IBM SPSS Modeler 17.1
CLEF-Entwicklerhandbuch

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen unter „Bemerkungen" auf Seite 335 gelesen werden.

Produktinformation

Diese Ausgabe bezieht sich auf Version 17, Release 1, Modifikation 0 von IBM(r) SPSS(r) Modeler und alle nachfolgenden Releases und Modifikationen, bis dieser Hinweis in einer Neuausgabe geändert wird.

Diese Veröffentlichung ist eine Übersetzung des Handbuchs
IBM SPSS Modeler 17.1 CLEF Developer's Guide,
herausgegeben von International Business Machines Corporation, USA

Herausgegeben von:
TSC Germany
Kst. 2877
August 2015

Inhaltsverzeichnis

Vorwort

IBM® SPSS Modeler ist die auf Unternehmensebene einsetzbare Data-Mining-Workbench von IBM. Mit SPSS Modeler können Unternehmen und Organisationen die Beziehungen zu ihren Kunden bzw. zu den Bürgern durch ein tief greifendes Verständnis der Daten verbessern. Organisationen verwenden die mithilfe von SPSS Modeler gewonnenen Erkenntnisse zur Bindung profitabler Kunden, zur Ermittlung von Cross-Selling-Möglichkeiten, zur Gewinnung neuer Kunden, zur Ermittlung von Betrugsfällen, zur Reduzierung von Risiken und zur Verbesserung der Verfügbarkeit öffentlicher Dienstleistungen.

Die visuelle Benutzerschnittstelle von SPSS Modeler erleichtert die Anwendung des spezifischen Fachwissens der Benutzer, was zu leistungsstärkeren Vorhersagemodellen führt und die Zeit bis zur Lösungserstellung verkürzt. SPSS Modeler bietet zahlreiche Modellierungsverfahren, beispielsweise Algorithmen für Vorhersage, Klassifizierung, Segmentierung und Assoziationserkennung. Nach der Modellerstellung ermöglicht IBM SPSS Modeler Solution Publisher die unternehmensweite Bereitstellung des Modells für Entscheidungsträger oder in einer Datenbank.

Informationen zu IBM Business Analytics

Die Software IBM Business Analytics liefert umfassende, einheitliche und korrekte Informationen, mit denen Entscheidungsträger die Unternehmensleistung verbessern können. Ein umfassendes Portfolio aus Anwendungen für Business Intelligence, Vorhersageanalyse, Finanz- und Strategiemanagement sowie Analysen bietet Ihnen sofort klare und umsetzbare Einblicke in die aktuelle Leistung und gibt Ihnen die Möglichkeit, zukünftige Ergebnisse vorherzusagen. Durch umfassende Branchenlösungen, bewährte Vorgehensweisen und professionellen Service können Unternehmen jeder Größe die Produktivität maximieren, Entscheidungen automatisieren und bessere Ergebnisse erzielen.

Als Teil dieses Portfolios unterstützt IBM SPSS Predictive Analytics-Software Unternehmen dabei, zukünftige Ereignisse vorherzusagen und proaktiv Maßnahmen zu ergreifen, um bessere Geschäftsergebnisse zu erzielen. Kunden aus Wirtschaft, öffentlichem Dienst und staatlichen Lehr- und Forschungseinrichtungen weltweit nutzen IBM SPSS-Technologie als Wettbewerbsvorteil für die Kundengewinnung, Kundenbindung und Erhöhung der Kundenumsätze bei gleichzeitiger Eindämmung der Betrugsmöglichkeiten und Minderung von Risiken. Durch die Einbindung von IBM SPSS-Software in ihre täglichen Operationen wandeln sich Organisationen zu "Predictive Enterprises", die Entscheidungen auf Geschäftsziele ausrichten und automatisieren und einen messbaren Wettbewerbsvorteil erzielen können. Wenn Sie weitere Informationen wünschen oder Kontakt zu einem Mitarbeiter aufnehmen möchten, besuchen Sie die Seite http://www.ibm.com/spss.

Technical Support

Kunden mit Wartungsvertrag können den Technical Support in Anspruch nehmen. Kunden können sich an den Technical Support wenden, wenn sie Hilfe bei der Arbeit mit IBM Produkten oder bei der Installation in einer der unterstützten Hardwareumgebungen benötigen. Zur Kontaktaufnahme mit dem Technical Support besuchen Sie die IBM Website unter http://www.ibm.com/support. Sie müssen bei der Kontaktaufnahme Ihren Namen, Ihre Organisation und Ihre Supportvereinbarung angeben.

Kapitel 1. Übersicht

Einführung in CLEF

Component-Level Extension Framework (CLEF) ist ein Mechanismus, mit dem der Standardfunktionalität von IBM SPSS Modeler benutzerdefinierte Erweiterungen hinzugefügt werden können. Eine Erweiterung enthält in der Regel eine gemeinsam genutzte Bibliothek, z. B. eine Datenverarbeitungsroutine oder einen Modellierungsalgorithmus, die bzw. der IBM SPSS Modeler hinzugefügt und als neuer Menüeintrag oder als neuer Knoten in der Knotenpalette zur Verfügung gestellt wird.

Dazu benötigt IBM SPSS Modeler Informationen über das benutzerdefinierte Programm, unter anderem dessen Namen, die an IBM SPSS Modeler übergebenen Befehlsparameter und Informationen darüber, wie IBM SPSS Modeler dem Programm Optionen und dem Benutzer Ergebnisse bereitstellen soll. Diese Informationen stellen Sie in einer XML-Datei, der **Spezifikationsdatei**, bereit. IBM SPSS Modeler überträgt die Informationen dieser Datei in einen neuen Menüeintrag oder eine Knotendefinition.

CLEF bietet unter anderem folgende Vorteile:

- Es stellt eine einfach zu verwendende, äußerst flexible und robuste Umgebung bereit, mit der Systementwickler, Berater und Endbenutzer neue Funktionen in IBM SPSS Modeler integrieren können.
- Es stellt sicher, dass Erweiterungsmodule genauso aussehen und sich genauso verhalten wie systemeigene IBM SPSS Modeler-Module.
- Es stellt sicher, dass die Geschwindigkeit und Effizienz der Erweiterungsknoten so nahe wie möglich an die der systemeigenen IBM SPSS Modeler-Knoten herankommen.

Systemarchitektur

Wie IBM SPSS Modeler verwendet CLEF eine zweischichtige Client-/Serverarchitektur, wobei sich die Schichten auf demselben Computer oder auf zwei verschiedenen Computern befinden können.

Clientseitige Komponenten

Die Komponenten der Clientschicht werden in nachfolgender Abbildung gezeigt.

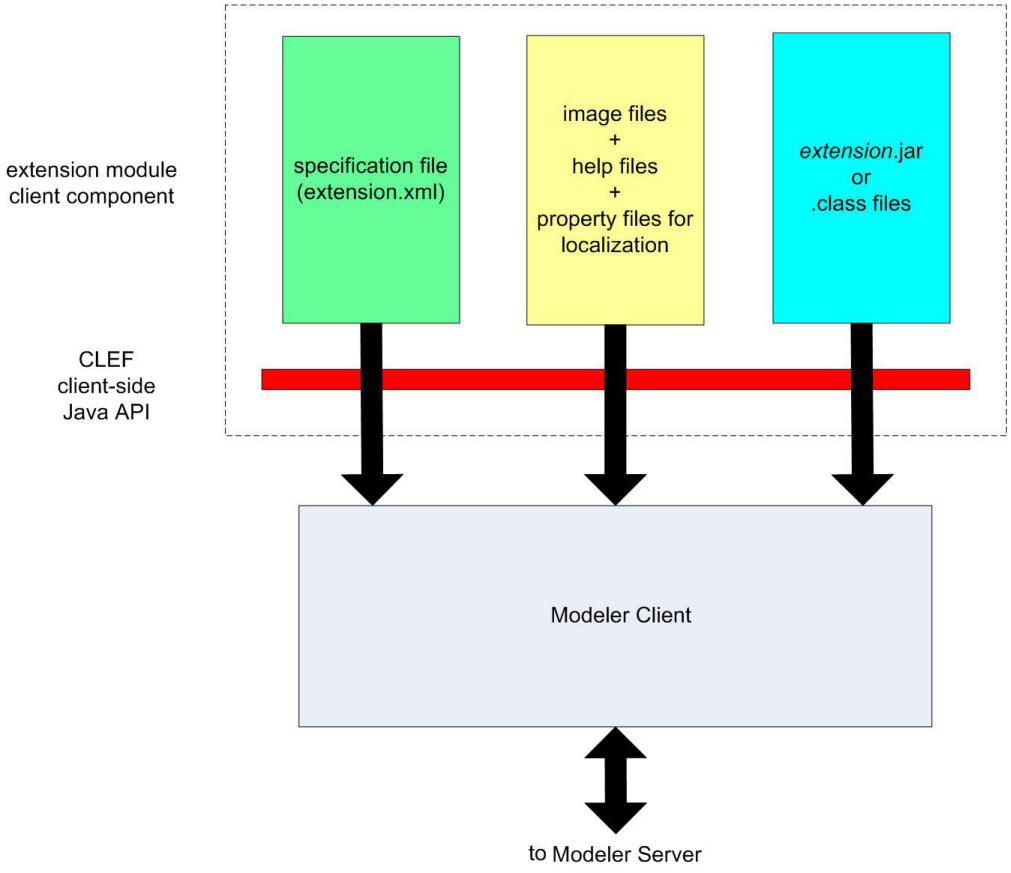

Abbildung 1. Clientseitige Komponenten

- **Spezifikationsdatei.** Definiert die Eigenschaften, Formate, Datenmodelländerungen, Steuerelemente und andere Merkmale der Erweiterung.
- **Bilddateien.** Enthalten die Bilder für die Darstellung eines Knotens in der Erweiterung.
- **Hilfedateien.** Enthalten die Hilfeinformationen für die Erweiterung.
- **Eigenschaftendateien.** Enthalten Textzeichenfolgen mit den Namen, Beschriftungen und Nachrichten, die von der Erweiterung auf dem Bildschirm angezeigt werden.
- **Java-Dateien (.jar oder .class).** Enthalten die von der Erweiterung verwendeten Java-Ressourcen.
- **Java-API (Anwendungsprogrammierschnittstelle).** Kann von Erweiterungen verwendet werden, für die Steuerelemente, Benutzerschnittstellenkomponenten oder Interaktivitätselemente erforderlich sind, die nicht in der Spezifikationsdatei bereitgestellt werden können.

Serverseitige Komponenten

Die Komponenten der Serverschicht werden in nachfolgender Abbildung gezeigt.

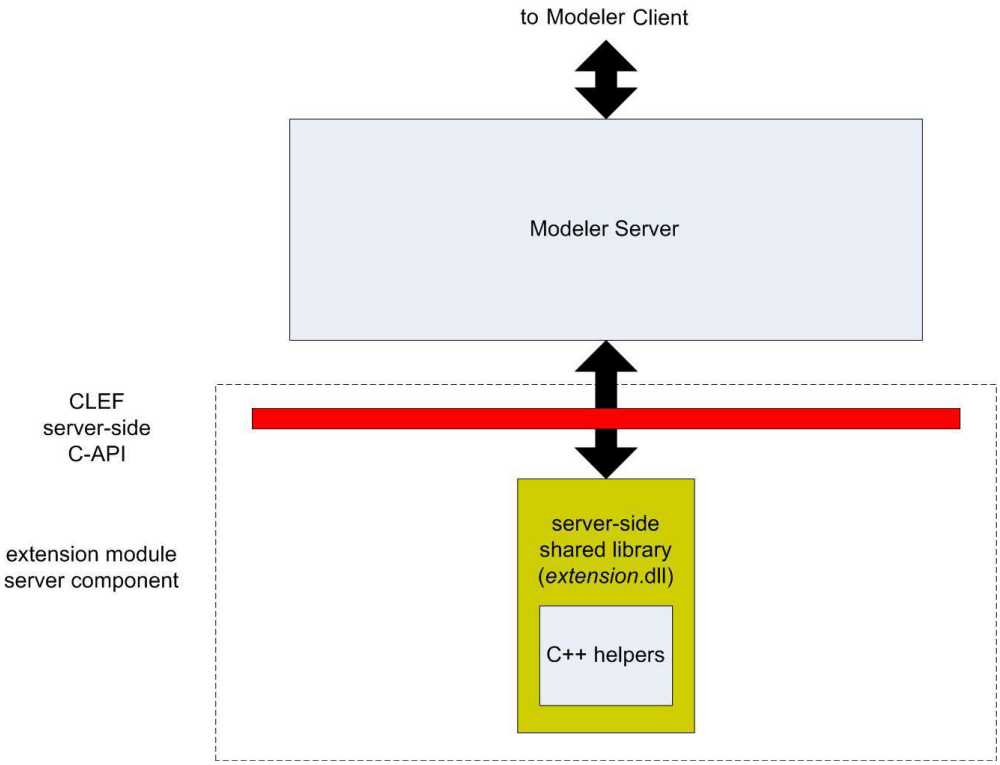

Abbildung 2. Serverseitige Komponenten

- **C-basierte API für gemeinsam genutzte Bibliotheken.** Definiert Aspekte wie die Festlegung und das Abrufen von Ausführungseinstellungen, die Persistenz dieser Einstellungen, Ausführungsfeedback, die Jobsteuerung (z. B. Unterbrechung der Ausführung), die SQL-Generierung und die zurückgegebenen Objekte.
- **Serverseitige gemeinsam genutzte Bibliothek.** Eine DLL (Dynamic Link Library), die die Knotenausführung unterstützt. Die C++-Helper sind Wrapper für einige der C-basierten APIs, die als Quellcode bereitgestellt werden und problemlos in C++-Module für CLEF kompiliert werden können.

Funktionen von CLEF

In den folgenden Abschnitten werden die wichtigsten Funktionen von CLEF vorgestellt:

- Spezifikationsdatei
- Knoten
- Datenmodell
- Eingabe- und Ausgabedateien
- Anwendungsprogrammierschnittstellen (APIs)

Spezifikationsdatei

Bei der Spezifikationsdatei von CLEF handelt es sich um eine XML-Datei mit strukturierten Spezifikationen, die das Verhalten der neuen Erweiterung beschreiben. Eine Spezifikationsdatei beschreibt folgende Komponenten:

- Die von der Erweiterung benötigten, gemeinsam verwendeten Ressourcen (z. B. lokalisierte Textbundles und serverseitige gemeinsam verwendete Bibliotheken)
- Allgemeine Definitionen wie Datei- oder Eigenschaftstypen
- Neue Objekte, wie Knoten und Ausgabemodelle, die vom Endbenutzer verwendet werden können

Beim Starten von IBM SPSS Modeler werden die Spezifikationsdateien aus ihrem jeweiligen Speicherort geladen, damit die in den Dateien definierten Funktionen sofort zur Verfügung stehen.

Weitere Informationen finden Sie in Kapitel 4, „Spezifikationsdatei", auf Seite 31.

Knoten

Wenn Sie IBM SPSS Modeler eine Erweiterung hinzufügen, die einen neuen Knoten implementieren soll, müssen Sie zunächst entscheiden, welchen Knotentyp Sie erstellen möchten (ob der Knoten z. B. ein Modell generieren oder lediglich Daten transformieren soll). Weitere Informationen finden Sie im Thema „Überblick über die Knoten" auf Seite 9.

Nachdem Sie die Spezifikationsdatei und alle erforderlichen Java-Klassen und gemeinsam verwendeten Bibliotheken erstellt haben, kopieren Sie die Dateien in bestimmte Verzeichnisse, in denen IBM SPSS Modeler die Dateien lesen kann. Wenn Sie IBM SPSS Modeler danach das nächste Mal starten, wird der neue, betriebsbereite Knoten der entsprechenden Palette hinzugefügt.

Datenmodell

Das **Datenmodell** stellt die Struktur der Daten dar, die durch den IBM SPSS Modeler-Stream fließen. Da das Modell die Daten an diesem Punkt des Streams beschreibt, entspricht es den im Typknoten angezeigten Informationen. Es listet die Namen der an einem bestimmten Punkt vorhandenen Felder auf und beschreibt deren Typ.

Beachten Sie beim Hinzufügen eines Knotens zu IBM SPSS Modeler, wie das an den Knoten übergebene Datenmodell das Verhalten dieses Knotens beeinflusst. Ein Ableitungsknoten nimmt beispielsweise ein Eingabedatenmodell an, fügt ein neues Feld hinzu und erstellt ein Ausgabedatenmodell, das an den nächsten Knoten im IBM SPSS Modeler-Stream übergeben wird. Ein Diagrammknoten hingegen nimmt ein Eingabedatenmodell an, erstellt aber kein Ausgabedatenmodell, da keine Daten an nachfolgende Knoten weitergeleitet werden müssen. IBM SPSS Modeler muss darüber informiert werden, was mit dem Datenmodell geschieht, damit nachfolgende Knoten korrekte Informationen über die verfügbaren Felder angeben können. Mit den Datenmodellinformationen der Spezifikationsdatei erhält IBM SPSS Modeler die notwendigen Informationen, um das Datenmodell im gesamten Stream konsistent zu halten.

Je nachdem, ob die Daten in den Knoten, aus dem Knoten oder durch den Knoten fließen, muss die Spezifikationsdatei das Datenmodell für die Eingabe, die Ausgabe oder beides beschreiben. Ein CLEF-Knoten kann das Datenmodell auf zweierlei Weise beeinflussen: durch Hinzufügen neuer Felder zu den Feldern, die an den Knoten übergeben werden, oder durch Ersetzen der an den Knoten übergebenen Felder durch neue Felder, die vom Programm selbst erzeugt werden. Die Auswirkungen eines CLEF-Knotens auf das Datenmodell werden im Element `OutputDataModel` der Spezifikationsdatei definiert. Weitere Informationen finden Sie im Thema „Ausgabedatenmodell" auf Seite 59.

Eingabe- und Ausgabedateien

Sie können festlegen, dass vor der Ausführung eines CLEF-Knotens mindestens eine temporäre Datei generiert wird. Diese Dateien werden als **Eingabedateien** bezeichnet, da sie die Eingabe für die Ausführung des Knotens auf dem Server bereitstellen. Ein Modellerstellungsknoten kann beispielsweise über einen Modellcontainer verfügen, dessen Inhalt bei der Ausführung des Knotens in eine bestimmte Eingabedatei übertragen wird. Weitere Informationen finden Sie im Thema „Eingabedateien" auf Seite 56.

Andere temporäre Dateien werden während der Ausführung des Knotens auf dem Server generiert. Diese enthalten z. B. das Ergebnis eines Modell- oder Dokumenterstellungsknotens. Diese Dateien werden als **Ausgabedateien** bezeichnet. Sie werden nach der Knotenausführung an den Client zurückgeleitet. Weitere Informationen finden Sie im Thema „Ausgabedateien" auf Seite 56.

Anwendungsprogrammierschnittstellen (APIs)

Je nachdem, welche Funktion Ihre Erweiterung haben soll, müssen Sie zu deren Entwicklung unter Umständen auch eine Anwendungsprogrammierschnittstelle (API) verwenden. Für die Festlegung der Verarbeitungsschritte einer einfachen Datentransformation reicht unter Umständen die Spezifikationsdatei aus. Für erweiterte Funktionen müssen Sie jedoch eine oder mehrere der verfügbaren APIs hinzuziehen:

- Clientseitige CLEF-API
- Serverseitige CLEF-API
- Predictive Server-API (PSAPI)

Bei der **clientseitigen API** von CLEF handelt es sich um eine Java-API, die von Erweiterungen verwendet werden kann, für die zusätzliche Steuerelemente, Benutzerschnittstellenkomponenten oder Interaktivitätselemente erforderlich sind, die nicht in der Spezifikationsdatei bereitgestellt werden.

Bei der **serverseitigen API** von CLEF handelt es sich um eine C-basierte API, in der Aspekte wie die Festlegung und das Abrufen von Ausführungseinstellungen, die Persistenz dieser Einstellungen, Ausführungsrückmeldungen, die Jobsteuerung (z. B. Unterbrechung der Ausführung), die SQL-Generierung und die zurückgegebenen Objekte definiert werden.

Bei der **Prective Server-API** handelt es sich um eine Java-API, die Anwendungen, die Data-Mining-Funktionen und die Mechanismen für die Vorhersage benötigen, die Funktionalität von IBM SPSS Modeler bereitstellt.

Weitere Informationen finden Sie in Kapitel 9, „Programmierung”, auf Seite 177.

Dateistruktur

Eine CLEF-Erweiterung besteht aus zwei Komponentensätzen:

- Clientseitige Komponenten
- Serverseitige Komponenten

Zu den **clientseitigen Komponenten** gehören die Spezifikationsdatei der Erweiterung, Java-Klassen und JAR-Dateien, Eigenschaftenbundles mit lokalisierbaren Ressourcen sowie Bild- und Hilfedateien.

Die **serverseitigen Komponenten** umfassen gemeinsam verwendete Bibliotheken und DLLs, die bei der Ausführung eines Erweiterungsknotens erforderlich sind.

Clientseitige Komponenten

Clientseitige Komponenten werden im Installationsverzeichnis von IBM SPSS Modeler im Ordner `\ext\lib` installiert. Es gibt folgende clientseitige Komponenten:

- Spezifikationsdatei
- Java-Klassen und `JAR`-Dateien
- Eigenschaftendateien
- Bilddateien
- Hilfedateien

Erweiterungsordner

Jede Erweiterung wird in einem eigenen **Erweiterungsordner** direkt unter dem Verzeichnis `\ext\lib` gespeichert.

Für den Erweiterungsordner wird folgendes Namensschema empfohlen:

`Provider-Tag.ID`

Dabei ist *Provider-Tag* die Kennung des Providers (providerTag) aus dem Element `ExtensionDetails` der Spezifikationsdatei und *ID* ist die Erweiterungskennung aus diesem Element.

Beispiel: Das Element `ExtensionDetails` beginnt wie folgt:

```
<ExtensionDetails providerTag="myco" id="sorter" ... />
```

Der Erweiterungsordner sollte dann den Namen `myco.sorter` erhalten.

Spezifikationsdatei

Die Spezifikationsdatei muss den Namen `extension.xml` erhalten und sie muss in der obersten Ebene des Erweiterungsordners gespeichert werden. Für das vorangegangene Beispiel würde der Pfad der Erweiterungsdatei relativ zum Installationsverzeichnis von IBM SPSS Modeler wie folgt lauten:

```
\ext\lib\myco.sorter\extension.xml
```

Java-Klassen und JAR-Dateien

Erweiterungen, die die clientseitige Java-API verwenden, enthalten kompilierten Java-Code. Dieser Code kann einer Gruppe von `.class`-Dateien oder verpackt in einer `JAR`-Datei bereitgestellt werden.

Die `.class`-Dateien von Java werden relativ zum obersten Erweiterungsordner angegeben. Eine Klasse, die die Schnittstelle `ActionHandler` implementiert, kann beispielsweise folgenden Pfad haben:

```
com.my_example.my_extension.MyActionHandler
```

In diesem Fall muss sich die `.class`-Datei im folgenden Ordner im Installationsverzeichnis von IBM SPSS Modeler befinden:

```
\Erweiterungsordner\com\my_example\my_extension\MyActionHandler.class
```

Eine `JAR`-Datei kann sich in einem beliebigen Verzeichnis des Erweiterungsordners befinden. Diesen Speicherort der `JAR`-Datei legen Sie mit dem Element `JarFile` der Spezifikationsdatei fest. Verwendet eine Erweiterung beispielsweise eine `JAR`-Datei mit dem folgenden Pfad:

```
\Erweiterungsordner\lib\common-utilities.jar
```

dann muss das Element `Resources` der Spezifikationsdatei folgenden Eintrag enthalten:

```
<Resources>
    <JarFile id="util" path="lib\common-utilities.jar"/>
     ...
</Resources>
```

Weitere Informationen finden Sie im Thema „JAR-Dateien” auf Seite 35.

Eigenschaftendateien

Lokalisierte Ressourcen (z. B. der auf dem Bildschirm angezeigte Text, Fehlernachrichten und die entsprechenden Übersetzungen) werden in Dateien mit der Erweiterung `.properties` gespeichert. Diese Dateien können in einem beliebigen Verzeichnis im Erweiterungsordner abgelegt werden. Weitere Informationen finden Sie im Thema „Eigenschaftendateien” auf Seite 170.

Bild- und Hilfedateien

Die Dateien mit den Grafiken für die angezeigten Symbole und die Dateien, die das Hilfesystem enthalten, können in einem beliebigen Verzeichnis im Erweiterungsordner abgelegt werden. Es empfiehlt sich jedoch, Bild- und Hilfedateien in separaten Unterordnern zu speichern.

Den Speicherort einer Bilddatei deklarieren Sie in der Spezifikationsdatei mit dem Attribut `imagePath` eines `Icon`-Elements. Weitere Informationen finden Sie im Thema „Symbole" auf Seite 108.

Den Speicherort eines Hilfesystems deklarieren Sie ebenso in der Spezifikationsdatei, allerdings mit dem Attribut `path` eines `HelpInfo`-Elements. Weitere Informationen finden Sie im Thema „Festlegen von Speicherort und Typ des Hilfesystems" auf Seite 166.

Beispiel

Die clientseitige Dateistruktur mit diesen Komponenten kann z. B. wie folgt aussehen:

```
\ext\lib\myco.sorter
\ext\lib\myco.sorter\extension.xml
\ext\lib\myco.sorter\sorter_en.properties
\ext\lib\myco.sorter\sorter_fr.properties
\ext\lib\myco.sorter\sorter_it.properties
\ext\lib\myco.sorter\com\my_example\my_extension\MyActionHandler.class
\ext\lib\myco.sorter\help\sorter.chm
\ext\lib\myco.sorter\images\lg_sorter.gif
\ext\lib\myco.sorter\images\sm_sorter.gif
\ext\lib\myco.sorter\lib\common-utilities.jar
```

Serverseitige Komponenten

Die für die Ausführung erforderlichen gemeinsam verwendeten Bibliotheken müssen sich in einem Unterordner des Ordners `\ext\bin` im Installationsverzeichnis von IBM SPSS Modeler befinden. Beispiel:

Installationsverzeichnis`\ext\bin\myco.sorter\my_lib.dll`

Die gemeinsam verwendeten Bibliotheken dürfen keinesfalls direkt in den Ordner `\ext\bin` gestellt werden.

Den Speicherort von gemeinsam verwendeten Bibliotheken, die IBM SPSS Modeler während der Ausführung aufruft, deklarieren Sie in einem `SharedLibrary`-Element der Spezifikationsdatei. Weitere Informationen finden Sie im Thema „Freigegebene Bibliotheken" auf Seite 36.

Die gemeinsam verwendete Hauptbibliothek muss eventuell auf weitere, abhängige Bibliotheken zugreifen. Damit die Hauptbibliothek diese Bibliotheken findet, müssen sich die abhängigen Bibliotheken im gleichen Verzeichnis befinden wie die Hauptbibliothek.

Beispiel

Die serverseitige Dateistruktur kann z. B. wie folgt aussehen:

```
\ext\bin\myco.sorter\my_lib.dll
\ext\bin\myco.sorter\my_lib2.dll
```

Kapitel 2. Knoten

Überblick über die Knoten

Wenn Sie eine Erweiterung mit einem neuen Knoten erstellen, sollten Sie sich zunächst mit den Merkmalen der IBM SPSS Modeler-Knoten vertraut machen. Dies hilft Ihnen, Ihren neuen Knoten korrekt in der Spezifikationsdatei zu definieren.

IBM SPSS Modeler-Knoten werden entsprechend ihrer Funktion als Quellen-, Prozess-, Ausgabe- und Modellierungsknoten klassifiziert. In CLEF werden die Knoten nach ihrem Typ unterschieden. Die Zuordnung zwischen den beiden Systemen ist in der folgenden Tabelle dargestellt.

Tabelle 1. CLEF-Knotentypen.

IBM SPSS Modeler-Klassifikation	**Palette**	**CLEF-Knotentyp**
Quellenknoten	Quellen	Data reader (Datenleser)
Prozessknoten	Datensatzoperationen	Data transformer (Datentransformer)
	Feldoperationen	
Ausgabeknoten	Grafiken	Document builder (Dokumenterstellung)
	Ausgabe (Berichtsknoten)	
	Exportieren	Data writer (Datenschreiber)
Modellierungsknoten	Modellierung	Model builder (Modellerstellung)

Bei der Erstellung eines neuen CLEF-Knotens weisen Sie ihm einen der CLEF-Knotentypen zu. Für welchen Knotentyp Sie sich entscheiden, richtet sich nach der Hauptfunktion des Knotens.

Tabelle 2. Knotentypen und ihre Funktionen.

CLEF-Knotentyp	**Beschreibung**	**Zugehörige Knotenpalette**	**Symbol**
Data reader (Datenleser)	Importiert Daten aus einem anderen Format in IBM SPSS Modeler.	Quellen	 *Abbildung 3. Quellenknotensymbol (Kreis)*
Data transformer (Datentransformer)	Liest Daten aus IBM SPSS Modeler ein, modifiziert sie und gibt die modifizierten Daten in den IBM SPSS Modeler-Stream zurück.	Datensatzoperationen, Feldoperationen	 *Abbildung 4. Operationsknotensymbol (Sechseck)*

Tabelle 2. Knotentypen und ihre Funktionen (Forts.).

CLEF-Knotentyp	Beschreibung	Zugehörige Knotenpalette	Symbol
Model builder (Modellerstellung)	Generiert aus IBM SPSS Modeler-Daten Modelle.	Modellierung	 *Abbildung 5. Modellerstellungs-knotensymbol*
Document builder (Dokumenterstellung)	Generiert aus IBM SPSS Modeler-Daten Diagramme oder Berichte.	Grafiken	 *Abbildung 6. Diagrammknotensymbol (Dreieck)*
		Ausgabe (Berichtsknoten)	 *Abbildung 7. Ausgabeknotensymbol (Rechteck)*
Model applier (Modellanwender, auch als Model Nugget (Modellnugget) bezeichnet)	Definiert einen Container für ein generiertes Modell, das wieder auf dem IBM SPSS Modeler-Erstellungsbereich abgelegt wurde.	-	 *Abbildung 8. Modellanwendungs-knotensymbol (goldener Diamant)*
Data writer (Datenschreiber)	Exportiert Daten aus dem IBM SPSS Modeler-Format in ein für eine andere An-wendung geeignetes Format.	Exportieren	 *Abbildung 9. Exportknotensymbol (Rechteck)*

Den Knotentyp legen Sie gemeinsam mit anderen Attributen in der Spezifikationsdatei in einem `Node`-Element fest. Beispiel:

```
<Node name="sort_process" type="dataTransformer"
      palette="recordOp" ... >
     -- Knotenelemente --
</Node>
```

Das Attribut `palette` bestimmt die Palette im Hauptfenster von IBM SPSS Modeler, aus der die Benutzer den Knoten aufrufen können - in diesem Beispiel die Palette "Datensatzoperationen". Falls dieses Attribut fehlt, wird der Knoten der Palette "Feldoperationen" hinzugefügt.

IBM SPSS Modeler stellt verschiedene Beispielknoten zur Verfügung. Weitere Informationen finden Sie im Thema „Informationen zu den Beispielen” auf Seite 25.

Datenleserknoten

Ein Datenleserknoten ermöglicht das Einlesen von Daten aus einer externen Quelle in einen IBM SPSS Modeler-Stream. Bei den Knoten der IBM SPSS Modeler-Palette "Datenquellen" handelt es sich um Datenleserknoten. Sie sind durch ein Kreissymbol gekennzeichnet.

Die Spezifikation eines Datenleserknotens enthält folgende Details:

- Datenquelle (z. B. eine Datei oder eine Datenbank)
- Datensatzvorverarbeitung (z. B. die Behandlung von führenden und nachfolgenden Leerzeichen oder die Festlegung des Datensatztrennzeichens)
- Ob Datensatzfelder ausgefiltert werden
- Datentyp (z. B. Bereich, Set oder Flag) und Speichertyp (Zeichenfolge, ganze Zahl oder reelle Zahl) jedes einzelnen Felds
- Ob das Eingabedatenmodell geändert wird

Der Datenleserknoten kann die Logik zum Einlesen der Quellendatensätze enthalten. In IBM SPSS Modeler kann diese Logik aber auch erst später anhand eines Typknotens definiert werden.

IBM SPSS Modeler enthält ein Beispiel für einen Datenleserknoten. Weitere Informationen finden Sie im Thema „Informationen zu den Beispielen” auf Seite 25.

Datentransformationsknoten

Ein Datentransformationsknoten liest die Daten eines IBM SPSS Modeler-Streams ein, modifiziert diese Daten auf eine bestimmte Weise und gibt die modifizierten Daten wieder an den Stream zurück. Bei den Knoten der IBM SPSS Modeler-Paletten "Datensatzoperationen" und "Feldoperationen" handelt es sich um Datentransformationsknoten. Sie sind durch ein sechseckiges Symbol gekennzeichnet.

Die Spezifikation eines Datentransformationsknotens enthält folgende Details:

- Welche Datensätze oder Felder transformiert werden sollen
- Wie die Daten transformiert werden sollen

IBM SPSS Modeler enthält ein Beispiel für einen Datentransformationsknoten. Weitere Informationen finden Sie im Thema „Informationen zu den Beispielen” auf Seite 25.

Modellerstellungsknoten

Eine Einführung in die Modellerstellung mit IBM SPSS Modeler erhalten Sie im Abschnitt "Einführung in die Modellierung" im *IBM SPSS Modeler 17.1-Anwendungshandbuch*.

Modellerstellungsknoten generieren Objekte, die auf der Registerkarte "Modelle" bzw. auf der Registerkarte "Ausgaben" des Managerfensters im Hauptfenster von IBM SPSS Modeler angezeigt werden.

Bei den Knoten der IBM SPSS Modeler-Palette "Modellierung" handelt es sich um Modellerstellungsknoten. Sie sind durch ein fünfeckiges Symbol gekennzeichnet.

Ein Modellerstellungsknoten generiert bei seiner Ausführung ein **Modellausgabeobjekt** (auch als "Modellnugget" bezeichnet), das auf der Registerkarte "Modelle" des Managerfensters angezeigt wird.

Wenn ein durch einen Modellerstellungsknoten generiertes Modell auf den Erstellungsbereich gezogen wird, nimmt es die Form eines Modellanwendungsknotens an.

Die Spezifikation eines Modellerstellungsknotens enthält folgende Details:

- Modellerstellungsdetails, beispielsweise der Algorithmus für die Generierung des Modells sowie die Eingabe- und Ausgabefelder, die für das Scoring der Daten im Modell verwendet werden
- Vom Modell verwendete Eigenschaften
- Container für die Ausgabeobjekte
- Benutzerschnittstelle des Knotendialogfelds
- Eigenschaften und Dateien, die bei der Ausführung des Knotens verwendet werden
- Auswirkung der Knotenausführung auf das Eingabedatenmodell
- Kennung des Modellausgabeobjekts sowie die Kennungen aller anderen bei der Knotenausführung erstellten Objekte
- Kennung des Modellanwendungsknotens (siehe „Modellanwendungsknoten")

Hinweis: Bei der Definition eines Modellerstellungsknotens fügen Sie die Definition des tatsächlichen Modellausgabeobjekts und des Modellanwendungsknotens an einer anderen Stelle derselben Spezifikationsdatei ein.

IBM SPSS Modeler enthält ein Beispiel für einen Modellerstellungsknoten. Weitere Informationen finden Sie im Thema „Informationen zu den Beispielen" auf Seite 25.

Dokumenterstellungsknoten

Dokumenterstellungsknoten generieren Objekte, die auf der Registerkarte "Ausgaben" des Managerfensters im Hauptfenster von IBM SPSS Modeler angezeigt werden. Bei den Knoten der Palette "Diagramme" handelt es sich um Dokumenterstellungsknoten. Sie sind durch ein dreieckiges Symbol gekennzeichnet.

Ein Dokumenterstellungsknoten generiert bei seiner Ausführung ein **Dokumentausgabeobjekt**, das auf der Registerkarte "Ausgaben" des Managerfensters angezeigt wird.

Im Gegensatz zu einem Modellausgabeobjekt kann ein Dokumentausgabeobjekt nicht zurück auf den IBM SPSS Modeler-Erstellungsbereich gezogen werden.

Die Spezifikation eines Dokumenterstellungsknotens enthält folgende Details:

- Dokumenterstellungsdetails, beispielsweise die Registerkarte, auf der sich die Steuerelemente für die Generierung des Dokuments befinden
- Vom Dokument verwendete Eigenschaften
- Container für die Ausgabeobjekte
- Benutzerschnittstelle des Knotendialogfelds
- Eigenschaften und Dateien, die bei der Ausführung des Knotens verwendet werden
- Kennung des Dokumentausgabeobjekts sowie die Kennungen aller anderen bei der Knotenausführung erstellten Objekte

Hinweis: Bei der Definition eines Dokumenterstellungsknotens fügen Sie die Definition des tatsächlichen Dokumentausgabeobjekts an einer anderen Stelle derselben Spezifikationsdatei ein.

Modellanwendungsknoten

Ein Modellanwendungsknoten legt den Container eines generierten Modells fest, in dem das Modell von der Registerkarte "Modelle" des Managerfensters auf den IBM SPSS Modeler-Erstellungsbereich gezogen wird.

Die Spezifikation eines Modellanwendungsknotens enthält folgende Details:

- Container für das Modell (bzw. mehrere Container, wenn die Modellausgabe in mehreren Formaten wie Text und HTML erstellt werden kann)

- Details der Benutzerschnittstelle des Dialogfelds, das angezeigt wird, wenn der Benutzer den Anwendungsknoten auf der Registerkarte "Modelle" durchsucht oder ihn im Erstellungsbereich öffnet
- Ausgabedatenmodell
- Die bei der Ausführung des Streams, der den Knoten enthält, durchzuführenden Verarbeitungsschritte
- Konstruktoren für die Behandlung der Objekte, die bei der Ausführung des Streams, der den Knoten enthält, erstellt werden

Datenschreiberknoten

Ein Datenschreiberknoten exportiert Daten aus dem IBM SPSS Modeler-Format in ein für eine andere Anwendung geeignetes Format. Bei den Knoten der IBM SPSS Modeler-Palette "Exportieren" handelt es sich um Datenschreiberknoten. Sie sind durch ein rechteckiges Symbol gekennzeichnet.

Die Spezifikation eines Datenschreiberknotens enthält folgende Details:

- Angaben zur Datei oder Datenbank, in die die Streamdaten geschrieben werden
- Optional, ob der gesamte Stream veröffentlicht werden soll, damit er in eine externe Anwendung eingebettet werden kann

---

Menüs, Symbolleisten und Paletten

Der Zugriff auf eine Erweiterung kann über ein IBM SPSS Modeler-Menü, die Symbolleiste oder eine Palette erfolgen. Eine Erweiterung kann entweder einen Knoten implementieren oder eine bestimmte Aktion ausführen.

Eine Erweiterung (Knoten oder Aktion), die über ein ausdrücklich angegebenes Menü aufgerufen wird, kann auch über die Symbolleiste zugänglich gemacht werden. Das Gleiche gilt auch im umgekehrten Fall.

Für einen Knoten, der über eine Palette aufgerufen wird, steht automatisch auch ein entsprechendes Menüelement im Menü "Einfügen" zur Verfügung.

Menüs und Untermenüs

Die Standardknoten von IBM SPSS Modeler können über das Menü "Einfügen" aufgerufen werden. Jedes Menüelement der letzten Gruppe dieses Menüs (mit Ausnahme des Menüelements "Modelle") verfügt über ein Untermenü, das Zugriff auf einen Satz verwandter Knoten bietet.

Die Elemente dieses Menüs entsprechen den Einträgen der Knotenpaletten. Ein Knoten, der einer Palette hinzugefügt wird, wird automatisch in der entsprechenden Gruppe im Menü "Einfügen" angezeigt.

Falls Ihre Erweiterung eine Aktion definiert, die nicht über einen Knoten aufgerufen werden kann, können Sie die Erweiterung durch Hinzufügen eines der folgenden Elemente zugänglich machen:

- Neues Element in einem Systemmenü oder einem Untermenü
- Neues Menü in IBM SPSS Modeler
- Neues Element in der Symbolleiste (siehe „Symbolleisten”)

Dem neuen Menü oder Menüelement können Sie optional auch das Symbol der Erweiterung hinzufügen, wie dies bei einigen Elementen des Menüs "Einfügen" bereits der Fall ist.

Weitere Informationen finden Sie in „Menüs” auf Seite 110 und „Menüelemente” auf Seite 111.

Symbolleisten

Falls Ihre Erweiterung eine Aktion definiert, die nicht über einen Knoten aufgerufen werden kann, können Sie die Erweiterung der Hauptsymbolleiste von IBM SPSS Modeler hinzufügen, um sie auf diese Weise zugänglich zu machen.

In diesem Fall empfiehlt es sich, die Beschriftung der Aktion auszublenden.

Sie können ein Element auch zur Symbolleiste eines Knotendialogfelds oder eines Ausgabefensters hinzufügen. Die Elementbeschriftung können Sie anzeigen oder ausblenden.

Weitere Informationen finden Sie im Thema „Symbolleistenelemente” auf Seite 112.

Paletten und Unterpaletten

Wenn Ihre Erweiterung einen neuen Knoten definiert, können Sie den Knoten an einer beliebigen Stelle in einer der Standardpaletten oder Unterpaletten von IBM SPSS Modeler einfügen.

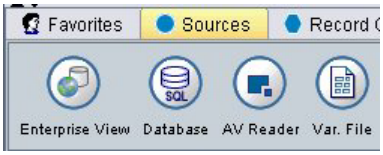

Abbildung 10. Neuer Knoten in einer Standardpalette

Sie können einer Standardunterpalette einen Eintrag hinzufügen und den neuen Knoten dort einfügen.

Abbildung 11. Neuer Knoten in benutzerdefiniertem Eintrag einer Standardunterpalette

Sie können eine benutzerdefinierte Palette erstellen und den neuen Knoten dort einfügen.

Abbildung 12. Neuer Knoten in einer benutzerdefinierten Palette

Einer benutzerdefinierten Palette können Sie benutzerdefinierte Unterpaletten hinzufügen.

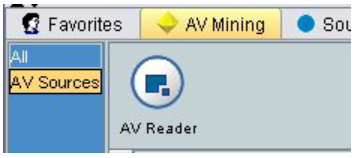

Abbildung 13. Neuer Knoten in einer benutzerdefinierten Unterpalette einer benutzerdefinierten Palette

Weitere Informationen finden Sie in „Knoten” auf Seite 48 und „Abschnitt 'UserInterface (Palettes)'” auf Seite 43.

Erstellen von Knotensymbolen

Sie können für jeden neuen Knoten, den Sie in CLEF erstellen, ein Bild bereitstellen, das in der Mitte des Symbols, das den Knoten auf dem Bildschirm darstellt, angezeigt wird.

Hinweis: Sie müssen kein Bild bereitstellen - IBM SPSS Modeler stellt ein Standardbild zur Verfügung, das angezeigt wird, wenn Sie kein eigenes Bild festlegen (dies kann z. B. in der anfänglichen Entwicklungsphase eines Knotens nützlich sein).

Abbildung 14. Standardbild für CLEF-Symbole

Die Standardsymbole von IBM SPSS Modeler bestehen aus drei Schichten:

- Rahmen
- Hintergrund
- Bild in der Mitte

Für das Symbol eines neuen Knotens brauchen Sie nur das Bild in der Mitte des Symbols bereitzustellen (auch als **Glyphe** bezeichnet). Die Rahmen- und Hintergrundverarbeitung wird von IBM SPSS Modeler übernommen. Das Glyphenbild muss einen transparenten Hintergrund haben, damit die Hintergrundschicht des Symbols nicht verdeckt wird. Die Glyphen in diesem Abschnitt sind mit einem farbigen Hintergrund dargestellt, der Transparenz darstellen soll.

Ein typisches Modellierungssymbol setzt sich in IBM SPSS Modeler wie nachfolgend beschrieben zusammen.

Tabelle 3. Zusammensetzung der Symbole für Knoten und generierte Modelle

	Knotensymbol	**Symbol für generiertes Modell**
Rahmen	 *Abbildung 15. Symbolrahmen*	None
Hintergrund	*Abbildung 16. Symbolhintergrund*	 *Abbildung 17. Hintergrund für generiertes Modell*
Glyphe	 *Abbildung 18. Symbolglyphe*	 *Abbildung 19. Glyphe für generiertes Modell*

Tabelle 3. Zusammensetzung der Symbole für Knoten und generierte Modelle (Forts.)

	Knotensymbol	Symbol für generiertes Modell
Bild angezeigt als	 *Abbildung 20. Angezeigtes Symbol*	 *Abbildung 21. Angezeigtes Symbol für generiertes Modell*

Rahmen

Die Funktion des Knotens wird durch die Form des Symbolrahmens angezeigt. Weitere Informationen finden Sie im Thema „Überblick über die Knoten" auf Seite 9.

Wenn für einen Knoten Caching aktiviert ist, wird der Rahmenform ein kleines Dokumentsymbol hinzugefügt. Ein weißes Dokumentsymbol an einem Knoten weist darauf hin, dass sein Cache leer ist. Wenn der Cache voll ist, wird das Dokumentsymbol grün.

