweistufige Knotenhierarchie: Fluggesellschaften und Flugverbindungen

Aktivieren Sie nun in dieser Hierarchie für den `SFLIGHT`-Knoten die Eigenschaft `Singleton`, werden auf dieser Hierarchieebene nur noch Context-Elemente des Eltern-Elements mit der Lead Selection instanziert. Haben Sie also im Knoten `SCARR` die Fluggesellschaft Lufthansa selektiert, enthält nur der zu Lufthansa gehörende Sub-Knoten `SFLIGHT` die zugehörigen Context-Elemente mit den Flugverbindungen. Wandert die Lead Selection vom Knoten `SCARR` zu einer anderen Fluggesellschaft, werden die Context-Elemente des

Lufthansa-`SFLIGHT`-Knotens freigegeben und neue Elemente im `SFLIGHT`-Knoten der anderen Fluggesellschaft erzeugt. Durch die Aktivierung der Singleton-Eigenschaft können Sie daher die Anzahl der Context-Elemente eines Sub-Knotens deutlich begrenzen. Die `Singleton`-Eigenschaft eignet sich besonders für die Verwendung im Zusammenspiel mit mehrstufigen Hierarchien und Supply-Funktionen, die bei Änderung einer Lead Selection auf oberer Ebene gezielt die Daten der zugehörigen Sub-Knoten auslesen.