Tabelle 4. Knotenrahmen und Caching-Status

Caching-Status	Beispiel
Kein Caching	 *Abbildung 22. Knoten ohne Caching*
Caching aktiviert	 *Abbildung 23. Knoten mit aktiviertem Caching*
Cache voll	 *Abbildung 24. Knoten mit vollem Cache*

Die verschiedenen Rahmensymbole werden vom System bereitgestellt. IBM SPSS Modeler sorgt dafür, dass die erforderliche Verarbeitung ausgeführt wird, um das richtige Symbol zur richtigen Zeit anzuzeigen.

Hintergründe

Die Hintergrundfarbe der Knotensymbole ändert sich, um den jeweiligen Status anzuzeigen. Hiervon ausgenommen sind Symbole für generierte Modelle und Symbole von Modellanwendungsknoten.

Tabelle 5. Knotenhintergründe

Status	Farbe	Beispiel
Nicht ausgewählt	Grau	*Abbildung 25. Symbolhintergrund (grau)*
Ausgewählt	Blau	 *Abbildung 26. Symbolhintergrund (blau)*
Fehler	Rot	 *Abbildung 27. Symbolhintergrund (rot)*
Aktion wird an einer Datenbank ausgeführt	Violett	 *Abbildung 28. Symbolhintergrund (violett)*

Auch die Hintergrundbilder werden vom System bereitgestellt und hier sorgt IBM SPSS Modeler wiederum dafür, dass die erforderliche Verarbeitung ausgeführt wird, um den richtigen Hintergrund für die jeweilige Situation anzuzeigen.

Grafikanforderungen

Erstellen Sie für jeden neuen CLEF-Knoten die folgenden Versionen der Glyphenbildschicht:

- Großes Format (49 x 49 Pixel) für das Knotensymbol im Streamerstellungsbereich
- Kleines Format (38 x 38 Pixel) für das Knotensymbol im Palettenmanager am unteren Bildschirmrand

Wenn Sie das Symbol in einem Menü oder einer Symbolleiste oder in der Titelleiste eines Browser- oder Ausgabefensters anzeigen möchten, müssen Sie zusätzlich folgende Bildschicht erstellen:

- Miniaturformat (16 x 16 Pixel)

Wenn der Knoten ein Modell generiert, müssen Sie zusätzlich folgende Bildschicht erstellen:

- Kleines Format (38 x 38 Pixel), wobei die Grafik in die linke untere Ecke verschoben werden muss, damit die Bildschicht über das Goldnugget-Symbol eines generierten Modells gelegt werden kann

Hinweis: Bilder, die größer als die angegebenen Formate sind, werden bei der Anzeige in IBM SPSS Modeler abgeschnitten.

Weitere Informationen finden Sie im Thema „Symbole” auf Seite 108.

Erstellen von benutzerdefinierten Bildern

Das Bild, das Sie für einen Knoten erstellen, sollte die Hauptfunktion des Knotens grafisch vermitteln. Wenn Sie Ihre Erweiterung einem internationalen Publikum bereitstellen möchten, sollten Sie darauf achten, dass Sie keine landestypischen Symbole verwenden und dass die Bilder in anderen Kulturen und Sprachräumen nicht missverstanden werden.

So erstellen Sie ein benutzerdefiniertes Bild für CLEF:

1. Verwenden Sie zur Erstellung Ihres Bildes eine Grafikanwendung, die Transparenz unterstützt, und richten Sie einen Erstellungsbereich mit dem erforderlichen Format ein. Zeichnen Sie danach die einzelnen Bildversionen.
2. Speichern Sie jede der erforderlichen Versionen (groß, klein usw.) als separate *GIF*-Datei mit folgenden Eigenschaften:
   - Hintergrund: Transparent
   - Farbtiefe: 16 Farben (4 Bit) oder höher

   Auf welche Art und Weise Sie den Bildhintergrund transparent gestalten, richtet sich nach der verwendeten Grafikanwendung. Möglicherweise können Sie für die Hintergrundfarbe ohne weitere Umwege "Transparent" festlegen oder Sie müssen zunächst eine Transparenzfarbe festlegen und den Bildhintergrund danach mit dieser Farbe "anmalen".

   Bei der Benennung der Bilddateien empfiehlt es sich, den von IBM SPSS Modeler intern verwendeten Dateinamenskonventionen zu folgen (siehe Darstellung in der folgenden Tabelle).

Tabelle 6. Dateinamenskonventionen für Bilddateien

Bildtyp	**Dateiname**
Large	lg_*Knoten*.gif
Klein	sm_*Knoten*.gif
Miniatur	*Knoten*16.gif
Generiertes Modell	sm_gm_*Knoten*.gif

3. Die Wirkung Ihres Bildes können Sie testen, indem Sie in der Spezifikationsdatei auf die Bilddateien verweisen (siehe „Hinzufügen der Bilddateien zur Knotenspezifikation") und den neuen Knoten zu IBM SPSS Modeler hinzufügen (siehe „Testen einer CLEF-Erweiterung" auf Seite 203).

Hinzufügen der Bilddateien zur Knotenspezifikation

Nachdem Sie die Bilddateien erstellt haben, kopieren Sie diese in einen Ordner auf dem Computer, auf dem Sie IBM SPSS Modeler ausführen. In der Spezifikationsdatei müssen Sie den Bildpfad relativ zu dem Ordner \ext\lib*Provider.Knotenname* im Installationsverzeichnis von IBM SPSS Modeler angeben. Sie sollten die Dateien daher in einen Ordner kopieren, der von diesem Speicherort leicht zu erreichen ist. Weitere Informationen finden Sie im Thema „Symbole" auf Seite 108.

Die Zuordnung der Grafikdateien mit dem großen und dem kleinen Symbol zu einem benutzerdefinierten Knoten erfolgt in der Spezifikationsdatei über das Element `Icons` im Abschnitt `UserInterface` der `Node`-Spezifikation. Beispiel:

```
<Icons>
     <Icon type="standardNode" imagePath="images/lg_mynode.gif" />
     <Icon type="smallNode" imagePath="images/sm_mynode.gif" />
</Icons>
```

Bei einem Modell- oder Dokumenterstellungsknoten müssen Sie zusätzlich auf die Miniaturversion (16 x 16 Pixel) verweisen. Bei einem Modellerstellungsknoten erfolgt dieser Verweis im Abschnitt `UserInterface` der `ModelOutput`-Spezifikation, bei einem Dokumenterstellungsknoten im Abschnitt `UserInterface` der `DocumentOutput`-Spezifikation. Beispiel:

```
<Icons>
     <Icon type="standardWindow" imagePath="images/mynode16.gif" />
</Icons>
```

Bei einem Modellanwendungsknoten müssen Sie im Abschnitt `UserInterface` der `Node`-Spezifikation zusätzlich auf die Bildversion für das generierte Modell verweisen. Beispiel:

```
<Icons>
     <Icon type="standardNode" imagePath="images/lg_gm_mynode.gif" />
     <Icon type="smallNode" imagePath="images/sm_gm_mynode.gif" />
</Icons>
```

Erstellen von Dialogfeldern

In diesem Abschnitt werden die Merkmale der Standardknotendialogfelder von IBM SPSS Modeler beschrieben. Der Abschnitt soll Ihnen dabei helfen, auch in CLEF konsistente Dialogfelder zu erstellen.

Informationen zu Knotendialogfeldern

Ein Knotendialogfeld bietet eine Benutzerschnittstelle, mit der der Endbenutzer Ausführungseinstellungen ändern kann. Die Darstellung eines Dialogfelds ist äußerst wichtig, da hier das Knotenverhalten geändert wird. Die Benutzerschnittstelle muss alle notwendigen Informationen enthalten und benutzerfreundlich sein.

Das Knotenverhalten wird durch verschiedene, für Dialogfelder typische **Steuerelemente**, d. h. durch Benutzerschnittstellenelemente, mit denen der Benutzer interagieren kann, geändert. Ein Dialogfeld kann zahlreiche Steuerelemente wie Optionsfelder, Kontrollkästchen, Textfelder und Menüs enthalten. CLEF bietet eine große Auswahl an Steuerelementen, die Sie in Ihre Dialogfelder einfügen können. Weitere Informationen finden Sie im Thema „Spezifikationen von Eigenschaftssteuerelementen” auf Seite 123.

Der durch ein Steuerelement geänderte Parametertyp bestimmt, welches Steuerelement im Dialogfeld angezeigt wird, wobei einige Typen alternative Steuerelemente bieten. Verwandte Optionen können Sie mittels des `Tab`-Elements der Spezifikationsdatei auf neuen Registerkarten zusammenfassen. Weitere Informationen finden Sie im Thema „Registerkartenbereich” auf Seite 22.

Hinweis: Das Look-and-feel der Benutzerschnittstelle einer Erweiterung können Sie testen, noch bevor Sie die durch die Erweiterung auszuführende Verarbeitung festgelegt haben. Weitere Informationen finden Sie im Thema „Testen von CLEF-Erweiterungen” auf Seite 203.

Richtlinien für die Erstellung von Dialogfeldern

Beachten Sie bei der Festlegung der Steuerelemente für ein Dialogfeld die folgenden Richtlinien:

- Überlegen Sie sich den Text für die Beschriftung eines Steuerelements gut. Der Text sollte möglichst knapp und präzise sein und gleichzeitig korrekte Informationen vermitteln. Wenn Sie Ihre Erweiterung für den internationalen Markt entwickeln, sollten Sie ausreichend Platz für die Übersetzung lassen, die häufig erheblich länger ist als das Original.
- Verwenden Sie das richtige Steuerelement für einen Parameter. Auch wenn ein Parameter nur zwei Werte haben kann, ist ein Kontrollkästchen nicht immer die beste Wahl. Im Dialogfeld des C5.0-Knotens von IBM SPSS Modeler werden beispielsweise Optionsfelder verwendet, um dem Benutzer die Möglichkeit zu geben, den Ausgabetyp auszuwählen, obwohl dieser nur einen der beiden folgenden Werte annehmen kann, **Entscheidungsbaum** oder **Regelset**.

  Diese Einstellung könnte durch ein Kontrollkästchen mit der Beschriftung **Entscheidungsbaum** dargestellt werden. Wenn das Kontrollkästchen aktiviert ist, ist der Ausgabetyp ein Entscheidungsbaum. Ist es nicht aktiviert, handelt es sich bei der Ausgabe um ein Regelset. Auch wenn das Ergebnis dasselbe wäre, so wären die zur Verfügung stehenden Optionen durch Verwendung von Optionsfeldern leichter zu verstehen.
- Steuerelemente für Dateinamen werden im Allgemeinen im oberen Bereich eines Dialogfelds platziert.

- Steuerelemente, die den Fokus des Knotens bilden, werden oben im Dialogfeld platziert. Diagrammknoten zeigen beispielsweise Felder aus den Daten an. Da die Auswahl dieser Felder die Hauptfunktion des Dialogfelds darstellt, werden Feldparameter oben platziert.
- Kontrollkästchen oder Optionsfelder ermöglichen dem Benutzer häufig die Auswahl einer Option, die weitere Informationen erfordert. Beispiel: Die Auswahl der Option **Boosting verwenden** im Dialogfeld des C5.0-Knotens erfordert, dass die Analyse eine Zahl für die **Anzahl der Versuche** enthält.

  Die Zusatzinformationen werden immer nach der Optionsauswahl platziert, und zwar entweder rechts oder direkt darunter.

Die CLEF-Dialogfelder verwenden die Commitbearbeitung von IBM SPSS Modeler auf dieselbe Weise wie die Dialogfelder von IBM SPSS Modeler. Die in den Dialogfeldern angezeigten Werte werden erst in den Knoten kopiert, wenn der Benutzer auf **OK**, **Anwenden** oder bei Endknoten auf **Ausführen** klickt. Ebenso werden die im Dialogfeld angezeigten Informationen erst aktualisiert (z. B. wenn sich die Eingabefelder des Knotens aufgrund von Vorgängen weiter oben im Stream des aktuellen Knotens geändert haben), wenn der Benutzer das Dialogfeld abbricht und erneut anzeigt oder auf die Schaltfläche **Aktualisieren** klickt.

Dialogfeldkomponenten

Dialogfelder enthalten die folgenden Komponenten:

- Titelleiste
- Symbolbereich
- Symbolleisten- und Menübereich mit:

  - "Datei", "Generieren", "Ansicht", "Vorschau", "Aktualisieren" und andere Schaltflächen (abhängig vom Knoten)

  - Schaltfläche "Maximieren/Normale Größe"

  - Schaltfläche "Hilfe"
- Statusbereich
- Fensterbereich
- Registerkartenbereich
- Schaltflächenbereich

Jeder benutzerdefinierte Knoten benötigt ein Dialogfeld, das angezeigt wird, sobald der Benutzer den Knoten öffnet. Vorausgesetzt Ihre Spezifikationsdatei enthält ein `Node`-Element mit dem Abschnitt `UserInterface`, der wiederum das Element `Tabs` enthält, dann werden beim Öffnen des Knotens alle oben aufgelisteten Dialogfeldkomponenten angezeigt. Abhängig vom Knotentyp enthalten der Registerkartenbereich und der Schaltflächenbereich mindestens die in der folgenden Tabelle aufgeführten Komponenten.

Tabelle 7. Standardkomponenten des Registerkarten- und Schaltflächenbereichs für verschiedene Knotentypen

Knotentyp	Registerkarten	Schaltflächen
Data reader (Datenleser)	Anmerkungen (mit Schaltfläche "Aktualisieren" im Symbolleistenbereich)	OK, Abbrechen, Anwenden, Zurücksetzen
Data transformer (Datentransformer)	Anmerkungen	OK, Abbrechen, Anwenden, Zurücksetzen
Data writer (Datenschreiber)	Veröffentlichen, Anmerkungen	OK, Abbrechen, Ausführen, Anwenden, Zurücksetzen
Model builder (Modellerstellung)	Anmerkungen	OK, Abbrechen, Ausführen, Anwenden, Zurücksetzen
Document builder (Dokumenterstellung)	Anmerkungen	OK, Abbrechen, Ausführen, Anwenden, Zurücksetzen
Model applier (Modellanwender)	Übersicht, Anmerkungen	OK, Abbrechen, Anwenden, Zurücksetzen

Knotendialogfelder sind zunächst so positioniert, dass das Knotensymbol beim Öffnen des Knotens über den Knoten, den es darstellt, gelegt wird. Der Benutzer kann das Dialogfeld verschieben, die neue Position wird aber nicht gespeichert und beim nächsten Öffnen des Knotens nicht wiederhergestellt. Wenn der Benutzer das Dialogfeld verschoben hat und es danach teilweise oder vollständig durch ein anderes Dialogfeld verdeckt wird, kann er es durch Doppelklicken auf den Originalknoten im Erstellungsbereich wieder in den Vordergrund verschieben. Das Dialogfeld ist moduslos (d. h., die gleiche Benutzereingabe bewirkt immer die gleiche Aktion) und seine Größe kann geändert werden.

Alle bearbeitbaren Felder im Dialogfeld unterstützen die in der folgenden Tabelle aufgeführten Tastenkombinationen.

Tabelle 8. Tastenkombinationen für bearbeitbare Felder in Dialogfeldern

Tastenkombination	**Effekt**
STRG-C	Kopieren
STRG-V	Einfügen
STRG-X	Ausschneiden

Titelleiste

Die Titelleiste eines Knotendialogfelds enthält eine Miniaturausgabe des Nuggetsymbols von IBM SPSS Modeler gefolgt vom Namen des Modells. Der Text wird der Einstellung der Steuerelemente für den Modellnamen entnommen. Auf der rechten Seite der Titelleiste befindet sich standardmäßig die Schaltfläche "Schließen" (X).

Symbolbereich

Das Knotensymbol befindet sich links oben im Dialogfeld im Symbolbereich. Hier wird die kleine Version des Symbols (38 x 38 Pixel) angezeigt, d. h. dieselbe Version, die auch unten im Hauptfenster in der Knotenpalette angezeigt wird (nicht die größere, für den Erstellungsbereich vorgesehene Version).

Hinweis: Die Miniaturausgabe des Nuggetsymbols auf der linken Seite der Titelleiste ist in allen Knotendialogfeldern fest codiert.

Symbolleisten- und Menübereich

Der obere Bereich des Dialogfelds ist für die Symbolleiste und den Menübereich reserviert.

Die Dialogfelder für Datenleser- oder Datentransformationsknoten verfügen in diesem Bereich über die Schaltfläche "Vorschau", mit deren Hilfe ein Beispiel der Eingabedaten angezeigt werden kann.

Dialogfelder für Datenleserknoten enthalten auch die Schaltfläche "Aktualisieren", die die vom Knoten angezeigten Informationen aktualisiert (z. B. wenn die Eingabefelder des Knotens geändert wurden).

Modellanwendungsknoten verfügen über die Schaltflächen "Datei", "Generieren" und "Ansicht", mit deren Hilfe Benutzer zahlreiche Operationen ausführen können, z. B. Exportieren eines Modells oder Generieren neuer Knoten. Modellanwendungsknoten verfügen auch über eine Schaltfläche "Vorschau", die in diesem Fall ein Beispiel der Eingabedaten zusammen mit den zusätzlichen Spalten anzeigt, die beim Anwenden des Knotens erstellt werden.

Auf der rechten Seite dieses Bereichs befinden sich in jedem Knotendialogfeld zwei Schaltflächen:

- Schaltfläche "Maximieren/Normale Größe"
- Schaltfläche "Hilfe"

Schaltfläche "Maximieren/Normale Größe": Mit dieser Schaltfläche kann das Dialogfeld auf Vollbildgröße vergrößert werden. Wenn das Dialogfeld bereits mit maximaler Größe angezeigt wird, wird es bei Betätigen der Schaltfläche wieder auf die vorherige Größe verkleinert.

Schaltfläche "Hilfe": Diese Schaltfläche ruft die kontextsensitive Hilfe für den Knoten auf. Bei einem Dialogfeld mit Registerkarten oder einem Ausgabefenster wird die Hilfe für die jeweilige Registerkarte bzw. die Hilfe für das Ausgabefenster angezeigt. Dieselbe Hilfe können Sie auch mit der Taste F1 aufrufen.

Statusbereich

Der verbleibende Teil des oberen Dialogfeldbereichs ist für Informationen, Warnungen oder Fehlernachrichten reserviert. Bei Quellenknoten wird hier der vollständige Pfad und Dateiname der Datenquelle angezeigt. In diesem Bereich werden je nach Knotentyp unterschiedliche Informationen angezeigt. Jeglicher für diesen Bereich vorgesehene Text sollte sich auf zwei Zeilen beschränken.

Fensterbereich

Dies ist der Hauptbereich des Dialogfelds. Er enthält alle Steuerelemente und Anzeigebereiche des Knotens. Der Fensterbereich sieht auf jeder Registerkarte anders aus. Es gibt folgende Fensterbereichtsypen:

- Textbrowser
- Erweiterungsobjekt
- Eigenschaften

Sie können auch Unterfensterbereiche definieren. Dies sind eigene Dialogfelder, die über Aktionsschaltflächen im Fensterbereich aufgerufen und in einem neuen Fenster geöffnet werden.

Weitere Informationen finden Sie im Thema „Fensterspezifikationen” auf Seite 117.

Registerkartenbereich

Knotendialogfelder enthalten die folgenden Registerkarten:

- Eine oder mehrere benutzerdefinierte knotenspezifische Registerkarten
- Registerkarte "Übersicht" (nur bei Modellausgabeobjekten und Modellanwendungsknoten)
- Registerkarte "Anmerkungen"

Die knotenspezifischen Registerkarten werden im `Tabs`-Abschnitt der Spezifikationsdatei von CLEF definiert. Weitere Informationen finden Sie im Thema „Registerkarten” auf Seite 114.

Den Dialogfeldern von Modellausgabeobjekten und Modellanwendungsknoten wird vom System automatisch die Registerkarte "Übersicht" hinzugefügt. Diese Registerkarte enthält zusammengefasste Informationen über das generierte Modell, einschließlich der Felder, der Generierungseinstellungen und des verwendeten Modellschätzungsprozesses. Die Ergebnisse werden in einer Baumansicht dargestellt, die durch Klicken auf bestimmte Elemente erweitert bzw. reduziert werden kann.

Die Registerkarte "Anmerkungen" wird allen Knotendialogfeldern automatisch vom System hinzugefügt. Hier können Benutzer Informationen zum jeweiligen Knoten wie den Knotennamen, QuickInfo-Text und einen längeren Kommentar eingeben.

Name: Der Standardknotenname wird im Element "Node" der Spezifikationsdatei mit dem Attribut `Label` angegeben (siehe „Knoten” auf Seite 48). Dieser Name kann vom Benutzer geändert werden. Dazu wählt er **Benutzerdefiniert** aus, gibt einen Namen im Bearbeitungsfeld "Benutzerdefiniert" ein und klickt auf **Anwenden** oder **OK**. Der neue Name wird auch in späteren Sitzungen beibehalten. Allerdings kann der Benutzer jederzeit zum Standardnamen zurückkehren, indem er auf **Auto** klickt. Ein auf der Registerkarte "Anmerkungen" angegebener benutzerdefinierter Name überschreibt jeden benutzerdefinierten Namen, der auf einer anderen Registerkarte des Dialogfelds festgelegt wurde.

QuickInfo-Text: Der hier angegebene Text wird im Erstellungsbereich als QuickInfo für den Knoten angezeigt. Falls kein Text angegeben ist, wird keine QuickInfo eingeblendet, wenn der Benutzer mit dem Cursor auf den Knoten zeigt.

Schlüsselwörter: Hier kann der Benutzer Schlüsselwörter eingeben, die in Projektberichten oder beim Suchen oder Verfolgen von Objekten im IBM SPSS Collaboration and Deployment Services Repository verwendet werden.

Kommentarbereich: In diesem Bereich kann der Benutzer Kommentare eingeben.

Erstell- und Speicherinformationen: In diesem nicht bearbeitbaren Textbereich werden Informationen zur Erstellung, der Dateiname sowie Speicherdatum und -uhrzeit einer Datei angezeigt (das Datum-/Zeitformat ist von der Ländereinstellung abhängig). Falls das Element noch nicht gespeichert wurde, enthält dieses Feld die Nachricht "This item has not been saved" (Dieses Element wurde noch nicht gespeichert).

Schaltflächenbereich

Im unteren Bereich jedes Dialogfelds befinden sich die Schaltflächen **Anwenden**, **Zurücksetzen**, **OK** und **Abbrechen**. Bei einem Endknoten (ein ausführbarer Knoten, der die Streamdaten verarbeitet) befindet sich dort zusätzlich die Schaltfläche **Ausführen**.

OK: Übernimmt die Einstellungen und schließt das Dialogfeld. Direkt nach dem Öffnen des Dialogfelds aus dem Knoten ist diese Schaltfläche durch ein blaues Rechteck hervorgehoben, d. h., sie hat den Fokus. Wenn Sie die Eingabetaste drücken, solange die Schaltfläche den Fokus hat, wird die Funktion der Schaltfläche sofort ausgeführt.

Abbrechen: Schließt das Dialogfeld, wobei die neuen Einstellungen nicht gespeichert werden. Es werden also die Einstellungen übernommen, die das Dialogfeld beim Öffnen hatte bzw. die beim letzten Betätigen der Schaltfläche "Anwenden" gespeichert wurden. Die Funktion der Schaltfläche "Abbrechen" können Sie auch mit der Escapetaste ausführen, sofern das gesamte Dialogfeld den Fokus hat.

Ausführen: Wendet die aktuellen Einstellungen an, schließt das Dialogfeld und führt den Endknoten aus.

Anwenden: Speichert die aktuellen Einstellungen des Dialogfelds, damit sie von den nächsten Operationen übernommen werden können.

Zurücksetzen: Setzt die Einstellungen des Dialogfelds auf diejenigen Einstellungen zurück, die das Dialogfeld beim Öffnen hatte bzw. die beim letzten Betätigen der Schaltfläche "Anwenden" gespeichert wurden.

Erstellen von Ausgabefenstern

In diesem Abschnitt werden die Merkmale der Standardausgabefenster von IBM SPSS Modeler beschrieben. Der Abschnitt soll Ihnen dabei helfen, auch in CLEF konsistente Ausgabefenster zu erstellen.

In Ausgabefenstern kann die Ausgabe folgender Elemente angezeigt werden:

- Die Ausgabe eines Modells, z. B. das Ergebnis aus dem Scoring eines Datasets (durch Scoring wird ein Modell angewendet)
- Die Ausgabe eines Dokuments, z. B. ein Diagramm oder ein Bericht

Weitere Informationen finden Sie im Thema „Informationen zu Benutzerschnittstellen” auf Seite 105.

Ausgabefenster unterscheiden sich nur in den folgenden Punkten von Knotendialogfeldern:

- Die Titelleiste enthält statt des allgemeinen Goldnuggetsymbols ein knotenspezifisches Miniatursymbol.
- Das Symbol des Hauptknotens fehlt.

- Im Symbolleisten- und Menübereich fehlt die Schaltfläche "Maximieren/Normale Größe". Diese kann in einem Dokumentausgabefenster durch die Schaltfläche "Schließen" oder "Löschen" ersetzt sein. Das Fenster kann jedoch mit der Maus vergrößert und verkleinert werden.
- Der Statusbereich fehlt.
- Die typischen Registerkarten sind:
  - eine Registerkarte "Modell" (für Modellausgabefenster), um die Daten zum Prädiktoreinfluss anzuzeigen, falls diese Option für den Modellknoten ausgewählt wurde
  - eine einzelne Registerkarte für die Ausgabe
  - eine Registerkarte "Übersicht" (für Modellausgabefenster), um die zusammenfassenden Details über das Modell anzuzeigen
  - eine Registerkarte "Anmerkungen" (Die Anmerkungswerte werden von dem Knoten übernommen, der die Ausgabe generiert hat)
- Der Schaltflächenbereich enthält die Schaltflächen "OK", "Abbrechen", "Anwenden" und "Zurücksetzen".

CLEF stellt Standardfenster für die Modell- und Dokumentausgabe bereit, die dem oben abgebildeten Fenster in etwa entsprechen. Diese Fenster werden standardmäßig angezeigt, wenn die Spezifikationsdatei ein `ModelOutput`-Element bzw. ein `DocumentOutput`-Element enthält. Weitere Informationen finden Sie im Thema „Objekt-ID” auf Seite 48.

Sie können das `ModelOutput`- oder `DocumentOutput`-Element jedoch auch so definieren, dass statt des Standardausgabefensters ein von Ihnen selbst erstelltes Ausgabefenster angezeigt wird. Weitere Informationen finden Sie im Thema „Benutzerdefinierte Ausgabefenster” auf Seite 162.

Kapitel 3. CLEF-Beispiele

Informationen zu den Beispielen

Damit Sie schneller mit CLEF vertraut werden, beinhaltet die IBM SPSS Modeler-Installation verschiedene Beispielknoten mit vollständigem Quellcode. Es handelt sich hier um grundlegende Knoten mit eingeschränkter Funktionalität, die Ihnen lediglich helfen sollen, die Funktionsweise und Verwendung von CLEF zu verstehen. Sie können diese Knoten sofort oder wann immer Sie Zeit dazu finden, ausprobieren.

Folgende Beispiele werden bereitgestellt:

- Datenleserknoten (mit dem Namen "Apache Log Reader")
- Datentransformationsknoten (mit dem Namen "URL Parser")
- Dokumenterstellungsknoten (mit dem Namen "Web Status Report")
- Modellerstellungsknoten (mit dem Namen "Interaction")

Die Beispiele müssen vor der Verwendung aktiviert werden.

Aktivieren der Beispiele

Die Beispiele werden bei der Installation von IBM SPSS Modeler in komprimiertem Format im Verzeichnis `Demos` installiert. Zur Aktivierung der Beispiele müssen Sie die Dateien in die korrekten Verzeichnisse extrahieren.

Führen Sie dazu auf dem Computer, auf dem IBM SPSS Modeler installiert ist, die folgenden Schritte aus:

1. Beenden Sie IBM SPSS Modeler, falls der Client ausgeführt wird.
2. Suchen Sie die Datei `clef_examples_ext_lib.zip` im Verzeichnis `Demos` der IBM SPSS Modeler-Installation.
3. Extrahieren Sie den Inhalt der Datei `clef_examples_ext_lib.zip` in den Ordner `\ext\lib` des Installationsverzeichnisses von IBM SPSS Modeler.

Führen Sie dazu auf dem Computer, auf dem IBM SPSS Modeler und/oder IBM SPSS Modeler Server installiert ist, die folgenden Schritte aus:

1. Extrahieren Sie den Inhalt der Datei `clef_examples_ext_bin.zip` in den Ordner `\ext\bin` die Installationsverzeichnisse von IBM SPSS Modeler sowie IBM SPSS Modeler Server.
2. Verwenden Sie auf Nicht-Windows-Systemen die in `clef_examples_ext_bin.zip` bereitgestellte Makefile, um den Quellcode für die relevanten Beispiele zu kompilieren. Weitere Informationen finden Sie im Thema „Untersuchen des Quellcodes” auf Seite 29.

   ODER

   Verwenden Sie unter Windows die folgenden Anweisungen zum Kompilieren des Quellcodes für die relevanten Beispiele (hierfür ist Visual Studio 2008 erforderlich):

   a. Navigieren Sie im Verzeichnis `\ext\bin`, in dem Sie `clef_examples_ext_bin.zip` in Schritt 1 extrahiert haben, zu dem Unterverzeichnis, das die CLEF-Beispielerweiterung enthält, die Sie kompilieren wollen (z. B. `spss.apachelogreader`).
   b. Doppelklicken Sie im Unterverzeichnis `src` auf die `.sln`-Datei, um die Lösung der CLEF-Erweiterung in Visual Studio zu öffnen (doppelklicken Sie z. B. auf `\ext\bin\spss.apachelogreader\src\apachelogreader.sln`).
   c. Rufen Sie in Visual Studio **Build** > **Configuration Manager** auf.
   d. Wählen Sie für Active Solution Configuration **Release** aus.
   e. Wählen Sie für Active Solution Platform **x64** aus.

f. Klicken Sie auf **Close**.

g. Wählen Sie **Build** > **Build Solution** aus oder klicken Sie auf F7, um das Projekt zu erstellen.

Die resultierende 64-Bit-DLL-Datei wird an den folgenden Speicherort geschrieben (relativ zum Ordner `src`):

```
..\bin\win64\release\spss.<Erweiterungsname>\<Erweiterungsname>.dll
```

(z. B. `..\bin\win64\release\spss.apachelogreader\apachelogreader.dll`).

Starten Sie in allen Fällen zum Abschluss IBM SPSS Modeler und stellen Sie sicher, dass die in der folgenden Tabelle aufgeführten Knoten in der Knotenpalette sichtbar sind.

Tabelle 9. In der Knotenpalette sichtbare Knoten.

Registerkarte "Palette"	Knoten
Quellen	Apache Log Reader
Feldoperationen	URL Parser
Modellierung	Interaction
Ausgabe	Web Status Report

Datenleserknoten (Apache Log Reader)

Der Datenleserknoten ist ein Datenquellenknoten, der die Daten aus der Zugriffsprotokolldatei eines Apache-HTTP-Web-Servers einliest. Das Zugriffsprotokoll enthält Informationen zu allen Anforderungen, die vom Web-Server verarbeitet wurden. Die Protokolldatensätze sind im kombinierten Protokollformat (Combined Log Format) gespeichert. Beispiel:

```
IP-Adresse - - [09/Jul/2007:07:57:38 +0000] "GET /lsearch.php?county_id=3 HTTP/1.1" 200 16348
"http://www.google.co.uk/search?q=thunderbirds+cliveden&hl=en&start=10&sa=N"
"Mozilla/4.0 (kompatibel; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322)"
```

Mit diesem Beispielknoten können Sie die Protokolldatensätze in ein Tabellenformat konvertieren, das leichter zu lesen ist.

So verwenden Sie den Knoten "Apache Log Reader":

1. Wenn Sie die Beispiele von CLEF noch nicht aktiviert haben, tun Sie es jetzt. Weitere Informationen finden Sie im Thema „Aktivieren der Beispiele” auf Seite 25.
2. Öffnen Sie IBM SPSS Modeler.
3. Wählen Sie auf der Registerkarte "Datenquellen" der Knotenpalette **Apache Log Reader** aus und fügen Sie den Knoten dem Erstellungsbereich hinzu.
4. Bearbeiten Sie den Knoten. Geben Sie auf der Registerkarte "Option" im Feld "Apache-Protokolldatei" Folgendes ein:

   Demo-Ordner`\combined_log_format.txt`

   Dabei ist *Demo-Ordner* der Speicherort des Ordners *Demos* im Installationsverzeichnis von IBM SPSS Modeler (verwenden Sie nicht das Format *$CLEO_DEMOS*).
5. Klicken Sie auf **OK**.
6. Fügen Sie einen Typknoten zum Stream hinzu.
7. Bearbeiten Sie den Typknoten. Klicken Sie auf **Werte lesen**, um die Daten einzulesen, und klicken Sie anschließend auf **OK**.
8. Verbinden Sie einen Tabellenknoten mit dem Typknoten und führen Sie den Stream aus. Der Inhalt der Protokolldatei wird nun in einer Tabelle angezeigt.
9. Speichern Sie den Stream für die nächsten beiden Beispiele.

Datentransformationsknoten (URL Parser)

Im Beispiel für den Datentransformationsknoten werden die im vorherigen Beispiel zurückgegebenen Daten weiter verarbeitet. Sie wählen ein ID-Feld aus (das für jede Zeile einen eindeutigen Wert enthalten sollte) sowie ein Eingabefeld mit einer URL. Die vom Knoten generierte Ausgabe besteht aus diesen beiden Feldern sowie den URL-Daten, die zusätzlich analysiert und in separat generierte Felder ausgegeben werden. Wenn ein URL-Datensatz beispielsweise einen Abfragezeichenfolge wie den folgenden enthält:

```
http://www.dummydomain.co.uk/resource.php?res_id=89
```

Der Datensatz wird wie in der folgenden Tabelle dargestellt analysiert.

Tabelle 10. Beispiel: URL-Datensatzparsing.

Generiertes Feld	**Inhalte**
*URL-Feld*_server	*http://www.dummydomain.co.uk*
*URL-Feld*_path	*/resource.php*
*URL-Feld*_field	*res_id*
*URL-Feld*_value	*89*

So verwenden Sie den Knoten "URL Parser":

1. Falls Sie den Stream des vorangegangenen Beispiels geschlossen haben, öffnen Sie ihn wieder. Der Stream enthält die Knoten "Apache Log Reader" und "Typ".
2. Verbinden Sie auf der Registerkarte "Feldoperationen" der Knotenpalette einen Knoten "URL Parser" mit dem Typknoten.
3. Bearbeiten Sie den Knoten "URL Parser". Wählen Sie in der Dropdown-Liste "ID-Feld" die Option **ReturnedContentSize** aus. Wählen Sie in der Dropdown-Liste "URL-Feld" die Option **ReferralURL** aus. Klicken Sie auf **OK**.
4. Verbinden Sie einen Tabellenknoten mit dem Knoten "URL Parser" und führen Sie den Stream aus. Die Felder **ReturnedContentSize** und **ReferralURL** werden angezeigt, wobei **ReferralURL** zusätzlich in die folgenden vier separat generierten Felder geparst wird: **ReferralURL_server**, **ReferralURL_path**, **ReferralURL_field** und **ReferralURL_value**.

Dokumenterstellungsknoten (Web Status Report)

Im Beispiel für den Dokumenterstellungsknoten werden die vom Web-Server-Protokoll übergebenen Daten eingelesen. Aus den eingelesenen Daten wird ein Bericht im Format HTML generiert. Der Bericht besteht aus einer Tabelle, die angibt, mit welcher Wahrscheinlichkeit (in Prozent) die einzelnen Protokolldatensätze verschiedene HTTP-Statuscodes zurückgeben (z. B. 200, 302, 404 usw.).

So verwenden Sie den Knoten "Web Status Report":

1. Falls Sie den Stream des ersten Beispiels geschlossen haben, öffnen Sie ihn wieder. Es handelt sich hier um den Stream, der die Knoten "Apache Log Reader" und "Typ" enthält. Wenn Ihr Stream bereits den Knoten "URL Parser" des zweiten Beispiels enthält, wird dieser Knoten in diesem Beispiel ignoriert.
2. Verbinden Sie auf der Registerkarte "Ausgabe" der Knotenpalette einen Knoten "Web Status Report" mit dem Typknoten.
3. Bearbeiten Sie den Knoten "Web Status Report". Wählen Sie in der Dropdown-Liste "Status Code Field" (Statuscodefeld) die Option **StatusCode** aus. Klicken Sie auf **Ausführen**. Ein Ausgabefenster mit dem Inhalt des Berichts wird angezeigt.

Modellerstellungsknoten (Interaction)

Das Beispiel für den Modellerstellungsknoten ist unabhängig von den anderen Beispielen. Es ermöglicht die Erstellung eines einfachen Modells mit dem nicht interaktiven Standardverfahren bzw. die Generierung eines Modells nach voriger Benutzerinteraktion. Das Modell erstellt Vorhersagen über die Kundenabwanderung für ein Telekommunikationsunternehmen.

Anmerkung: Dieses Beispiel verwendet Windows-spezifische API-Aufrufe zum Erstellen und Verwalten von Threads. Es wird daher nur auf Windows-Plattformen unterstützt.

Verwenden des Knotens "Interaction"

1. Wenn Sie die Beispiele von CLEF noch nicht aktiviert haben, tun Sie es jetzt. Weitere Informationen finden Sie in „Aktivieren der Beispiele" auf Seite 25.
2. Erstellen Sie in IBM SPSS Modeler einen neuen Stream.
3. Fügen Sie einen Statistikdateiquellenknoten hinzu, der die Datei *telco.sav* aus dem Verzeichnis *Demos* importiert.
4. Klicken Sie auf der Registerkarte "Typen" auf **Werte lesen** und zur Bestätigung im Nachrichtenfenster auf **OK**.
5. Setzen Sie die Rolle des Felds **Churn** ("Abwanderung", das letzte Feld in der Liste) auf **Ziel** und klicken Sie dann auf **OK**.
6. Verbinden Sie auf der Registerkarte "Modellierung" der Knotenpalette einen Knoten "Interaction" mit dem Quellenknoten.

Testen des nicht interaktiven Standardverfahrens der Modellerstellung

1. Führen Sie den Stream aus, um ein Modellnugget im Stream und in der Modellpalette oben rechts im Fenster zu erstellen.
2. Verbinden Sie einen Tabellenknoten mit dem Modellnugget.
3. Führen Sie den Tabellenknoten aus. Blättern Sie im Tabellenausgabefenster nach rechts, um die Verlustvorhersagen anzuzeigen. Im Feld `$I-churn` werden die Vorhersagen angezeigt, während das Feld `$IP-churn` die Konfidenzwerte für die einzelnen Vorhersagen enthält (von 0,0 bis 1,0).

Testen der interaktive Modellerstellung

1. Wählen Sie auf der Registerkarte "Modell" des Dialogfelds "Interaction Model Builder" die Option **Start an interactive session** aus.
2. Klicken Sie auf **Ausführen**, um das Dialogfeld "Interaction Test" zu öffnen.
3. Klicken Sie im Dialogfeld "Interaction Test" auf **Start Build Task**, um den Fortschritt der Modellerstellung anzuzeigen.
4. Wählen Sie nach Beendigung der Modellerstellung die Zeile aus, die im Dialogfeld der Tabelle mit den Erstellungsaufgaben hinzugefügt wurde.
5. Klicken Sie oben im Symbolleistenbereich des Dialogfelds auf die Schaltfläche mit dem gelben, kristallförmigen Symbol. Dadurch wird das Modellausgabeobjekt (mit dem Namen **model_1**) in der Modellpalette oben rechts im Fenster erstellt.

Das interaktiv generierte Modell ist mit Ausnahme seines Namens identisch mit dem zuvor erstellten Modell. Wenn Sie diesen Vorgang ab **Start Build Task** wiederholen, wird ein weiteres identisches Modell mit dem Namen **model_2** erstellt.

Untersuchen der Spezifikationsdateien

Wenn Sie wissen möchten, wie CLEF funktioniert, sollten Sie sich die Spezifikationsdateien der bereitgestellten Beispiele näher ansehen. Diese Dateien befinden sich in folgenden Verzeichnissen:

Installationsverzeichnis`\ext\lib\`*Erweiterungsordner*`\extension.xml`

Dabei ist *Installationsverzeichnis* das Installationsverzeichnis von IBM SPSS Modeler und *Erweiterungsordner* ist einer der folgenden Ordner:

- *spss.apachelogreader*
- *spss.interaction*
- *spss.urlparser*
- *spss.webstatusreport*

Eventuell befinden sich unter *\ext\lib* noch weitere Erweiterungsordner. Diese gehören zu IBM SPSS Modeler-Knoten, die während der Verwendung von CLEF vom System erstellt wurden. Welche Knoten in Ihrer Installation vorhanden sind, richtet sich nach den von Ihnen lizenzierten IBM SPSS Modeler-Modulen. Sie können sich diese Spezifikationsdateien gerne ansehen, um mehr über das Produkt zu erfahren, jedoch sollten Sie **diese Dateien in keiner Weise bearbeiten**. Andernfalls besteht die Gefahr, dass die Knoten nicht mehr richtig funktionieren. Sie müssten in diesem Fall die betreffenden IBM SPSS Modeler-Produkte neu installieren. Änderungen an den vom System bereitgestellten Dateien werden von IBM nicht unterstützt.

Untersuchen des Quellcodes

Als Referenz wird der vollständige Quellcode der Beispielknoten bereitgestellt. Alle Beispielknoten verwenden serverseitige C++-Bibliotheken. Der Knoten "Interaction" verwendet zusätzlich clientseitige Java-Klassen.

Die Quellcodedateien werden, sobald Sie die Beispiele aktivieren, automatisch extrahiert und wie in der folgenden Tabelle dargestellt) installiert.

Tabelle 11. Installation der Quellcodedatei

Speicherort	Inhalte
...\ext\lib\spss.interaction\src	Java-Quellcode für die `.class`-Dateien in der Datei `ui.jar` im übergeordneten Ordner
...\ext\bin\spss.apachelogreader\src *...\ext\bin\spss.interaction\src* *...\ext\bin\spss.urlparser\src* *...\ext\bin\spss.webstatusreport\src*	C++-Quellen- und Projektdateien für die DLLs im übergeordneten Ordner

Entfernen der Beispiele

Wenn Sie in IBM SPSS Modeler keine Beispielknoten anzeigen möchten, können Sie diese wie folgt entfernen:

1. Beenden Sie IBM SPSS Modeler.
2. Löschen Sie die Beispielordner sowohl aus dem Verzeichnis *\ext\bin* als auch aus dem Verzeichnis *\ext\lib* Ihrer IBM SPSS Modeler-Installation. Achten Sie darauf, nicht versehentlich die IBM SPSS Modeler-Standardordner zu löschen. In diesem Fall müssten Sie das betreffende IBM SPSS Modeler-Produkt neu installieren. Folgende Ordner dürfen gelöscht werden:

- *spss.apachelogreader*
- *spss.urlparser*
- *spss.webstatusreport*
- *spss.interaction*

Die Änderungen werden wirksam, sobald Sie IBM SPSS Modeler das nächste Mal starten.

Kapitel 4. Spezifikationsdatei

Überblick über Spezifikationsdateien

Alle CLEF-Erweiterungen müssen über eine XML-Datei verfügen, in der sämtliche Erweiterungsmerkmale definiert sind. Diese Datei wird als **Spezifikationsdatei** bezeichnet und hat stets den Namen `extension.xml`. Eine Spezifikationsdatei enthält folgende Abschnitte:

- **XML-Deklaration**. Optionale Deklaration der XML-Version und weitere Informationen.
- **Extension-Element**. Hauptteil der Datei; enthält alle nachfolgenden Abschnitte.
- **Abschnitt 'ExtensionDetails'**. Legt grundlegende Informationen über die Erweiterung fest.
- **Abschnitt 'Resources'**. Gibt externe Ressourcen an, die für das Funktionieren der Erweiterung erforderlich sind, wie Ressourcenbundles, JAR-Dateien und freigegebene Bibliotheken.
- **Abschnitt 'CommonObjects'**. (Optional) Definiert Elemente, die von anderen Objekten in der Erweiterung verwendet oder referenziert werden können, wie Modelle, Dokumente und Eigenschaftstypen.
- **Abschnitt 'UserInterface (Palettes)'**. (Optional) Definiert eine benutzerdefinierte Palette oder Unterpalette, auf der ein Knoten angezeigt werden soll.
- **Abschnitt 'ObjectDefinition'**. Identifiziert die durch die Erweiterung definierten Objekte, wie Knoten, Modellausgaben und Dokumentausgaben.

Jeder Abschnitt kann statische Deklarationen enthalten (wie Komponenten in einem Element) oder einfache dynamische Prozesse (wie die Berechnung eines Ausgabedatenmodells eines Knotens) oder beides. Das Gesamtformat einer CLEF-Spezifikationsdatei sieht wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<Extension ... >
   <ExtensionDetails ... />
   <Resources
      Abschnitt 'Resources'
   </Resources>
   <CommonObjects>
      Abschnitt 'CommonObjects'
   </CommonObjects>
   <UserInterface>
      Abschnitt 'UserInterface (Palettes)'
   </UserInterface>
   Abschnitt 'ObjectDefinition'
     Objektdefinition
     Objektdefinition
     Objektdefinition
    ...
</Extension>
```

Kommentarzeilen

In der Spezifikationsdatei können Sie jederzeit eine Kommentarzeile im folgenden Format einfügen:

```
<!- Kommentartext ->
```

Erforderlich oder optional?

In den Elementdefinitionen in nachfolgenden Abschnitten (normalerweise durch die Überschrift **Format** gekennzeichnet) sind Elementattribute und untergeordnete Elemente optional, sofern sie nicht als "(erforderlich)" gekennzeichnet sind. Die vollständige Elementsyntax finden Sie in „CLEF-XML-Schema”, auf Seite 207.

Beispiel für eine Spezifikationsdatei

Hier sehen Sie ein vollständiges Beispiel für eine CLEF-Spezifikationsdatei für einen einfachen Datentransformationsknoten.

```
<?xml version="1.0" encoding="UTF-8"?>
<Extension version="1.0" debug="true">
     <ExtensionDetails id="urlparser" providerTag="spss" label="URL CLEF Module" version="1.0"
provider="IBM Corp." copyright="(c) 2005-2011 IBM Corp." description="A Url Transform CLEF Extension"/>
     <Resources>
          <SharedLibrary id="urlparser_library" path="spss.urlparser/urlparser" />
     </Resources>
     <Node id="urlparser_node" type="dataTransformer" palette="fieldOp" label="URL Parser">
          <Properties>
               <Property name="id_fieldname" valueType="integer" label="ID field" />
               <Property name="url_fieldname" valueType="string" label="URL field" />
          </Properties>
          <UserInterface>
               <Icons />
               <Tabs>
                    <Tab label="Types" labelKey="optionsTab.LABEL">
                         <PropertiesPanel>
                              <SingleFieldChooserControl property="id_fieldname"  storage="integer" />
                              <SingleFieldChooserControl property="url_fieldname"  storage="string" />
                         </PropertiesPanel>
                    </Tab>
               </Tabs>
               <Controls />
          </UserInterface>
          <Execution>
               <Module libraryId="urlparser_library" name="">
                    <StatusCodes>
                         <StatusCode code="0" status="error" message="Peer kann nicht initialisiert werden" />
                         <StatusCode code="1" status="error" message="Fehler beim Lesen der Eingabedaten" />
                         <StatusCode code="2" status="error" message="Interner Fehler" />
                         <StatusCode code="3" status="error" message="Eingabefeld nicht vorhanden" />
                    </StatusCodes>
               </Module>
          </Execution>
          <OutputDataModel mode="replace">
               <AddField name="${id_fieldname}" fieldRef="${id_fieldname}"/>
               <AddField name="${url_fieldname}" fieldRef="${url_fieldname}"/>
               <AddField name="${url_fieldname}_server" storage="string" />
               <AddField name="${url_fieldname}_path" storage="string" />
               <AddField name="${url_fieldname}_field" storage="string" />
               <AddField name="${url_fieldname}_value" storage="string" />
          </OutputDataModel>
     </Node>
</Extension>
```

Das Element `ExtensionDetails` liefert grundlegende Informationen über die Erweiterung, die intern durch IBM SPSS Modeler verwendet wird.

Das Element `Resources` gibt den Speicherort einer serverseitigen Bibliothek an, die später in der Datei referenziert wird. Die Pfadspezifikation gibt an, dass sich die Bibliothek im IBM SPSS Modeler-Installationsverzeichnis unter `\ext\bin\spss.urlparser\urlparser.dll` befindet.

Diese Spezifikationsdatei enthält kein `CommonObjects`-Element.

Das Element `Node` gibt alle Informationen über den Knoten als solchen an:

- Unter `Properties` werden anfangs zwei Eigenschaften deklariert, die später auf einer Registerkarte des Knotendialogfelds verwendet werden.
- Das Element `UserInterface` definiert das Aussehen und das Layout der Registerkarte des Kontendialogfelds, das zu dieser Erweiterung gehört (weitere Registerkarten werden durch IBM SPSS Modeler bereitgestellt).

- Das Element `Execution` definiert Elemente, die beim Ausführen des Knotens verwendet werden. In diesem Fall handelt es sich bei den Elementen um die serverseitige Bibliothek, die zuvor in der Datei deklariert wurde, und um einen Satz Nachrichten, die angezeigt werden, wenn bei der Ausführung ein bestimmter Statuscode zurückgegeben wird.
- Das Element `OutputDataModel` definiert die Datentransformation, die dieser Knoten durchführt. Es legt fest, dass das Eingabedatenmodel (der Satz an Feldern, der als Eingabe für den Knoten verwendet wird) durch den hier definierten Satz an Feldern ersetzt werden soll, wodurch das Ausgabedatenmodell erstellt wird (der Satz an Feldern, der von hier aus an alle nachgeordneten Knoten übergeben wird, sofern das Modell anschließend nicht weiter angepasst wird). In diesem Beispiel übergibt der Knoten die beiden Originalfelder (`id_fieldname` und `url_fieldname`) unverändert, fügt aber vier weitere Felder hinzu, deren Namen von `url_fieldname` abgeleitet sind.

Die spezielle Spezifikationsdatei wird von einem der Beispielknoten übernommen, die im Rahmen der IBM SPSS Modeler-Installation bereitgestellt werden. Weitere Informationen finden Sie im Thema „Datentransformationsknoten (URL Parser)” auf Seite 27.

XML-Deklaration

Die XML-Deklaration ist optional und gibt an, welche XML-Version verwendet sowie Details des Zeichencodierungsformats.

Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
```

Extension-Element

Das Extension-Element bildet den Hauptteil der Datei und enthält alle anderen Abschnitte. Format:

```
<Extension version="Versionsnummer" debug="true_false">
     Abschnitt 'ExtensionDetails'
     Abschnitt 'Resources'
     Abschnitt 'CommonObjects'
     Abschnitt 'UserInterface (Palettes)'
     Abschnitt 'ObjectDefinition'
</Extension>
```

Dabei gilt Folgendes:

`version` ist die Versionsnummer der Erweiterung.

`debug` ist optional; wenn `true` festgelegt ist, wird allen CLEF-Knoten oder -Ausgaben zugeordneten Dialogfeldern oder Rahmen eine Registerkarte **Debuggen** hinzugefügt und es wird Zugriff auf die für das Objekt definierten Eigenschaften und Container bereitgestellt. Der Standardwert lautet `false`. Weitere Informationen finden Sie im Thema „Verwenden der Registerkarte "Debuggen"” auf Seite 204.

Abschnitt 'ExtensionDetails'

Der Abschnitt 'ExtensionDetails' enthält grundlegende Informationen zur Erweiterung.

Format

```
<ExtensionDetails providerTag="Erweiterungs-Provider-Tag"
     id="eindeutige_Kennung_der_Erweiterung"
     label="Anzeigename" version="Versionsnummer_der_Erweiterung"
     provider="Provider_der_Erweiterung" copyright="Copyrightvermerk"
     description="Beschreibung_der_Erweiterung"/>
```

Dabei gilt Folgendes:

`providerTag` (erforderlich) ist der Name, der den Provider dieser Erweiterung eindeutig definiert. Achten Sie darauf, dass die Zeichenfolge `spss` innerhalb des Werts nicht vorkommen darf, da diese für die interne Verwendung reserviert ist.

`id` (erforderlich) ist ein Name, der diese Erweiterung eindeutig identifiziert und in Systemnachrichten verwendet wird. In der Regel wird die Erweiterungsdatei in einem Ordner namens `\ext\lib\` *providerTag.id* im IBM SPSS Modeler-Installationsverzeichnis gespeichert.

`label` (erforderlich) ist die Anzeigebezeichnung der Erweiterung. Dieser Text wird im Feld 'Name' des Palettenmanagers angezeigt, wenn der Knoten hinzugefügt wird. Weitere Informationen finden Sie im Thema „Testen einer CLEF-Erweiterung" auf Seite 203.

`version` ist die Versionsnummer der Erweiterung.

`provider` ist eine Zeichenfolge, die den Provider der Erweiterung angibt. Dieser Text wird im Feld 'Provider' des Palettenmanagers angezeigt, wenn der Knoten hinzugefügt wird. Der Standardwert ist die Zeichenfolge `(unknown)`.

`copyright` ist der Copyrighthinweis für die Erweiterung. Dieser Text wird im Feld 'Copyright' des Palettenmanagers angezeigt, wenn der Knoten hinzugefügt wird.

`description` ist eine kurze Beschreibung des Zwecks der Erweiterung. Dieser Text wird im Feld 'Beschreibung' des Palettenmanagers angezeigt, wenn der Knoten hinzugefügt wird.

Beispiel

```
<ExtensionDetails providerTag="myco" id="sorter" name="Sort Data" version="1.2"
     provider="My Company Inc." copyright="(c) 2005-2006 My Company Inc."
     description="Beispielerweiterung, die Daten nach integrierten Betriebssystembefehlen sortiert."/>
```

Abschnitt 'Resources'

Dieser Abschnitt definiert, welche externen Ressourcen erforderlich sind, damit die Erweiterung korrekt funktioniert.

Format

```
<Resources>
     <Bundle .../>
      ...
     <JarFile .../>
      ...
     <SharedLibrary .../>
      ...
     <HelpInfo .../>
</Resources>
```

Dabei gilt Folgendes:

`Bundle` identifiziert einen Satz clientseitig lokalisierter Ressourcen. Weitere Informationen finden Sie im Thema „Bundles" auf Seite 35.

`JarFile` identifiziert eine clientseitige Java-JAR-Datei. Weitere Informationen finden Sie im Thema „JAR-Dateien" auf Seite 35.

`SharedLibrary` identifiziert eine serverseitige Bibliothek oder DLL. Weitere Informationen finden Sie im Thema „Freigegebene Bibliotheken" auf Seite 36.

`HelpInfo` gibt den Typ der Hilfeinformationen für die Erweiterung an, sofern welche vorliegen. Weitere Informationen finden Sie im Thema „Implementieren eines Hilfesystems" auf Seite 165.

Beispiel

```
<Resources>
     <SharedLibrary id="discriminantnode" path="spss.xd/Discriminant"/>
     <Bundle id="translations.discrim" type="properties" path="messages"/>
     <JarFile id="java" path="discriminant.jar"/>
     <HelpInfo id="help" type="native"/>
</Resources>
```

Bundles

Das Element `Bundle` gibt ein clientseitiges Ressourcenbundle an (wie einen Satz Nachrichtentexte für die Lokalisierung), das entweder als `.properties`-Datei oder als Java-Klassendatei (`.class`) implementiert ist. Weitere Informationen finden Sie im Thema „Lokalisierung" auf Seite 169.

Format

```
<Bundle id="ID" path="Pfad"/>
```

Dabei gilt Folgendes:

`id` (erforderlich) ist ein eindeutiges Kennzeichen für dieses Bundle.

`path` (erforderlich) gibt den Speicherort der Bundledatei relativ zum übergeordneten Ordner der Spezifikationsdatei an. Wenn sich das Bundle auf eine `.properties`-Datei bezieht, muss der Pfad keine Spracherweiterungen oder den `.properties`-Suffix enthalten.

Beispiel

```
<Bundle id="translations.discrim" path="messages"/>
```

Dies gibt an, dass ein Ressourcenbundle in einer Datei namens `messages.properties` in demselben Ordner wie die Spezifikationsdatei vorhanden ist.

JAR-Dateien

Das Element `JarFile` gibt eine clientseitige Java-Archivdatei (`.jar`) an, die Java-Klassen und sonstige clientseitige Ressourcen für die Erweiterung bereitstellt.

Format

```
<JarFile id="ID" path="Pfad"/>
```

Dabei gilt Folgendes:

`id` (erforderlich) ist ein eindeutiges Kennzeichen für die `JAR`-Datei.

`path` (erforderlich) gibt den Speicherort der `JAR`-Datei relativ zum übergeordneten Ordner der Spezifikationsdatei an.

Beispiel

```
<JarFile id="java" path="coxreg_model_terms.jar"/>
```

Dies gibt an, dass sich die JAR-Datei für diese Erweiterung in demselben Ordner wie die Spezifikationsdatei befindet.

Freigegebene Bibliotheken

Das Element SharedLibrary gibt eine serverseitig freigegebene Bibliothek oder DLL an. Dies ist in der Regel nur zur Unterstützung der Knotenausführung erforderlich. Wenn eine Bibliothek mehrere Module implementiert, identifiziert ein Module-Element im Ausführungsabschnitt der Knotenspezifikation ein bestimmtes Modul innerhalb der Bibliothek.

Format

```
<SharedLibrary id="Kennung" path="Pfad"/>
```

Dabei gilt Folgendes:

id (erforderlich) ist ein eindeutiges Kennzeichen für die freigegebene Bibliothek.

path (erforderlich) gibt den Speicherort der freigegebenen Bibliothek relativ zum Ordner \ext\bin im serverseitigen Installationsverzeichnis an. Beachten Sie, dass der Pfad die Dateierweiterung (z. B. .dll) der freigegebenen Bibliothek nicht enthalten muss.

Beispiel

Folgende Deklaration einer freigegebenen Bibliothek:

```
<SharedLibrary id="binning" path="spss.binning/Binning" />
```

gibt an, dass die freigegebene Bibliothek geladen wird aus:

```
Installationsverzeichnis\ext\bin\spss.binning\Binning.dll
```

Dabei ist *Installationsverzeichnis* das Verzeichnis, in dem die serverseitigen CLEF-Komponenten installiert sind. Da diese Bibliothek mehr als ein Modul implementiert, wird das konkret benötigte Modul (supervisedBinning) mittels eines Module-Elements in der Spezifikationsdatei des Erstellungsknotens identifiziert, wobei das Bibliothekskennzeichen wie folgt referenziert wird:

```
<Execution>
     <Module libraryId="binning" name="supervisedBinning" .../>
 ...
</Execution>
```

Hilfeinformationen

Das optionale Element HelpInfo gibt an, welche der möglichen Hilfetypen für die Erweiterung bereitgestellt werden. Weitere Informationen finden Sie im Thema „Implementieren eines Hilfesystems” auf Seite 165.

Abschnitt 'CommonObjects'

Der optionale Abschnitt 'CommonObjects' definiert Objekte, die von an anderer Stelle in der Spezifikationsdatei definierten Elementen gemeinsam genutzt werden können. Einige Objekttypen in diesem Abschnitt (wie Eigenschaftsaufzählungen) können auch dort, wo sie benötigt werden, lokal definiert werden, während andere (wie Modelle und Dokumente) nur hier definiert werden können.

Format

```
<CommonObjects>
     <PropertyTypes .../>
     <PropertySets .../>
```

```
     <ContainerTypes .../>
     <Actions .../>
     <Catalogs .../>
</CommonObjects>
```

Dabei gilt Folgendes:

`PropertyTypes` erlaubt, dass allgemeine Eigenschaftsdefinitionen von Objekten gemeinsam verwendet werden. Weitere Informationen finden Sie im Thema „Eigenschaftstypen".

`PropertySets` wird in der Regel verwendet, wenn Modellierungsknoten, Modellausgabeobjekte und Modellanwendungsknoten denselben Satz an Eigenschaften enthalten. Weitere Informationen finden Sie im Thema „Eigenschaftssätze" auf Seite 38.

`ContainerTypes` definiert Typen von Containern, bei denen es sich um Objekte handelt, die komplexe Datenstrukturen aufnehmen können. Weitere Informationen finden Sie im Thema „Containertypen" auf Seite 39.

`Actions` definiert grundlegende Informationen zu Benutzerinteraktionen, z. B. mithilfe von Menüs oder Symbolleisten. Weitere Informationen finden Sie im Thema „Aktionen" auf Seite 40.

`Catalogs` implementiert ein Steuerelement, mit dessen Hilfe Sie eine oder mehrere Optionen aus einer Werteliste auswählen können, die der Server dynamisch generiert. Weitere Informationen finden Sie im Thema „Kataloge" auf Seite 41.

Beispiel

```
<CommonObjects>
     <ContainerTypes>
          <ModelType id="discriminant_model" format="utf8" />
          <DocumentType id="html_output" />
          <DocumentType id="zip_outputType" format="binary"/>
     </ContainerTypes>
</CommonObjects>
```

Eigenschaftstypen

Der optionale Abschnitt 'PropertyTypes' erlaubt die gemeinsame Verwendung von Eigenschaftsdefinitionen durch Objekte. Dies dient teilweise einer leichteren Wartung - so kann die Definition einer Eigenschaft z. B. an einer einzelnen Stelle hinterlegt sein und muss nicht an mehrere Stellen kopiert werden. Die Freigabe von Definitionen wird außerdem verwendet, um die Kompatibilität von Eigenschaften in verschiedenen Objekten sicherzustellen, wenn deren Werte kopiert werden, sobald eine neue Instanz eines Objekts erstellt wird.

Eigenschaftstypen können nur im Abschnitt 'CommonObjects' definiert werden.

Format

```
<PropertyTypes>
     <PropertyType id="ID" isKeyed="true_false" isList="true_false" max="Maximalwert"
       min="Mindestwert" valueType="Werttyp">
          <Enumeration ... />
          <Structure ... />
          <DefaultValue ... />
     </PropertyType>
     <PropertyType ... />
      ...
</PropertyTypes>
```

Die PropertyType-Attribute sind im Folgenden aufgeführt.

id (erforderlich) ist ein eindeutiges Kennzeichen für den Eigenschaftstyp.

isKeyed gibt, sofern die Einstellung true vorgenommen wurde, an, dass der Eigenschaftstyp verschlüsselt ist. Eine verschlüsselte Eigenschaft ordnet eine Gruppe von Operationen durch ein benutzerdefiniertes Steuerelement einem Feld zu (siehe „Eigenschaftssteuerelement" auf Seite 140). Wenn isKeyed auf true eingestellt ist, muss das Attribut valueType auf structure eingestellt sein. Weitere Informationen zu strukturierten Eigenschaften finden Sie in „Strukturierte Eigenschaften" auf Seite 62.

isList gibt an, ob es sich bei der Eigenschaft um eine Liste von Werten des angegebenen Werttyps (true) handelt oder um einen einzelnen Wert (false).

max und min geben den maximalen und den minimalen Wert für einen Bereich an.

valueType kann einen der folgenden Werte haben:

- string
- encryptedString
- fieldName
- integer
- double
- boolean
- date
- enum (siehe „Aufgezählte Eigenschaften" auf Seite 61)
- structure (siehe „Strukturierte Eigenschaften" auf Seite 62)
- databaseConnection

Die untergeordneten Elemente Enumeration und Structure schließen sich gegenseitig aus. Die untergeordneten Elemente Enumeration, Structure und DefaultValue werden in speziellen Fällen verwendet. Siehe hierzu „Aufgezählte Eigenschaften" auf Seite 61, „Strukturierte Eigenschaften" auf Seite 62 und „Standardwerte" auf Seite 64.

Eigenschaftssätze

Eigenschaftssätze werden in der Regel verwendet, wenn Modellierungsknoten, Modellausgabeobjekte und Modellanwendungsknoten denselben Satz an Eigenschaften enthalten. Beispiel: Ein Modellierungsknoten kann einen Standardsatz von Eigenschaften definieren, die im Builder festgelegt werden können, die aber erst bei der Modellanwendung verwendet werden. Damit sie automatisch übertragen werden, müssen sie außerdem in der Modellausgabe enthalten sein.

Format

```
<PropertySets>
     <PropertySet id="ID">
          <Property ... />
          <Property ... />
            ...
     </PropertySet>
       ...
</PropertySets>
```

Dabei ist id ein eindeutiges Kennzeichen für den Eigenschaftssatz.

Eine Beschreibung des Elements 'Property' finden Sie in „Eigenschaften" auf Seite 52.

Beispiel

Dieses Beispiel demonstriert die Definition eines Sets von zwei Eigenschaften: die Anzahl der zu erzeugenden Vorhersagen und ob Wahrscheinlichkeiten enthalten sein sollen. Im Abschnitt 'CommonObjects' erfolgt folgende Definition:

```
<PropertySets>
     <PropertySet id="common_model_properties">
          <Property name="prediction_count" valueType="integer" min="1" max="10"/>
          <Property name="include_probabilities" valueType="boolean" defaultValue="false"/>
     </PropertySet>
       ...
</PropertySets>
```

Anschließend wird in alle Definitionen für den Modellierungsknoten, das Modellausgabeobjekt und den Modellanwendungsknoten ein `includePropertySets`-Attribut eingefügt, das wie folgt aussieht (diese Definition gilt nur für den Modellierungsknoten):

```
<Node id="my_builder" type="modelBuilder" ... >
     <Properties includePropertySets="[common_model_properties]">
      ...
     </Properties>
      ...
</Node>
```

Containertypen

Container sind Objekte, die als Platzhalter für komplexe Datenstrukturen wie Modelle und Dokumente agieren. Ein Container wird als bestimmter Containertyp definiert. Im Folgenden sind die Typdefinitionen aufgeführt. Folgende Containertypen können definiert werden:

- Modelltypen
- Dokumenttypen

Containertypen können geklont zwischen Client und Server übertragen und in einer Datei oder einem Inhalts-Repository gespeichert werden. Ein Modell wird geklont, wenn aus einem Modellausgabeobjekt ein Modellanwendungsknoten generiert wird.

Jeder Containertyp verfügt über einen vordefinierten Satz von Eigenschaften, obwohl benutzerdefinierte Eigenschaften hinzugefügt werden können. Containertypen können nur im Abschnitt 'CommonObjects' definiert werden.

Format

Das Format des Abschnitts 'ContainerTypes' sieht wie folgt aus:

```
<ContainerTypes>
     <ModelType ... />
      ...
     <DocumentType ... />
      ...
</ContainerTypes>
```

Dabei gilt Folgendes:

`ModelType` gibt das Format für einen bestimmten Modelltyp an. Weitere Informationen finden Sie im Thema „Modelltypen" auf Seite 40.

`DocumentType` gibt das Format für einen bestimmten Dokumenttyp an. Weitere Informationen finden Sie im Thema „Dokumenttypen" auf Seite 40.

Beispiel

```
<ContainerTypes>
     <ModelType id="discriminant_model" format="utf8">
     <DocumentType id="html_output" />
     <DocumentType id="zip_outputType" format="binary"/>
</ContainerTypes>
```

Modelltypen

Ein Modell muss Informationen bereitstellen - wie Algorithmusname, Modelltyp sowie Eingabe- und Ausgabedatenmodell. Eine Modelltypdefinition gibt das Format für einen bestimmten Modelltyp an.

Die Modelltypdefinition kann hier statisch in der Spezifikationsdatei festgelegt sein oder dynamisch, wenn das Modell durch den Modellierungsknoten erstellt wird.

Format

```
<ModelType id="Kennung" format="Format_des_Modelltyps" />
```

Dabei gilt Folgendes:

- `id` (erforderlich) ist ein eindeutiges Kennzeichen für den Modelltyp.
- `format` (erforderlich) ist das Format des Modelltyps. Es kann entweder `utf8` (Text) oder `binary` sein. Das Modellformat muss als Teil der statischen Informationen angegeben sein.

Beispiel

```
<ModelType id="my_model" format="utf8" />
```

Dokumenttypen

Ein **Dokument** ist ein Ausgabeobjekt wie ein Diagramm oder ein Bericht. Eine Dokumenttypdefinition gibt das Format für einen bestimmten Dokumenttyp an.

Format

```
<DocumentType id="Kennung" format="Format_des_Dokumenttyps" />
```

Dabei gilt Folgendes:

- `id` (erforderlich) ist ein eindeutiges Kennzeichen für den Dokumenttyp.
- `format` (erforderlich) ist das Format des Dokumenttyps. Es kann entweder `utf8` (Text) oder `binary` sein.

Beispiele

```
<DocumentType id="html_output" format="utf8" /> <DocumentType id="zip_outputType" format="binary"/>
```

Aktionen

Aktionen definieren grundlegende Informationen zu Benutzerinteraktionen, z. B. mittels Menüs oder Symbolleisten. Jede Aktion definiert, wie sie auf der Benutzerschnittstelle dargestellt werden soll, wie z. B. als Beschriftung, QuickInfo oder Symbol. Eine Sammlung von Aktionen wird durch eine clientseitige Java-Klasse behandelt, die für jede Aktionsgruppe definiert ist. Aktionen können auch innerhalb bestimmter Objekte definiert werden.

Format

```
<Actions>
     <Action id="ID" label="Anzeigebeschriftung" labelKey="Beschriftungsschlüssel"
          description="Aktionsbeschreibung" descriptionKey="Beschreibungsschlüssel" imagePath="Bildpfad"
```

```
        imagePathKey="Bildpfadschlüssel" mnemonic="mnemonisches_Zeichen" mnemonicKey="mnemonische_Taste"
        shortcut="Verknüpfungszeichenfolge" shortcutKey="Tastenkombination" />
    ...
</Actions>
```

Dabei gilt Folgendes:

`id` (erforderlich) ist ein eindeutiges Kennzeichen für die Aktion.

`label` (erforderlich) ist der Anzeigename für die Aktion, mit dem sie auf der Benutzerschnittstelle angezeigt wird.

`labelKey` gibt die Beschriftung für Lokalisierungszwecke an.

`description` ist eine Beschreibung der Aktion. Für ein benutzerdefiniertes Menüelement oder ein Aktionsschaltflächensymbol in einer Symbolleiste wäre dies beispielsweise der Text der QuickInfo für das Menüelement oder die Schaltfläche.

`descriptionKey` gibt die Beschreibung für Lokalisierungszwecke an.

`imagePath` ist der Speicherort der Grafikdatei, wie beispielsweise ein Symbolbild. Der Speicherort wird relativ zu dem Verzeichnis angegeben, in dem die Spezifikationsdatei installiert ist.

`imagePathKey` gibt den Bildpfad für Lokalisierungszwecke an.

`mnemonic` ist der Buchstabe, der zusammen mit der Taste Alt gedrückt wird, um dieses Steuerelement zu aktivieren (wenn z. B. der Wert S angegeben ist, kann der Benutzer dieses Steuerelement über Alt+S aktivieren).

`mnemonicKey` gibt das mnemonische Zeichen für Lokalisierungszwecke an. Wenn weder `mnemonic` noch `mnemonicKey` verwendet wird, ist kein Direktaufruf dieser Aktion verfügbar. Weitere Informationen finden Sie im Thema „Zugriffstasten und Tastenkombinationen” auf Seite 115.

`shortcut` ist eine Zeichenfolge, die eine Tastenkombination angibt (z. B. `STRG+UMSCHALT+A`), die zum Initieren dieser Aktion verwendet werden kann.

`shortcutKey` gibt die Tastenkombination für Lokalisierungszwecke an. Wenn weder `shortcut` noch `shortcutKey` verwendet wird, ist kein Direktzugriff auf diese Aktion verfügbar. Weitere Informationen finden Sie im Thema „Zugriffstasten und Tastenkombinationen” auf Seite 115.

Beispiel

```
<Actions>
     <Action id="generateSelect" label="Select Node..." labelKey="generate.selectNode.LABEL"
      imagePath="images/generate.gif" description="Generates a select node"
      descriptionKey="generate.selectNode.TOOLTIP"/>
     <Action id="generateDerive" label="Derive Node..." labelKey="generate.deriveNode.LABEL"
      imagePath="images/generate.gif" description="Generates a derive node"
      descriptionKey="generate.deriveNode.TOOLTIP"/>
</Actions>
```

Kataloge

Kataloge ermöglichen es Ihnen, eine Eigenschaft einem Steuerelement zuzuordnen, damit der Benutzer eine oder mehrere Optionen aus einer Werteliste wählen kann, die der Server dynamisch erzeugt.

Die Werte werden im Steuerelement als Popup-Liste angezeigt, wenn der Benutzer auf den Eintrag **<Auswahl>** klickt.

Wenn der Benutzer eine Zeile in der Liste auswählt, wird der Zeilenwert aus einer Spalte, die im Catalog-Element angegeben ist, im Steuerelement platziert.

Format

```
<CommonObjects>
     <Catalogs>
          <Catalog id="ID" valueColumn="ganze_Zahl">
               <Attribute label="Anzeigename" />
                ...
          </Catalog>
           ...
     </Catalogs>
</CommonObjects>
```

Dabei gilt Folgendes:

id (erforderlich) ist ein eindeutiges Kennzeichen für den Katalog.

valueColumn (erforderlich) ist die Nummer der Spalte, deren Wert in das Steuerelement übertragen wird, sobald der Benutzer eine Zeile auswählt. Die Spaltennummerierung beginnt bei 1.

Verwenden Sie ein Attribute-Element pro Spalte in Spaltenreihenfolge (siehe folgendes Beispiel).

Wenn der Benutzer ein Steuerelement aktiviert, das einem Katalog zugeordnet ist, wird der Katalog mit der Werteliste durch einen Aufruf der Funktion getCatalogInformation vom Server abgerufen. Diese Funktion gibt ein XML-Dokument zurück, das die Werteliste enthält. Weitere Informationen finden Sie im Thema „Peerfunktionen” auf Seite 181.

Beispiel

In diesem Beispiel ist ein Auszug aus dem Code zur Definition von Steuerelementen für Kataloge dargestellt. Es werden drei Kataloge definiert und mit drei verschiedenen Steuerelementen auf der Registerkarte eines Dialogfelds verbunden.

Zuerst werden die Kataloge im Abschnitt 'CommonObjects' definiert.

```
<CommonObjects>
     <Catalogs>
          <Catalog id="cat1" valueColumn="1">
               <Attribute label="col1" />
               <Attribute label="col2" />
          </Catalog>
          <Catalog id="cat2" valueColumn="2">
               <Attribute label="col1" />
               <Attribute label="col2" />
               <Attribute label="col3" />
          </Catalog>
          <Catalog id="cat3" valueColumn="1">
               <Attribute label="col1" />
          </Catalog>
     </Catalogs>
</CommonObjects>
```

Als Nächstes werden die Eigenschaften, die den Steuerelementen zugeordnet werden sollen, im Abschnitt "Properties" der Knotendefinition definiert:

```
<Node id="catalognode" type="dataReader" palette="import" label="Catalog">
     <Properties>
          <Property name="sometext" valueType="string" label="Some Text" />
```

```
        <Property name="selection1" valueType="string" label="Selection 1" />
        <Property name="selection2" valueType="string" isList="true" label="Selection 2" />
        <Property name="selection3" valueType="string" label="Selection 3" />
    </Properties>
```

Im Abschnitt "UserInterface" der Knotendefinition werden die Steuerelemente definiert und durch Verweise auf die Eigenschaften den Katalogdefinitionen zugeordnet:

```
<UserInterface>
    <Tabs>
        <Tab label="Catalog Controls" labelKey="Catalog.LABEL" >
            <PropertiesPanel>
                <TextBoxControl property="sometext" />
                <SingleItemChooserControl property="selection1" catalog="cat1" />
                <MultiItemChooserControl property="selection2" catalog="cat2" />
                <SingleItemChooserControl property="selection3" catalog="cat3" />
            </PropertiesPanel>
        </Tab>
```

Abschnitt 'UserInterface (Palettes)'

Dies ist ein optionaler Abschnitt, der nur dann angegeben wird, wenn die Erweiterung eine benutzerdefinierte Palette oder Unterpalette definiert, auf der der Knoten angezeigt werden soll.

Wenn eine Erweiterung eine benutzerdefinierte Palette oder Unterpalette definiert, kann bei nachfolgend geladenen Erweiterungen, die Knoten definieren, die auf derselben Palette oder Unterpalette enthalten sein sollen, der Abschnitt 'UserInterface (Palettes)' entfallen. In diesem Fall ist es lediglich erforderlich, dass für das Element `Node` ein Attribut `customPalette` angegeben ist, das die Palette referenziert. Erweiterungen werden in alphabetischer Reihenfolge des Werts für *providerTag*.*id* geladen, wobei dies die Werte der Attribute `providerTag` und `id` des Elements `ExtensionDetails` für die Erweiterung sind (siehe „Abschnitt 'ExtensionDetails'" auf Seite 33). So wird die Erweiterung `myco.abc` z. B. vor der Erweiterung `myco.def` geladen.

Hinweis: Der Abschnitt 'UserInterface (Palettes)' unterscheidet sich vom Hauptabschnitt 'UserInterface', der als Teil einer einzelnen Objektdefinition angegeben wird. Eine Beschreibung finden Sie in Kapitel 6, „Erstellen von Benutzerschnittstellen", auf Seite 105.

Format

Das Format des Abschnitts 'UserInterface (Palettes)' sieht wie folgt aus:

```
<UserInterface>
    <Palettes>
        <Palette id="Name" systemPalette="Palettenname" customPalette="Palettenname"
           relativePosition="Position" relativeTo="Palette" label="Anzeigebeschriftung"
           labelKey="Beschriftungsschlüssel" description="Beschreibung"
           descriptionKey="Beschreibungsschlüssel"
           imagePath="Bildpfad" />
        <Palette ... />
     ...
    </Palettes>
</UserInterface>
```

Tabelle 12. Palettenattribute.

Attribut	Beschreibung
`id`	(erforderlich) Eine eindeutige Kennung für die Palette oder Unterpalette, die definiert wird.

Tabelle 12. Palettenattribute (Forts.).

Attribut	Beschreibung
`systemPalette`	Wird nur verwendet, wenn eine Unterpalette zu einer Systempalette hinzugefügt wird. Gibt die Systempalette an, in der die Unterpalette vorkommen soll: `import` - Quellen `recordOp` - Datensatzoperationen `fieldOp` - Feldoperationen `graph` - Diagramme `modeling` - Modellierung (siehe unten) `dbModeling` - Datenbankmodellierung `output` - Ausgabe `export` - Export
`customPalette`	Wird nur verwendet, wenn eine Unterpalette zu einer benutzerdefinierten Palette hinzugefügt wird. Gibt die benutzerdefinierte Palette an, in der die Unterpalette vorkommen soll. Der Wert des `id`-Attributs des Elements `Palette`, das die benutzerdefinierte Palette definiert.

`relativePosition`	Wird nur verwendet, wenn eine benutzerdefinierte Palette definiert wird. Legt die Position auf dem Palettenstreifen am unteren Bildschirmrand fest.
	
	Mögliche Werte:
	
	`first`
	`last`
	`before`
	`after`
	
	Wenn der Wert `before` (vor) oder `after` (nach) ist, wird außerdem das Attribut `relativeTo` benötigt (siehe unten).
	
	Wenn `relativePosition` nicht angegeben wird, wird die Palette am Ende des Streifens platziert.

`relativeTo`	Wenn der Wert für `relativePosition` jeweils `before` oder `after` ist, wird mit `relativeTo` die Kennung der Palette angegeben, die vor oder nach dieser benutzerdefinierten Palette erscheint. Paletten-IDs werden als Werte des Palettenattributs des Knotenelements aufgeführt (siehe „Knoten” auf Seite 48).
`label`	(erforderlich) Der Anzeigename für die Palette oder Unterpalette, der auf der Benutzerschnittstelle angezeigt wird.
`labelKey`	Gibt die Beschriftung für Lokalisierungszwecke an.
`description`	Der Text der QuickInfo, die angezeigt wird, wenn der Cursor über die Registerkarte 'Palette' bewegt wird (wird für Unterpaletten nicht verwendet). Dieser Wert wird auch als lange aufrufbare Beschreibung des Steuerelements verwendet. Weitere Informationen finden Sie im Thema „Zugriffsmöglichkeiten” auf Seite 175.
`descriptionKey`	Gibt die Beschreibung für Lokalisierungszwecke an.
`imagePath`	Gibt den Speicherort des Bildes an, das auf der Registerkarte 'Palette' verwendet wird (wird für Unterpaletten nicht verwendet). Der Speicherort wird relativ zu dem Verzeichnis angegeben, in dem die Spezifikationsdatei installiert ist. Wenn dieses Attribut nicht angegeben ist, wird kein Bild verwendet.

Beispiel - Hinzufügen eines Knotens zu einer Systempalette

Angenommen ihr Unternehmen hat einen neuen Algorithmus zum Mining von Audio- und Videodaten entwickelt und Sie möchten den Algorithmus in IBM SPSS Modeler integrieren. Sie definieren zuerst einen benutzerdefinierten Knoten zum Lesen von Daten, der die eingegebenen Audio- und Videodateien liest.

Zuerst beschließen Sie, Ihren neuen Datenleserknoten zur Systempalette 'Quellen' hinzuzufügen. Hierzu müssen Sie lediglich die Palette 'Quellen' mithilfe des Attributs `palette` des Elements `Node` angeben. Weitere Informationen finden Sie im Thema „Knoten" auf Seite 48.

Um den Knoten nach dem Knoten 'Datenbank' zur Palette 'Quellen' hinzuzufügen, erstellen Sie folgende Anweisung:

```
<Node id="AVreader" type="dataReader" palette="import" relativePosition="after"
      relativeTo="database" label="AV Reader">
```

Beispiel - Hinzufügen einer benutzerdefinierten Palette

Es bietet sich zwar an, eine IBM SPSS Modeler-Standardpalette zu verwenden, Sie möchten Ihrem neuen Knoten aber zu mehr Prominenz verhelfen. Sie beschließen, für ihn eine benutzerdefinierte Palette zu definieren, die Sie nach der Favoritenpalette aber vor den Quellen platzieren möchten. Sie müssen zuerst einen Abschnitt 'UserInterface (Palettes)' hinzufügen, um die benutzerdefinierte Palette wie folgt zu definieren:

```
<UserInterface>
     <Palettes>
          <Palette id="AV_mining" label="AV Mining" relativePosition="before"
            relativeTo="import" description="Audio video mining"  />
     </Palettes>
</UserInterface>
```

Das Attribut `relativeTo` verwendet die interne ID der Quellenpalette, die `import` lautet.

Dann verändern Sie die `Node`-Definition wie folgt:

```
<Node id="AVreader" type="dataReader" customPalette="AV_mining" label="AV Reader">
```

Dadurch wird die Palette **AV Mining** zwischen der Favoriten- und der Quellenpalette platziert.

Beispiel - Hinzufügen einer benutzerdefinierten Unterpalette zu einer benutzerdefinierten Palette

Ausgehend vom vorherigen Beispiel, beschließen Sie nun beispielsweise, dass Sie den Datenleserknoten lieber in seine eigene Unterpalette **AV Sources** der Palette **AV Mining** eintragen möchten. Hierzu müssen Sie zuerst die Unterpalette angeben, indem Sie dem Abschnitt 'UserInterface (Palettes)' ein zweites Element `Palette` hinzufügen:

```
<UserInterface>
     <Palettes>
          <Palette id="AV_mining" label="AV Mining" description="Audio video mining" />
          <Palette id="AV_mining.sources" customPalette="AV_mining" label="AV Sources" />
     </Palettes>
<UserInterface>
```

Dann ändern Sie das `Node`-Element dahingehend, dass es die Unterpaletten-ID referenziert:

```
<Node id="AVreader" type="dataReader" customPalette="AV_mining.sources" label="AV-Leser">
```

Wenn der Benutzer jetzt auf die Registerkarte **AV Mining** klickt, werden zwei Unterpaletten angezeigt, eine mit der Beschriftung **Alle** und eine mit der Beschriftung **AV Sources**. Der AV-Leserknoten wird auf beiden angezeigt.

Wenn Sie einen weiteren neuen Knoten zu einer anderen neuen Unterpalette von **AV Mining** hinzufügen, wird der neue Knoten unter **Alle** und auf der neuen Unterpalette angezeigt, aber nicht auf der Unterpalette **AV Sources**.

Beispiel - Hinzufügen eines Knotens zu einer Systemunterpalette

Um die Audio- und Videoquellendaten zu verarbeiten, definieren Sie jetzt einen Modellierungsknoten. Sie beschließen, ihn zur Standardmodellierungspalette hinzuzufügen, die eine Anzahl von Standardunterpaletten enthält. Sie beschließen, ihn zur Unterpalette 'Klassifikation' hinzuzufügen und direkt vor dem Knoten für neuronale Netze zu platzieren:

```
<Node id="AVmodeler" type="modelBuilder" palette="modeling.classification"
      relativePosition="before" relativeTo="neuralnet" label="AV Modeler">
```

Beachten Sie, dass der Knoten in derselben relativen Position auch auf der Modellierungspalette in der Unterpalette 'Alle' hinzugefügt wird.

Beispiel - Hinzufügen einer benutzerdefinierten Unterpalette zu einer Systempalette

Wenn Sie sich die Anzahl der Modellierungsknoten auf der Unterpalette 'Klassifikation' noch einmal ansehen, werden Sie feststellen, dass es für Benutzer nicht einfach ist, Ihren neuen Knoten zu erkennen. Um Ihren Knoten sichtbarer zu machen, können Sie Ihre eigene Unterpalette zur Modellierungspalette hinzufügen und den Knoten dort platzieren.

Hierzu müssen Sie zuerst Ihre benutzerdefinierte Unterpalette definieren, indem Sie die Datei um einen Abschnitt 'UserInterface (Palettes)' ergänzen:

```
<UserInterface>
     <Palettes>
          <Palette id="modeling.av_modeling" systemPalette="modeling" label="AV Modeling"
             labelKey="av_modeling.LABEL" description="Enthält AV-Mining-relevante
             Modellierungsknoten" descriptionKey="av_modeling.TOOLTIP"/>
     </Palettes>
</UserInterface>
```

Beachten Sie, dass Sie `systemPalette` explizit angeben müssen, um die Systempalette festzulegen, die Sie erweitern.

Anschließend geben Sie im Hauptabschnitt 'UserInterface' für den Knoten an, dass sie auf dieser Unterpalette angezeigt werden soll:

```
<Node id="my.avmodeler" type="modelBuilder" customPalette="modeling.av_modeling"
      label="AV Modeler">
```

Benutzerdefinierte Unterpaletten werden immer nach den Systemunterpaletten platziert.

Hinweis: Wenn Sie weitere Knoten zur AV-Modellierungsunterpalette hinzufügen möchten, benötigen deren Spezifikationsdateien **keinen** Abschnitt 'UserInterface (Palettes)', sofern die AV-Modellierungserweiterung zuvor geladen wurde.

Ausblenden oder Löschen einer benutzerdefinierten Palette oder Unterpalette

Wenn eine benutzerdefinierte Palette oder Unterpalette nicht mehr angezeigt werden soll, können Sie diese mit dem IBM SPSS Modeler-Palettenmanager entweder ausblenden oder löschen.

Denken Sie daran, dass das Ausblenden für alle IBM SPSS Modeler-Sitzungen gilt, über das entsprechende Kontrollkästchen aber wieder rückgängig gemacht werden kann. Das Löschen kann innerhalb einer Sitzung nicht rückgängig gemacht werden. Beim Neustart von IBM SPSS Modeler wird das Symbol aller-

dings so lange wieder angezeigt, bis Sie es aus der Spezifikationsdatei entfernen oder die gesamte Erweiterung löschen. Weitere Informationen finden Sie im Thema „Deinstallieren von CLEF-Erweiterungen” auf Seite 206.

So blenden Sie eine Palette aus oder löschen sie:

1. Wählen Sie im IBM SPSS Modeler-Hauptmenü Folgendes aus:
   Tools > **Paletten verwalten**
2. Wählen Sie im Feld 'Palettenname' eine Palette aus und gehen Sie wie folgt vor:
   - Um die Palette auszublenden, inaktivieren Sie das entsprechende Kontrollkästchen 'Angezeigt?'.
   - Um die Palette zu löschen, klicken Sie auf die Schaltfläche 'Auswahl löschen'.
3. Klicken Sie auf 'OK'.

So blenden Sie eine Unterpalette aus oder löschen sie:

1. Wählen Sie im IBM SPSS Modeler-Hauptmenü Folgendes aus:
   Tools > **Paletten verwalten**
2. Wählen Sie im Feld 'Palettenname' eine Palette aus:
3. Klicken Sie auf die Schaltfläche 'Unterpaletten'.
4. Wählen Sie im Feld 'Unterpalettenname' eine Unterpalette aus und verfahren Sie wie folgt:
   - Um die Unterpalette auszublenden, inaktivieren Sie das entsprechende Kontrollkästchen 'Angezeigt?'.
   - Um die Unterpalette zu löschen, klicken Sie auf die Schaltfläche 'Auswahl löschen'.
5. Klicken Sie auf 'OK'.

Objektdefinitionsabschnitt

Elemente stellen die sichtbarsten Bestandteile einer Erweiterung dar. Der Objektdefinitionsabschnitt bildet den Abschluss der CLEF-Spezifikationsdatei und wird verwendet, um die verschiedenen Objekte in der Erweiterung zu definieren. Folgende Objekttypen können definiert werden:

- Knoten
- Modellausgabeobjekte
- Dokumentausgabeobjekte
- interaktive Ausgabeobjekte

Knoten sind Objekte, die in einem Stream enthalten sind. **Modellausgabeobjekte** werden durch Modellierungsknoten generiert und im Hauptfenster auf der Registerkarte 'Modelle' im Managerfenster angezeigt. Auf ähnliche Weise werden **Dokumentausgabeobjekte** durch Dokumenterstellungsknoten generiert und in demselben Bereich auf der Registerkarte 'Ausgaben' angezeigt. **Interaktive Ausgabeobjekte** werden durch interaktive Modellierungsknoten generiert und im Managerfenster auf der Registerkarte 'Ausgaben' angezeigt.

Der Objektdefinitionsabschnitt besteht aus mindestens einer solchen Objektdefinition.

Die Elemente, die für die verschiedenen Objekttypen definiert werden können, werden in den folgenden Abschnitten beschrieben. Einige dieser Elemente haben alle Objekttypen gemeinsam, während andere spezifisch für Knoten- oder Modellausgabedefinitionen sind. Objektspezifische Elemente werden im Text entsprechend gekennzeichnet.

- Objekt-ID
- Modellerstellung
- Dokumenterstellung
- Eigenschaften

- Container
- Benutzerschnittstelle
- Ausführung
- Ausgabedatenmodell
- Konstruktoren

Objekt-ID

Die Objekt-ID gibt den Objekttyp an, der wie folgt aussehen kann:

```
<Node .../>

<ModelOutput .../>

<DocumentOutput .../>

<InteractiveModelBuilder .../>
```

Die Objekt-ID bietet außerdem Informationen darüber, wie das Objekt in Scripts zur Verfügung steht. Das Attribut `scriptName` stellt einen eindeutigen Namen des Objekts dar. Scripts können dieses Attribut verwenden, um ein bestimmtes Objekt anzugeben (z. B. einen Knoten in einem Stream oder eine Ausgabe auf der Registerkarte 'Ausgaben').

Knoten

Eine Knotendefinition beschreibt ein Objekt, das in einem Stream vorkommen kann.

Format

```
<Node id="ID" type="Knotentyp" palette="Palette" customPalette="benutzerdefinierte_Palette"
     relativePosition="Position" relativeTo="Knoten" label="Anzeigebeschriftung"
     labelKey="Beschriftungsschlüssel"
     scriptName="Scriptname" helpLink="ID_des_Themas" description="Beschreibung"
     descriptionKey="Beschreibungsschlüssel" delegate="Java-Klasse">
     <ModelBuilder ... >
      ...
     </ModelBuilder>
     <DocumentBuilder ... >
      ...
     </DocumentBuilder>
     <ModelProvider ... />
     <Properties>
      ...
     </Properties>
     <Containers>
      ...
     </Containers>
     <UserInterface>
      ...
     </UserInterface>
     <Execution>
      ...
     </Execution>
     <OutputDataModel ...>
      ...
     </OutputDataModel>
```

```
    <Constructors>
      ...
    </Constructors>
</Node>
```

Die in der Knotendefinition zulässigen Elemente werden in den Abschnitten ab „Eigenschaften" auf Seite 52 beschrieben.

Tabelle 13. Node-Attribute.

Attribut	Beschreibung
`id`	(erforderlich) Ein Kennzeichen für den Knoten als Textzeichenfolge.
`type`	(erforderlich) Der Knotentyp: `dataReader` - Knoten, der Daten liest (z. B. Quellenpalettenknoten) `dataWriter` - Knoten, der Daten schreibt (z. B. Exportpalettenknoten) `dataTransformer` - Knoten, der Daten umwandelt (z. B. Datensatz-/Feldoperationsknoten) `modelBuilder` - Modellierungsknoten (z. B. Modellierungspalettenknoten) `documentBuilder` - Knoten, der ein Diagramm oder einen Bericht erstellt `modelApplier` - Knoten, der ein generiertes Modell enthält Der Knotentyp legt die Form des Knotensymbols in der Palette und im Erstellungsbereich fest. Weitere Informationen finden Sie im Thema „Überblick über die Knoten" auf Seite 9. Beim Knotentyp `modelBuilder` muss die Knotendefinition ein `ModelBuilder`-Element enthalten. Siehe „Modellerstellung" auf Seite 51. Beim Knotentyp `documentBuilder` muss die Knotendefinition ein `DocumentBuilder`-Element enthalten. Siehe „Dokumenterstellung" auf Seite 51.
`palette`	Die ID einer der IBM SPSS Modeler-Standardpaletten oder -Standardunterpaletten, auf denen der Knoten angezeigt wird: `import` - Quellen `recordOp` - Datensatzoperationen `fieldOp` - Feldoperationen `graph` - Diagramme `modeling` - Modellierung (siehe unten) `dbModeling` - Datenbankmodellierung `output` - Ausgabe `export` - Export Die Modellierungspalette verfügt über eine Anzahl von Standardunterpaletten: `modeling.classification` - Klassifikation `modeling.association` - Assoziation `modeling.segmentation` - Segmentierung `modeling.auto` - Automatisch Wenn Sie das Attribut `palette` weglassen, wird der Knoten auf der Palette 'Feldoperationen' angezeigt. Hinweis: Das Attribut `palette` wird nur für Modellerstellungsknoten verwendet.
`customPalette`	Die ID einer benutzerdefinierten Palette oder Unterpalette, auf der der Knoten angezeigt wird. Dies ist der Wert des `id`-Attributs eines `Palette`-Elements, der im Abschnitt 'UserInterfaces (Palettes)' der Datei angegeben ist. Weitere Informationen finden Sie im Thema „Abschnitt 'UserInterface (Palettes)'" auf Seite 43.

Tabelle 13. Node-Attribute (Forts.).

Attribut	Beschreibung

`relativePosition`	Gibt die Position des Knotens innerhalb der Palette an. Mögliche Werte:
	
	`first`
	`last`
	`before`
	`after`
	
	Wenn der Wert `before` (vor) oder `after` (nach) ist, wird außerdem das Attribut `relativeTo` benötigt (siehe unten).
	
	Wenn `relativePosition` nicht angegeben wird, wird der Knoten an das Ende der Palette platziert.

`relativeTo`	Wenn der Wert für `relativePosition` jeweils `before` oder `after` ist, wird mit `relativeTo` der Knoten in der Palette angegeben, die vor oder nach diesem Knoten erscheint. Der Wert von `relativeTo` ist der Scriptname des Knotens. Für einen IBM SPSS Modeler-Standardknoten finden Sie den Scriptnamen im Abschnitt 'Eigenschaftenreferenz' des IBM SPSS Modeler-Handbuchs für Scripterstellung und Automatisierung, allerdings ohne das Suffix `...node` (für den Datenbankknoten verwenden Sie z. B. `database` anstelle von `databasenode`). Für einen CLEF-Knoten ist dies der Wert des Attributs `scriptName` des Knotens.
`label`	(erforderlich) Der Anzeigename für den Knoten, der in der Palette, dem Erstellungsbereich und den Dialogfeldern angezeigt wird.
`labelKey`	Gibt die Beschriftung für Lokalisierungszwecke an.
`scriptName`	Wird zur eindeutigen Kennzeichnung des Knotens verwendet, wenn ein Script auf diesen verweist. Weitere Informationen finden Sie im Thema „Verwenden von CLEF-Knoten in Scripts” auf Seite 76.
`helpLink`	Eine optionale ID für ein Hilfethema, das angezeigt wird, wenn der Benutzer das Hilfesystem aufruft, sofern vorhanden. Das Format der ID hängt vom Hilfesystemtyp ab (siehe Kapitel 7, „Hinzufügen eines Hilfesystems”, auf Seite 165) : HTML Help - URL des Hilfethemas JavaHelp - ID des Themas
`description`	Eine Textbeschreibung des Knotens.
`descriptionKey`	Gibt die Beschreibung für Lokalisierungszwecke an.
`delegate`	Wird dieses Element angegeben, definiert es den Namen einer Java-Klasse, die die NodeDelegate-Schnittstelle implementiert. Eine Instanz der angegebenen Klasse wird für jede Instanz des zugeordneten Knotens erstellt.

Die in der Knotendefinition zulässigen Elemente werden in den Abschnitten ab „Modellerstellung” auf Seite 51 beschrieben.

Beispiel

Ein Beispiel für eine Knotendefinition finden Sie in „Beispiel für eine Spezifikationsdatei” auf Seite 32.

Modellausgabe

Eine Modellausgabedefinition beschreibt ein generiertes Modell, d. h. ein Objekt, das nach der Ausführung eines Streams im Managerfenster auf der Registerkarte 'Modelle' angezeigt wird.

Die vollständigen Details zum Codieren dieses Teils der Datei finden Sie in „Modellausgabe” auf Seite 88.

Dokumentausgabe

Eine Dokumentausgabedefinition beschreibt ein Objekt, wie eine generierte Tabelle oder ein generiertes Diagramm, das nach der Ausführung eines Streams im Managerfenster auf der Registerkarte 'Ausgaben' angezeigt wird.

Die vollständigen Details zum Codieren dieses Teils der Datei finden Sie in „Dokumentausgabe” auf Seite 99.

Interaktive Modellerstellung

Die vollständigen Details zum Codieren dieses Teils der Datei finden Sie in „Erstellen von interaktiven Modellen” auf Seite 89.

Modellerstellung

Dieses Element wird nur in Knotenelementdefinitionen verwendet.

Die vollständigen Details zum Codieren dieses Teils der Datei finden Sie in Kapitel 5, „Erstellen von Modellen und Dokumenten”, auf Seite 79.

Dokumenterstellung

Dieses Element wird nur in Knotenelementdefinitionen verwendet.

Die vollständigen Details zum Codieren dieses Teils der Datei finden Sie in Kapitel 5, „Erstellen von Modellen und Dokumenten”, auf Seite 79.

Modellprovider

Dieses Element wird nur in Knotenelementdefinitionen verwendet.

Wenn Sie ein Modellausgabeobjekt und einen Modellanwendungsknoten definieren, können Sie mit dem Element `ModelProvider` den Container angeben, in dem das Element aufgenommen wird. Sie können außerdem festlegen, ob das Modell im PMML-Format gespeichert wird. PMML-Modelle können entweder über einen benutzerdefinierten Viewer oder mit dem IBM SPSS Modeler-Standardmodellausgabeviewer angezeigt werden, der über das Element `ModelViewerPanel` bereitgestellt wird. Weitere Informationen finden Sie im Thema „Modellviewerfenster” auf Seite 122.

Format

```
<ModelProvider container="Containername" isPMML="true_false" />
```

Dabei gilt Folgendes:

`container` ist der Name des Containers, der das Modell enthält.

`isPMML` gibt an, ob das Modell im PMML-Format gespeichert wird.

Beispiel

```
<ModelProvider container="Modell" isPMML="true" />
```

Ein Beispiel für die Verwendung von `ModelProvider` im Kontext eines Modellanwendungsknotens finden Sie in „Modellviewerfenster” auf Seite 122.

Eigenschaften

Eine Eigenschaftsdefinition besteht aus einem Satz von Namens- und Wertepaaren. Einzelne Eigenschaftsdefinitionen, von denen es eine Vielzahl geben kann, werden in einem einzigen Abschnitt 'Properties' aufgeführt.

Hinweis: Wenn eine Eigenschaft im Abschnitt 'Properties' definiert ist, ist es nicht erforderlich, sie für ein einzelnes Eigenschaftssteuerelement zu definieren, da die Definitionen des Abschnitts 'Properties' Vorrang haben. Aus diesem Grund empfiehlt es sich, Eigenschaften im Abschnitt 'Properties' zu definieren.

Die einzige Ausnahme von dieser Regel betrifft das Attribut `label`. Wenn das Attribut `label` für ein Eigenschaftssteuerelement definiert wird, hat *jede* innerhalb der Deklaration vorhandene Eigenschaftsdefinition (nicht nur die Definition für `label`) Vorrang vor der entsprechenden Definition im Abschnitt 'Properties'. Beachten Sie, dass diese Ausnahme nur für Eigenschaftssteuerelemente gilt und nicht für andere Steuerelementtypen wie Menüs, Menüelemente und Symbolleistenelemente. Für diese muss explizit eine Beschriftung definiert sein, entweder direkt (Menüs) oder indirekt über ein `Action`-Element (Menüelemente und Symbolleistenelemente).

Format

```
<Properties>
     <Property name="Name" scriptName="Scriptname" valueType="Wertetyp" isList="true_false"
          defaultValue="Standardwert" label="Anzeigebeschriftung" labelKey="Beschriftungsschlüssel"
          description="Beschreibung" descriptionKey="Beschreibungsschlüssel" />
          <Enumeration ... />
          <Structure ... />
          <DefaultValue ... />
     ...
</Properties>
```

Die Elemente `Enumeration`, `Structure` und `DefaultValue`werden in spezifischen Fällen verwendet. Weitere Informationen finden Sie im Thema „Werttypen" auf Seite 60.

Die folgende Tabelle enthält die Attribute des Elements `Property`.

Tabelle 14. Attribute des Elements 'Property'.

Attribut	Beschreibung
`name`	(erforderlich) Ein eindeutiger Name für die Eigenschaft.
`scriptName`	Der Name, über den die Eigenschaft in einem Script referenziert wird. Weitere Informationen finden Sie im Thema „Verwenden von CLEF-Knoten in Scripts" auf Seite 76.

`valueType`	Legt den Typ des Werts fest, den die Eigenschaft annehmen kann:
	
	`string`
	`encryptedString`
	`fieldName`
	`integer`
	`double`
	`boolean`
	`date`
	`enum`
	`structure`
	`databaseConnection`
	
	Weitere Informationen finden Sie im Thema „Werttypen" auf Seite 60.

`isList`	Gibt an, ob es sich bei der Eigenschaft um eine Liste von Werten des angegebenen Werttyps (`true`) handelt oder um einen einzelnen Wert (`false`).

Tabelle 14. Attribute des Elements 'Property' (Forts.).

Attribut	Beschreibung
`defaultValue`	Der Standardwert für die Eigenschaft. Dieser kann als einfaches Wertattribut ausgedrückt werden oder als zusammengesetztes Element und muss mit den angegebenen gültigen Werten übereinstimmen.
`label`	Der Anzeigename für den Eigenschaftswert, der auf der Benutzerschnittstelle angezeigt wird.
`labelKey`	Gibt die Beschriftung für Lokalisierungszwecke an.
`description`	Eine Beschreibung der Eigenschaft.
`descriptionKey`	Gibt die Beschreibung für Lokalisierungszwecke an.

Eigenschaften können optional deklarieren, wie gültige Bereiche ermittelt werden:

- Für numerische Werte sind dies der Minimal- und/oder Maximalwert.
- Für Zeichenfolgen ist dies in der Regel eine Feldauswahl (z. B. alle Felder, alle numerischen Felder, alle diskreten Felder usw.), es kann aber auch eine Dateiauswahl sein.
- Für Aufzählungen ist dies der Satz der gültigen Werte.

Für verschlüsselte Eigenschaften muss außerdem deklariert werden, wie gültige Schlüssel ermittelt werden. Beachten Sie, dass der Schlüsseltyp einer verschlüsselten Eigenschaft entweder eine Zeichenfolge oder eine Aufzählung sein muss. Weitere Informationen finden Sie im Thema „Eigenschaftstypen" auf Seite 37.

Der der Eigenschaft zugeordnete optionale Standardwert wird evaluiert, wenn das zugeordnete Objekt erstellt wird. Beispiel: Standardwerte für Knoteneigenschaften werden jedes Mal evaluiert, wenn eine neue Instanz des Knotens erstellt wird; Ausführungseigenschaften werden jedes Mal evaluiert, wenn der Knoten ausgeführt wird. Die Evaluierung erfolgt in der Reihenfolge, in der die Eigenschaften deklariert sind.

Beachten Sie, dass eine Eigenschaftsdefinition einen Eigenschaftstyp referenzieren kann, der im Abschnitt 'CommonObjects' deklariert ist.

Container

Ein Container ist ein Platzhalter für ein Ausgabeobjekt, dessen Generierung im Abschnitt 'Constructors' definiert ist.

Format

```
<Containers>
     <Container name="Containername" />
      ...
</Containers>
```

Dabei gilt Folgendes:

`name` entspricht dem Wert des Zielattributs eines `CreateModel`- oder `CreateDocument`-Elements (siehe „Verwenden von Konstruktoren" auf Seite 100) und ordnet den Container indirekt einem der im Abschnitt 'CommonObjects' deklarierten Containertypen zu.

Beispiel

Zuerst werden die Containertypen im Abschnitt 'CommonObjects' deklariert. Es gibt einen Containertyp für Modelle, im Textformat, und zwei Containertypen für Dokumentausgabeobjekte, einer im Standardtextformat für HTML-Ausgaben und einer im Binärformat für komprimierte ZIP-Ausgaben.

```
<CommonObjects>
     <ContainerTypes>
          <ModelType id="my_model" format="utf8" />
          <DocumentType id="html_output" />
          <DocumentType id="zip_outputType" format="binary" />
     </ContainerTypes>
</CommonObjects>
```

Im Abschnitt 'Execution' der Knotendefinition werden die Ausgabedateien als Containerdateien definiert, mit Containertypen, die den im Abschnitt 'CommonObjects' angegebenen IDs entsprechen:

```
<Node id="mynode" ... >
 ...
     <Execution>
      ...
          <OutputFiles>
               <ContainerFile id="pmml" path="${tempfile}.pmml" containerType="my_model" />
               <ContainerFile id="htmloutput" path="${tempfile}.html" containerType="html_
               output" />
               <ContainerFile id="zipoutput" path="${tempfile}.zip" containerType="zip_
               outputType" />
          </OutputFiles>
```

Anschließend werden im Abschnitt 'Constructors' die Ausgabeobjekte definiert, die bei der Ausführung des Knotens generiert werden. Hier haben die Elemente `CreateModel` und `CreateDocument` ein `sourceFile`-Attribut, das einer Containerdatei entspricht, die zuvor im Abschnitt 'OutputFiles' festgelegt wurde:

```
          <Constructors>
               <CreateModelOutput type="myoutput">
                    <CreateModel target="model" sourceFile="pmml" />
                    <CreateDocument target="advanced_output" sourceFile="htmloutput" />
                    <CreateDocument target="zip_output" sourceFile="zipoutput" />
               </CreateModelOutput>
          </Constructors>
     </Execution>
</Node>
```

Zuletzt erfolgt im Abschnitt 'ModelOutput' die Zuordnung eines Containers zu einem Modellausgabe- oder einem Dokumentausgabeobjekt. Im `Container`-Element entspricht das Attribut `name` dem Attribut `target` in den gerade angegebenen Elementen `CreateModel` und `CreateDocument`:

```
<ModelOutput id="myoutput" label="My Model">
     <Containers>
          <Container name="model" />
          <Container name="advanced_output" />
          <Container name="zip_output" />
     </Containers>
      ...
</ModelOutput>
```

Benutzerschnittstelle

Die Spezifikationsdatei unterstützt eine Palette von Benutzerschnittstellenkomponenten, die die Anzeige von Objekten ermöglichen, sowie Steuerelemente und Eigenschaften, die geändert werden sollen. Es gibt Funktionen für die Angabe des Layouts und die Anpassung des Verhaltens der Komponente sowie zur Aktivierung oder Anzeige der Komponente, wenn andere Steuerelemente geändert werden.

Der Abschnitt 'UserInterface' gibt die sichtbare Erscheinung eines Objekts an. Die Spezifikation kann verwendet werden, um eine grundlegende Komponente der Benutzerschnittstelle anzupassen, wie das Eigenschaftsdialogfeld eines Knotens oder ein Ausgabefenster.

Der Abschnitt 'UserInterface' ist als Teil der Spezifikation des `Node`-Elements erforderlich.

Die vollständigen Details zum Codieren dieses Teils der Datei finden Sie in Kapitel 6, „Erstellen von Benutzerschnittstellen", auf Seite 105.

Ausführung

Dieses Element wird nur in Knotenelementdefinitionen verwendet.

Der Abschnitt 'Execution' definiert Eigenschaften und Dateien, die verwendet werden, wenn ein Knoten ausgeführt wird.

Format

```
<Execution>
     <Properties>
      ...
     </Properties>
     <InputFiles>
          <ContainerFile ... />
           ...
     </InputFiles>
     <OutputFiles>
          <ContainerFile ... />
           ...
     </OutputFiles>
     <Module ... >
          <StatusCodes ... />
     </Module>
     <Constructors ... />
</Execution>
```

Der Abschnitt 'Execution' enthält die Definition eines Satzes von Eigenschaften, die jedes Mal neu erstellt werden, wenn der Knoten ausgeführt wird, und nur während der Ausführung des Knotens verfügbar sind.

Die Ausführungsinformationen können auch den Satz der Eingabedateien definieren, die vor der Ausführung des Knotens generiert werden, sowie alle während der Ausführung generierten Ausgabedateien.

Es kann eine beliebige Anzahl von Ein- und Ausgabedateien angegeben werden. Jede Eingabedatei wird einem durch den Knoten definierten Container zugeordnet. Jede Ausgabedatei wird in der Regel verwendet, um Container für generierte Objekte zu erstellen. Das Format einer Eingabe- oder Ausgabedatei wird durch die Deklaration des Containers im Abschnitt 'CommonObjects' festgelegt.

Beispiel

Ein Beispiel für den Abschnitt 'Execution' finden Sie in „Beispiel für eine Spezifikationsdatei" auf Seite 32.

Laufzeiteigenschaften

Dieser Abschnitt definiert die Laufzeiteigenschaften, die nur dann verfügbar sind, wenn der Knoten ausgeführt wird.

Format

Das Format ist ähnlich wie das des Abschnitts `Properties` im Hauptteil der Elementdefinition. Weitere Informationen finden Sie im Thema „Eigenschaften" auf Seite 52.

Während der Ausführung eines Modellierungs- oder Dokumenterstellungsknotens wird eine **temporäre Serverdatei** erstellt, in der das Modellausgabe- oder Dokumentausgabeobjekt gespeichert wird. Der Server greift auf diese Datei zu und bringt das Objekt auf den Client, wo es in einem Container eingeschlossen wird. Sie müssen diese Datei hier angeben.

Beispiel

Dieses Beispiel zeigt, wie die temporäre Serverdatei angegeben wird.

```
<Properties>
     <Property name="tempfile" valueType="string">
          <DefaultValue>
               <ServerTempFile basename="datatmp"/>
          </DefaultValue>
     </Property>
</Properties>
```

Eingabedateien

Dieser Abschnitt definiert den Satz der Eingabedateien, der vor der Ausführung des Knotens erzeugt wird. Eingabedateien sind in diesem Kontext Dateien, die als Eingabe für die Ausführung eines Knotens auf dem Server dienen. Beispiel: Ein Modellanwendungsknoten hat einen Modellcontainer, der an die angegebene Eingabedatei für die Ausführung des Knotens übertragen wird.

Format

```
<InputFiles>
     <ContainerFile id="ID" path="Pfad" container="Container">
      ...
</InputFiles>
```

Im Element `ContainerFile` für eine Eingabedatei gibt es die in der folgenden Tabelle aufgeführten Attribute.

Tabelle 15. ContainerFile-Attribute - Eingabedateien.

Attribut	Beschreibung
`id`	Eine eindeutige ID für die Containerdatei.
`path`	Der Speicherort auf dem Server, an dem die Eingabedatei generiert werden soll (z. B. der Speicherort einer temporären Serverdatei. Siehe „Laufzeiteigenschaften" auf Seite 55).
`container`	Die ID des Containers, der das Objekt enthält, das als Eingabe an den Server gesendet werden soll.

Beispiel

```
<InputFiles>
     <ContainerFile id="pmml" path="${tempfile}.pmml" container="model"/>
</InputFiles>
```

Ausgabedateien

In diesem Abschnitt werden die Ausgabedateien festgelegt, die während des Ausführens des Knotens auf dem Server erstellt werden. Ausgabedateien (z. B. die Ergebnisse der Ausführung des Modellierungs- oder Dokumenterstellungsknotens) werden nach der Ausführung zurück auf den Client übertragen.

Format

```
<OutputFiles>
     <ContainerFile id="ID" path="Pfad" containerType="Container">
      ...
</OutputFiles>
```

Im Element `ContainerFile` gibt es die in der folgenden Tabelle aufgeführten Attribute.

Tabelle 16. ContainerFile-Attribute - Ausgabedateien.

Attribut	Beschreibung
`id`	Eine eindeutige ID für die Containerdatei.
`path`	Der Speicherort des Objekts, das auf den Client übertragen werden soll, auf dem Server (z. B. der Speicherort einer temporären Serverdatei. Siehe „Laufzeiteigenschaften" auf Seite 55).
`containerType`	Die ID des Containertyps für das Objekt (d. h. die ID des Modelltyps oder Dokumenttyps), über die die Übertragung des Objekts im richtigen Format aktiviert wird. Weitere Informationen finden Sie im Thema „Containertypen" auf Seite 39.

Beispiel

```
<OutputFiles>
     <ContainerFile id="pmml" path="${tempfile}.pmml" containerType="mynode_model" />
     <ContainerFile id="htmloutput" path="${tempfile}.html" containerType="html_output" />
     <ContainerFile id="zipoutput" path="${tempfile}.zip" containerType="zip_outputType" />
</OutputFiles>
```

Module

In diesem Abschnitt wird festgelegt, dass während der Ausführung eine serverseitige freigegebene Bibliothek verwendet wird (z. B. eine DLL, die in den Hauptspeicher geladen wird).

Format

```
<Module libraryId="ID_gemeinsam_genutzte_Bibliothek" name="Knotenname">
     <StatusCodes ... />
</Module>
```

Dabei gilt Folgendes:

`libraryId` ist die ID einer Bibliothek, die im SharedLibrary-Element im Abschnitt 'Resources' deklariert wird. Weitere Informationen finden Sie im Thema „Freigegebene Bibliotheken" auf Seite 36.

`name` wird verwendet, wenn die Bibliothek durch mehrere Knoten verwendet wird. Hierüber wird der auszuführende Knoten identifiziert. Wenn die Bibliothek nur von einem Knoten verwendet wird, muss der Name nicht angegeben werden.

Beispiel

```
<Module libraryId="mynode1" name="mynode">
     <StatusCodes>
          <StatusCode code="0" status="error" message="eine Ausnahme ist aufgetreten" />
          <StatusCode code="1" status="error" message="Fehler beim Lesen der Eingabedaten" />
           ...
     </StatusCodes>
</Module>
```

Statuscodes

Die meisten Programme führen eine Art Fehlerprüfung durch und zeigen dem Benutzer alle erforderlichen Nachrichten an. In der Regel werden dabei Ganzzahlen zurückgegeben, die den erfolgreichen Abschluss oder einen anderen Status anzeigen. Die serverseitige API kann nach der Ausführung eines Streams, der den Knoten enthält, einen Statuscode zurück geben. Weitere Informationen finden Sie im Thema „Statusdetaildokument" auf Seite 196.

Im Abschnitt 'StatusCodes' können Sie einem bestimmten Statuscode eine Nachricht zuordnen, die für Benutzer angezeigt wird.

Format

```
<StatusCodes>
     <StatusCode code="Codenummer" status="Status" message="Nachrichtentext"
       messageKey="Nachrichtenschlüssel" />
      ...
</StatusCodes>
```

Die folgende Tabelle enthält die Statuscode-Attribute.

Tabelle 17. Statuscode-Attribute.

Attribut	Beschreibung
code	Der Statuscode (ein ganzzahliger Wert), dem die Nachricht zugeordnet wird.
status	Die Statusklassifizierung: success - Knoten wurde erfolgreich ausgeführt warning - Knoten wurde mit Warnungen ausgeführt error - Knoten wurde nicht erfolgreich ausgeführt
message	Die Nachricht, die angezeigt werden soll, wenn der entsprechende Statuscode zurückgegeben wird.
messageKey	Gibt die Nachricht für Lokalisierungszwecke an.

Beispiel

In diesem Beispiel ist der Text der Fehlernachricht im Element StatusCode enthalten:

```
<StatusCodes>
     <StatusCode code="0" status="error" message="Peer kann nicht initialisiert werden" />
     <StatusCode code="1" status="error" message="Fehler beim Lesen der Eingabedaten" />
     <StatusCode code="2" status="error" message="Interner Fehler" />
     <StatusCode code="3" status="error" message="Eingabefeld nicht vorhanden" />
</StatusCodes>
```

Bei der Ausführung wird dem Benutzer die folgende Nachricht angezeigt, wenn die serverseitige API den Statuscode '3' zurückgibt.

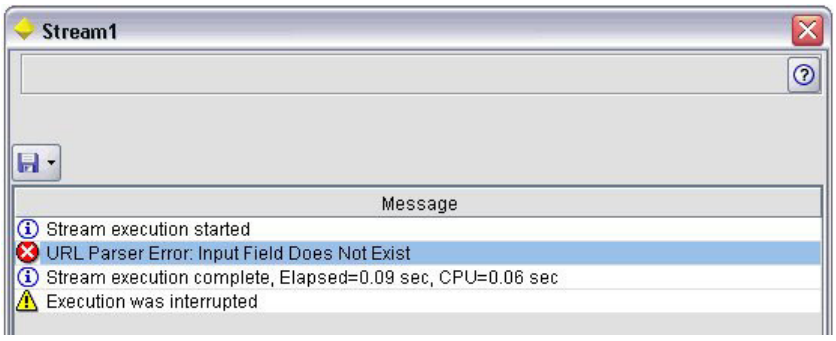

Abbildung 29. Fehlernachrichtenanzeige

Im nächsten Beispiel werden die Fehlernachrichtentexte durch ein messageKey-Attribut referenziert:

```
<StatusCodes>
     <StatusCode code="0" status="error" messageKey="initErrMsg.LABEL"/>
     <StatusCode code="1" status="error" messageKey="inputErrMsg.LABEL"/>
```

```
    <StatusCode code="2" status="error" messageKey="internalErrMsg.LABEL"/>
    <StatusCode code="3" status="error" messageKey="invalidMetadataErrMsg.LABEL"/>
     ...
</StatusCodes>
```

Eine Eigenschaftsdatei (z. B. `messages.properties`), die sich in demselben Ordner befindet wie die Spezifikationsdatei, enthält den Nachrichtentext zusammen mit anderen Anzeigetexten::

```
 ...
initErrMsg.LABEL=Initialisierung fehlgeschlagen.
inputErrMsg.LABEL=Fehler beim Lesen der Eingabedaten.
internalErrMsg.LABEL=Interner Fehler.
invalidMetadataErrMsg.LABEL=Metadaten (in Eingabe-/Ausgabefeldern) nicht gültig.
 ...
```

Dieses Verfahren ist nützlich, wenn der Anzeigetext z. B. für ausländische Märkte lokalisiert werden muss, da sich der gesamte zu übersetzende Text in einer einzigen Datei befindet. Weitere Informationen finden Sie im Thema „Lokalisierung” auf Seite 169.

Ausgabedatenmodell

Dieses Element wird nur in Knotenelementdefinitionen verwendet.

Im Abschnitt 'OutputDataModell' wird festgelegt, wie sich verschiedene Eigenschaften auf das Datenmodell auswirken.

Das Ausgabedatenmodell kann auf drei verschiedenen Weisen bestimmt werden:

- Mithilfe der Definitionsfunktionen für Feldsets in der Spezifikationsdatei. Weitere Informationen finden Sie im Thema „Feldsets” auf Seite 70.
- Mithilfe einer clientseitigen Java-Klasse, die eine Datenmodellproviderschnittstelle implementiert, die einen Satz Eigenschaften und das Eingabedatenmodell empfängt und eine Datenmodellinstanz zurück gibt.
- Mithilfe einer serverseitigen freigegebenen Bibliothekskomponente, die einen Satz Eigenschaften und ein Eingabedatenmodell empfängt und ein Metadatenmodell zurück gibt.

Im Abschnitt 'OutputDataModell' wird definiert, wie sich die Eigenschaften eines Knotens auf die durch den Knoten fließenden Felder auswirken. Das Ausgabedatenmodell bietet folgende Optionen:

- Das Eingabedatenmodell unverändert lassen
- Das Eingabedatenmodell ändern
- Das Eingabedatenmodell durch ein anderes Eingabedatenmodell ersetzen

Beispiel: Ein Sortierknoten wirkt sich nicht auf die Eigenschaften als solche aus, er sortiert sie lediglich. Ein Ableitungsknoten ändert das Datenmodell, indem er ein neues Feld hinzufügt. Ein Aggregatknoten ersetzt das Datenmodell vollständig.

In den Fällen, in denen das Eingabedatenmodell geändert wird, können über die Definition neue Felder hinzugefügt oder vorhandene Felder geändert oder entfernt werden. Wenn das Datenmodell ersetzt wird, können nur neue Felder hinzugefügt werden. Die Spezifikationsdatei unterstützt diese Basisoperationen (auch die Möglichkeit, neue Felder zu erstellen, deren Typen auf Eingabefeldern basieren) sowie die Möglichkeit, das Eingabefeldset oder einen Schlüssel oder eine Listeneigenschaft zu durchlaufen, die eine Feldgruppe im Eingabefeldset darstellen.

Format

Nachfolgend ist das allgemeine Format des Abschnitts für das Ausgabedatenmodell ('OutputDataModel') dargestellt, spezifische Formate für diese Fälle finden Sie aber in den Abschnitten „Steuerelement für die Auswahl mehrerer Felder" auf Seite 137 und „Steuerelement für die Auswahl eines einzelnen Felds" auf Seite 144.

```
<OutputDataModel mode="Modus" libraryId="Containername">
     -- Datenmodelloperationen --
</OutputDataModel>
```

Die OutputDataModel-Attribute sind in der folgenden Tabelle dargestellt.

Tabelle 18. OutputDataModel-Attribute.

Attribut	Beschreibung
`mode`	Auswirkung auf das Datenmodell: `extend` - fügt ein neues Feld zum vorhandenen Modell hinzu `fixed` - unverändert `modify` - ändert vorhandene Felder (z. B. entfernen oder umbenennen) `replace` - ersetzt das vorhandene Modell
`libraryId`	Der Name eines serverseitigen Containers, aus dem das Datenmodell erhalten wird.

Datenmodelloperationen sind die Operationen, die neue Felder hinzufügen oder vorhandene Felder ändern oder entfernen. Weitere Informationen finden Sie im Thema „Datenmodelloperationen" auf Seite 64.

Beispiel

Ein `OutputDataModel`-Element ist im Beispiel für eine Spezifikationsdatei enthalten. Weitere Informationen finden Sie im Thema „Beispiel für eine Spezifikationsdatei" auf Seite 32.

Konstruktoren

Konstruktoren definieren die Objekte, die als Ergebnis der Ausführung eines Knotens in einem Stream oder beim Generieren eines Objekts zurück in den Stream produziert werden.

Die vollständigen Details zum Codieren dieses Teils der Datei finden Sie in den Abschnitten ab „Verwenden von Konstruktoren" auf Seite 100.

Allgemeine Funktionen

Einige Funktionen können in mehreren Abschnitten der Spezifikationsdatei verwendet werden:

- Wertetypen
- evaluierte Zeichenfolgen
- Operationen
- Felder und Feldmetadaten
- Feldsets
- Rollen
- logische Operatoren
- Bedingungen

Werttypen

Werttypdeklarationen geben den Typ des Werts an, den eine Spalten-, Eigenschafts- oder Eigenschaftstypspezifikation annehmen kann.

Zeichenfolgen und verschlüsselte Zeichenfolgen

Das Format `valueType="string"` gibt an, dass der Wert eine Textzeichenfolge ist. Die Deklaration `valueType="encryptedString"` wird für eine Eigenschaft verwendet, die sich auf ein Feld bezieht, dessen Inhalt bei der Eingabe durch den Benutzer ausgeblendet werden, wie z. B. ein Kennwortfeld.

Feldnamen

Wenn ein Wert die Form eines Feldnamens annimmt, verwenden Sie das Format `valueType="fieldName"`.

Mathematische, logische und Datumsausdrücke

Wenn der Wert ein mathematischer Ausdruck (Ganzzahl oder doppelte Genauigkeit), ein logischer Ausdruck (wahr/falsch) oder ein Datumsausdruck ist, geben Sie für `valueType` entsprechend `integer`, `double`, `boolean` oder `date` an.

Aufgezählte Eigenschaften

Aufgezählte Eigenschaften sind im Abschnitt 'Enumeration' enthalten, der direkt auf eine `valueType="enum"`-Deklaration folgt. Weitere Informationen finden Sie im Thema „Aufgezählte Eigenschaften".

Strukturdeklarationen

Eine `valueType="structure"`-Deklaration gibt einen zusammengesetzten Wert an, der andere benannte Attribute enthält. Attribute sind ähnlich wie Eigenschaften, können jedoch nicht strukturiert oder verschlüsselt werden. Weitere Informationen finden Sie im Thema „Strukturierte Eigenschaften" auf Seite 62.

- **Verschlüsselter Indikator.** Gibt an, ob die Eigenschaft ein einzelner Wert oder eine Hashtabelle ist, in der alle Werte den angegebenen Werttyp haben.
- **Wertegruppe.** Legt fest, wie der Satz der verfügbaren Werte ermittelt wird.
- **Schlüsselsatz.** Für verschlüsselte Eigenschaften. Wird verwendet, um festzulegen wie der Satz der verfügbaren Schlüssel bestimmt wird. Diese Information wird außerdem verwendet, um an der Benutzerschnittstelle Hinweise zum Controller-Typ zu geben, der verwendet werden sollte.

Datenbankverbindungen

Dies sind Verbindungszeichenfolgen, mit denen sich Benutzer bei einer Datenbank anmelden können, wie z. B. `user1@testdb`. Die Anmeldedetails müssen für die Datenbank bereits definiert sein. Weitere Informationen finden Sie im Thema „Steuerelement für die Auswahl der Datenbankverbindung" auf Seite 135.

Aufgezählte Eigenschaften

Eine aufgezählte Eigenschaft kann einen Wert aus einer vordefinierten Werteliste annehmen.

Format

Das Format für aufgezählte Eigenschaften verwendet einen Abschnitt `Enumeration`, in dem die Liste der Werte definiert ist:

```
<PropertyTypes>
     <PropertyType id="ID" valueType="enum">
          <Enumeration>
               <Enum value="Wert" label="Anzeigebeschriftung" labelKey="Beschriftungsschlüssel"
                    description="Beschreibung" descriptionKey="Beschreibungsschlüssel" />
           ...
          </Enumeration>
     </PropertyType>
<PropertyTypes>
```

Dabei sehen die `PropertyType`-Attribute wie folgt aus:

- id ist ein eindeutiges Kennzeichen für den Eigenschaftstyp.
- valueType gibt an, dass der Eigenschaftstyp aufgezählt ist.

Die Enum-Attribute sehen wie folgt aus:

- value (erforderlich) ist der Eigenschaftswert, der in der Werteliste angezeigt werden soll.
- label (erforderlich) ist der Anzeigename für die Eigenschaft, mit dem sie an der Benutzerschnittstelle angezeigt wird.
- labelKey gibt die Beschriftung für Lokalisierungszwecke an.
- description ist eine Beschreibung des aufgezählten Werts.
- descriptionKey gibt die Beschreibung für Lokalisierungszwecke an.

Beispiel

```
<PropertyTypes>
     <PropertyType id="shared_enum1" valueType="enum">
          <Enumeration>
               <Enum value="value1" label="Value 5.1" labelKey="enum5.value1.LABEL" />
               <Enum value="value2" label="Value 5.2" labelKey="enum5.value2.LABEL" />
               <Enum value="value3" label="Value 5.3" labelKey="enum5.value3.LABEL" />
          </Enumeration>
     </PropertyType>
<PropertyTypes>
```

Strukturierte Eigenschaften

Eine strukturierte Eigenschaft wird in einer rasterartigen Struktur verwendet, wie z. B. ein Steuerelement für Tabellen in einem Dialogfeld.

Format

Das Format für strukturierte Eigenschaften verwendet einen Abschnitt Structure, in dem die Struktur definiert wird, und besteht aus einer Reihe von Attribute-Elementen, die wie folgt aussehen:

```
<PropertyTypes>
     <PropertyType id="ID" valueType="structure" isList="true_false">
          <Structure>
               <Attribute name="Spalten-ID" valueType="Werttyp" isList="true_false"
                    label="Spaltenbeschriftung" labelKey="Beschriftungsschlüssel" defaultValue="Wert"
                    description="Beschreibung" descriptionKey="Beschreibungsschlüssel" />
           ...
          </Structure>
     </PropertyType>
</PropertyTypes>
```

Dabei sehen die Attribute des PropertyType-Elements wie folgt aus:

- id ist ein eindeutiges Kennzeichen für den Eigenschaftstyp.
- valueType gibt an, dass der Eigenschaftstyp strukturiert ist.
- isList gibt an, ob es sich bei der Eigenschaft um eine Liste von Werten des angegebenen Werttyps (true) handelt oder um einen einzelnen Wert (false).

Die Attribute des Attribute-Elements sehen wie folgt aus:

- name (erforderlich) ist die ID der Spalte.
- valueType gibt den Werttyp an, den die Inhalte der Spalte annehmen können:

  string

  encryptedString

  integer

`double`

`boolean`

`date`

`enum`

- `isList` gibt an, ob es sich bei dem Attribut um eine Liste von Werten des angegebenen Werttyps (`true`) handelt oder um einen einzelnen Wert (`false`). Auf diese Weise kann eine verschlüsselte Eigenschaft einem festen Satz an bekannten Attributen (z. B. Boolean-Attribute für verschiedene Aggregationsvorgänge, die an einem bestimmten Feld ausgeführt werden sollen) oder einer Werteliste (z. B. Verbinden einer Feldnamenliste mit anderen Feldnamen) zugeordnet werden.
- `label` (erforderlich) ist der Anzeigename für die Spalte, mit dem sie auf der Benutzerschnittstelle angezeigt wird.
- `labelKey` gibt die Beschriftung für Lokalisierungszwecke an.
- `defaultValue` ist ein Wert, der in der Spalte erschient, wenn sie angezeigt wird.
- `description` ist eine Beschreibung der Spalte.
- `descriptionKey` gibt die Beschreibung für Lokalisierungszwecke an.

Beispiel - Steuerelement für Tabellen

Ein Beispiel zur Verwendung strukturierter Eigenschaften in einem Steuerelement für Tabellen finden Sie in „Tabellensteuerelement” auf Seite 147.

Beispiele - Verschlüsselte Eigenschaftstypen

Die ersten dieser Beispiele illustrieren die Verwendung eines verschlüsselten Eigenschaftstyps, bei dem jeder zugeordnete Wert eine Struktur darstellt, die angibt, welcher Aggregationsvorgang aus einer vorgegebenen Menge an Vorgängen auf ein Feld angewendet wird:

```
<PropertyType id="aggregateOps" isKeyed="true" valueType="structure">
  <Structure>
    <Attribute name="MIN" valueType="boolean" label="Min" />
    <Attribute name="MAX" valueType="boolean" label="Max" defaultValue="true"/>
    <Attribute name="SUM" valueType="boolean" label="Sum" defaultValue="false"/>
    <Attribute name="MEAN" valueType="boolean" label="Mean" defaultValue="false"/>
    <Attribute name="SDEV" valueType="boolean" label="SDev" defaultValue="false"/>
  </Structure>
</PropertyType>
```

Daher könnte eine Eigenschaft, die für die Verwendung des Eigenschaftstyps `aggregateOps` deklariert wurde, Folgendes sein:

```
<Property name="aggregationSettings" scriptName="aggregation_settings" type="aggregateOps"/>
```

Hier besteht die Eigenschaft aus mehreren Werten, wobei jeder Wert über einen anderen Schlüssel verfügt. Der Schlüssel `name` beispielsweise ist der Name eines Felds (MIN, MAX usw.).

Im nächsten Beispiel eines verschlüsselten Eigenschaftstyps stellt jeder zugeordnete Wert eine Struktur dar, die ein einzelnes Attribut enthält. In diesem Fall ist das Attribut eine Liste mit Ausdrücken mit doppelter Genauigkeit, die sich als Multiplikatoren auf ein Feld anwenden lassen:

```
<PropertyType id="multiplierOps" isKeyed="true" valueType="structure">
  <Structure>
    <Attribute name="multipliers" valueType="double" isList="true"/>
  </Structure>
</PropertyType>
```

Eine Eigenschaft, die für die Verwendung des Eigenschaftstyps `multiplierOps` deklariert wurde, könnte Folgendes sein:

```
<Property name="multiplierSettings" scriptName="multiplier_settings" type="multiplierOps"/>
```

Standardwerte

Das Element `DefaultValue` wird verwendet, um ein temporäres Serververzeichnis, eine temporäre Serverdatei oder beides anzugeben. Diese werden erstellt, um ein Modellausgabe- oder ein Dokumentausgabeobjekt zu speichern.

Format

```
<DefaultValue>
     <ServerTempDir basename="Name"/>
     <ServerTempFile basename="Name"/>
</DefaultValue>
```

Dabei ist `basename` (erforderlich) der Name des temporären Verzeichnisses oder der temporären Datei.

Beispiel

```
<DefaultValue>
     <ServerTempFile basename="datatmp"/>
</DefaultValue>
```

Evaluierte Zeichenfolgen

Einige in der Spezifikationsdatei deklarierte Zeichenfolgen enthalten möglicherweise Referenzen auf Eigenschaftsnamen. Diese Zeichenfolgen werden als evaluierte Zeichenfolgen bezeichnet.

Die Syntax für eine Eigenschaftsreferenz lautet:

`"${Eigenschaftsname}"`

Wenn auf eine evaluierte Zeichenfolge zugegriffen wird, werden alle Eigenschaftsreferenzen durch den Wert der referenzierten Eigenschaft ersetzt. Wenn die Eigenschaft nicht vorhanden ist, tritt ein Fehler auf. Beispiel: Wenn ein neues Feld hinzugefügt wird, ist in der Knotendefinition möglicherweise ein Feld namens `my_new_field` vorhanden und im Abschnitt 'UserInterface' gibt es ein Steuerelement, das dem Benutzer die Möglichkeit gibt, den Wert dieser Eigenschaft zu bearbeiten.

Beispiel

```
<AddField name="${my_new_field}" ... >
```

Vorgänge

Bestimmte Abschnitte der Spezifikationsdatei unterstützen Operationen wie das Hinzufügen von Feldern, das Erstellen von Komponenten und das Initialisieren von Eigenschaften. Zu den Abschnitten, die Operationen unterstützen, gehören:

- Ausgabedatenmodell (Quellen- und Prozessknoten)
- Eingabe- und Ausgabedatenmodell (Komponenten)
- Ausgabeobjekterstellung (Modellierungs- und Dokumenterstellungsknoten)
- Modellanwendungserstellung (Modellausgaben)

Operationen werden in folgende Typen eingeteilt:

- Datenmodelloperationen: `AddField`, `ChangeField`, `RemoveField`
- Iteration: `ForEach`

Datenmodelloperationen

Folgende Operationen können Sie auf einem Datenmodell durchführen:

- Hinzufügen eines neuen Felds zu einem vorhandenen Datenmodell

- Ändern eines vorhandenen Felds in einem Datenmodell
- Entfernen eines Felds aus einem Datenmodell

Hinzufügen eines Felds: Das Element `AddField` ermöglicht das Hinzufügen eines neuen Felds zu einem vorhandenen Datenmodell.

Format

```
<AddField prefix="Präfix" name="Name" direction="Feldrolle" directionRef="Feldrollendefinition"
     fieldRef="Feldreferenz" group="Gruppen-ID" label="Beschriftung" missingValuesRef="FWert-Referenz"
     storage="Speichertyp" storageRef="Speicherreferenz" targetField="Zielfeld"
     type="Datentyp" typeRef="Typreferenz" role="Rolle" tag="Propensity-Typ" value="Wert" depth="ganze_Zahl"
     <Range min="Mindestwert" max="Maximalwert" />
</AddField>
```

Die Attribute für `AddField` sind im Folgenden aufgeführt.

Tabelle 19. AddField-Attribute.

Attribut	Beschreibung
`prefix`	Ein Präfix, das zum Feldnamen hinzugefügt wird, um z. B. ein Modellausgabefeld anzugeben.
`name`	(erforderlich) Der Name des Felds, das hinzugefügt wird. Dies kann entweder eine fest codierte Zeichenfolge (z. B. `Feld8`) oder eine berechnete Zeichenfolge (z. B. `${target}`) sein, die auf ein Feld verweist. Weitere Informationen finden Sie im Thema „Evaluierte Zeichenfolgen" auf Seite 64.
`direction`	Die Feldrolle, d. h., ob das Feld eine Eingabe oder ein Ziel ist. Entweder `in`, `out`, `both`, `partition` oder `none`.
`directionRef`	Gibt an, dass die Feldrichtung von dem Feld übernommen werden soll, das durch eine berechnete Zeichenfolge (z. B. `${field1}`) identifiziert wird und auf ein Feld verweist. Weitere Informationen finden Sie im Thema „Evaluierte Zeichenfolgen" auf Seite 64.
`fieldRef`	Gibt an, dass alle referenzierten Werte (`directionRef`, `missingValuesRef` , `storageRef` und `typeRef`) von den entsprechenden Werten des Felds abzuleiten sind, das durch eine berechnete Zeichenfolge (z. B. `${field1}`) identifiziert wird, der auf ein Feld verweist. Weitere Informationen finden Sie im Thema „Evaluierte Zeichenfolgen" auf Seite 64.
`group`	Gibt an, dass das Feld einer Feldgruppe angehört. Weitere Informationen finden Sie im Thema „Modellfelder" auf Seite 86.
`label`	Eine Beschriftung, die dem Feld hinzugefügt werden soll.
`missingValuesRef`	Gibt an, dass die Behandlung von fehlenden Werten gemäß der Spezifikation fehlender Werte des Felds übernommen werden soll, das durch eine berechnete Zeichenfolge (z. B. `${field1}`) identifiziert wird, die auf ein Feld verweist. Weitere Informationen finden Sie im Thema „Evaluierte Zeichenfolgen" auf Seite 64.
`storage`	Der Datenspeichertyp für den Feldwert. Entweder `integer`, `real`, `string`, `date`, `time`, `timestamp`, `list` oder `unknown`.

Tabelle 19. AddField-Attribute (Forts.).

Attribut	Beschreibung
storageRef	Gibt an, dass der Speichertyp von dem Feld übernommen werden soll, das durch eine berechnete Zeichenfolge (z. B. ${field1}) identifiziert wird, die auf ein Feld verweist. Weitere Informationen finden Sie im Thema „Evaluierte Zeichenfolgen" auf Seite 64.
targetField	Gibt für ein Modellausgabefeld das Zielfeld an, aus dem Daten für dieses neue Feld übernommen werden. Dies kann entweder eine fest codierte Zeichenfolge (z. B. Feld8) oder eine berechnete Zeichenfolge (z. B. ${target}) sein, die auf ein Feld verweist. Weitere Informationen finden Sie im Thema „Evaluierte Zeichenfolgen" auf Seite 64.
type	Der Datentyp des Felds. Entweder auto, range, discrete, set, orderedSet, flag, collection, geospatial oder typeless.
typeRef	Gibt an, dass der Datentyp von dem Feld übernommen werden soll, das durch eine berechnete Zeichenfolge (z. B. ${field1}) identifiziert wird, die auf ein Feld verweist. Weitere Informationen finden Sie im Thema „Evaluierte Zeichenfolgen" auf Seite 64.
role	Die Art der Daten, die sich im Modellausgabefeld befinden - entweder unknown, predictedValue, predictedDisplayValue, probability, residual, standardError, entityId, entityAffinity, upperConfidenceLimit, lowerConfidenceLimit, propensity, value oder supplementary. Weitere Informationen finden Sie im Thema „Rollen" auf Seite 71.
tag	Wird nur verwendet, wenn role den Wert propensity hat; gibt den Neigungstyp an und ist entweder RAW oder ADJUSTED.
value	Gibt an, dass die Werte, die das neue Feld enthalten soll, von dem Feld abgeleitet werden sollen, das von einer evaluierten Zeichenfolge (z. B. ${field1}) angegeben wird, die auf ein Feld verweist. Weitere Informationen finden Sie im Thema „Evaluierte Zeichenfolgen" auf Seite 64.
depth	Wird nur verwendet, wenn storage den Wert list hat. Der Wert definiert die Tiefe der Liste, die >= -1 betragen muss.
valueStorage	Wird nur verwendet, wenn storage den Wert list hat. Der Wert definiert den Speichertyp der Listenwerte. Entweder integer, real, string, date, time oder timestamp.

Die Attribute für Range sind im Folgenden aufgeführt.

Tabelle 20. Range-Attribute

Attribut	Beschreibung
min	Der kleinste Wert, den das Feld annehmen kann.
max	Der größte Wert, den das Feld annehmen kann.

Beispiele

Das folgende Beispiel fügt ein Zeichenfolgenfeld namens field8 hinzu:

```
<AddField name="field8" storage="string" />
```

Das nächste Beispiel zeigt, wie beim Hinzufügen eines Felds eine Referenz auf einen Eigenschaftsnamen verwendet werden kann. Das Feld wird hier mit einem Namen hinzugefügt, der mit dem Wert der zuvor definierten Eigenschaft prop1 übereinstimmt:

```
<AddField name="${prop1}" ... />
```

Wenn im folgenden Beispiel das Zielfeld den Namen field1 hat, erstellt das Modell ein Ausgabefeld mit dem Namen $S-field1, das den vorhergesagten Wert für field1 aufnimmt:

```
<AddField prefix="$S" name="${target}" role="predictedValue" targetField="${target}"/>
```

Das nächste Beispiel fügt ein Modellausgabefeld hinzu, das einen Wahrscheinlichkeitsscore zwischen 0 und 1 enthält:

```
<AddField prefix="$SC" name="${target}" storage="real" role="probability" targetField=
"${target}">
     <Range min="0,0" max="1,0"/>
</AddField>
```

Im letzten Beispiel wird für jedes Modellausgabefeld ein Ausgabefeld hinzugefügt, das einen Wahrscheinlichkeitsscore zwischen 0,0 und 1,0 enthält und dessen Wert aus der Variablen fieldValue übernommen wird:

```
<ForEach var="fieldValue" inFieldValues="${field}">
     <AddField prefix="$SP" name="${fieldValue}" storage="real" role="probability" targetField=
     "${field}" value="${fieldValue}">
          <Range min="0,0" max="1,0"/>
     </AddField>
</ForEach>
```

Weitere Informationen finden Sie im Thema „Evaluierte Zeichenfolgen" auf Seite 64.

Feld ändern: Mit dem Element ChangeField kann ein vorhandenes Feld in einem Datenmodell geändert werden.

Format

```
<ChangeField name="Name" fieldRef="Feldreferenz" direction="Feldrolle" storage="Speichertyp"
type="Datentyp">
     <Range min="Mindestwert" max="Maximalwert" />
</ChangeField>
```

Die Attribute für ChangeField sind im Folgenden aufgeführt.

Tabelle 21. ChangeField-Attribute.

Attribut	Beschreibung
name	(erforderlich) Der Name des Felds, das geändert wird.
fieldRef	Ein Referenzwert für das Feld.
direction	Die Feldrolle, d. h., ob das Feld eine Eingabe oder ein Ziel ist. Entweder in, out, both, partition oder none.
storage	Der Datenspeichertyp für den Feldwert. Entweder integer, real, string, date, time, timestamp oder unknown.
type	Der Datentyp des Felds. Entweder auto, range, discrete, set, orderedSet, flag oder typeless.

Die Attribute für Range sind im Folgenden aufgeführt.

Tabelle 22. Range-Attribute

Attribut	Beschreibung
min	Der kleinste Wert, den das Feld annehmen kann.
max	Der größte Wert, den das Feld annehmen kann.

RemoveField: Mit dem RemoveField-Element kann ein Feld aus einem Datenmodell entfernt werden.

Format

```
<RemoveField fieldRef="Feldreferenz" />
```

Dabei ist fieldRef ein Referenzwert für das Feld.

Iteration mit dem ForEach-Element

An manchen Stellen ist es nützlich, wenn man bestimmte Operationen wiederholt durchführen kann, um alle Elemente eines Wertesets zu verarbeiten. Die Spezifikationsdatei unterstützt einen einfachen ForEach-Iterator, der jeden Wert eines angegebenen Sets mit einer temporären Eigenschaft verbindet. Die ForEach-Schleife kann auf eine der folgenden Weisen eingerichtet werden:

- zwischen zwei ganzzahligen Werten mit einer optionalen Schrittweite
- über Werte in einer Listeneigenschaft
- über Schlüssel in einer verschlüsselten Eigenschaft
- über Felder in einer Feldgruppe

Format

```
<ForEach var="Feldname" from="erwartete_Ganzzahl" to="erwartete_Ganzzahl" step="erwartete_Ganzzahl"
inFields="Felder" inFieldValues="Feldname" inProperty="Eigenschaftsname" >
   -- Datenmodelloperation --
</ForEach>
```

Dabei gilt Folgendes:

var (erforderlich) gibt das Feld an, das die Werte enthält, auf die die Iteration angewendet wird.

from und to geben Ganzzahlen an (oder einen Ausdruck, dessen Auswertung Ganzzahlen liefert), die die untere und die obere Iterationsgrenze bestimmen, mit dem optionalen Attribut step, das eine ganzzahlige Schrittweite angibt.

inFields, inFieldValues und inProperty sind Alternativen zum Format from/to/step:

- inFields gibt ein Feldset an, für das die Iteration durchgeführt wird, und hat einen der folgenden Werte:

  inputs - die Eingabefelder für den Knoten

  outputs - die Ausgabefelder aus dem Knoten

  modelInput - die in der Modellsignatur angegebenen Eingabefelder

  modelOutput - die in der Modellsignatur angegebenen Ausgabefelder
- inFieldValues gibt einen Feldnamen (oder eine Eigenschaft, die für einen Feldnamen steht) an und iteriert durch die Werte in den Metadaten für das Feld
- inProperty gibt den Namen einer Eigenschaft an, für die die Iteration durchgeführt wird

Die Datenmodeloperation, die in einem ForEach-Element angegeben werden kann, ist eines der Elemente AddField, ChangeField oder RemoveField. Weitere Informationen finden Sie im Thema „Datenmodelloperationen” auf Seite 64. ForEach-Elemente können auch geschachtelt werden.

Beispiele

Das folgende Beispiel führt eine Operation zehn Mal durch:

```
<ForEach var="val" from="1" to="10">
 ...
</ForEach>
```

Das folgende Beispiel führt eine Operation so oft durch, wie durch eine ganzzahlige Eigenschaft angegeben:

```
<ForEach var="val" from="1" to="${history_count}">
 ...
</ForEach>
```

Im nächsten Beispiel iteriert die Verarbeitung durch die Werte in den Ausgabefeldern für den Knoten:

```
<ForEach var="field" inFields="outputs">
 ...
</ForEach>
```

Das folgende Beispiel iteriert durch die Werte in den Metadaten für die Felder, die durch `${field}` abgegeben werden:

```
<ForEach var="fieldValue" inFieldValues="${field}">
 ...
</ForEach>
```

Das nächste Beispiel iteriert durch die Werte in einer Listeneigenschaft:

```
<ForEach var="val" inProperty="Meine_Listeneigenschaft">
 ...
</ForEach>
```

Das folgende Beispiel iteriert durch die Schlüsselwerte in einer verschlüsselten Eigenschaft:

```
<ForEach var="key" inProperty="Meine_verschlüsselte_Eigenschaft">
 ...
</ForEach>
```

Felder und Feldmetadaten

Knoten, Modelle und Datenquellen agieren als **Datenmodellprovider** und können Feldmetadaten definieren, auf die von anderen Objekten zugegriffen werden kann.

Datenmodellprovider haben ein Eingabedatenmodell und ein Ausgabedatenmodell. Ein Ausgabedatenmodell kann abhängig vom Eingabedatenmodell beschrieben werden, indem z. B. ein Eingabedatenmodell um ein Feld erweitert wird oder indem ein vorhandenes Modell geändert wird.

Alle Objekte haben leicht unterschiedliche Anforderungen.

Knoten. Das Eingabedatenmodell kann referenziert werden, ist aber nicht änderbar. Das Ausgabedatenmodell kann auf dem Eingabedatenmodell basieren oder dieses ersetzen. Das Ausgabedatenmodell wird jedes Mal neu berechnet, wenn die Knoteneigenschaften oder das Eingabedatenmodell geändert werden. Das Ausgabedatenmodell eines Modellanwendungsknotens kann auch das Ausgabedatenmodell der Modellkomponente referenzieren.

Modelle. Standardmäßig basieren die Eingabe- und Ausgabedatenmodelle (die Modellsignatur) auf den Eingabe- und Ausgabefeldeinstellungen, die beim Erstellen des Modells verwendet werden. Idealerweise gibt der Modellerstellungsprozess eine Metadatendatei zurück, die die erforderlichen Eingabefelder und die generierten Ausgabefelder definiert. Die Modellsignatur kann nach ihrer Definition nicht mehr geändert werden. Die Eigenschaften in einem Modellanwendungsknoten können jedoch die Datenmodellaus-

gabe des Anwendungsknotens verändern. Dise Eigenschaften können z. B. definieren, ob eine Cluster-ID als Zeichenfolge oder als Ganzzahl zurückgegeben wird, oder wie viele Sequenz-IDs generiert werden. Zusätzlich gibt die Modellsignatur in der Regel Ausgaben mit der Feldrolle (`direction`) `"out"` an, während der Knoten diese eher mit der Feldrolle `"in"` generiert.

Datenquellen. Datenquellen, die in Datenleserknoten verwendet werden, können ein Ausgabedatenmodell angeben. Das Eingabedatenmodell ist immer leer.

Feldsets

Ein Feldset kann an verschiedenen Stellen verwendet werden, um ein Subset von Feldern aus einem Datenmodellprovider auszuwählen. Der Datenmodellprovider kann entweder das umschließende Objekt oder ein Container des umschließenden Objekts sein. Im Ausgangszustand kann ein Feldfilter entweder alle verfügbaren Felder einschließen und dann bestimmte Feldtypen ausschließen oder mit einem leeren Feldsatz beginnen, um dann die erforderlichen Felder einzuschließen oder neue Felder hinzuzufügen.

Das folgende Beispiel zeigt, wie ein Erweiterungsknoten das Ausgabedatenmodell angeben kann. Die Schlüsselfelder werden durch eine Eigenschaftsliste namens `keys` angegeben, auf die ein optionales Datensatzzählerfeld folgt, das generiert werden kann und dessen Name auch durch eine Eigenschaft angegeben wird.

```
<OutputDataModel mode="replace">
     <ForEach var="field" inProperty="keys">
          <AddField name="${field}" fieldRef="${field}"/>
     </ForEach>
     <AddField name="${record_count_name}" storage="integer">
          <Condition property="include_record_count" op="equals" value="true"/>
     </AddField>
</OutputDataModel>
```

Feldsets und Modellierung

Das folgende Beispiel zeigt, wie ein Modellanwender die Informationen in der zuvor erstellten Modellkomponente verwenden kann, um seine Ausgabefelder zu erstellen:

```
<OutputDataModel mode="modify">
     <AddField provider="model" dataModel="output">
</OutputDataModel>
```

Sowohl `AddField` als auch `ForEach` geben einen Datenmodellprovider an und geben an, welche Eingabe- oder Ausgabedatenmodelle verwendet werden sollen. Sie bieten einen Mechanismus zur Angabe eines Sets (oder eines Subsets) von Feldern vom Datenmodellprovider. Der Standardprovider ist `this`, was für des einschließende Element steht (d. h. nicht das zu erstellende Objekt), dessen Eingabefeldset standardmäßig verwendet wird. Wenn kein Feldset angegeben ist, werden alle verfügbaren Felder verwendet.

Feldsets können auf Speicherung, Typ, Feldrolle oder Namen basieren. Wenn sie auf Namen basieren, ist eine Referenz auf eine Listeneigenschaft erforderlich. Das Feldset kann entweder voll sein (Standard) oder leer; im ersten Fall können Felder ausgeschlossen werden, im letzten eingeschlossen. Für jeden der einzelnen Filter können mehrere Werte festgelegt werden, die als "intersection"- oder "and"-Operatoren agieren, wie z. B.:

```
<FieldSet include="none"> <Include direction="in" storage="string"/> </FieldSet>
```

Dieses Beispiel beginnt mit einem leeren Feldset (durch `include="none"` angegeben) und schließt dann Felder ein, die über die Feldrolle (`direction`) `"in"` verfügen und als Zeichenfolge gespeichert werden.

Weiteres Beispiel:

```
<FieldSet include="all"> <Exclude type="typeless"/> </FieldSet>
```

Dieses Beispiel schließt alle verfügbaren Felder ein (festgelegt durch das Attribut `include="all"`, das das Standardverhalten ist) und schließt dann alle Felder mit dem Typ `typeless` aus. Dies schließt Felder ein, für die `direction` auf `"in"` oder `"both"` gesetzt ist.

Es können auch mehrere Filter angegeben werden, die als "union"- oder "or"-Operatoren agieren, wie z. B.:

```
<FieldSet include="all"> <Exclude type="discrete" storage="real"/>
<Exclude type="discrete" storage="integer"/> </FieldSet>
```

Dies schließt Felder aus, die entweder diskrete Zahlen sind, die als reelle gespeichert werden, oder diskrete Zahlen, die als ganze Zahlen gespeichert werden.

Beachten Sie, dass die Reihenfolge der `include`-Anweisungen beim Einschließen von Feldern in ein anfangs leeres Feldset in der Regel keine Auswirkung auf die Reihenfolge hat, in der die Felder eingeschlossen werden. Dies bedeutet, dass Felder von einem Feldset-Provider in ihrer natürlichen Reihenfolge in Bezug auf die Bedingung ausgewertet werden, um festzustellen, ob sie in das Feldset eingeschlossen werden.

Rollen

Eine Rolle beschreibt die Art der Daten, die im Ausgabefeld eines Datenmodells enthalten sind. Eine Rolle kann durch ein `AddField`-Element angegeben und durch ein `Condition`-Element getestet werden.

Im Folgenden sind die möglichen Rollen aufgeführt.

Tabelle 23. Modellausgaberollen

Rolle	Bedeutung
`unknown`	Keine Rolle angegeben (sollte nicht vorkommen).
`predictedValue`	Dieses Feld enthält den vorhergesagten Wert des Zielfelds.
`probability`	Die Wahrscheinlichkeit oder Konfidenz der Vorhersage.
`Residuen`	Der Wert des Residuums.
`standardError`	Der Standardfehler der Vorhersage.
`entityId`	Die ID des Elements; stellt in der Regel die Cluster-ID in einem Clustermodell dar.
`entityAffinity`	Die Affinität des Elements; stellt in der Regel die Entfernung vom Clusterzentrum in einem Modell dar.
`upperConfidenceLimit`	Die obere Konfidenzgrenze der Vorhersage.
`lowerConfidenceLimit`	Die untere Konfidenzgrenze der Vorhersage.
`propensity`	Der Propensity-Score. Ein zusätzliches `tag`-Attribut gibt an, ob sich diese Angabe auf eine Raw Propensity oder auf eine Adjusted Propensity bezieht.
`value`	Wird für die Darstellung eines Werts für Modelle verwendet, denen eine andere Ausgabe zugeordnet ist (siehe unten).
`supplementary`	Vom Modell erzeugte Informationen, die nicht durch andere Rollen abgedeckt werden.

Beispiel für `value`: Das Anomalieerkennungsmodell generiert Gruppen von Feldern, bei denen jede Gruppe aus zwei Feldern besteht, von denen das eine einen Feldnamen angibt und das andere ein Maß der Anomalie des Felds. In diesem Fall ist `value` der Feldname.

Logische Operatoren

Eine Anzahl von Elementen kann die logischen Operatoren `And`, `Or` und `Not` verwenden, um verschiedene Verarbeitungsarten anzugeben, z. B. wenn zusammengesetzte Bedingungen festgelegt werden (siehe „Zusammengesetzte Bedingungen" auf Seite 76).

Format

Das Format des `And`-Elements sieht wie folgt aus. Das Format der Elemente `Or` und `Not` ist immer identisch, der einzige Unterschied besteht in den einschließenden Tags `<Or>...</Or>` bzw. `<Not>...</Not>`.

```
<And>
     <Condition .../>
     <And ... />
     <Or ... />
     <Not ... />
</And>
```

Das Element `Condition` gibt eine zu testende Bedingung an. Weitere Informationen finden Sie im Thema „Bedingungen".

Beachten Sie, dass die untergeordneten Elemente `And`, `Or` und `Not` geschachtelt werden können.

Bedingungen

Das Verhalten einiger Objekte kann mithilfe von Bedingungen geändert werden. Diese werden in `Condition`-Elementen angegeben (die IF-Anweisungen entsprechen). Beispiel: Ein Knoten, der durch einen Befehl ausgeführt wird, kann zur Ausführungsinformation eine Bedingung hinzufügen, wie z. B., dass eine bestimmte Option nur dann eingeschlossen wird, wenn eine Eigenschaft einen bestimmten Wert hat. Genauso kann ein Eigenschaftssteuerelement auf der Benutzerschnittstelle nur dann aktiviert oder sichtbar sein, wenn ein anderes Steuerelement einen bestimmten Wert hat.

Bedingungen können einfach oder zusammengesetzt sein. Eine **einfache Bedingung** hat folgende Bestandteile:

- eine Wertquelle (entweder eine Eigenschaft oder ein Steuerelement)
- einen Test
- einen optionalen Testwert

Eine **zusammengesetzte Bedingung** ermöglicht die Kombination von Bedingungen zu komplexen logischen Bedingungen. Für zusammengesetzte Bedingungen kommen zum Einsatz:

- `And`
- `Or`
- `Not`

Format

```
<Condition container="Containername" control="Eigenschaftsname" property="Name" op="Operator"
     value="Wert" />
```

Dabei gilt Folgendes:

`container` gibt den Namen eines bestimmten Containers an, dessen Wert durch die Bedingung getestet wird.

`control` gibt ein Eigenschaftssteuerelement an, dessen Wert durch die Bedingung getestet wird. *Eigenschaftsname* ist der Wert des `property`-Attributs des Elements, in dem das Steuerelement definiert ist (z. B. in einem Eigenschaftsfenster auf einer Dialogfeldregisterkarte).

`property` gibt eine Eigenschaft an, deren Wert durch die Bedingung getestet wird. *Name* ist der Wert des `name`-Attributs des `Property`-Elements, in dem die Eigenschaft definiert ist.

`op` ist der Bedingungsoperator. Weitere Informationen finden Sie im Thema „Bedingungsoperatoren".

`value` ist der spezifische Wert, der durch die Bedingung getestet wird.

Beispiele

Beispiele für das Festlegen von Bedingungen finden Sie in „Einfache Bedingungen" auf Seite 75 und „Zusammengesetzte Bedingungen" auf Seite 76.

Bedingungsoperatoren

Es steht ein Satz Operatoren zur Verfügung, der die Formulierung der meisten Bedingungen ermöglicht.

Tabelle 24. Für beliebige Werte unterstützte Tests

Operator	Wert	Beschreibung
`equals`	*Wert*	Wahr, wenn die Eigenschaft gleich dem angegebenen Wert ist (bei Zeichenfolgen wird zwischen Groß- und Kleinschreibung unterschieden).
`notEquals`	*Wert*	Wahr, wenn die Eigenschaft nicht dem angegebenen Wert entspricht (bei Zeichenfolgen wird zwischen Groß- und Kleinschreibung unterschieden).
`in`	*Werteliste*	Wahr, wenn sich die Eigenschaft in der angegebenen Werteliste befindet.

Tabelle 25. Für numerische Werte unterstützte Tests

Operator	Wert	Beschreibung
`lessThan`	*Zahl*	Wahr, wenn die Eigenschaft kleiner als die angegebene Zahl ist.
`lessOrEquals`	*Zahl*	Wahr, wenn die Eigenschaft kleiner-gleich der angegebenen Zahl ist.
`greaterThan`	*Zahl*	Wahr, wenn die Eigenschaft größer als die angegebene Zahl ist.
`greaterOrEquals`	*Zahl*	Wahr, wenn die Eigenschaft größer-gleich der angegebenen Zahl ist.

Tabelle 26. Für Zeichenfolgen unterstützte Tests

Operator	Wert	Beschreibung
`isEmpty`	-	Wahr, wenn die Eigenschaft eine Zeichenfolge der Länge null ist.
`isNotEmpty`	-	Wahr, wenn die Eigenschaft keine Zeichenfolge der Länge null ist.
`startsWith`	*Zeichenfolge*	Wahr, wenn die Eigenschaft mit der angegebenen Zeichenfolge beginnt (Groß- und Kleinschreibung wird unterschieden).
`startsWithIgnoreCase`	*Zeichenfolge*	Wahr, wenn die Eigenschaft mit der angegebenen Zeichenfolge beginnt, wobei die Groß- und Kleinschreibung ignoriert wird.

Tabelle 26. Für Zeichenfolgen unterstützte Tests (Forts.)

Operator	Wert	Beschreibung
`endsWith`	*Zeichenfolge*	Wahr, wenn die Eigenschaft auf die angegebene Zeichenfolge endet (Groß- und Kleinschreibung wird unterschieden).
`endsWithIgnoreCase`	*Zeichenfolge*	Wahr, wenn die Eigenschaft auf die angegebene Zeichenfolge endet, wobei die Groß- und Kleinschreibung ignoriert wird.
`equalsIgnoreCase`	*Zeichenfolge*	Wahr, wenn die Eigenschaft gleich der angegebenen Zeichenfolge ist, wobei die Groß- und Kleinschreibung ignoriert wird.
`hasSubstring`	*Zeichenfolge*	Wahr, wenn die Eigenschaft die angegebene Zeichenfolge enthält (Groß- und Kleinschreibung wird unterschieden).
`hasSubstringIgnoreCase`	*Zeichenfolge*	Wahr, wenn die Eigenschaft die angegebene Zeichenfolge enthält, wobei die Groß- und Kleinschreibung ignoriert wird.
`isSubstring`	*Zeichenfolge*	Wahr, wenn die Eigenschaft eine Teilzeichenfolge der angegebenen Zeichenfolge ist (Groß- und Kleinschreibung wird unterschieden).
`isSubstringIgnoreCase`	*Zeichenfolge*	Wahr, wenn die Eigenschaft eine Teilzeichenfolge der angegebenen Zeichenfolge ist, wobei die Groß- und Kleinschreibung ignoriert wird.

Tabelle 27. Für Listeneigenschaften unterstützte Tests

Operator	Wert	Beschreibung
`isEmpty`	-	Wahr, wenn die Anzahl der in der Liste vorhandenen Elemente 0 ist.
`isNotEmpty`	-	Wahr, wenn die Anzahl der in der Liste vorhandenen Elemente ungleich 0 ist.
`countEquals`	*Zahl*	Wahr, wenn die Anzahl der in der Liste enthaltenen Elemente gleich dem angegebenen Wert ist.
`countLessThan`	*Zahl*	Wahr, wenn die Anzahl der in der Liste enthaltenen Elemente kleiner als der angegebene Wert ist.
`countLessOrEquals`	*Zahl*	Wahr, wenn die Anzahl der in der Liste enthaltenen Elemente kleiner oder gleich der angegebene Wert ist.
`countGreaterThan`	*Zahl*	Wahr, wenn die Anzahl der in der Liste enthaltenen Elemente größer als der angegebene Wert ist.
`countGreaterOrEquals`	*Zahl*	Wahr, wenn die Anzahl der in der Liste enthaltenen Elemente größer oder gleich der angegebene Wert ist.
`contains`	*Wert*	Wahr, wenn das angegebene Element in der Liste enthalten ist.

Tabelle 28. Für Feldeigenschaften unterstützte Tests

Operator	Beschreibung
`storageEquals`	Wahr, wenn der Speicher gleich dem angegebenen Wert ist.
`typeEquals`	Wahr, wenn der Typ gleich dem angegebenen Wert ist.
`directionEquals`	Wahr, wenn die Feldrolle (`direction`) gleich dem angegebenen Wert ist.

Tabelle 28. Für Feldeigenschaften unterstützte Tests (Forts.)

Operator	Beschreibung
isMeasureDiscrete	Wahr, wenn der Felddatentyp mit discrete angegeben ist (d. h., dass er ausschließlich set, flag oder orderedSet sein kann).
isMeasureContinuous	Wahr, wenn der Felddatentyp range ist.
isMeasureTypeless	Wahr, wenn der Felddatentyp typeless ist.
isMeasureUnknown	Wahr, wenn der Felddatentyp unknown ist.
isStorageString	Wahr, wenn der Feldspeichertyp string ist.
isStorageNumeric	Wahr, wenn der Feldspeichertyp numeric ist.
isStorageDatetime	Wahr, wenn der Feldspeichertyp datetime ist.
isStorageUnknown	Wahr, wenn der Feldspeichertyp unknown ist.
isModelOutput	Wahr, wenn das Feld ein Modellausgabefeld ist.
modelOutputRoleEquals	Wahr, wenn die Rolle für das Feld eine der in der nächsten Tabelle enthaltenen gültigen Rollen ist.
modelOutputTargetFieldEquals	Wahr, wenn das Zielfeld gleich der angegebenen Wert (Zeichenfolge) ist.
modelOutputHasValue	Wahr, wenn das Feld ein Modellausgabefeld ist, dem ein Wert zugeordnet ist.
modelOutputTagEquals	Wahr, wenn das Tag gleich der angegebenen Wert (Zeichenfolge) ist.

Bedingungsoperatoren, die verschlüsselten Eigenschaften unterstützen:

- isEmpty
- isNotEmpty
- countEquals
- countLessThan
- countLessOrEquals
- countGreaterThan
- countGreaterOrEquals
- contains

Einfache Bedingungen

Eine einfache Bedingung besteht aus einer ersten Wertquelle, die getestet werden soll (entweder eine Eigenschaft oder ein Controllername oder ein ausgewerteter Ausdruck), dem durchzuführenden Test und optional einem Wert, gegen den der Test durchgeführt werden soll.

Beispiele

Die folgende Bedingung liefert das Ergebnis 'True', wenn eine boolesche Eigenschaft namens values_grouped wahr ist:

```
<Condition property="values_grouped" op="equals" value="true"/>
```

Das nächste Beispiel liefert das Ergebnis 'True', wenn ein Steuerelement namens values_grouped, das boolesche Werte anzeigt, aktiviert ist:

```
<Condition control="values_grouped" op="equals" value="true"/>
```

Das folgende Beispiel liefert das Ergebnis 'True', wenn eine Listeneigenschaft namens plot_fields mindestens einen Wert enthält:

```
<Condition property="plot_fields" op="countGreaterThan" value="0"/>
```

Das nächste Beispiel liefert das Ergebnis 'True', wenn eine Listeneigenschaft namens `input_fields` nur instanziierte Werte enthält:

```
<Condition property="input_fields" op="instantiated" listMode="all"/>
```

Das letzte Beispiel liefert das Ergebnis 'True', wenn eine Listeneigenschaft namens `input_fields` mindestens einen Wert hat, der ein nicht instanziiertes Feld darstellt:

```
<Condition property="input_fields" op="uninstantiated" listMode="any "/>
```

Zusammengesetzte Bedingungen

Gruppen einfacher Bedingungen können mithilfe logischer Operatoren kombiniert werden.

Beispiele

Die folgende Bedingung liefert das Ergebnis 'True', wenn die boolesche Eigenschaft `values_grouped` wahr ist und `group_fields` mindestens einen Wert enthält:

```
<And>
     <Condition property="values_grouped" op="equals" value="true"/>
     <Condition property="group_fields" op="countGreaterThan" value="0"/>
</And>
```

Die folgende Bedingung liefert das Ergebnis 'True', wenn die boolesche Eigenschaft `values_grouped` wahr ist oder wenn `group_fields` mindestens einen Wert enthält:

```
<Or>
     <Condition property="values_grouped" op="equals" value="true"/>
     <Condition property="group_fields" op="countGreaterThan" value="0"/>
</Or>
```

Das folgende Beispiel liefert das Ergebnis 'True', wenn `group_fields` mindestens einen Wert enthält:

```
<Not>
     <Condition property="group_fields" op="equals" value="0"/>
</Not>
```

Zusammengesetzte Bedingungen können geschachtelt werden, um beliebige Kombinationen von Bedingungen zu realisieren.

Verwenden von CLEF-Knoten in Scripts

Sie können in einem Script auf einen CLEF-Knoten verweisen, indem Sie das Attribut `scriptName` des Elements `Node` verwenden. Auf dieselbe Weise können Sie in einem Script mithilfe des Attributs `scriptName` des Elements `Property` auf eine Eigenschaft des Knotens verweisen.

Das Attribut `scriptName` ist in beiden Fällen optional, seine Verwendung wird jedoch empfohlen, um Namenskonflikte zwischen Erweiterungen und Eigenschaften zu vermeiden.

Wenn Sie den Scriptnamen in einer Knotendefinition weglassen, kann das Script mithilfe des Werts vom Attribut `id`, dem der Erweiterungsname als Präfix vorangestellt ist, auf den Knoten verweisen. Beispiel: Eine Erweiterung namens `myext`, die einen Datenleserknoten mit der ID `import` definiert, kann in einem Script als `myextimport` auf den Knoten verweisen.

Wenn Sie den Scriptnamen in einer Eigenschaftsdefinition unterdrücken, kann ein Script auf die Eigenschaft durch den Wert ihres Attributs `name` verweisen.

Weitere Informationen finden Sie im *IBM SPSS Modeler-Handbuch für Scripterstellung und Automatisierung.*

Beispiel - Bearbeiten und Ausführen eines Knotens

Das folgende Beispiel zeigt, wie Sie mithilfe eines Scripts die Aufgaben der Bearbeitung und Ausführung des im Beispiel verwendeten Datenleserknotens (siehe „Datenleserknoten (Apache Log Reader)" auf Seite 26) automatisieren können.

In der Spezifikationsdatei für den Knoten "Apache Log Reader" beginnt die Knotenspezifikation wie folgt:

```
<Node id="apachelogreader" type="dataReader" palette="import" labelKey="apacheLogReader.LABEL">
     <Properties>
          <Property name="log_filename" valueType="string" labelKey="logfileName.LABEL" />
     </Properties>
```

Im Script würden Sie wie folgt auf den Knoten und die Eigenschaft verweisen:

```
create apachelogreader
set :apachelogreader.log_filename='Installationsverzeichnis\Demos\combined_log_format.txt'
create tablenode at 200 100
connect :apachelogreader to :tablenode
execute :tablenode
```

Dabei bezeichnet *Installationsverzeichnis* das Verzeichnis, in dem IBM SPSS Modeler installiert ist.

Das Ausführen dieses Scripts bewirkt Folgendes:

- Erstellen des Datenleserknotens
- Angabe von *combined_log_format.txt* als zu lesende Apache-Protokolldatei
- Erstellung eines Tabellenknotens
- Verbindung des Datenleserknotens mit dem Tabellenknoten
- Ausführung des Tabellenknotens

Beispiel - Verschlüsselte Eigenschaften

Verschlüsselte Eigenschaften unterstützen Standardsyntax für die Scripterstellung. Beispielsweise könnte die Struktur aus dem ersten Beispiel für verschlüsselte Eigenschaftstypen in „Strukturierte Eigenschaften" auf Seite 62 in einem Script wie folgt definiert werden:

```
set :mynode.aggregation_settings.Age = {true true false false false}
```

Ein einzelnes Attribut könnte wie folgt geändert werden:

```
set :mynode.aggregation_settings.Age.MIN = true
```

Erhaltung der Abwärtskompatibilität

Beim Planen von Aktualisierungen für eine vorhandene Erweiterung müssen Sie darauf achten, dass die Kompatibilität zu vorher verteilten Versionen der Erweiterung erhalten bleibt. Einige Änderungen haben keine problematischen Auswirkungen, einige stellen ein signifikantes Risiko dar und bei anderen ist bekannt, dass sie die Kompatibilität aufheben und daher vermieden werden sollten.

Änderungen ohne Risiko

Die folgenden Änderungen haben keine Auswirkungen auf die Abwärtskompatibilität:

- Hinzufügen neuer `Node`-, `ModelOutput`-, `DocumentOutput`- oder `InteractiveModelBuilder`-Elemente
- Hinzufügen neuer `Property`-Definitionen und der diesen Elementen zugeordneten neuen Steuerelemente
- Hinzufügen neuer Container dieser Elemente*

- Hinzufügen neuer Werte zu einer vorhandenen Aufzählungseigenschaft

* Beachten Sie, dass jeder Code, der diese neuen Container verwendet, zulassen sollte, dass diese Container für Objekte, die mit der vorherigen Version der Erweiterung erstellt wurden, leer sind.

Änderungen mit signifikantem Risiko

Änderungen an vorhandenen Deklarationen bergen ein signifikantes Risiko hinsichtlich der Kompatibilität. Sie sollten vor der Freigabe sorgfältig getestet werden.

Zu vermeidende Änderungen

Bei den folgenden Änderungen ist bekannt, dass sie die Kompatibilität beeinträchtigen. Sie sollten daher vermieden werden:

- Änderung des Werts der `id`- oder `providerTag`-Attribute im `ExtensionDetail`-Element
- Änderung des Werts des `id`-Attributs in einem `Node`-, `ModelOutput`-, `DocumentOutput`- oder `InteractiveModelBuilder`-Element
- Entfernen eines `Node`-, `ModelOutput`-, `DocumentOutput`- oder `InteractiveModelBuilder`-Elements aus der Erweiterung
- Änderung des Werts des `valueType`-Attributs eines `Property`- oder `PropertyType`-Elements

Kapitel 5. Erstellen von Modellen und Dokumenten

Einführung in die Modell- und Dokumenterstellung

Die Standardmodule von IBM SPSS Modeler umfassen Knoten für die Generierung bzw. Erstellung der verschiedensten Modelle und Diagramme. CLEF ermöglicht die Definition weiterer Knoten für die Generierung von Modellen und Dokumenten (Diagrammen und Berichten), die nicht durch die Standardmodule bereitgestellt werden.

Bei der Definition von Modell- oder Dokumenterstellungsknoten müssen Sie auch die Objekte definieren, die bei der Ausführung dieser Knoten erstellt werden. Dazu verwenden Sie Elemente, die als "Konstruktoren" bezeichnet werden.

In den folgenden Abschnitten wird dieser Vorgang ausführlich beschrieben.

Modelle

Ein **Modell** ist ein Regelset, eine Formel oder eine Gleichung, die ein Ergebnis auf Grundlage von bestimmten Eingabefeldern vorhersagen können. Die Vorhersage eines Ergebnisses ist das zentrale Ziel der Vorhersageanalyse. In IBM SPSS Modeler erreichen Sie dies durch folgende Schritte:

- Generieren eines Modells aus vorhandenen Daten
- Anwenden des generierten Modells auf die Daten, die für die Vorhersage hinzugezogen werden

Die Generierung eines Modells wird auch als "Modellerstellung" bezeichnet. In IBM SPSS Modeler erstellen Sie ein Modell mithilfe eines Modellierungsknotens. In CLEF werden Modellierungsknoten als **Model Builder-Knoten** bezeichnet, zu Deutsch "Modellerstellungsknoten". Diese Bezeichnung leitet sich von der Syntax der XML-Anweisung ab, die zur Definition dieser Knoten verwendet wird. Weitere Informationen finden Sie im Thema „Modellerstellungsknoten” auf Seite 11.

Die Anwendung eines Modells auf Daten wird auch als "Datenscoring" bezeichnet. Beim Scoring werden die aus der Modellerstellung gewonnenen Informationen für Vorhersagen für neue Datensätze verwendet. In IBM SPSS Modeler ziehen Sie dazu das Symbol eines generierten Modells auf den Streamerstellungsbereich. Da das Symbol wie ein Goldnugget aussieht, wird ein generiertes Modell in IBM SPSS Modeler auch als "Modellnugget" bezeichnet. In CLEF wird ein Modellnugget auf der Registerkarte "Modelle" des Managerfensters auch als **Modellausgabeobjekt** bezeichnet, während es im Erstellungsbereich als **Modellanwendungsknoten** bezeichnet wird. Weitere Informationen finden Sie im Thema „Modellanwendungsknoten” auf Seite 12.

Dokumente

Vermutlich möchten Sie nicht nur Modelle, sondern auch andere Objekte, beispielsweise Diagramme oder Berichte, erstellen. In IBM SPSS Modeler werden diese Objekte als **Dokumente** bezeichnet. Sie werden mithilfe eines **Dokumenterstellungsknotens** generiert. Weitere Informationen finden Sie im Thema „Dokumenterstellungsknoten” auf Seite 12.

Konstruktoren

Konstruktoren definieren die Objekte, die entweder als Ergebnis der Ausführung eines Knotens in einen Stream zurückgeliefert oder als Ergebnis der Rückgabe eines Objekts in den Stream erstellt werden.

Konstruktoren können für die folgenden Elemente definiert werden:

- Modellerstellungsknoten
- Dokumenterstellungsknoten

- Modellanwendungsknoten
- Modellausgabeobjekt

Bei **Modell-** und **Dokumenterstellungsknoten** legt der Konstruktor fest, wie das Ausgabeobjekt bei der Ausführung des Knotens generiert wird. Die Definition eines Ausgabeobjekts kann mehrere Eigenschaften und Komponenten umfassen und im Abschnitt "Constructors" wird festgelegt, wie diese initialisiert bzw. aus dem bei der Ausführung generierten Objekt erstellt werden.

Bei **Modellanwendungsknoten** legt der Konstruktor fest, welche Art von Objekt der Knoten zurück in den Stream liefert bzw. welche Art von Objekt er auf die Registerkarte "Modelle" ausgibt.

Bei **Modellausgabeobjekten** erfüllen Konstruktoren folgende Aufgaben:

- Festlegung des Modellanwendungsknotens, der beim Ablegen des Modellausgabeobjekts auf den Streamerstellungsbereich erstellt wird
- Generieren eines Modellerstellungsknotens mit denselben Einstellungen, die für die Erstellung des Modellausgabeobjekts verwendet wurden

Weitere Informationen finden Sie im Thema „Verwenden von Konstruktoren” auf Seite 100.

Erstellen von Modellen

Wenn Sie einen Knoten definieren, der ein Modell generieren kann (d. h., Sie definieren einen Modellerstellungsknoten), müssen Sie festlegen, wie der Knoten mit der Model Builder-Komponente von IBM SPSS Modeler kommuniziert. Dadurch definieren Sie den Prozess, der das Modell tatsächlich erstellt. Diese Definition legen Sie in der Spezifikationsdatei in einem `Node`-Element fest.

Sofern nicht anders angegeben, beginnt die Modellerstellung, sobald der Endbenutzer im Dialogfeld des Modellerstellungsknotens auf die Schaltfläche **Ausführen** klickt. Allerdings kann auch ein **interaktives Modell** definiert werden. In diesem Fall kann der Endbenutzer die Datenwerte nach dem Anklicken der Schaltfläche **Ausführen** noch vor der Erstellung des Modells einstellen bzw. anpassen. Für die interaktive Modellerstellung sind zusätzlich bestimmte Elemente erforderlich, die das interaktive Verhalten steuern. Weitere Informationen finden Sie im Thema „Erstellen von interaktiven Modellen” auf Seite 89.

Für die Definition eines Modellerstellungsknotens müssen Sie dem `Node`-Element folgende Elemente hinzufügen:

- Ein Attribut `type="modelBuilder"`
- Ein untergeordnetes Element `ModelBuilder`
- Ein untergeordnetes Element `Constructors` mit einem Element `CreateModelOutput` (siehe „Verwenden von Konstruktoren” auf Seite 100)

Informationen zum Format einer `Node`-Elementspezifikation finden Sie in „Knoten” auf Seite 48.

Hinweis: Die in den nachfolgenden Elementdefinitionen (Abschnitte mit der Überschrift **Format**) angegebenen Elementattribute und untergeordneten Elemente sind optional, sofern Sie nicht als "(erforderlich)" gekennzeichnet sind. Die vollständige Elementsyntax finden Sie in „CLEF-XML-Schema”, auf Seite 207.

Die Erweiterung muss für die Modellerstellung auch über ein `ModelOutput`-Element verfügen, das das generierte Modell beschreibt (siehe „Modellausgabe” auf Seite 88). Das `ModelOutput`-Element muss ein untergeordnetes `Constructors`-Element mit einer `CreateModelApplier`-Definition enthalten. Weitere Informationen finden Sie im Thema „Erstellen des Modellanwenders” auf Seite 102.

Modellerstellung (ModelBuilder)

Das `ModelBuilder`-Element legt das Verhalten eines Modellerstellungsknotens fest. Die Definition erfolgt über die Attribute des Elements und ein oder mehrere untergeordnete Elemente.

Format

```
<ModelBuilder allowNoInputs="true_false" allowNoOutputs="true_false" nullifyBlanks="true_false"
     miningFunctions="[Funktion1 Funktion2 ... ]" >
     <Algorithm ... />
     <ModelingFields ... />
     <ModelGeneration ... />
     <ModelFields ... />
     <AutoModeling ... />
</ModelBuilder>
```

Dabei gilt Folgendes:

- `allowNoInputs` bzw. `allowNoOutputs` muss explizit angegeben werden, wenn Sie ein Modell erstellen möchten, das entweder über keine Eingabefelder oder über keine Ausgabefelder verfügt.
- Wenn `nullifyBlanks` auf `false` gesetzt ist, wird die Funktion inaktiviert, wobei leere Werte in den Daten, die an die Model Builder-Komponente von IBM SPSS Modeler übergeben werden, durch Nullwerte ersetzt werden (dargestellt durch `$null$`). Standardmäßig werden leere Werte durch Nullwerte ersetzt. Wenn Ihr Algorithmus leere Werte auf eine andere Weise behandeln muss, müssen Sie diese Funktion jedoch inaktivieren.
- `miningFunctions` (erforderlich) legt die Data-Mining-Funktionen fest (d. h. die Funktionen, die das Modell ausführt).

Tabelle 29. Data-Mining-Funktionen.

Funktion	Beschreibung
`classification`	Sagt aus Datensätzen mit bekannten Zielwerten einen diskreten Wert (d. h. einen Wert mit dem Datentyp `set`, `flag` oder `orderedSet`) eines unbekannten Zielattributs vorher.
`approximation`	Sagt aus Datensätzen mit bekannten Zielwerten einen stetigen Wert (d. h. einen Wert mit dem Datentyp `range`) eines unbekannten Zielattributs vorher.
`clustering`	Ermittelt Gruppen mit ähnlichen Datensätzen und beschriftet sie entsprechend.
`association`	Ermittelt aus Daten verwandte Ereignisse oder Attribute.
`sequence`	Sucht in zeitlich strukturierten Daten nach sequenziellen Mustern.
`reduction`	Reduziert die Datenkomplexität, z. B. mittels abgeleiteter Felder, die den Inhalt der ursprünglichen Datenfelder zusammenfassen.
`conceptExtraction`	Wird im Textmining verwendet.
`categorize`	Wird im Textmining verwendet.
`timeSeries`	Sagt aus vergangenen Datenmustern zukünftige Werte vorher.
`anomalyDetection`	Sucht anhand von Abweichungen von den Normen der jeweiligen Clustergruppen nach ungewöhnlichen Fällen.
`attributeImportance`	Ermittelt die Attribute, die den größten Einfluss auf ein Zielattribut haben.
`supervisedMultiTarget`	Schätzt für eine von mehreren Möglichkeiten die Wahrscheinlichkeit eines Ergebnisses (*Ja* oder *Nein*).

Wenn das Modell mehrere Funktionen ausführt, werden die Namen der Funktionen innerhalb der eckigen Klammern durch Leerzeichen getrennt, wie das folgende Beispiel zeigt:

```
<ModelBuilder miningFunctions="[classification approximation]">
 ...
</ModelBuilder>
```

Untergeordnete Elemente

Das Element ModelBuilder kann die in der folgenden Tabelle aufgeführten untergeordneten Elemente enthalten.

Tabelle 30. Untergeordnete Elemente der ModelBuilder-Deklaration.

Untergeordnetes Element	Beschreibung	Siehe...
Algorithm	(Erforderlich) gibt den Algorithmus für die Generierung des Modells an.	„Algorithmus”
ModelingFields	Gibt die Kennung an, die nachfolgend im Abschnitt "UserInterface" zur Angabe der Position der Steuerelemente für die Eingabe- und Ausgabefelder des Modells verwendet wird. Die Steuerelemente selbst werden in den untergeordneten Elementen InputFields und OutputFields des Elements ModelingFields definiert.	„Modellierungsfelder”
ModelGeneration	Gibt die Kennung an, die nachfolgend im Abschnitt "UserInterface" zur Angabe der Position der Steuerelemente für den Modellnamen des generierten Modells verwendet wird.	„Modellgenerierung” auf Seite 85
ModelFields	Gibt den Satz der Eingabe- und Ausgabefelder an, der für das Datenscoring in diesem Modell verwendet wird.	„Modellfelder” auf Seite 86
AutoModeling	Ermöglicht die Verwendung dieses Modells in einem Ensemblemodellierungsknoten (z. B. in den Knoten "Automatisches Klassifikationsmerkmal", "Autom. Cluster" oder "Autonumerisch").	„Automatisierte Modellierung” auf Seite 93

Algorithmus

Das Element Algorithm legt die Details des Algorithmus fest, der für die Generierung des Modells verwendet wird.

```
<Algorithm value="Modellausgabe-ID" label="Anzeigebeschriftung" labelKey="Beschriftungsschlüssel"/>
```

Dabei gilt Folgendes:

- value (erforderlich) ist der interne Name des Algorithmus. Dieser Name wird an zahlreichen anderen Stellen der Spezifikationsdatei referenziert. Weitere Informationen finden Sie im Thema „Beispiel für den Abschnitt "modelBuilder"” auf Seite 86.
- label (erforderlich) ist die Beschreibung des Algorithmus.
- labelKey gibt die Beschriftung für Lokalisierungszwecke an.

Modellierungsfelder

Standardmäßig werden die Eingabe- und Ausgabefelder des Modells anhand des Typknotens festgelegt. Die Benutzer legen die Feldrolle nach Bedarf mit **Prädiktor** oder **Ziel** fest. In einem Modellerstellungsknoten können Sie den Benutzern optional die Möglichkeit einräumen, die Einstellungen in einem vorgeordneten Typknoten zu überschreiben und benutzerdefinierte Einstellungen zu verwenden.

Dazu verwenden Sie das Element ModelingFields. Dieses Element legt eine Kennung fest, die nachfolgend im Abschnitt "UserInterface" der Deklaration des Modellerstellungsknotens verwendet wird, um die Position der Steuerelemente der Eingabe- und Ausgabefelder des Modells anzugeben. Die Steuerelemente selbst werden in den untergeordneten Elementen InputFields und OutputFields definiert.

Format

```
<ModelingFields controlsId="Steuerelement-ID" ignoreBOTH="true_false" >
      <InputFields ... />
      <OutputFields ... />
</ModelingFields>
```

Dabei gilt Folgendes:

- `controlsId` (erforderlich) ist die Kennung, die nachfolgend in der Deklaration des Modellerstellungsknotens im `SystemControls`-Element des Abschnitts "UserInterface" verwendet wird. Damit legen Sie fest, auf welcher Registerkarte des Knotendialogfelds sich die Steuerelemente für die Eingabe- und Ausgabefelder des Modells befinden.
- Wenn `ignoreBOTH` auf `true` (Standardwert) gesetzt ist, werden Felder, deren Feldrolle auf **both** gesetzt ist, vom Modell ignoriert.

Die Elemente `InputFields` und `OutputFields` werden ab dem Abschnitt „InputFields" beschrieben.

Beispiel

Dieses Beispiel veranschaulicht die Verwendung eines ModelingFields-Steuerelementsatzes auf der Registerkarte "Felder" im Dialogfeld eines Modellerstellungsknotens. Zunächst wird für den Steuerelementsatz eine Kennung festgelegt:

```
<ModelBuilder miningFunctions="[classification]">
 ...
     <ModelingFields controlsId="modelingFields">
          <InputFields property="inputs" onlyNumeric="true" multiple="true" label="Inputs"
            labelKey="inputFields.LABEL"/>
          <OutputFields property="target" multiple="false" types="[set flag]" label="Target"
            labelKey="targetField.LABEL"/>
     </ModelingFields>
 ...
</ModelBuilder>
```

Die `modelingFields`-Kennung wird nachfolgend im Abschnitt "UserInterface" des Knotendialogfelds an der Stelle referenziert, an der die Registerkarte "Felder" definiert wird:

```
<UserInterface ... >
     <Tabs defaultTab="1">
          <Tab label="Fields" labelKey="Fields.LABEL" helpLink="modeling_fieldstab.htm">
               <PropertiesPanel>
                    <SystemControls controlsId="modelingFields">
                    </SystemControls>
               </PropertiesPanel>
          </Tab>
 ...
</UserInterface>
```

InputFields: Das Element `InputFields` legt den Feldsatz fest, aus dem die Benutzer ein oder mehrere Eingabefelder (d. h. Prädiktoren) für das Modell auswählen können.

Das Set besteht aus allen Feldern, die an diesem Knoten sichtbar sind. Wenn Felder oberhalb dieses Knotens gefiltert wurden, sind nur die Felder sichtbar, die den Filter durchlaufen haben. Die Liste kann noch weiter eingeschränkt werden, indem angegeben wird, dass nur Felder mit bestimmten Speicher- und Datentypen zur Auswahl stehen sollen.

```
<InputFields storage="Speichertypen" onlyNumeric="true_false" onlySymbolic="true_false"
     onlyDatetime="true_false" types="Datentypen" onlyRanges="true_false"
     onlyDiscrete="true_false" property="Eigenschaftsname" multiple="true_false" label="Beschriftung"
     labelKey="Beschriftungsschlüssel"/>
```

Sie können die Liste der als Eingabefelder möglichen Felder durch die Angabe von nur zwei Attributen einschränken - eines dieser Attribute muss aus folgender Liste stammen:

- `storage` ist eine Listeneigenschaft, die den Speichertyp der Felder angibt, die in der Liste zulässig sein sollen. So bedeutet `storage="[integer real]"`, dass nur Felder mit diesen Speichertypen aufgelistet werden. Die möglichen Speichertypen können Sie der Tabelle unter „Daten- und Speichertypen" auf Seite 186 entnehmen.
- `onlyNumeric` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit numerischem Speichertyp aufgelistet werden sollen.
- `onlySymbolic` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit symbolischem Speichertyp (also Zeichenfolgen) aufgelistet werden sollen.
- `onlyDatetime` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit einem Speichertyp für Datum und Uhrzeit aufgelistet werden sollen.

Das zweite angegebene Attribut muss aus dieser Liste stammen:

- `types` ist eine Listeneigenschaft, die den Datentyp der Felder angibt, die in der Liste zulässig sein sollen. So bedeutet `types="[range flag]"`, dass nur Felder mit diesen Speichertypen aufgelistet werden. Folgende Datentypen sind möglich:

  `range`

  `flag`

  `set`

  `orderedSet`

  `numeric`

  `discrete`

  `typeless`
- `onlyRanges` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit dem Datentyp "range" aufgelistet werden sollen.
- `onlyDiscrete` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit einem Datentyp 'discrete' (also 'flag', 'set' oder 'typeless') aufgelistet werden sollen.

So stellt beispielsweise ein Steuerelement mit der Angabe `storage="[integer]"` und `types="[flag]"` sicher, dass nur ganzzahlige Felder, bei denen es sich um Flags handelt, in der Liste angezeigt werden.

Die restlichen Attribute lauten wie folgt:

- `property` ist die Kennung der Eigenschaft, die zum Speichern der Feldwerte verwendet werden soll.
- `multiple` gibt an, ob es sich bei den Feldwerten um eine Aufzählungsliste handelt (`true`) oder nicht (`false`).
- `label` ist der Anzeigename des Steuerelements.
- `labelKey` gibt die Beschriftung für Lokalisierungszwecke an.

Ausgabefelder: Das Element `OutputFields` legt den Feldsatz fest, aus dem die Benutzer ein oder mehrere Ausgabefelder (d. h. Ziele) für das Modell auswählen können.

Das Set besteht aus allen Feldern, die an diesem Knoten sichtbar sind. Wenn Felder oberhalb dieses Knotens gefiltert wurden, sind nur die Felder sichtbar, die den Filter durchlaufen haben. Die Liste kann noch weiter eingeschränkt werden, indem angegeben wird, dass nur Felder mit bestimmten Speicher- und Datentypen zur Auswahl stehen sollen.

```
<OutputFields storage="Speichertypen" onlyNumeric="true_false" onlySymbolic="true_false"
     onlyDatetime="true_false" types="Datentypen" onlyRanges="true_false"
     onlyDiscrete="true_false" property="Eigenschaftsname" multiple="true_false" label="Beschriftung"
     labelKey="Beschriftungsschlüssel"/>
```

Sie können die Liste der als Ausgabefelder möglichen Felder durch die Angabe von nur zwei Attributen einschränken - eines dieser Attribute muss aus folgender Liste stammen:

- `storage` ist eine Listeneigenschaft, die den Speichertyp der Felder angibt, die in der Liste zulässig sein sollen. So bedeutet `storage="[integer real]"`, dass nur Felder mit diesen Speichertypen aufgelistet werden. Die möglichen Speichertypen können Sie der Tabelle unter „Daten- und Speichertypen" auf Seite 186 entnehmen.
- `onlyNumeric` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit numerischem Speichertyp aufgelistet werden sollen.
- `onlySymbolic` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit symbolischem Speichertyp (also Zeichenfolgen) aufgelistet werden sollen.
- `onlyDatetime` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit einem Speichertyp für Datum und Uhrzeit aufgelistet werden sollen.

Das zweite angegebene Attribut muss aus dieser Liste stammen:

- `types` ist eine Listeneigenschaft, die den Datentyp der Felder angibt, die in der Liste zulässig sein sollen. So bedeutet `types="[range flag]"`, dass nur Felder mit diesen Speichertypen aufgelistet werden. Folgende Datentypen sind möglich:

  ```
  range
  flag
  set
  orderedSet
  numeric
  discrete
  typeless
  ```

- `onlyRanges` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit dem Datentyp "range" aufgelistet werden sollen.
- `onlyDiscrete` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit einem Datentyp 'discrete' (also 'flag', 'set' oder 'typeless') aufgelistet werden sollen.

So stellt beispielsweise ein Steuerelement mit der Angabe `storage="[integer]"` und `types="[flag]"` sicher, dass nur ganzzahlige Felder, bei denen es sich um Flags handelt, in der Liste angezeigt werden.

Die restlichen Attribute lauten wie folgt:

- `property` ist die Kennung der Eigenschaft, die zum Speichern der Feldwerte verwendet werden soll.
- `multiple` gibt an, ob es sich bei den Feldwerten um eine Aufzählungsliste handelt (`true`) oder nicht (`false`).
- `label` ist der Anzeigename des Steuerelements.
- `labelKey` gibt die Beschriftung für Lokalisierungszwecke an.

Modellgenerierung

Das Element `ModelGeneration` legt die Kennung fest, die an anderen Stellen der Datei zur Angabe der Registerkarte des Dialogfelds des Modellerstellungsknotens verwendet werden soll, die die Steuerelemente für den Modellnamen des generierten Modells enthalten soll.

Format:

```
<ModelGeneration controlsId="Steuerelementekennung" />
```

Das Attribut `controlsId` legt die Kennung fest, die nachfolgend in der Spezifikation des Modellerstellungsknotens im `SystemControls`-Element des Abschnitts "UserInterface" verwendet wird. Die Steuerelemente für den Modellnamen werden auf der Registerkarte eingefügt, deren Spezifikation dieses `SystemControls`-Element enthält.

Modellfelder

Das Element ModelFields wird zur Erstellung der **Modellsignatur** verwendet. Hiermit ist der Satz der Eingabe- und Ausgabefelder gemeint, der für das Datenscoring in diesem Modell verwendet wird.

```
<ModelFields inputDirections="[in]" outputDirections="[out]">
     <AddField prefix="Feldpräfix" ... />
      ...
     <ForEach ... >
          <AddField prefix="Feldpräfix" ... />
     </ForEach>
      ...
</ModelFields>
```

Dabei geben inputDirections und outputDirections an, wie die Modellsignatur erstellt wird. Die Werte können in, out oder both sein.

Die Felder werden durch ein oder mehrere AddField-Elemente definiert. Das Attribut prefix gibt das Präfix an, das einem Feldnamen hinzugefügt wird, um die von dem Modell erstellten Felder zu kennzeichnen. Beispiel: Wenn der Feldname Feld1 lautet und das Präfix $S, dann erhält das generierte Feld den Namen $S-Feld1. Weitere Informationen finden Sie im Thema „Hinzufügen eines Felds" auf Seite 65.

Iteration wird durch das Element ForEach möglich. Weitere Informationen finden Sie im Thema „Iteration mit dem ForEach-Element" auf Seite 68.

Mithilfe von **Feldgruppen** können Sie zwei oder mehr Ausgabefelder des Modells zu Iterationszwecken gruppieren. Ausgabefeldnamen erhalten ein Suffix, das die Iteration angibt, z. B. $S-field1-1 , $S-field1-2 usw. Feldgruppen lassen sich beispielsweise dafür verwenden, dieselbe Feldgruppe mehrmals in der Modellausgabe anzuzeigen. Weitere Informationen finden Sie im Thema „Beispiel für Feldgruppe" auf Seite 87.

Automatische Modellierung

Das Element "AutoModeling" ermöglicht die Verwendung des Modells in einem Ensemblemodellierungsknoten (z. B. in den Knoten "Automatisches Klassifikationsmerkmal", "Autom. Cluster" oder "Autonumerisch"). Weitere Informationen finden Sie im Thema „Automatisierte Modellierung" auf Seite 93.

Beispiel für den Abschnitt "modelBuilder"

Nachfolgend ist der vollständige Abschnitt "modelBuilder" der Spezifikationsdatei am Beispiel des Interaktionsknotens dargestellt (siehe „Modellerstellungsknoten (Interaction)" auf Seite 28):

```
<Node id="interaction.builder" type="modelBuilder" palette="modeling" label="Interaction">
     <ModelBuilder miningFunctions="[classification]">
          <Algorithm value="robd" label="Robert's Algorithm" />
          <ModelingFields controlsId="modellingFields">
               <InputFields property="inputs" multiple="true" label="Inputs"
                onlyDiscrete="true" />
               <OutputFields property="target" multiple="false" label="Target"
                onlyDiscrete="true" />
          </ModelingFields>
          <ModelFields inputDirections="[in]" outputDirections="[out]">
               <ForEach var="field" inFields="outputs">
                    <AddField prefix="$I" name="${field}" fieldRef="${field}" role=
                     "predictedValue" targetField="${field}" />
                    <AddField prefix="$IP" name="${field}" storage="real" role="probability"
                     targetField="${field}">
                         <Range min="0.0" max="1.0"/>
                    </AddField>
               </ForEach>
```

```
          </ModelFields>
     </ModelBuilder>
 ...
</Node>
```

Beispiel für Feldgruppe

Dieses Beispiel entstammt dem SLRM-Knoten und fügt der Modellsignatur eine Gruppe von zwei Feldern hinzu, die die generierten Daten enthalten sollen, wenn das Modell bewertet wird. Für jeden Eingabedatensatz werden die Daten für jedes der neuen Felder mit der benutzerdefinierten Häufigkeit bewertet, was durch den Wert der Eigenschaft `max_predictions` festgelegt wird.

Die beiden neuen Felder sind:

- $S-*target* - enthält den vorhergesagten Wert des Zielfelds.
- $SC-*target* - enthält den Wahrscheinlichkeitswert für diese Vorhersage.

Um diese beiden Felder zu gruppieren, wird ihnen dieselbe Gruppenkennung zugewiesen wie bei ihrer Deklaration im Abschnitt `ModelFields`. Gruppenkennungen werden mithilfe des Attributs `group` des Elements `AddField` zugewiesen.

Die Deklaration für den Modellerstellungsknoten enthält daher:

```
<Node ... type="modelBuilder" ... >
     <ModelBuilder ... >
     ...
          <ModelFields inputDirections="[in]" outputDirections="[out]">
               <AddField prefix="$S" name="${target}" fieldRef="${target}" role=
                "predictedValue" targetField="${target}" group="[1]"/>
               <AddField prefix="$SC" name="${target}" storage="real" role="probability"
                  targetField="${target}" group="[1]">
                    <Range min="0.0" max="1.0"/>
               </AddField>
          </ModelFields>
     </ModelBuilder>
</Node>
```

Die Deklaration für den Modellanwendungsknoten enthält:

```
<Node ... type="modelApplier" ... >
...
     <OutputDataModel mode="extend">
          <ForEach var="group" from="1" to="${max_predictions}">
               <ForEach var="field" inFields="modelOutputs" container="model">
                    <AddField name="${field}" group="[${group}]" fieldRef="${field}" />
               </ForEach>
          </ForEach>
     </OutputDataModel>
</Node>
```

Das Zielfeld hat den Namen **campaign** und der Benutzer gibt 2 in das Feld ein, das der Eigenschaft `max_predictions` entspricht. Durch Ausführung des Modellerstellungsknotens werden dem Modell die folgenden Felder angehängt:

- `$S-campaign-1`
- `$SC-campaign-1`
- `$S-campaign-2`
- `$SC-campaign-2`

Modellausgabe

Das Element `ModelOutput` beschreibt ein Modellausgabeobjekt, d. h. ein Objekt, das nach der Ausführung eines Streams auf der Registerkarte "Modelle" des Managerfensters angezeigt wird.

Format

```
<ModelOutput id="ID" label="Anzeigebeschriftung" labelKey="Beschriftungsschlüssel" delegate="Java-Klasse" >
     <ModelProvider ... />
     <Properties>
          <Property ... />
           ...
     </Properties>
     <Containers ... />
     <UserInterface ... />
     <Constructors ... />
</ModelOutput>
```

Dabei gilt Folgendes:

- `id` (erforderlich) ist eine eindeutige Kennung für das generierte Modell.
- `label` (erforderlich) ist der auf der Registerkarte "Modelle" angezeigte Name des generierten Modells.
- `labelKey` gibt die Beschriftung für Lokalisierungszwecke an.

Das Element `ModelOutput` kann die in der folgenden Tabelle aufgeführten untergeordneten Elemente enthalten.

Tabelle 31. Untergeordnete Elemente der ModelOutput-Deklaration.

Untergeordnetes Element	Definition	Siehe...
`ModelProvider`	Der Container der Modellausgabe; definiert außerdem, ob die Ausgabe im PMML-Format erfolgt.	„Modellprovider” auf Seite 51
`Properties`	Die vom generierten Modell verwendeten Eigenschaften.	„Eigenschaften” auf Seite 52
`Containers`	Die Container für die generierte Modellausgabe.	„Container” auf Seite 53
`UserInterface`	Die Komponenten der Benutzerschnittstelle, in der die generierte Modellausgabe angezeigt wird, z. B. die Registerkarten "Modell" und "Einstellungen" eines Modellausgabeobjekts.	„Benutzerschnittstelle” auf Seite 54
`Constructors`	Die vom generierten Modell erstellten Objekte.	„Verwenden von Konstruktoren” auf Seite 100
`delegate`	Wird dieses Element angegeben, definiert es den Namen einer Java-Klasse, die die OutputDelegate-Schnittstelle implementiert. Eine Instanz der angegebenen Klasse wird für jede Instanz der zugeordneten Ausgabe erstellt.	

Beispiel

```
<ModelOutput id="interaction.model" label="Interaction Model">
     <Properties>
     </Properties>
     <Containers>
          <Container name="model" />
     </Containers>
     <UserInterface>
          <Tabs>
               <Tab label="Model">
```

```
                    <TextBrowserPanel container="model" textFormat="plainText" />
               </Tab>
          </Tabs>
     </UserInterface>
     <Constructors>
          <CreateModelApplier type="interaction.applier">
               <SetContainer target="model" source="model" />
          </CreateModelApplier>
     </Constructors>
</ModelOutput>
```

Erstellen von interaktiven Modellen

Mit der interaktiven Modellierungsfunktion können Sie ein Ausgabeobjekt erstellen, mit dem der Endbenutzer interagieren kann, bevor er das Modell generiert. Das interaktive Ausgabeobjekt wird auf der Registerkarte "Ausgaben" des Managerfensters angezeigt. Es enthält ein vorläufiges Dataset, mit dessen Hilfe das Modell vor der Generierung verfeinert oder vereinfacht werden kann. Für die interaktive Modellierung müssen der Spezifikation eines normalen Modellerstellungsknotens einige zusätzliche Elemente hinzugefügt werden:

- Der Abschnitt `Constructors` der `Node`-Definition muss das Element `CreateInteractiveModelBuilder` enthalten.
- Die Erweiterung muss ein dediziertes `InteractiveModelBuilder`-Element enthalten.

Die Benutzerinteraktion am vorläufigen Dataset erfolgt im sogenannten **Interaktionsfenster**, das angezeigt wird, sobald das Ausgabeobjekt erstellt ist.

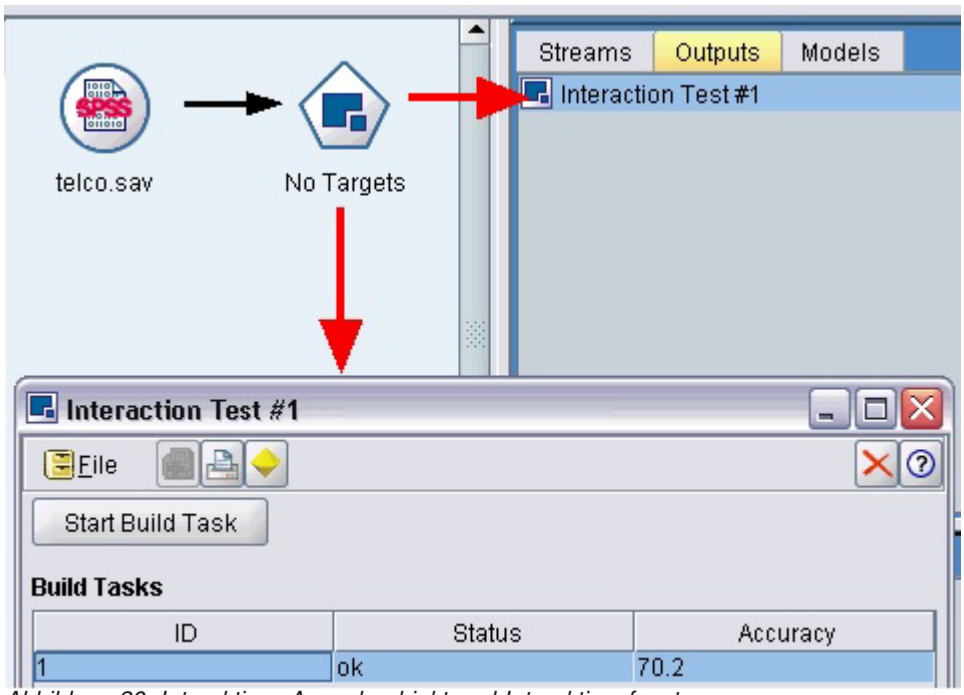

Abbildung 30. Interaktives Ausgabeobjekt und Interaktionsfenster

Die Art der Interaktion richtet sich nach dem verwendeten Algorithmus. Die Interaktion wird durch die Erweiterung implementiert. Das Interaktionsfenster wird im Abschnitt "UserInterface" des Elements `InteractiveModelBuilder` festgelegt. Es wird durch eines der folgenden Elemente definiert:

- Eine Frameklasse (frameClass) (siehe „Abschnitt 'UserInterface'" auf Seite 106), die das Fenster vollständig definiert
- Eine Anzeigenklasse (panelClass) für jede Registerkarte des Fensters; diese wird als Attribut des Erweiterungsobjektfensters angegeben (siehe „Erweiterungsobjektfenster" auf Seite 119)

Wenn das Interaktionsfenster geschlossen wurde, kann es mit einem Doppelklick auf den Objektnamen auf der Registerkarte "Ausgaben" wieder geöffnet werden.

Die Spezifikation des Interaktionsfensters muss Code enthalten, durch den das Modell generiert wird, sobald der Benutzer die Interaktion abgeschlossen hat. Im Beispiel in diesem Abschnitt ist dieser Code durch die Symbolleistenschaltfläche mit dem Goldnugget-Symbol implementiert, die einer Aktion zugeordnet ist, durch die das Modell generiert wird. Der Code wird im Abschnitt `InteractiveModelBuilder` unter „Beispiel für die interaktive Modellierung” auf Seite 92 dargestellt.

Erstellen der interaktiven Modellerstellung

Das Element `CreateInteractiveModelBuilder` beschreibt das Ausgabeobjekt, über das die Benutzerinteraktion erfolgt. Im Prinzip handelt es sich hier um nichts anderes als um eine interaktive Version des Elements `CreateModelOutput`.

Format

Dieses Element wird im Abschnitt "Execution" der Definition des Modellerstellungsknotens verwendet:

```
<Node ... type="modelBuilder" ... >
 ...
     <Execution>
      ...
         <Constructors>
          ...
             <CreateInteractiveModelBuilder ... >
               ...
             </CreateInteractiveModelBuilder>
         </Constructors>
     </Execution>
 ...
</Node>
```

Das Format des Elements selbst sieht wie folgt aus:

```
<CreateInteractiveModelBuilder type="Ausgabeobjekt-ID">
     <Condition ... ./>
     <And>
     <Or>
     <Not>
     <CreateModel type="Modell-ID" target="Container-ID" sourceFile="Containerdatei-ID" />
     <CreateDocument type="Modell-ID" target="Container-ID" sourceFile="Containerdatei-ID" />
</CreateInteractiveModelBuilder>
```

Dabei ist `type` (erforderlich) die Kennung des durch das Element `InteractiveModelBuilder` erstellten Ausgabeobjekts.

Im Abschnitt `Condition` können Sie eine oder mehrere Bedingungen festlegen. Weitere Informationen finden Sie im Thema „Bedingungen” auf Seite 72.

Hier können Sie auch komplexe Bedingungen mit den Operatoren `And`, `Or` und `Not` eingeben. Weitere Informationen finden Sie im Thema „Logische Operatoren” auf Seite 72.

In den Elementen `CreateModel` und `CreateDocument` gilt:

- `type` ist die Kennung für das zu definierende Modell bzw. Dokument.
- `type` (erforderlich) ist die Kennung des Containers für das Modell. Dieser Container wird in einem Abschnitt `ModelOutput` definiert. Weitere Informationen finden Sie im Thema „Modellausgabe” auf Seite 88.

- `sourceFile` (erforderlich) ist die Kennung einer Ausgabedatei, die während der Knotenausführung generiert wird. Diese Datei wird in einem Abschnitt `OutputFiles` definiert. Weitere Informationen finden Sie im Thema „Ausgabedateien" auf Seite 56.

Beispiel

```
<CreateInteractiveModelBuilder type="my.interaction">
     <Condition property="interactive" op="equals" value="true" />
</CreateInteractiveModelBuilder>
```

In diesem Beispiel wird bei der Ausführung des Modellerstellungsknotens, dessen Spezifikation dieses Element enthält, ein Ausgabeobjekt mit der Kennung `my.interaction` erstellt. Das Ausgabeobjekt selbst wird an einer anderen Stelle der Spezifikationsdatei durch ein `InteractiveModelBuilder`-Element definiert, das auf diese Kennung verweist. Beispiel:

```
<InteractiveModelBuilder id="my.interaction" label=...>
 ...
</InteractiveModelBuilder>
```

Interaktive Modellerstellung

Dieses Element definiert ein interaktives Ausgabeobjekt, mit dem der Endbenutzer ein Modell verfeinern oder vereinfachen kann, bevor es generiert wird.

Das Element `InteractiveModelBuilder` folgt auf die Definition eines Modellerstellungsknotens mit einem entsprechenden `CreateInteractiveModelBuilder`-Element. Weitere Informationen finden Sie im Thema „Erstellen der interaktiven Modellerstellung" auf Seite 90.

Format

Das Format des Elements `InteractiveModelBuilder` sieht wie folgt aus:

```
<Node ... type="modelBuilder" ... >
 ...
     -- Erstellung des Abschnitts "InteractiveModelBuilder" --
 ...
</Node>
 ...
<InteractiveModelBuilder id="ID" label="Anzeigebeschriftung" labelKey="Beschriftungsschlüssel">
     <Properties>
          <Property name=... />
           ...
     </Properties>
     <Containers>
          <Container name="Containername" />
     </Containers>
     <UserInterface ... />
     <Constructors ... />
</InteractiveModelBuilder>
```

Dabei gilt Folgendes:

- `id` (erforderlich) ist eine eindeutige Kennung für das generierte Modell.
- `label` (erforderlich) ist der auf der Registerkarte "Modelle" angezeigte Name des generierten Modells.
- `labelKey` gibt die Beschriftung für Lokalisierungszwecke an.

Weitere Informationen zu den Elementen `Properties`, `Containers`, `UserInterface` und `Constructors` finden Sie in den Abschnitten „Eigenschaften" auf Seite 52, „Container" auf Seite 53 sowie „Verwenden von Konstruktoren" auf Seite 100 und „Abschnitt 'UserInterface'" auf Seite 106.

Beispiel für die interaktive Modellierung

Dieses Beispiel zeigt, wie Sie einen Modellerstellungsknoten generieren, in dem die Benutzer anhand eines Kontrollkästchens angeben können, ob sie das Modell vor der Generierung anpassen möchten.

Diese Funktion können Sie an dem in diesem Release enthaltenen Beispielknoten "Interaction" testen. Weitere Informationen finden Sie im Thema „Modellerstellungsknoten (Interaction)" auf Seite 28.

Zunächst gibt der Modellerstellungsknoten eine boolesche Eigenschaft an:

```
<Node id="interaction.builder" type="modelBuilder" ... >
 ...
     <Properties>
      <Property name="interactive" valueType="boolean" />
    </Properties>
```

Im Abschnitt "UserInterface" der Knotenspezifikation enthält der Abschnitt, der die Registerkarte "Modell" definiert, einen Verweis auf diese Eigenschaft:

```
<Tab label="Model">
     <PropertiesPanel>
          <CheckBoxControl property="interactive" label="Start an interactive session" />
     </PropertiesPanel>
</Tab>
```

Im Abschnitt `CreateInteractiveModelBuilder` desselben Knotens wird die Einstellung der Eigenschaft untersucht. Falls diese `true` lautet, wird ein interaktives Ausgabeobjekt erstellt:

```
<CreateInteractiveModelBuilder type="my.interaction">
     <Condition property="interactive" op="equals" value="true" />
</CreateInteractiveModelBuilder>
```

Das referenzierte Ausgabeobjekt ist im Abschnitt `InteractiveModelBuilder` der Erweiterung definiert:

```
<InteractiveModelBuilder id="my.interaction" label="Interaction Test">
     <Properties>
     </Properties>
     <Containers>
     </Containers>
     <UserInterface actionHandler="ui.InteractionHandler">
          <Controls>
               <ToolbarItem action="generateModel" showLabel="false" />
          </Controls>
          <Tabs>
               <Tab label="Model">
                    <ExtensionObjectPanel id="model.panel" panelClass=
                     "ui.SampleInteractionPanel" />
               </Tab>
               <Tab label="Generic">
                    <ExtensionObjectPanel id="generic.panel" panelClass=
                     "ui.GenericInteractionPanel" />
               </Tab>
          </Tabs>
     </UserInterface>
</InteractiveModelBuilder>
```

Die Aktion, durch die das Modell generiert wird, wird durch die im Element `ToolbarItem` definierte Symbolleistenschaltfläche gesteuert.

Achten Sie besonders auf die Verwendung des Attributs `panelClass` im Element `ExtensionObjectPanel`. Hiermit wird die Java-Klasse festgelegt, die die Benutzerschnittstelle der einzelnen Registerkarten des Interaktionsfensters steuert.

Automatisierte Modellierung

IBM SPSS Modeler stellt als Standard eine Gruppe von Ensemblemodellierungsknoten bereit, zu denen unter anderem die Knoten "Automatisches Klassifikationsmerkmal", "Autom. Cluster" und "Autonumerisch" gehören. Mit diesen Knoten kann die gleichzeitige Erstellung verschiedener Modelle automatisiert werden. Der Endbenutzer kann die Ergebnisse anschließend vergleichen und das beste Modell für seine Daten auswählen. Für die Festlegung, dass ein durch das `ModelBuilder`-Element angegebenes Modell von einem dieser Ensemble-Knoten verwendet werden soll, stellt CLEF das `AutoModeling`-Element bereit.

Das Format des `AutoModeling`-Elements sieht wie folgt aus:

```
<AutoModeling enabledByDefault="true_false">
     <SimpleSettings ... />
     <ExpertSettings ... />
     <Constraint ... />
     <Constraint ... />
      ...
</AutoModeling>
```

Dabei gibt `enabledByDefault` an, ob das Modell standardmäßig zur Verwendung im Ensemblemodellierungsknoten aktiviert ist (d. h. die Spalte **Verwenden?** ist für dieses Modell standardmäßig aktiviert). Wenn dieses Attribut weggelassen wird, wird der Wert `true` angenommen.

Auf der Registerkarte "Experten" im Dialogfeld eines Ensemblemodellierungsknotens werden die Modelle aufgelistet, die vom Benutzer erstellt werden können.

Wenn der Benutzer im Feld **Modellparameter** eines bestimmten Modells auf **Angeben** klickt, wird das Dialogfeld "Algorithmuseinstellungen" geöffnet, in dem er die Optionen für den Modelltyp auswählen kann.

Das Dialogfeld "Algorithmuseinstellungen" enthält entsprechend den Ausführungsmodi des Modellierungsknotens die Registerkarten "Einfach" und "Experten". Der Inhalt der Registerkarten "Einfach" und "Experten" wird durch die in den nachfolgenden Abschnitten beschriebenen Elemente `SimpleSettings` und `ExpertSettings` gesteuert.

Darüber hinaus ermöglichen `Constraint`-Elemente die Angabe von Bedingungen, die es dem Endbenutzer erlauben, Parameter im Dialogfeld "Algorithmuseinstellungen" zu bearbeiten, bzw. unter denen die Bearbeitung eingeschränkt ist. Weitere Informationen finden Sie im Thema „Nebenbedingungen” auf Seite 96.

Einigen Parametern im Dialogfeld "Algorithmuseinstellungen" können mehrere Werte zugewiesen sein. In diesem Fall versucht der Ensemble-Knoten, Modelle für alle möglichen Kombinationen dieser Parameterwerte zu erstellen. Gibt der Benutzer bei einem verallgemeinerten linearen Modell z. B. zwei Verteilungen (normale Verteilung und Gammaverteilung) und drei Linkfunktionen (Identität, Log und Exponent) an, dann versucht der Knoten "Autonumerisch" sechs verallgemeinerte lineare Modelle zu erstellen - eines für jede mögliche Kombination dieser Parameter.

Einfache Einstellungen

Das Element `SimpleSettings` bestimmt, welche Parameter im Dialogfeld "Algorithmuseinstellungen" dieses Modells eines Ensemblemodellierungsknotens auf der Registerkarte "Einfach" angezeigt werden. Weitere Informationen finden Sie im Thema „Automatisierte Modellierung”.

Format

```
<SimpleSettings>
     <PropertyGroup label="Gruppenname" labelKey="Ressourcenschlüssel"
      properties="[Eigenschaftsname1
      Eigenschaftsname2 ...]"/>
     <PropertyGroup ... />
</SimpleSettings>
```

In einem PropertyGroup-Element (mindestens eines ist erforderlich):

label ist eine Anzeigebeschriftung für die Eigenschaftsgruppe, das als Unterbeschriftung in das Dialogfeld oberhalb des ersten Parameters in der Gruppe eingefügt wird.

labelKey gibt die Beschriftung für Lokalisierungszwecke an. Wenn weder label noch labelKey verwendet wird, wird keine Unterbeschriftung eingefügt.

properties (erforderlich) ist eine Liste mit einer oder mehreren Eigenschaften, die auf der Registerkarte angezeigt werden sollen. Der Wert von *Eigenschaftsname1*, *Eigenschaftsname2* usw. ist der Wert des Attributs name des Elements Property, in dem diese Eigenschaft definiert wird. Weitere Informationen finden Sie im Thema „Eigenschaften” auf Seite 52.

Beispiel

```
<SimpleSettings>
     <PropertyGroup properties="[method]"/>
</SimpleSettings>
```

Dieses Beispiel aus dem Diskriminanzknoten legt fest, dass für dieses Modell des relevanten Ensemblemodellierungsknotens (in diesem Fall der Knoten "Automatisches Klassifikationsmerkmal") auf der Registerkarte "Einfach" des Dialogfelds "Algorithmuseinstellungen" nur der Parameter **Methode** angezeigt wird. Da kein label- oder labelKey-Attribut angegeben ist, enthält das Dialogfeld für den Parameter keine Unterbeschriftung.

Experteneinstellungen

Das Element ExpertSettings bestimmt, welche Parameter im Dialogfeld "Algorithmuseinstellungen" dieses Modells eines Ensemblemodellierungsknotens auf der Registerkarte "Experten" angezeigt werden. Weitere Informationen finden Sie im Thema „Automatisierte Modellierung” auf Seite 93.

Format

```
<ExpertSettings>
     <Condition ... />
     <PropertyGroup label="Gruppenname" labelKey="Ressourcenschlüssel"
      properties="[Eigenschaft1 Eigenschaft2 ...]"/>
     <PropertyGroup ... />
      ...
</ExpertSettings>
```

Das Element Condition gibt eine Bedingung an, die, sofern sie true ist, bewirkt, dass die durch die nachfolgenden PropertyGroup-Elemente angegebenen Parameter aktiviert werden. Weitere Informationen finden Sie im Thema „Bedingungen” auf Seite 72.

In einem PropertyGroup-Element (mindestens eines ist erforderlich):

label ist eine Anzeigebeschriftung für die Eigenschaftsgruppe, das als Unterbeschriftung in das Dialogfeld oberhalb des ersten Parameters in der Gruppe eingefügt wird.

labelKey gibt die Beschriftung für Lokalisierungszwecke an. Wenn weder label noch labelKey verwendet wird, wird keine Unterbeschriftung eingefügt.

properties (erforderlich) ist eine Liste mit einer oder mehreren Eigenschaften, die auf der Registerkarte angezeigt werden sollen. Der Wert von *Eigenschaftsname1*, *Eigenschaftsname2* usw. ist der Wert des Attributs name des Elements Property, in dem diese Eigenschaft definiert wird. Weitere Informationen finden Sie im Thema „Eigenschaften” auf Seite 52.

Beispiele

Im folgenden Beispiel ist der Parameter **Modus** auf der Registerkarte "Experten" des Dialogfelds "Algorithmuseinstellungen" zunächst auf **Einfach** gesetzt.

Abbildung 31. Inaktivierte Experteneinstellungen

Der folgende Code legt fest, dass die übrigen Parameter der Registerkarte "Experten" nur dann aktiviert werden, wenn der Benutzer den Parameter **Mode** auf **Expert** setzt:

```
<ExpertSettings>
     <Condition property="mode" op="equals" value="Expert"/>
     <PropertyGroup properties="[mode prior_probabilities covariance_matrix]"/>
      ...
</ExpertSettings>
```

Sobald die Einstellung **Modus** auf **Experten** gesetzt wird, werden die beiden Parameter der Eigenschaftengruppe aktiviert.

Abbildung 32. Aktivierte Experteneinstellungen

Das nächste Beispiel veranschaulicht die Verwendung von Beschriftungen in der Eigenschaftengruppe:

```
<ExpertSettings>
      ...
     <PropertyGroup labelKey="automodel.stepping_criteria_options"
      properties="[stepwise_method V_to_enter criteria]"/>
      ...
</ExpertSettings>
```

Hier steuert das Element `PropertyGroup` die in der folgenden Abbildung hervorgehobenen Parameter.

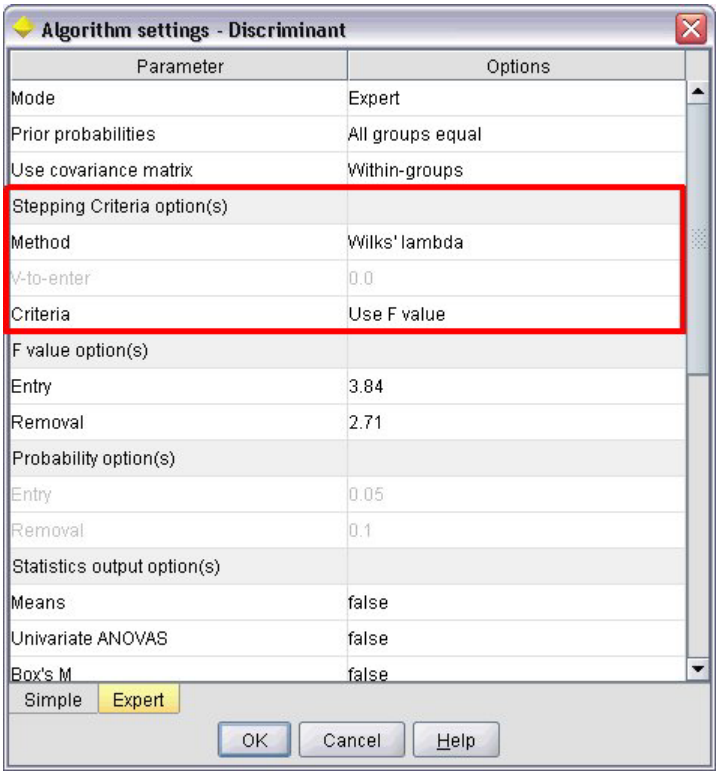

Abbildung 33. Inaktivierte Experteneinstellungen

Das Attribut `labelKey` veranlasst CLEF, den Anzeigetext für die Unterbeschriftung der Eigenschaftengruppe aus dem entsprechenden Eintrag der Eigenschaftendatei der Erweiterung abzurufen:

```
automodel.stepping_criteria_options=Schrittkriterien - Option(en)
```

Weitere Informationen finden Sie im Thema „Eigenschaftendateien” auf Seite 170.

Nebenbedingungen

Das Element `Constraint` bestimmt die Bedingungen, unter denen die Bearbeitung der Parameter im Dialogfeld "Algorithmuseinstellungen" eines Modells in einem Ensemblemodellierungsknoten erlaubt ist bzw. unter denen die Bearbeitung eingeschränkt ist. Bestimmte Parameter können z. B. inaktiviert sein, wenn der Endbenutzer diese zurzeit nicht bearbeiten darf.

Format

```
<Constraint property="Eigenschaftsname" singleSelection="true_false">
     <Condition property="Eigenschaftsname" op="Operator" value="Wert"/>
     <And ... />
     <Or ... />
     <Not ... />
</Constraint>
```

Dabei gilt Folgendes:

- `property` (erforderlich) gibt den Parameter an, der bearbeitet werden darf bzw. dessen Bearbeitung eingeschränkt wird. *Eigenschaftsname* ist der Wert des Attributs `name` des `Property`-Elements, in dem die zu diesem Parameter gehörige Eigenschaft definiert ist. Weitere Informationen finden Sie im Thema „Eigenschaften” auf Seite 52.
- `singleSelection` legt fest, ob der Endbenutzer mehrere der für einen Parameter zur Verfügung stehenden Werte auswählen darf. Wenn diese Einstellung auf `true` gesetzt ist, darf nur ein Wert ausgewählt werden, selbst wenn das Feld "Optionen" dieses Parameters im Dialogfeld "Algorithmuseinstellungen"

mehrere Werte enthält. Wenn diese Einstellung auf `false` gesetzt ist (Standardwert), kann der Benutzer, wie im nachfolgenden Beispiel gezeigt, ein oder mehrere der verfügbaren Werte auswählen.

Das Element `Condition` legt die tatsächliche Einschränkung fest. Weitere Informationen finden Sie im Thema „Bedingungen” auf Seite 72.

Die Elemente `And`, `Or` und `Not` können zur Angabe zusammengesetzter Bedingungen verwendet werden. Weitere Informationen finden Sie im Thema „Logische Operatoren” auf Seite 72.

Beispiel

Dieses Beispiel stammt aus der Spezifikationsdatei des verallgemeinerten linearen Knotens. Selbst im Expertenmodus sind einige Parameter standardmäßig nicht aktiviert.

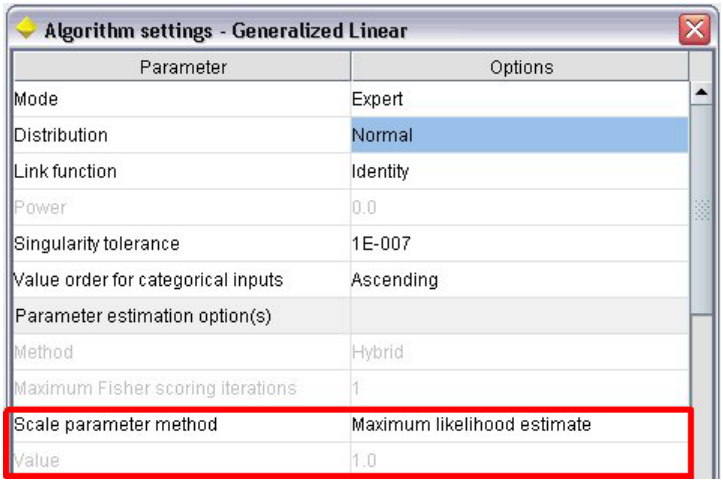

Abbildung 34. Auswirkung einer Einschränkung - inaktivierter Parameter

Die Einschränkung legt die Bedingung fest, unter der der Parameter "Wert" aktiviert ist:

```
<Constraint property="scale_value">
     <And>
          <Condition property="scale_method" op="equals" value="FixedValue"/>
          <Condition property="distribution" op="in" value="[IGAUSS GAMMA NORMAL]"/>
     </And>
</Constraint>
```

Damit der Parameter **Wert** aktiviert wird, muss der Parameter **Skalenparametermethode** (festgelegt durch die Eigenschaft `scale_method`) auf **Fester Wert** gesetzt sein und **Verteilung** muss auf **Normal**, **Invers normal** oder **Gamma** gesetzt sein.

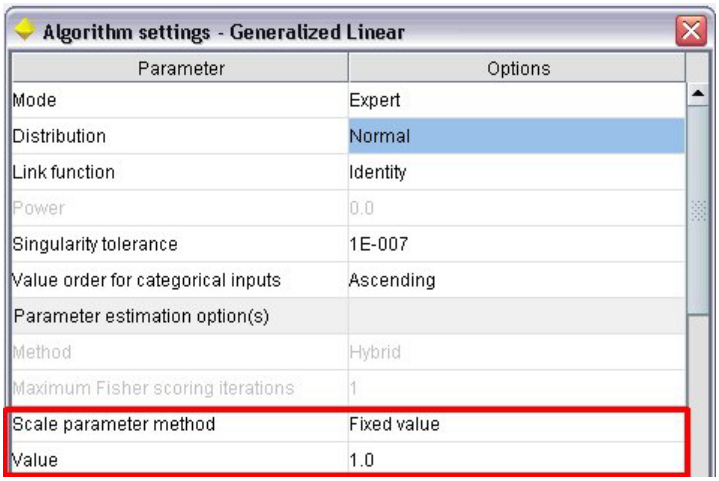

Abbildung 35. Auswirkung einer Einschränkung - inaktivierter Parameter

Anwenden von Modellen

Das Anwenden eines Modells bedeutet, ein generiertes Modell für das Scoring von Daten zu verwenden. Dabei werden die aus der Modellerstellung gewonnenen Informationen zur Erstellung von Vorhersagen für neue Datensätze verwendet. In IBM SPSS Modeler verwenden Sie dazu einen Modellanwendungsknoten. Weitere Informationen finden Sie im Thema „Modellanwendungsknoten" auf Seite 12.

Die Definition eines Modellanwendungsknotens in der Spezifikationsdatei bildet das Framework für die Anwendung eines generierten Modells. In IBM SPSS Modeler erstellen Sie eine Instanz eines Modellanwendungsknotens, indem Sie das Symbol des Modellausgabeobjekts von der Registerkarte "Modelle" des Managerfensters auf den Streamerstellungsbereich ziehen. Ohne Definition des Modellanwendungsknotens würde die Ausführung des Modellerstellungsknotens lediglich ein nicht verfeinertes Modell ergeben, das dem Streamerstellungsbereich nicht hinzugefügt werden kann.

Für die Definition eines Modellanwendungsknotens müssen Sie dem `Node`-Element folgende Elemente hinzufügen:

- Das Attribut `type="modelApplier"`
- Ein untergeordnetes Element `Constructors` mit einem Element `CreateModelOutput` (siehe „Verwenden von Konstruktoren" auf Seite 100)

Informationen zum Format einer `Node`-Elementspezifikation finden Sie in „Knoten" auf Seite 48.

Erstellen von Dokumenten

Für die Definition eines Dokumenterstellungsknotens müssen Sie dem `Node`-Element folgende Elemente hinzufügen:

- Das Attribut `type="documentBuilder"`
- Das untergeordnete Element `DocumentBuilder`

Die Erweiterung muss für die Dokumenterstellung auch über ein `DocumentOutput`-Element verfügen, das das generierte Dokument beschreibt. Weitere Informationen finden Sie im Thema „Modellausgabe" auf Seite 88.

Informationen zum Format einer `Node`-Elementspezifikation finden Sie in „Knoten" auf Seite 48.

Dokumenterstellung (DocumentBuilder)

Das DocumentBuilder-Element legt das Verhalten eines Dokumenterstellungsknotens fest. Die Definition muss das untergeordnete Element DocumentGeneration enthalten. Dieses Element legt fest, welcher Registerkarte des Dialogfelds des Dokumenterstellungsknotens die Steuerelemente für die Dokumentgenerierung hinzugefügt werden. Die Steuerelemente selbst werden im Abschnitt zur Benutzerschnittstelle definiert (siehe Kapitel 6, „Erstellen von Benutzerschnittstellen", auf Seite 105).

Format

```
<DocumentBuilder>
     <DocumentGeneration controlsId="Steuerelementekennung />
</DocumentBuilder>
```

Dabei ist controlsId (erforderlich) die von den Systemsteuerelementen verwendete Kennung, die angibt, wo die Steuerelemente für die Dokumentgenerierung angezeigt werden.

Beispiel

```
<DocumentBuilder>
     <DocumentGeneration controlsId="1"/>
</DocumentBuilder>
```

Dokumentausgabe

Das Element DocumentOutput beschreibt ein Dokumentausgabeobjekt, d. h. ein Objekt, das nach der Ausführung eines Streams auf der Registerkarte "Ausgaben" des Managerfensters angezeigt wird.

Format

```
<DocumentOutput id="ID" label="Anzeigebeschriftung" labelKey="Beschriftungsschlüssel" delegate="Java-Klasse"
     <Properties>
          <Property ... />
           ...
     </Properties>
     <Containers>
          <Container ... />
           ...
     </Containers>
     <UserInterface ... />
     <Constructors ... />
</DocumentOutput>
```

Dabei gilt Folgendes:

- id (erforderlich) ist eine eindeutige Kennung für das generierte Dokument.
- label (erforderlich) ist der auf der Registerkarte "Ausgaben" angezeigte Name des generierten Dokuments.
- labelKey gibt die Beschriftung für Lokalisierungszwecke an.

Das Element DocumentOutput kann die in der folgenden Tabelle aufgeführten untergeordneten Elemente enthalten.

Tabelle 32. Untergeordnete Elemente der DocumentOutput-Deklaration.

Untergeordnetes Element	Definition	Siehe...
Properties	Die vom generierten Dokument verwendeten Eigenschaften.	„Eigenschaften" auf Seite 52
Containers	Die Container für die generierte Dokumentausgabe.	„Container" auf Seite 53

Tabelle 32. Untergeordnete Elemente der DocumentOutput-Deklaration (Forts.).

Untergeordnetes Element	Definition	Siehe...
`UserInterface`	Die Komponente der Benutzerschnittstelle, in der die generierte Dokumentausgabe angezeigt wird.	„Benutzerschnittstelle" auf Seite 54
`Constructors`	Die vom generierten Dokument erstellten Objekte.	„Verwenden von Konstruktoren"
`delegate`	Wird dieses Element angegeben, definiert es den Namen einer Java-Klasse, die die OutputDelegate-Schnittstelle implementiert. Eine Instanz der angegebenen Klasse wird für jede Instanz der zugeordneten Ausgabe erstellt.	

Beispiel

```
<DocumentOutput id="webstatusreport">
     <Containers>
          <Container name="webstatusreportdata" />
     </Containers>
     <UserInterface>
          <Tabs>
               <Tab label="Advanced" labelKey="advancedTab.LABEL" >
                    <TextBrowserPanel container="webstatusreportdata" textFormat="html" />
               </Tab>
          </Tabs>
     </UserInterface>
</DocumentOutput>
```

Verwenden von Konstruktoren

`Constructors`-Elemente können an zwei Stellen der Spezifikationsdatei eingefügt werden:

- Im Abschnitt "Execution" einer Knotendefinition (bei Modell- oder Dokumentausgabeobjekten)
- In einer ModelOutput-Definition (bei Modellanwendungsknoten)

Ein Knoten kann nur ein Ausgabeobjekt generieren. Diese Einschränkung wurde aus Konsistenzgründen mit bereits bestehenden Knoten und den Scripting- und Client-API-Schnittstellen für diese Knotentypen beibehalten.

Format

Im Abschnitt "Execution" sieht das Format des `Constructors`-Elements wie folgt aus:

```
<Constructors>
     <CreateModelOutput ... />
      ...
     <CreateDocumentOutput ... />
      ...
     <CreateInteractiveModelBuilder ... />
      ...
</Constructors>
```

In einer ModelOutput-Definition sieht das Format wie folgt aus:

```
<Constructors>
     <CreateModelApplier ... />
</Constructors>
```

Erstellen der Modellausgabe

In diesem Abschnitt wird beschrieben, wie ein Modellausgabeobjekt für die Registerkarte "Modelle" bzw. ein Dokumentausgabeobjekt für die Registerkarte "Ausgaben" erstellt wird.

Format

```
<CreateModelOutput type="Ausgabeobjekt-ID">
     <Condition ... ./>
     <And>
     <Or>
     <Not>
     <CreateModel type="Modell-ID" target="Container-ID" sourceFile="Containerdatei-ID" />
      ...
     <CreateDocument type="Dokument-ID" target="Container-ID" sourceFile="Containerdatei-ID" />
      ...
</CreateModelOutput>
```

Im Element `CreateModelOutput`:

- `type` (erforderlich) ist die Kennung eines Modellausgabeobjekts, das in einem Abschnitt `ModelOutput` definiert wird. Weitere Informationen finden Sie im Thema „Modellausgabe" auf Seite 50.

Im Abschnitt `Condition` können Sie eine oder mehrere Bedingungen festlegen. Weitere Informationen finden Sie im Thema „Bedingungen" auf Seite 72.

Hier können Sie auch komplexe Bedingungen mit den Operatoren `And`, `Or` und `Not` eingeben. Weitere Informationen finden Sie im Thema „Logische Operatoren" auf Seite 72.

In den Elementen `CreateModel` und `CreateDocument` gilt:

- `type` ist die Kennung für das zu definierende Modell bzw. Dokument.
- `type` (erforderlich) ist die Kennung des Containers für das Modell. Dieser Container wird in einem Abschnitt `ModelOutput` definiert. Weitere Informationen finden Sie im Thema „Modellausgabe" auf Seite 88.
- `sourceFile` (erforderlich) ist die Kennung einer Ausgabedatei, die während der Knotenausführung generiert wird. Diese Datei wird in einem Abschnitt `OutputFiles` definiert. Weitere Informationen finden Sie im Thema „Ausgabedateien" auf Seite 56.

Beispiel

```
<Constructors>
     <CreateModelOutput type="naivebayes">
          <CreateModel type="naivebayes_model" target="model" sourceFile="pmml"/>
          <CreateDocument type="html_output" target="advanced_output" sourceFile="htmloutput" />
          <CreateDocument type="zip_outputType" target="zip_output" sourceFile="zipoutput" />
     </CreateModelOutput>
</Constructors>
```

Erstellen der Dokumentausgabe

Dieses Element wird im Abschnitt "Execution" der Definition eines Dokumenterstellungsknotens verwendet. Es gibt das zu erstellende Dokumentausgabeobjekt an.

Format

```
<CreateDocumentOutput type="Ausgabeobjekt-ID">
     <Condition ... ./>
     <And>
     <Or>
     <Not>
     <CreateModel type="Modell-ID" target="Container-ID" sourceFile="Containerdatei-ID" />
```

```
     ...
     <CreateDocument type="Dokument-ID" target="Container-ID" sourceFile="Containerdatei-ID" />
     ...
</CreateModelOutput>
```

Dabei ist `type` (erforderlich) die Kennung eines Dokumentausgabeobjekts, das in einem Abschnitt `DocumentOutput` definiert wird. Weitere Informationen finden Sie im Thema „Dokumentausgabe" auf Seite 51.

Im Abschnitt `Condition` können Sie eine oder mehrere Bedingungen festlegen. Weitere Informationen finden Sie im Thema „Bedingungen" auf Seite 72.

Hier können Sie auch komplexe Bedingungen mit den Operatoren `And`, `Or` und `Not` eingeben. Weitere Informationen finden Sie im Thema „Logische Operatoren" auf Seite 72.

In den Elementen `CreateModel` und `CreateDocument` gilt:

- `type` ist die Kennung für das zu definierende Modell bzw. Dokument.
- `type` (erforderlich) ist die Kennung des Containers für das Modell. Dieser Container wird in einem Abschnitt `ModelOutput` definiert. Weitere Informationen finden Sie im Thema „Modellausgabe" auf Seite 88.
- `sourceFile` (erforderlich) ist die Kennung einer Ausgabedatei, die während der Knotenausführung generiert wird. Diese Datei wird in einem Abschnitt `OutputFiles` definiert. Weitere Informationen finden Sie im Thema „Ausgabedateien" auf Seite 56.

Beispiel

```
<Constructors>
     <CreateDocumentOutput type="webstatusreport">
          <CreateDocument type="webstatusreport" target="webstatusreportdata"
               sourceFile="webstatusreport_output_file" />
     </CreateDocumentOutput>
</Constructors>
```

Erstellen der interaktiven Modellerstellung

Dieses Element wird im Abschnitt "Execution" der Definition eines interaktiven Modellerstellungsknotens verwendet. Es gibt das Ausgabeobjekt an, mit dem der Benutzer interagiert. Weitere Informationen finden Sie im Thema „Erstellen der interaktiven Modellerstellung" auf Seite 90.

Erstellen des Modellanwenders

Dieses Element wird in einem `Constructors`-Element im Abschnitt "ModelOutput" der Definition eines Modellerstellungsknotens verwendet (siehe „Modellausgabe" auf Seite 88). Das Element `CreateModelApplier` legt fest, wie der Modellanwendungsknoten erstellt wird, wenn ein vom Modellerstellungsknoten generiertes Modellausgabeobjekt auf den Erstellungsbereich gezogen wird.

Format

```
<CreateModelApplier type="Anwendungsknoten-ID">
     <Condition ... ./>
     <And>
     <Or>
     <Not>
     <CreateModel type="Modell-ID" target="Container-ID" sourceFile="Containerdatei-ID" />
     ...
     <CreateDocument type="Dokument-ID" target="Container-ID" sourceFile="Containerdatei-ID" />
     ...
</CreateModelApplier>
```

Im Element CreateModelApplier:

- type (erforderlich) ist die Kennung des zu erstellenden Modellanwendungsknotens. Der Knoten selbst wird erst später im Element Node ... type="modelApplier" dieser Datei definiert.

Im Abschnitt Condition können Sie eine oder mehrere Bedingungen festlegen. Weitere Informationen finden Sie im Thema „Bedingungen" auf Seite 72.

Hier können Sie auch komplexe Bedingungen mit den Operatoren And, Or und Not eingeben. Weitere Informationen finden Sie im Thema „Logische Operatoren" auf Seite 72.

In den Elementen CreateModel und CreateDocument gilt:

- type ist die Kennung für das zu definierende Modell bzw. Dokument.
- type (erforderlich) ist die Kennung des Containers für das Modell. Dieser Container wird in einem Abschnitt ModelOutput definiert. Weitere Informationen finden Sie im Thema „Modellausgabe" auf Seite 88.
- sourceFile (erforderlich) ist die Kennung einer Ausgabedatei, die während der Knotenausführung generiert wird. Diese Datei wird in einem Abschnitt OutputFiles definiert. Weitere Informationen finden Sie im Thema „Ausgabedateien" auf Seite 56.

Beispiel

Im folgenden Beispiel enthält das Element CreateModelApplier einen Vorwärtsverweis auf den Modellanwendungsknoten mit dem Namen myapplier. Dieser Knoten wird im nachfolgenden Node-Element definiert.

```
<ModelOutput>
     <Constructors>
          <CreateModelApplier type="myapplier"></CreateModelApplier>
     </Constructors>
</ModelOutput>
<Node id="myapplier" type="modelApplier">
 ...
</Node>
```

Kapitel 6. Erstellen von Benutzerschnittstellen

Informationen zu Benutzerschnittstellen

Wenn Sie einen neuen CLEF-Knoten hinzufügen, müssen Sie definieren, wie der Endbenutzer mit dem Knoten und durch die Erweiterung angegebenen Modellausgaben, Dokumentausgaben oder Anwendungsknoten interagieren soll. Die Benutzerschnittstelle für die einzelnen Objekte ist im Abschnitt `UserInterface` der Spezifikationsdatei für die Erweiterung definiert. Dieses Kapitel bietet eine detaillierte Beschreibung des Abschnitts `UserInterface`.

Hinweis: In einer einzelnen Spezifikationsdatei können mehrere `UserInterface`-Abschnitte vorhanden sein. Dies hängt davon ab, welche Objekttypen in der Datei definiert sind.

Die Benutzerschnittstelle für ein CLEF-Objekt kann folgende Elemente enthalten:

- Menüeintrag
- Paletteneintrag
- Symbole
- Ein oder mehrere Dialogfenster
- Ein oder mehrere Ausgabefenster

Menüeinträge und **Paletteneinträge** ermöglichen den Zugriff auf das Objekt aus dem Menüsystem bzw. aus den Knotenpaletten von IBM SPSS Modeler. **Symbole** kennzeichnen Objekte in den Menüs, im Erstellungsbereich und in den verschiedenen Knotenpaletten. **Dialogfenster** enthalten Registerkarten und Steuerelemente, mit denen Benutzer verschiedene Optionen festlegen können, bevor ein Stream ausgeführt wird, und um optional eine Ausgabe festzulegen, die nach Abschluss der Ausführung erfolgt. **Ausgabefenster** werden verwendet, um Modellausgaben (wie Suchergebnisse, die aus der Anwendung eines Modells auf ein Dataset stammen) oder Dokumentausgaben (wie Diagramme) anzuzeigen.

CLEF bietet Ihnen die Möglichkeit, zu den mit IBM SPSS Modeler gelieferten Standardobjekten neue Objekttypen hinzuzufügen und unterstützt einen durchgängigen Ansatz zur Definition der Benutzerschnittstelle für diese neuen Objekte. Richtlinien zum Erstellen von Symbolen und Dialogfeldern in CLEF finden Sie in „Erstellen von Knotensymbolen" auf Seite 15 und „Erstellen von Dialogfeldern" auf Seite 19.

Symbole sind Grafikdateien, mit denen ein bestimmter Knoten identifiziert wird und die im Rahmen der geometrischen Formen dargestellt werden, mit denen der Knotentyp identifiziert wird.

Dialogfelder und **Ausgabefenster** haben folgende Merkmale:

- Titelleiste mit Miniatursymbol und Objekttitel
- Menuleiste mit:
  - Menüs
  - Objektspezifischen Aktionsschaltflächen
  - Allgemeinen Aktionsschaltflächen (z. B. zum Maximieren oder für die Hilfe)
- Hauptinhaltsbereich
- Mehrere Registerkarten, mit denen Komponenten der Benutzerschnittstelle in logischen Gruppen angeordnet werden
- Größenanpassung

Eine Registerkarte ändert den Hauptinhaltsbereich eines Fensters auf verschiedene Weisen. Bei einem Dialogfenster zeigen unterschiedliche Registerkarten verschiedene Sätze von Steuerelementen für die Ob-

jekteigenschaften an. Die Steuerelemente können geändert werden und die Ergebnisse werden auf das zugrunde liegende Objekt angewendet, wenn der Benutzer auf die Schaltflächen **Anwenden** oder **OK** klickt.

Bei einem Ausgabefenster können Benutzer mit Registerkarten verschiedene mit der Ausgabe zusammenhängende Aktionen durchführen, wie die Anzeige einer Zusammenfassung der Ergebnisse oder das Hinzufügen von Anmerkungen.

Abschnitt 'UserInterface'

Die Benutzerschnittstelle für ein Objekt wird innerhalb der Objektdefinition in der Spezifikationsdatei in einem `UserInterface`-Element deklariert. In einer Spezifikationsdatei können mehrere `UserInterface`-Elemente vorhanden sein - je nachdem, wie viele Objekte (z. B. Modellierungsknoten, Modellausgabeobjekte, Modellanwendungsknoten) in der Datei definiert sind.

In jedem Abschnitt `UserInterface` können folgende Definitionen vorgenommen werden:

- Symbole für die Anzeige im Erstellungsbereich oder in Paletten
- Steuerelemente (benutzerdefinierte Menü- und Symbolleistenelemente), die in Dialogfeldern oder Ausgabefenstern angezeigt werden
- Registerkarten, die Sätze von Eigenschaftssteuerelementen definieren (für Dialogfelder oder Ausgabefenster)

Hinweis: In den folgenden Elementdefinitionen (normalerweise durch die Überschrift **Format** gekennzeichnet) sind Elementattribute und untergeordnete Elemente optional, sofern sie nicht als "(erforderlich)" gekennzeichnet sind. Die vollständige Syntax für alle Elemente finden Sie in „CLEF-XML-Schema”, auf Seite 207.

Für jede Benutzerschnittstelle wird die durchzuführende Verarbeitung festgelegt. Dies erfolgt entweder mittels eines Aktionshandlers oder eines Frameklassenattributs, wobei beide optional sind. Wenn weder ein Aktionshandler noch ein Frameklassenattribut angegeben ist, ist die durchzuführende Verarbeitung an anderer Stelle in der Datei angegeben.

Format

Das Basisformat des Elements `UserInterface` sieht wie folgt aus:

```
<UserInterface>
     <Icons>
          <Icon ... />
           ...
     </Icons>
     <Controls>
          <Menu ... />
          <MenuItem ... />
           ...
          <ToolbarItem ... />
           ...
     </Controls>
     <Tabs>
          <Tab ... />
           ...
     </Tabs>
</UserInterface>
```

Ein **Benutzerschnittstellendelegate für Erweiterungsobjekte** ist einem Knotendialog oder einem Modell- oder Dokumentausgabefenster zugeordnet und ermöglicht es der Erweiterung, benutzerdefinierte Aktionen zu verarbeiten, die im Dialog aufgerufen wurden. Der Benutzerschnittstellendelegate gibt die von ei-

ner Erweiterung bereitgestellte Java-Klasse an, die zur selben Zeit wie das IBM SPSS Modeler-Fenster erstellt wird und eine Implementierung der Klasse `ExtensionObjectUIDelegate` darstellt. Weitere Informationen finden Sie im Thema „Clientseitige API-Klassen" auf Seite 178.

Das Format für ein Benutzerschnittstellendelegate ist mit dem Basisformat identisch, mit Ausnahme der ersten Zeile:

```
<UserInterface uiDelegate="Java-Klasse" >
      ...
</UserInterface>
```

Dabei ist `uiDelegate` der Name der Java-Klasse des Benutzerschnittstellendelegates.

Erweiterungen dürfen nicht davon ausgehen, dass die Benutzerschnittstelle aufgerufen werden kann. Erweiterungen können z. B. in Umgebungen ohne GUI, etwa im Stapelmodus oder in Anwendungsservern, ausgeführt werden.

Ein **Aktionshandler** wird verwendet, wenn benutzerdefinierte Aktionen einem IBM SPSS Modeler-Standardfenster hinzugefügt werden. Er ähnelt dem **Benutzerschnittstellendelegate für Erweiterungsobjekte**, obwohl ein Aktionshandler auch verwendet werden kann, wenn kein Erweiterungsknoten oder Ausgabefenster vorhanden ist, z. B. wenn ein neues Tool definiert wird, das direkt im IBM SPSS Modeler-Hauptfenster aufgerufen werden kann. Der Aktionshandler gibt die Java-Klasse an, die aufgerufen wird, wenn ein Benutzer in einem Knotendialogfeld, einem Modellausgabefenster oder einem Dokumentausgabefenster eine benutzerdefinierte Menüoption oder Symbolleistenschaltfläche auswählt. Es handelt sich um eine Implementierung einer `ExtensionObjectFrame`-Klasse oder einer `ActionHandler`-Klasse. In beiden Fällen werden die Standardfensterkomponenten (Standardmenüs, Registerkarten und Symbolleistenschaltflächen) automatisch eingeschlossen. Weitere Informationen finden Sie im Thema „Clientseitige API-Klassen" auf Seite 178.

Das Format für einen Aktionshandler ist wiederum mit dem Basisformat identisch:

```
<UserInterface actionHandler="Java-Klasse" >
      ...
</UserInterface>
```

Dabei ist `actionHandler` der Name der Java-Klasse des Aktionshandlers.

Es wird empfohlen, einen **Aktionhandler** zu verwenden, wenn die Erweiterung ein Tool hinzufügt, das direkt im IBM SPSS Modeler-Hauptfenster aufgerufen werden kann, und einen **Benutzerschnittstellendelegate für Erweiterungsobjekte** bei Fenstern zu verwenden, die Knoten und Ausgaben zugeordnet sind, die von der Erweiterung definiert sind. Der Grund ist, dass der Benutzerschnittstellendelegate einen einfacheren Zugriff auf das zugrundeliegende Knoten- oder Ausgabeobjekt bietet. Eine Erweiterung darf nicht `uiDelegate` und `actionHandler` in demselben Element `UserInterface` definieren.

Eine **Frameklasse** (frameClass) wird für Modellausgabe- und Dokumentausgabeobjekte verwendet, bei denen die Erweiterung ihr eigenes Fenster liefert, anstatt ein IBM SPSS Modeler-Standardfenster anzupassen. Eine Frameklasse ist eine Java-Klasse, die das gesamte Fenster und dessen Verarbeitung vollständig angibt. Standardfensterkomponenten werden nicht automatisch aufgenommen, sondern müssen in der Klasse einzeln angegeben werden. Frameklassen können nur für Modellausgabe- und Dokumentausgabeobjekte verwendet werden und nicht für Knoten, die immer IBM SPSS Modeler-Dialogfelder verwenden. Weitere Informationen finden Sie im Thema „Benutzerdefinierte Ausgabefenster" auf Seite 162.

Das Format für eine Frameklasse ist einfach:

```
<UserInterface frameClass="Java-Klasse" />
```

Dabei ist frameClass der Name der Frameklasse für ein Modellausgabe- oder Dokumentausgabeobjekt. Alle Symbole, Steuerelemente und Registerkarten werden durch die Frameklasse selbst angegeben, daher werden diese Elemente in diesem Format nicht verwendet.

Die untergeordneten Elemente eines UserInterface-Elements werden in den folgenden Abschnitten beschrieben.

Beispiele

Das erste Beispiel zeigt die Benutzerschnittstelle für einen Modellierungsknoten, der ein Benutzerschnittstellendelegate definiert:

```
<UserInterface uiDelegate="com.spss.myextension.MyNodeUIDelegate">
     <Icons>
          <Icon type="standardNode" imagePath="images/lg_discriminant.gif" />
          <Icon type="smallNode" imagePath="images/sm_discriminant.gif" />
     </Icons>
     <Tabs defaultTab="1">
      ...
     </Tabs>
</UserInterface>
```

Der zugehörige Abschnitt für ein Modellausgabeobjekt ist:

```
<UserInterface>
     <Icons>
          <Icon type="standardWindow" imagePath="images/browser_discriminant.gif" />
     </Icons>
     <Tabs>
          <Tab label="Advanced" labelKey="advancedTab.LABEL"
           helpLink="discriminant_output_advancedtab.htm">
               <ExtensionObjectPanel id="DiscriminantPanel"
                panelClass="com.spss.clef.discriminant.DiscriminantPanel"/>
          </Tab>
     </Tabs>
</UserInterface>
```

Der Abschnitt UserInterface für einen Modellanwendungsknoten sieht wie folgt aus:

```
<UserInterface>
     <Icons>
          <Icon type="standardNode" imagePath="images/lg_gm_discriminant.gif" />
          <Icon type="smallNode" imagePath="images/sm_gm_discriminant.gif" />
     </Icons>
     <Tabs>
          <Tab label="Advanced" labelKey="advancedTab.LABEL"
           helpLink="discriminant_output_advancedtab.htm">
               <ExtensionObjectPanel id="DiscriminantPanel"
                panelClass="com.spss.clef.discriminant.DiscriminantPanel"/>
          </Tab>
     </Tabs>
</UserInterface>
```

Symbole

Dieser Abschnitt definiert die dem Objekt zugeordneten Symbole.

Für Knoten sollte dieser Abschnitt zwei Icon-Elemente definieren:

- Eine große Version für die Anzeige im Erstellungsbereich
- Eine kleine Version für die Anzeige auf der Palette

Für Modell- und Dokumentausgaben wird damit das Miniatursymbol definiert, das in der oberen linken Ecke des Fensterrahmens angezeigt wird.

„Erstellen von Knotensymbolen” auf Seite 15 bietet Anleitungen zum Erstellen von Symbolen in CLEF.

Format

```
<Icons>
     <Icon type="Symboltyp" imagePath="Bildpfad" />
      ...
</Icons>
```

Dabei gilt Folgendes:

`type` (erforderlich) ist einer der in der folgenden Tabelle aufgeführten Symboltypen:

Tabelle 33. Symboltypen.

Typ	Beispiel	Beschreibung
`standardNode`	 *Abbildung 36. Standardknoten-symbol*	Das große Symbol (49 x 49 Pixel), das in der Knotenform des Erstellungsbereichs angezeigt wird.
`smallNode`	 *Abbildung 37. Kleines Knotensymbol*	Das kleinere Symbol (38 x 38 Pixel), das in der Knotenform auf der Knotenpalette angezeigt wird.
`window`	 *Abbildung 38. Fenstersymbol*	Das Miniatursymbol (16 x 16 Pixel), das in einem Browser oder einem Ausgabefenster angezeigt wird.

`imagePath` (erforderlich) identifiziert den Speicherort des für das Symbol verwendeten Bildes. Der Speicherort wird relativ zu dem Verzeichnis angegeben, in dem die Spezifikationsdatei installiert ist.

Beispiele

```
<Icon type="standardNode" imagePath="images/lg_mynode.gif" />
<Icon type="smallNode" imagePath="images/sm_mynode.gif" />
<Icon type="window" imagePath="images/mynode16.gif" />
```

Steuerelemente

Dieser Abschnitt definiert benutzerdefinierte Menü- und Symbolleistenelemente, mit denen Aktionen aufgerufen werden, die im Abschnitt 'CommonObjects' der Spezifikationsdatei deklariert sind. Weitere Informationen finden Sie im Thema „Aktionen” auf Seite 40.

Format

```
<Controls>
     <Menu ... />
      ...
```

```
     <MenuItem ... />
      ...
     <ToolbarItem ... />
      ...
</Controls>
```

Die Elemente `Menu`, `MenuItem` und `ToolbarItem` werden in anschließenden Abschnitten beschrieben.

Beispiel

Das folgende Beispiel fügt ein Element zum Menü 'Generieren' und zur Symbolleiste des Dialogfelds hinzu, in dessen Spezifikation es enthalten ist. Beide Elemente implementieren eine zuvor definierte Aktion namens `generateDerive`, die einen Ableitungskonten generiert.

```
<Controls>
     <MenuItem action="generateDerive" systemMenu="generate"/>
     <ToolbarItem action="generateDerive" showLabel="false"/>
      ...
</Controls>
```

Menüs

Sie können ein benutzerdefiniertes Menü definieren, das zu einem der Standardmenüs hinzugefügt werden soll.

Format

```
<Menu id="Name" label="Anzeigebeschriftung" labelKey="Beschriftungsschlüssel" systemMenu="Menüname"
showLabel="true_false" showIcon="true_false" separatorBefore="true_false" separatorAfter="true_
false" offset="ganze_Zahl" mnemonic="mnemonischesZeichen" mnemonicKey="mnemonische_Taste"/>
```

Dabei gilt Folgendes:

`id` (erforderlich) ist eine eindeutige ID für das Menü, das hinzugefügt wird.

`label` (erforderlich) ist der Anzeigename für das Menü, der auf der Benutzerschnittstelle angezeigt wird (sofern `showLabel` auf `true` gesetzt ist).

`labelKey` gibt die Beschriftung für Lokalisierungszwecke an.

`systemMenu` (erforderlich) identifiziert das Standardmenü, zu dem das benutzerdefinierte Menü hinzugefügt wird. Der Wert von *Menüname* hat einen der folgenden Werte:

- `file`
- `edit`
- `insert*`
- `view*`
- `tools*`
- `window*`
- `generate`
- `help*`
- `file.project`
- `file.outputs`
- `file.models`
- `edit.stream`
- `edit.node`

- edit.outputs
- edit.models
- edit.project
- tools.repository
- tools.options
- tools.streamProperties

* nur gültig beim Hinzufügen zum IBM SPSS Modeler-Hauptfenster

showLabel gibt an, ob die Anzeigebeschriftung des Elements auf der Benutzerschnittstelle verwendet werden soll.

showIcon gibt an, ob das dem Element zugeordnete Symbol auf der Benutzerschnittstelle angezeigt werden soll.

separatorBefore gibt an, ob vor diesem neuen Element ein Trennzeichen (z. B. ein horizontaler Balken für Menüelemente oder ein Leerraum für Symbolleistenschaltflächen) zum Menü hinzugefügt werden soll.

separatorAfter gibt an, ob nach diesem neuen Element ein Trennzeichen zum Menü hinzugefügt werden soll.

offset ist eine nicht negative ganze Zahl, die die Position des neuen Elements angibt. Ein Offset von 0 beispielsweise fügt das neue Element als erstes Element hinzu (standardmäßig wird es am Ende hinzugefügt).

mnemonic ist der Buchstabe, der zusammen mit der Taste Alt gedrückt wird, um dieses Steuerelement zu aktivieren (wenn z. B. der Wert S angegeben ist, kann der Benutzer dieses Steuerelement über Alt+S aktivieren).

mnemonicKey gibt das mnemonische Zeichen für Lokalisierungszwecke an. Wenn weder mnemonic noch mnemonicKey verwendet wird, ist kein Direktaufruf dieser Aktion verfügbar. Weitere Informationen finden Sie im Thema „Zugriffstasten und Tastenkombinationen" auf Seite 115.

Menüelemente

Sie können ein benutzerdefiniertes Menüelement definieren, das zu einem der Standardmenüs oder zu einem benutzerdefinierten Menü hinzugefügt werden soll.

Format

```
<MenuItem action="ID" systemMenu="Menüname" customMenu="Menüname"
     showLabel="true_false" showIcon="true_false" separatorBefore="true_false"
     separatorAfter="true_false" offset="ganze_Zahl" />
```

Dabei gilt Folgendes:

action (erforderlich) ist die ID einer Aktion, die im Abschnitt 'CommonObjects' definiert ist und durch dieses Menüelement ausgeführt wird.

systemMenu legt fest, dass das Element in dem durch *Menüname* angegebenen Standardmenü angezeigt wird, bei dem es sich um eines der folgenden Menüs handelt:

- file
- edit
- insert*

- view*
- tools*
- window*
- generate
- help*
- file.project
- file.outputs
- file.models
- edit.stream
- edit.node
- edit.outputs
- edit.models
- edit.project
- tools.repository
- tools.options
- tools.streamProperties

* nur gültig beim Hinzufügen zum IBM SPSS Modeler-Hauptfenster

customMenu ist eine ID von einem Menu-Element, die angibt, dass das Menüelement in dem benutzerdefinierten Menü angezeigt werden soll.

showLabel gibt an, ob die Anzeigebeschriftung des Elements auf der Benutzerschnittstelle verwendet werden soll.

showIcon gibt an, ob das dem Element zugeordnete Symbol auf der Benutzerschnittstelle angezeigt werden soll.

separatorBefore gibt an, ob vor diesem neuen Element ein Trennzeichen (z. B. ein horizontaler Balken für Menüelemente oder ein Leerraum für Symbolleistenschaltflächen) zum Menü hinzugefügt werden soll.

separatorAfter gibt an, ob nach diesem neuen Element ein Trennzeichen zum Menü hinzugefügt werden soll.

offset ist eine nicht negative ganze Zahl, die die Position des neuen Elements angibt. Ein Offset von 0 beispielsweise fügt das neue Element als erstes Element hinzu (standardmäßig wird es am Ende hinzugefügt).

Beispiel

```
<MenuItem action="generateSelect" systemMenu="generate" showIcon="true"/>
```

Symbolleistenelemente

Sie können ein benutzerdefiniertes Symbolleistenelement für ein Dialog- oder Ausgabefenster definieren.

Format

```
<ToolbarItem action="Aktion" showLabel="true_false" showIcon="true_false"
     separatorBefore="true_false" separatorAfter="true_false" offset="ganze_Zahl" />
```

Dabei gilt Folgendes:

action (erforderlich) ist die ID einer Aktion, die im Abschnitt 'CommonObjects' definiert ist und durch dieses Symbolleistenelement ausgeführt wird.

showLabel gibt an, ob die Anzeigebeschriftung des Elements auf der Benutzerschnittstelle verwendet werden soll.

showIcon gibt an, ob das dem Element zugeordnete Symbol auf der Benutzerschnittstelle angezeigt werden soll.

separatorBefore gibt an, ob vor diesem neuen Element ein Trennzeichen (z. B. ein horizontaler Balken für Menüelemente oder ein Leerraum für Symbolleistenschaltflächen) zum Menü hinzugefügt werden soll.

separatorAfter gibt an, ob nach diesem neuen Element ein Trennzeichen zum Menü hinzugefügt werden soll.

offset ist eine nicht negative ganze Zahl, die die Position des neuen Elements angibt. Ein Offset von 0 beispielsweise fügt das neue Element als erstes Element hinzu (standardmäßig wird es am Ende hinzugefügt).

Beispiel

```
<ToolbarItem action="generateDerive" showLabel="true"/>
```

Beispiel: Hinzufügen zum Hauptfenster

Dies ist ein Beispiel für eine Spezifikationsdatei, die dem Menü "Tools" im Hauptfenster einen neuen Eintrag hinzufügt. Es definiert keine Standardobjekte (z. B. Knoten, Modellausgabefenster oder Dokumentausgabefenster), hat aber ein UserInterface-Element in der Extension der obersten Ebene, was soviel bedeutet, wie 'ändere das Hauptfenster'. Alle anderen UserInterface-Abschnitte müssen dann in einer der Standardobjektdefinitionen angegeben werden und wirken sich auf Dialogfelder aus, die diesen Objekten zugeordnet sind.

```
<?xml version="1.0" encoding="UTF-8"?>
<Extension version="1.0">
     <ExtensionDetails providerTag="example" id="main_window" label="Main Window" version="1.0"
          provider="IBM" copyright="(c) 2005-2006 IBM"
          description="An example extension that plugs into the main window"/>
     <Resources/>
     <CommonObjects>
          <Actions>
               <Action id="customTool1" label="Custom Tool..." labelKey="customTool.LABEL"
                    imagePath="images/generate.gif" description="Invokes the custom tool"
                    descriptionKey="customTool.TOOLTIP"/>
               <Action id="customTool2" label="Custom Tool..." labelKey="customTool.LABEL"
                    imagePath="images/generate.gif" description="Invokes the custom tool"
                    descriptionKey="customTool.TOOLTIP"/>
          </Actions>
     </CommonObjects>
     <UserInterface actionHandler="com.spss.cleftest.MainWindowActionHandler">
          <Controls>
               <Menu systemMenu="tools" id ="toolsExtension" separatorBefore="true"
                    label="Extension Items" offset="3"/>
               <MenuItem action="customTool2" customMenu="toolsExtension" showIcon="true"/>
               <MenuItem action="customTool1" systemMenu="file.models" showIcon="true"/>
               <ToolbarItem action="customTool1" offset="0"/>
          </Controls>
     </UserInterface>
</Extension>
```

Diese Anweisungen sorgen dafür, dass ein neues Untermenü namens **Extension Items** zum Menü 'Extras' hinzugefügt wird. Dieses neue Untermenü enthält einen einzigen Eintrag namens **Custom Tool**.

Sie können dieses Beispiel ausprobieren, indem Sie den XML-Code in einer Datei mit dem Namen *extension.xml* speichern und dann CLEF die Erweiterung hinzufügen. Weitere Informationen finden Sie im Thema „Testen einer CLEF-Erweiterung” auf Seite 203.

Registerkarten

Im Abschnitt `Tabs` werden die Registerkarten definiert, die an folgenden Stellen angezeigt werden können:

- Dem Dialogfeld, das angezeigt wird, wenn der Benutzer im Erstellungsbereich einen Knoten öffnet
- Einem neuen Modellausgabefenster
- Einem Dokumentausgabefenster

Jeder Abschnitt `Tabs` kann mehrere `Tab`-Elemente enthalten, die jeweils eine anzuzeigende benutzerdefinierte Registerkarte deklarieren:

```
<Tabs defaultTab="ganze_Zahl">
     <Tab ... />
     <Tab ... />
      ...
</Tabs>
```

Dabei ist `defaultTab` eine positive ganze Zahl, die angibt, welche Registerkarte angezeigt wird, wenn das Knotendialogfeld oder das Fenster zum ersten Mal in einem Stream geöffnet wird. Wenn der Benutzer eine andere Registerkarte auswählt, wird beim Schließen und erneuten Öffnen des Dialogfelds oder Fensters während der Stream aktiv ist, statt der Standardregisterkarte die zuletzt angezeigte Registerkarte angezeigt. Die Nummerierung der Registerkarten beginnt bei 0.

Beachten Sie, dass andere Registerkarten automatisch in das Dialogfeld oder Fenster aufgenommen werden können. Beispielsweise enthalten alle Dialogfelder und Ausgabefenster eine Registerkarte **Anmerkungen** und alle Dialogfelder für Datenquellen enthalten die Registerkarten **Filter** und **Typen**.

In einem `Tab`-Element muss die Registerkartenbeschriftung deklariert sein (der Text, der auf der Registerkarte angezeigt wird). Es sollte außerdem einen Beschriftungsschlüssel enthalten, nach dem gesucht wird, wenn die Registerkartenbeschriftung übersetzt werden soll.

Innerhalb aller `Tab`-Elemente befindet sich eine Fensterspezifikation, die definiert, wie der Hauptinhaltsbereich der Registerkarte aufgebaut ist. Die Fensterspezifikation kann einen der folgenden Typen haben: Textbrowser (TextBrowserPanel), Erweiterungsobjekt (ExtensionsObjectPanel) oder Eigenschaften (PropertiesViewerPanel). Weitere Informationen finden Sie im Thema „Fensterspezifikationen” auf Seite 117.

Das Format eines einzelnen `Tab`-Elements lautet:

```
<Tab id="ID" label="Anzeigebeschriftung" labelKey="Beschriftungsschlüssel" helpLink="Hilfe-ID"
          mnemonic="mnemonisches_Zeichen" mnemonicKey="mnemonische_Taste" >
     <TextBrowserPanel ... />
     <ExtensionObjectPanel ... />
     <PropertiesPanel ... />
     <ModelViewerPanel ... />
</Tab>
```

Dabei gilt Folgendes:

`id` ist eine eindeutige ID, über die die Registerkarte in Java-Code referenziert werden kann.

`label` (erforderlich) ist der Anzeigename für die Registerkarte, mit dem sie auf der Benutzerschnittstelle angezeigt wird.

`labelKey` gibt die Beschriftung für Lokalisierungszwecke an.

`helpLink` ist die Kennung für ein Hilfethema, das angezeigt wird, wenn der Benutzer das Hilfesystem aufruft, sofern vorhanden. Das Format der Kennung hängt vom Typ des Hilfesystems ab (siehe „Festlegen von Speicherort und Typ des Hilfesystems" auf Seite 166):

HTML-Hilfe: URL des Hilfethemas

JavaHelp: ID des Themas

`mnemonic` ist der Buchstabe, der zusammen mit der Taste Alt gedrückt wird, um dieses Steuerelement zu aktivieren (wenn z. B. der Wert S angegeben ist, kann der Benutzer dieses Steuerelement über Alt+S aktivieren).

`mnemonicKey` gibt das mnemonische Zeichen für Lokalisierungszwecke an. Wenn weder `mnemonic` noch `mnemonicKey` verwendet wird, ist kein Direktaufruf dieser Aktion verfügbar. Weitere Informationen finden Sie im Thema „Zugriffstasten und Tastenkombinationen".

Beispiele

Beispiele zu `Tab`-Elementen finden Sie in den Abschnitten über die verschiedenen Fensterspezifizikationen, die folgende Inhalte aufweisen können: „Textbrowserfenster" auf Seite 117, „Erweiterungsobjektfenster" auf Seite 119, „Eigenschaftsfenster" auf Seite 120 und „Modellviewerfenster" auf Seite 122.

Zugriffstasten und Tastenkombinationen

Als Alternative für den Mauszugriff auf Funktionen der Benutzerschnittstelle können Sie verschiedene Tastenkombinationen angeben, damit Benutzer über die Tastatur auf Funktionen zugreifen können.

Zugriffstasten

Zugriffstasten sind Tasten, die zusammen mit der Taste Alt verwendet werden. Für Menüeinträge, Registerkarten und viele andere Steuerelemente für Dialogfelder können Sie Zugriffstasten durch Verwendung des Attributs `mnemonic` der folgenden Elemente festlegen.

Tabelle 34. Funktionen der Benutzerschnittstelle.

Feature	Element	Siehe...
Bildschirmaktion (z. B. für einen Menüeintrag)	`action`	„Aktionen" auf Seite 40
Menü	`menu`	„Menüs" auf Seite 110
Registerkarte	`tab`	„Registerkarten" auf Seite 114
Controller	Verschiedene	„Controller" auf Seite 129

Nehmen Sie das folgende Menü als Beispiel.

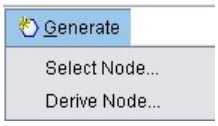

Abbildung 39. Menüelemente

Für die Angabe von Zugriffstasten für dieses Menü würden Sie den folgenden Code verwenden:

```
<Actions>
     <Action id="generateSelect" label="Select Node..." labelKey="generate.selectNode.LABEL"
          imagePath="images/generate.gif" description="Generates a select node"
          descriptionKey="generate.selectNode.TOOLTIP" mnemonic="S" />
     <Action id="generateDerive" label="Derive Node..." labelKey="generate.deriveNode.LABEL"
          imagePath="images/generate.gif" description="Generates a derive node"
          descriptionKey="generate.deriveNode.TOOLTIP"  mnemonic="D" />
   ...
</Actions>
```

Damit erhalten Sie die Unterstriche in den folgenden Menüelementen.

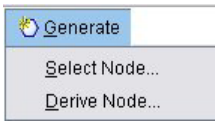

Abbildung 40. Verwenden von Zugriffstasten mit Menüeinträgen

Benutzer können nun mithilfe von Alt+S und Alt+D und per Mausklick auf die Menüeinträge zugreifen.

Tastenkombinationen

Tastenkombinationen dienen zur Navigation in der Benutzerschnittstelle. IBM SPSS Modeler bietet eine Reihe von Tastenkombinationen als Standard. In CLEF können Sie Tastenkombinationen nur für benutzerdefinierte Menüelemente, die Sie hinzugefügt haben, definieren.

Für die Angabe von Tastenkombinationen für Elemente im Beispiel für Zugriffstasten würden Sie folgenden Code verwenden:

```
<Actions>
     <Action id="generateSelect" label="Select Node..." labelKey="generate.selectNode.LABEL"
          imagePath="images/generate.gif" description="Generates a select node"
          descriptionKey="generate.selectNode.TOOLTIP" mnemonic="S" shortcut="CTRL+SHIFT+S" />
     <Action id="generateDerive" label="Derive Node..." labelKey="generate.deriveNode.LABEL"
          imagePath="images/generate.gif" description="Generates a derive node"
          descriptionKey="generate.deriveNode.TOOLTIP"  mnemonic="D"  shortcut="CTRL+SHIFT+D" />
     ...
</Actions>
```

Die Tastenkombinationen werden nun den Menüelementen hinzugefügt.

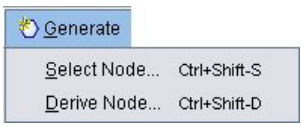

Abbildung 41. Verwenden von Tastenkombinationen mit Menüeinträgen

Benutzer können nun mithilfe der Tastenkombinationen und per Mausklick auf die Menüeinträge zugreifen. Sie können Tastenkombinationen wie im Beispiel zusammen mit Zugriffstasten angeben.

Achten Sie sorgfältig darauf, dass Sie keine Zugriffstasten verwenden, die bereits in demselben Dialogfeld angegeben sind oder zu den IBM SPSS Modeler-Standardkombinationen gehören.

Fensterspezifikationen

Jedes `Tab`-Element kann die Spezifikation für ein einzelnes Fenster enthalten, das einen der in der folgenden Tabelle aufgeführten Typen aufweisen kann.

Tabelle 35. Typen von Fensterspezifikationen

Fenster	Anzeigen...	Siehe...
Textbrowserfenster	Textinhalt eines angegebenen Containers.	„Textbrowserfenster"
Erweiterungsobjekt-fenster	Durch eine angegeben Java-Klasse definierter Inhalt.	„Erweiterungsobjektfenster" auf Seite 119
Eigenschaftsfenster	Eigenschaftssteuerelemente (z. B. Schaltflächen, Kontrollkästchen, Eingabefelder).	„Eigenschaftsfenster" auf Seite 120
Modellviewerfenster	Modellausgabe im PMML-Format aus einem festgelegten Container.	„Modellviewerfenster" auf Seite 122

Textbrowserfenster

Ein Textbrowserfenster zeigt den Textinhalt eines in der Erweiterung angegeben Containers. Unterstützte Formate (UTF-8-Codierung) sind reiner Text, HTML und Rich Text Format (RTF).

Das folgende Beispiel für ein Modellausgabefenster enthält ein Textbrowserfenster im HTML-Format.

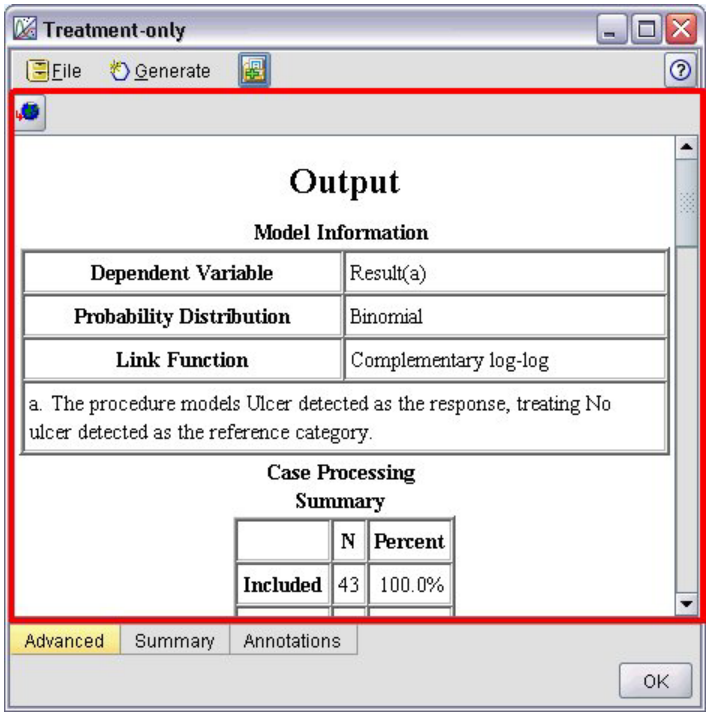

Abbildung 42. Modellausgabefenster mit hervorgehobenem Textbrowserfenster

Format

```
<TextBrowserPanel container="Name" textFormat="Textformat" rows="ganze_Zahl"
     columns="ganze_Zahl" wrapLines="true_false" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</TextBrowserPanel>
```

Dabei gilt Folgendes:

`container` (erforderlich) ist der Name des Containers, aus dem die Inhalte des Fensters erhalten werden.

`textFormat` (erforderlich) gibt das Format des im Fenster angezeigten Texts an, das folgende Werte haben kann:

- `plainText`
- `html`
- `rtf`

`rows` und `columns` geben die Anzahl der Textzeilen und -spalten an, die angezeigt werden, wenn das Fenster geöffnet wird.

`wrapLines` gibt an, ob lange Zeilen umgebrochen werden (`true`) oder ob ein horizontales Blättern erforderlich ist, um lange Textzeilen zu lesen (`false`). Der Standardwert ist `false`.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout” auf Seite 153.

Beispiel

Das folgende Beispiel zeigt den Registerkartenabschnitt, der das zuvor angezeigte Textbrowserfenster definiert:

```
<Tab label="Advanced" labelKey="advancedTab.LABEL" helpLink="genlin_output_advancedtab.htm">
     <TextBrowserPanel container="advanced_output" textFormat="html" />
</Tab>
```

Die Modellausgabe wird an einen Container gesendet, der in demselben Abschnitt `ModelOutput` definiert ist, wie die Registerkartenspezifikation:

```
<ModelOutput id="generalizedlinear" label="Generalized Linear">
     <Containers>
      ...
          <Container name="advanced_output" />
     </Containers>
     <UserInterface>
      ...
          <Tabs>
               <Tab label="Advanced" ... >
                    <TextBrowserPanel container="advanced_output" ... />
               </Tab>
          </Tabs>
     </UserInterface>
</ModelOutput>
```

Der Container wird in einem `CreateDocument`-Element im Ausführungsabschnitt für den zuständigen Erstellungsknoten referenziert:

```
<Execution>
 ...
     <Constructors>
```

```
        <CreateModelOutput type="generalizedlinear">
             <CreateModel type="generalizedlinear_model" target="model" sourceFile="pmml"/>
             <CreateDocument type="html_output" target="advanced_output" sourceFile=
              "htmloutput"/>
        </CreateModelOutput>
    </Constructors>
</Execution>
```

Erweiterungsobjektfenster

Ein Erweiterungsobjektfenster funktioniert ähnlich wie ein Textbrowserfenster, anstelle der Textinhalte eines Containers erstellt es jedoch eine Instanz der angegebenen Java-Klasse, die die von der CLEF-Java-API definierte `ExtensionObjectPanel`-Schnittstelle implementiert.

Hier sehen Sie ein Beispiel für das Dialogfeld eines Modellanwendungsknotens, das ein Erweiterungsobjektfenster enthält.

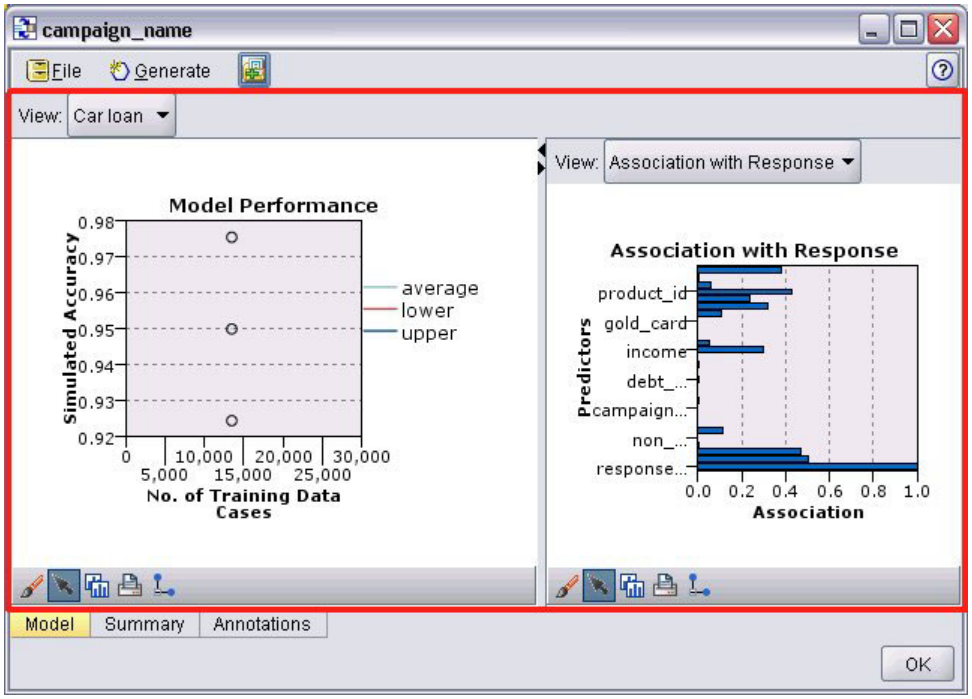

Abbildung 43. Modellausgabefenster mit hervorgehobenem Erweiterungsobjektfenster

Format

```
<ExtensionObjectPanel id="ID" panelClass="Java-Klasse" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</ExtensionObjectPanel>
```

Dabei gilt Folgendes:

`id` ist eine eindeutige ID, über die das Fenster im Java-Code referenziert werden kann.

`panelClass` (erforderlich) ist der Name der Java-Klasse, die das Fenster implementiert.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout” auf Seite 153.

Beispiel

Das folgende Beispiel zeigt den Registerkartenabschnitt, der das zuvor angezeigte Erweiterungsobjektfenster definiert:

```
<Tab label="Model" labelKey="Model.LABEL" helpLink="selflearnnode_output.htm">
     <ExtensionObjectPanel id="SelfLearningPanel"
          panelClass="com.spss.clef.selflearning.SelfLearningPanel"/>
</Tab>
```

Eigenschaftsfenster

In einem Eigenschaftsfenster können eine Registerkarte oder ein Unterfenster für Eigenschaften angezeigt werden (siehe „Eigenschaftsunterfenster" auf Seite 127), um **Eigenschaftssteuerelemente** anzuzeigen, bei denen es sich um Anzeigenkomponenten handelt (wie Schaltflächen, Kontrollkästchen und Eingabefelder), mit denen die Eigenschaften eines am Bildschirm angezeigten Objekts geändert werden können. Das Eigenschaftsfenster wendet die über diese Steuerelemente vorgenommenen Änderungen automatisch an, wenn der Benutzer auf **OK** oder auf **Anwenden** klickt. Wenn der Benutzer auf **Abbrechen** oder **Zurücksetzen** klickt, verwirft das Fenster alle Änderungen, die seit dem letzten Anwendungsvorgang durchgeführt wurden.

Das folgende Beispiel zeigt ein Knotendialogfeld, das ein Eigenschaftsfenster enthält.

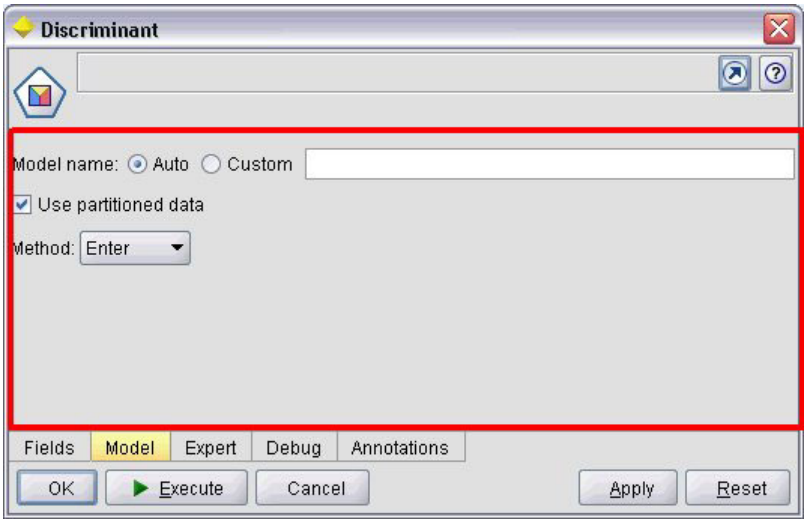

Abbildung 44. Knotendialogfeld mit hervorgehobenem Eigenschaftsfenster

Das nächste Beispiel zeigt ein Eigenschaftsunterfenster, das drei Eigenschaftsfenster enthält.

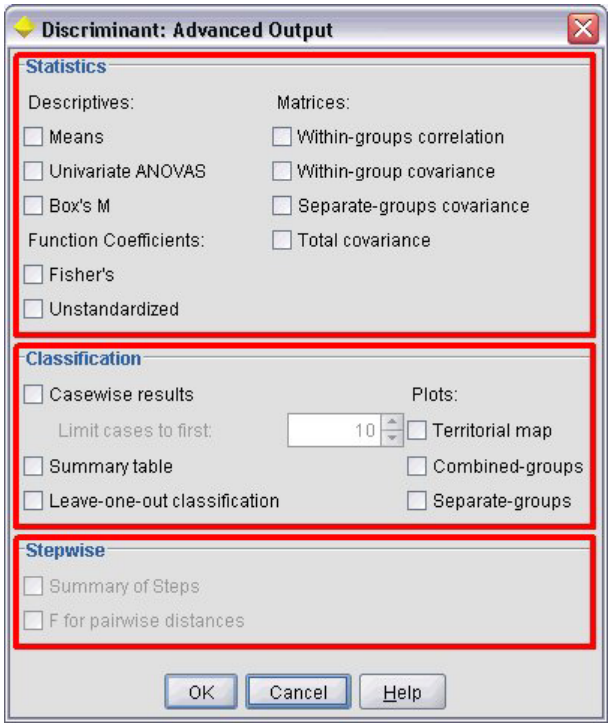

Abbildung 45. Eigenschaftsunterfenster mit hervorgehobenen Eigenschaftsfenstern

Format

```
<PropertiesPanel id="ID" label="Anzeigebeschriftung" labelKey="Beschriftungsschlüssel">
     -- erweiterte Optionen für benutzerdefiniertes Layout --
     -- Spezifikationen für Eigenschaftssteuerelemente --
</PropertiesPanel>
```

Dabei gilt Folgendes:

`id` ist eine eindeutige ID, über die das Fenster im Java-Code referenziert werden kann.

`label` ist die Anzeigeüberschrift für eine Gruppe von Eigenschaftssteuerelementen (z. B. **Statistics**, **Classification** und **Stepwise** im letzten Beispiel).

`labelKey` gibt die Beschriftung für Lokalisierungszwecke an.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Eine Beschreibung der Spezifikationen der einzelnen Eigenschaftssteuerelemente finden Sie in „Spezifikationen von Eigenschaftssteuerelementen" auf Seite 123.

Beispiel

```
<Tab label="Model" labelKey="Model.LABEL" helpLink="discriminant_node_model.htm">
     <PropertiesPanel>
          <SystemControls controlsId="ModelGeneration" />
          <ComboBoxControl property="method">
               <Layout fill="none" />
          </ComboBoxControl>
     </PropertiesPanel>
</Tab>
```

Modellviewerfenster

Ein Modellviewerfenster enthält alle Modellausgaben im PMML-Format eines in der Erweiterung angegebenen Containers.

Das folgende Beispiel zeigt das Fenster eines Modellanwendungsknotens, das ein Modellviewerfenster enthält.

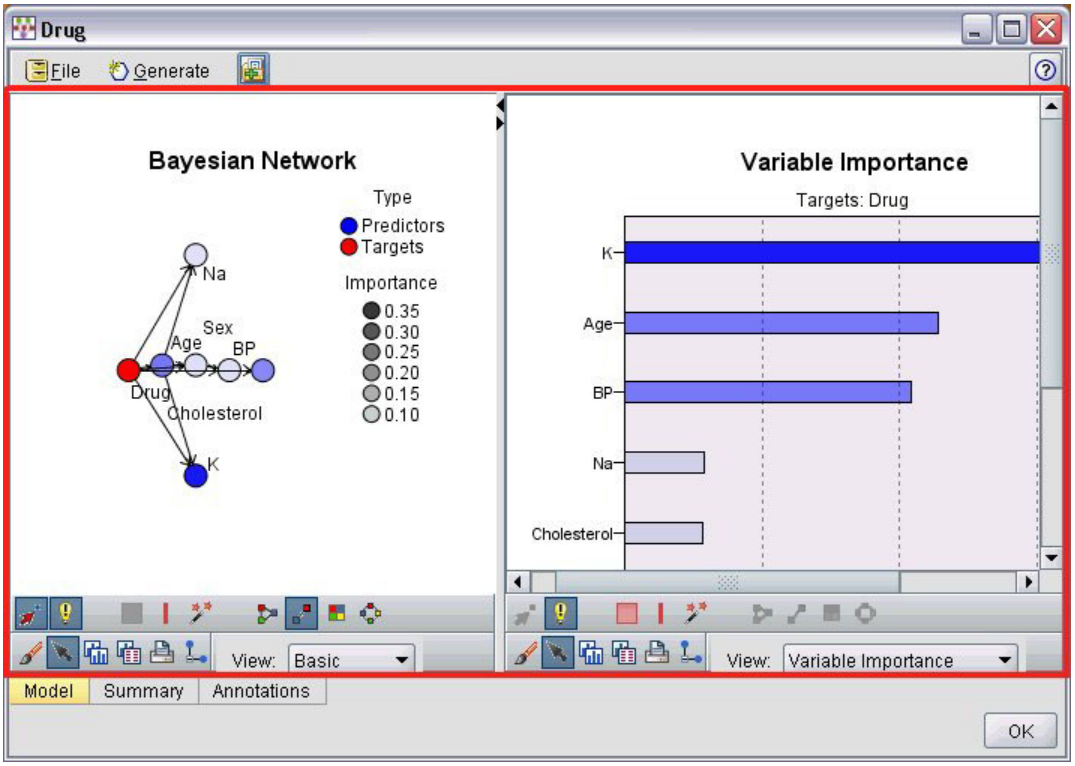

Abbildung 46. Ausgabefenster mit hervorgehobenem Modellviewerfenster

Format

```
<ModelViewerPanel container="Containername">
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</ModelViewerPanel>
```

Dabei ist `container` (erforderlich) der Name des Containers, dem die Modellausgabe zugewiesen wird.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout” auf Seite 153.

Beispiel

Dieses Beispiel zeigt die Verwendung eines Modellviewerfensters in einer Modellanwenderspezifikation. Die Modellausgabe wurde zuvor dem Container namens `model` zugewiesen. Hier nimmt die Modellanwenderspezifikation den Container und ordnet ihn dem Modellviewerfenster zu:

```
<Node id="applyBN" type="modelApplier">
     <ModelProvider container="model" isPMML="true" />
      ...
     <Containers>
          <Container name="model" />
     </Containers>
```

```
     <UserInterface>
      ...
     <Tabs>
          <Tab label="Model" labelKey="modelTab.LABEL" helpLink="BN_output_modeltab.htm">
               <ModelViewerPanel container="model"/>
          </Tab>
      ...
     </Tabs>
     </UserInterface>
      ...
</Node>
```

Spezifikationen von Eigenschaftssteuerelementen

Eigenschaftssteuerelemente sind Anzeigenkomponenten, wie Schaltflächen, Kontrollkästchen und Eingabefelder, mit deren Hilfe die Eigenschaften eines auf dem Bildschirm angezeigten Objekts geändert werden können. Das Format der Spezifikation eines Eigenschaftssteuerelements hängt vom Typ des Eigenschaftsteuerelements ab, der folgende Ausprägungen haben kann:

- Benutzerschnittstellenkomponente
- Eigenschaftsfenster
- Controller

Steuerelemente für **Benutzerschnittstellenkomponenten** sind Aktionsschaltflächen, statischer Text in der Anzeige und Systemsteuerelemente (ein Satz von Steuerelementen, die Eigenschaften behandeln, die in allen Dialogfeldern gleich sind).

Steuerelemente für **Eigenschaftsfenster** sind einzelnen Fenster innerhalb der Spezifikation des Eigenschaftsfensters.

Controller bilden die größte Gruppe der Eigenschaftssteuerelemente. Hierzu gehören Elemente wie Kontrollkästchen, Kombinationsfelder und Drehfelder.

Steuerelemente der Benutzerschnittstellenkomponente

In der folgenden Tabelle sind Steuerelemente der Benutzerschnittstellenkomponente dargestellt.

Tabelle 36. Steuerelemente der Benutzerschnittstellenkomponente

Steuerelement	Beschreibung
`ActionButton`	Eine Bildschirmschaltfläche, die beim Anklicken eine vordefinierte Aktion ausführt.
`StaticText`	Eine nicht variierbare Textzeichenfolge, die auf dem Bildschirm angezeigt wird.
`SystemControls`	Standardsätze von Steuerelementen, die Eigenschaften behandeln, die alle Modelle gemeinsam haben.

Aktionsschaltfläche

Definiert eine Dialogfeld- oder Symbolleistenschaltfläche, die eine Aktion ausführt, die im Abschnitt 'CommonObjects' definiert ist. Die Aktion (z. B. die Anzeige eines neuen Bildschirms) wird durchgeführt, wenn der Benutzer auf diese Schaltfläche klickt.

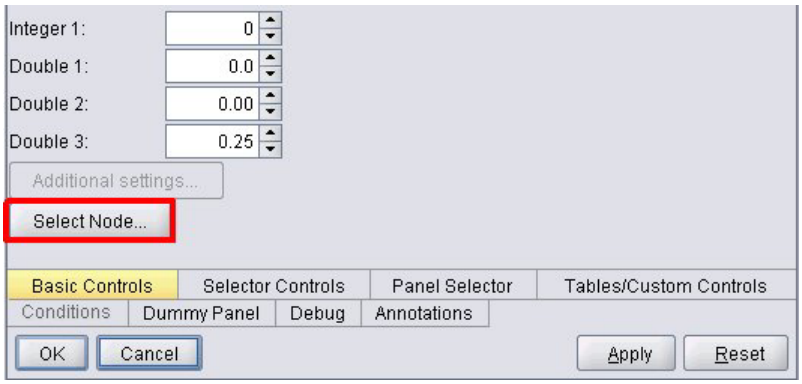

Abbildung 47. Dialogfeld mit hervorgehobener Aktionsschaltfläche

Format

```
<ActionButton action="Aktion" showLabel="true_false" showIcon="true_false" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</ActionButton>
```

Dabei gilt Folgendes:

`action` (erforderlich) ist die ID für die durchzuführende Aktion.

`showLabel` gibt an, ob die Beschriftung der Schaltfläche angezeigt (`true`) oder ausgeblendet (`false`) wird (in einer Symbolleiste wird z. B. anstelle einer Beschriftung eher ein reines Symbol angezeigt). Der Standardwert ist `true`.

`showIcon` gibt an, ob ein der Schaltfläche zugeordnetes Symbol angezeigt (`true`) oder ausgeblendet (`false`) werden soll. Der Standardwert ist `false`.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout” auf Seite 153.

Beispiel

Der Code für die oben dargestellte Aktionsschaltfläche lautet:

```
<ActionButton action="generateSelect"/>
```

Die Aktion wird wie folgt im Abschnitt 'CommonObjects' definiert (beachten Sie, dass hier auch die Schaltflächenbeschriftung definiert wird):

```
<CommonObjects extensionListenerClass="com.spss.cleftest.TestExtensionListener">
     ...
    <Actions>
      <Action id="generateSelect" label="Select Node..." labelKey="generate.selectNode.LABEL"
           imagePath="images/generate.gif" description="Generates a select node"
           descriptionKey="generate.selectNode.TOOLTIP"/>
       ...
    </Actions>
</CommonObjects>
```

Feststehender Text

Dieses Element bietet die Möglichkeit, eine nicht variierbare Textzeichenfolge in ein Dialogfeld oder ein Ausgabefenster einzufügen. Das folgende Beispiel zeigt ein Eigenschaftsfenster, das statischen Text ent-

hält.

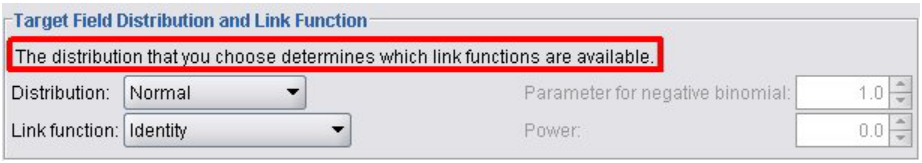

Abbildung 48. Eigenschaftsfenster mit hervorgehobenem statischen Text

Format

```
<StaticText text="statischer_Text" textKey="Textschlüssel" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</StaticText>
```

Dabei gilt Folgendes:

`text` ist die zu verwendende Textzeichenfolge.

`textKey` gibt den statischen Text für Lokalisierungszwecke an.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

Das folgende Beispiel zeigt die Deklaration, die für den oben gezeigten statischen Text verwendet wird:

```
<StaticText text="The distribution that you choose determines which link functions are
     available." textKey="Genlin_staticText1"/>
```

Systemsteuerelemente

Einige Eigenschaften sind für alle Modelle identisch. In einem Modellierungsknoten handelt es sich bei den Systemsteuerelementen um Standardsätze von Steuerelementen, die diese Eigenschaften behandeln.

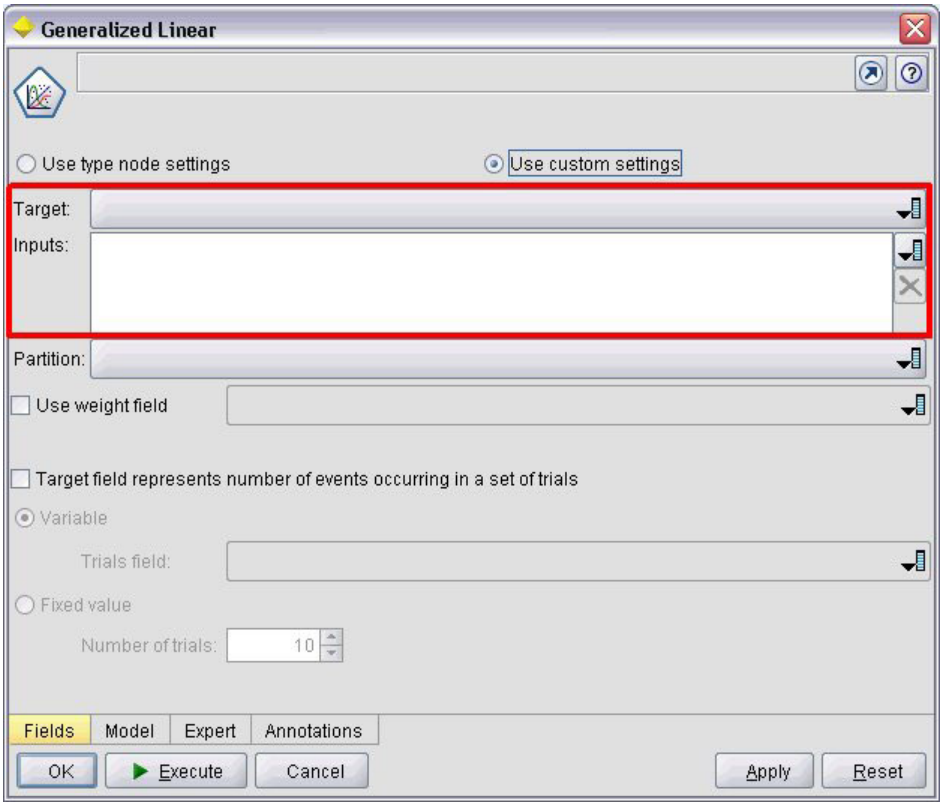

Abbildung 49. Beispiel für ein Dialogfeld mit hervorgehobenen Systemsteuerelementen

Format

```
<SystemControls controlsID="ID" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</SystemControls>
```

Dabei ist `controlsID` die ID des Steuerelementsatzes. Diese ID muss identisch sein mit der, die im Attribut `controlsID` eines `ModelingFields`-Elements in einer Modellierungsdeklaration festgelegt ist (siehe „Modellerstellung" auf Seite 51).

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

In diesem Beispiel wird die Deklaration dargestellt, die für die Systemsteuerelemente in der vorigen Abbildung verwendet wurde.

Im Modellierungsabschnitt der Knotenspezifikation definieren die folgenden Anweisungen ein Set mit Systemsteuerungselementen, zu denen in diesem Fall die Feldauswahllisten für Ein- und Ausgabefelder des Modells gehören:

```
<ModelBuilder ... >
     <ModelingFields controlsId="modelingFields">
          <InputFields property="inputs" multiple="true" label="Inputs" types="[range set
orderedSet flag]" labelKey="inputFields.LABEL"/>
          <OutputFields property="target" multiple="false" types="[range flag]"
```

```
label="Target" labelKey="targetField.LABEL"/>
     </ModelingFields>
      ...
</ModelBuilder>
```

Im weiteren Verlauf der Datei wird dieser Steuerelementsatz in der Definition für die Registerkarte des Modellierungsknotendialogfelds referenziert, auf der er angezeigt wird:

```
<Tab label="Fields" labelKey="Fields.LABEL" helpLink="genlin_node_fieldstab.htm">
     <PropertiesPanel>
          <SystemControls controlsId="modelingFields">
          </SystemControls>
      ...
     </PropertiesPanel>
</Tab>
```

Steuerelemente des Eigenschaftsfensters

In der folgenden Tabelle sind Steuerelemente des Eigenschaftsfensters dargestellt.

Tabelle 37. Steuerelemente des Eigenschaftsfensters

Steuerelement	Beschreibung
`PropertiesSubPanel`	Separates Dialogfeld, das angezeigt wird, wenn der Benutzer in einem Eigenschaftsfenster auf eine Schaltfläche klickt.
`PropertiesPanel`	Eigenschaftsfenster, das in die Deklaration eines Eigenschaftsunterfensters oder in eine Eigenschaftsfensterdeklaration auf oberster Ebene eingebettet ist.

Eigenschaftsunterfenster

Definiert ein separates Dialogfeld, das angezeigt wird, wenn der Benutzer in einem Eigenschaftsfenster auf eine Schaltfläche klickt. Die Deklaration des Eigenschaftsunterfensters erfolgt im Rahmen der Spezifikation des Haupteigenschaftsfensters für eine Registerkarte.

Format

```
<PropertiesSubPanel buttonLabel="Anzeigebeschriftung" buttonLabelKey="Beschriftungsschlüssel"
     dialogTitle="Anzeigetitel" dialogTitleKey="Titelschlüssel" helplink="Hilfe-ID"
     mnemonic="mnemonisches_Zeichen" mnemonicKey="mnemonische_Taste" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
     -- Spezifikationen für Eigenschaftssteuerelemente --
</PropertiesSubPanel>
```

Dabei gilt Folgendes:

`buttonLabel` ist die Beschriftung der Schaltfläche, die Zugriff auf das Unterfenster gewährt.

`buttonLabelKey` gibt die Schaltflächenbeschriftung für Lokalisierungszwecke an.

`dialogTitle` ist der Text, der in der Titelleiste des Unterfensterdialogfelds angezeigt wird.

`dialogTitleKey` identifiziert den Titel des Dialogfelds des Unterfensters für Lokalisierungszwecke.

`helpLink` ist die Kennung für ein Hilfethema, das angezeigt wird, wenn der Benutzer das Hilfesystem aufruft, sofern vorhanden. Das Format der Kennung hängt vom Typ des Hilfesystems ab (siehe „Festlegen von Speicherort und Typ des Hilfesystems” auf Seite 166):

HTML-Hilfe: URL des Hilfethemas

JavaHelp: ID des Themas

`mnemonic` ist der Buchstabe, der zusammen mit der Taste Alt gedrückt wird, um dieses Steuerelement zu aktivieren (wenn z. B. der Wert S angegeben ist, kann der Benutzer dieses Steuerelement über Alt+S aktivieren).

`mnemonicKey` gibt das mnemonische Zeichen für Lokalisierungszwecke an. Wenn weder `mnemonic` noch `mnemonicKey` verwendet wird, ist kein Direktaufruf dieser Aktion verfügbar. Weitere Informationen finden Sie im Thema „Zugriffstasten und Tastenkombinationen" auf Seite 115.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Eine Beschreibung der Spezifikationen der einzelnen Eigenschaftssteuerelemente finden Sie in „Spezifikationen von Eigenschaftssteuerelementen" auf Seite 123.

Beispiel

Das folgende Beispiel zeigt ein Eigenschaftsunterfenster, das angezeigt wird, wenn der Benutzer im Haupteigenschaftsbereich einer Registerkarte auf die Schaltfläche **Ausgabe** klickt.

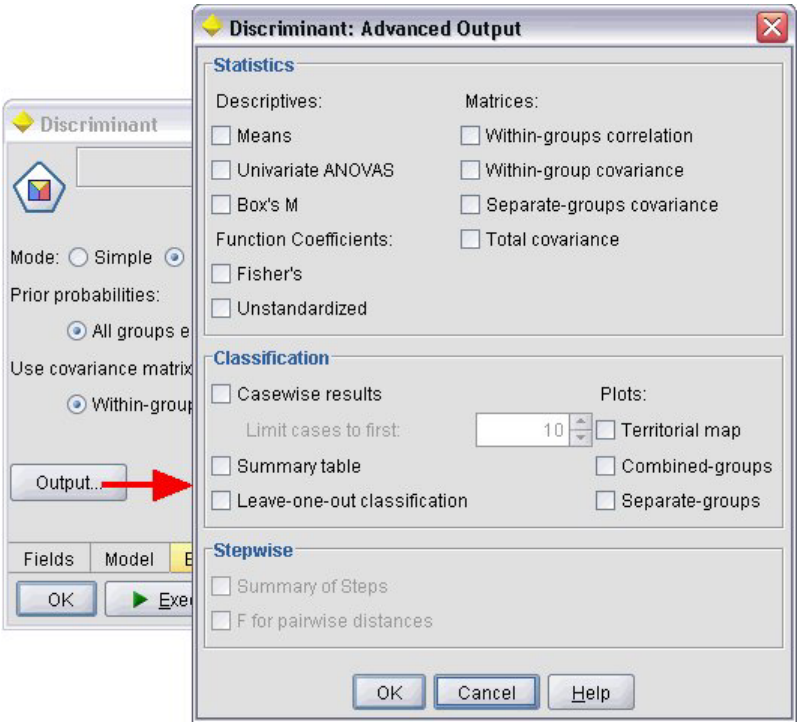

Abbildung 50. Eigenschaftsunterfenster

Der folgende Code zeigt die entscheidenden Teile der Deklaration, die für das Eigenschaftsunterfenster verwendet werden. Beachten Sie, dass innerhalb der Unterfensterdeklaration jede Feldgruppe (**Statistics**, **Classification** und **Stepwise**) ihre eigene Spezifikation des Eigenschaftsfensters hat:

```
<PropertiesSubPanel buttonLabel="Output..." buttonLabelKey="OutputSubPanel.LABEL"
     dialogTitle="Discriminant: Advanced Output" dialogTitleKey="AdvancedOutputSubDialog.LABEL"
     helpLink="discriminant_node_outputdlg.htm">
     ...
     <PropertiesPanel>
          <PropertiesPanel label="Statistics" ... >
```

```
          ...
        </PropertiesPanel>
        <PropertiesPanel label="Classification" ... >
          ...
        </PropertiesPanel>
        <PropertiesPanel label="Stepwise" ... >
          ...
        </PropertiesPanel>
    </PropertiesPanel>
</PropertiesSubPanel>
```

Eigenschaftsfenster (verschachtelt)

Sie können eine Eigenschaftsfensterspezifikation in einer Eigenschaftsunterfensterdeklaration verschachteln, um die Inhalte des vom Unterfenster angezeigten Dialogfelds zu definieren. Weitere Informationen finden Sie im Thema „Eigenschaftsunterfenster" auf Seite 127.

Sie können auch eine Eigenschaftsfensterspezifikation in der Eigenschaftsfensterdeklaration der obersten Ebene verschachteln. Dies bietet sich z. B. dann an, wenn der Inhalt einer gesamten Registerkarte, die aus mehreren Eigenschaftsfenstern besteht, abhängig davon aktiviert oder inaktiviert werden soll, ob eine bestimmte Schaltfläche auf der Registerkarte ausgewählt wird. In diesem Fall sieht die Registerkartenspezifikation etwa wie folgt aus:

```
<Tab .... >
    <PropertiesPanel>
     --- Schaltflächenspezifikation ---
        <PropertiesPanel>
            <Enabled>
             --- Bedingung mit Schaltflächenwert ---
            </Enabled>
          ...
        </PropertiesPanel>
        <PropertiesPanel>
            <Enabled>
             --- Bedingung mit Schaltflächenwert ---
            </Enabled>
          ...
        </PropertiesPanel>
          ...
    </PropertiesPanel>
</Tab>
```

Eine verschachtelte Eigenschaftsfensterspezifikation hat das dasselbe Format wie das Element auf der obersten Ebene. Weitere Informationen finden Sie im Thema „Eigenschaftsfenster" auf Seite 120.

Controller

Controller bilden die größte Gruppe der Eigenschaftssteuerelemente.

Tabelle 38. Controller

Steuerelement	Beschreibung
`CheckBoxControl`	Kontrollkästchen.
`CheckBoxGroupControl`	Satz von Kontrollkästchen, ein Kontrollkästchen pro `enum`-Wert.
`ClientDirectoryChooserControl`	Einzeiliges Textfeld und zugehörige Schaltfläche mit der der Benutzer ein Verzeichnis auf dem Client auswählen kann.
`ClientFileChooserControl`	Einzeiliges Textfeld und zugehörige Schaltfläche mit der der Benutzer eine Datei auf dem Client auswählen kann.
`ComboBoxControl`	Dropdown-Liste eines Kombinationsfelds, die die `enum`-Werte enthält.

Tabelle 38. Controller (Forts.)

Steuerelement	Beschreibung
DBConnectionChooserControl	Ermöglicht dem Benutzer die Auswahl einer Datenquelle und das Herstellen einer Verbindung zur Datenbank.
DBTableChooserControl	Ermöglicht dem Benutzer die Auswahl einer Datenbanktabelle, nachdem erfolgreich eine Datenbankverbindung hergestellt wurde.
MultiFieldChooserControl	(Nur Knoten) Liste mit Feldnamen, aus der der Benutzer mindestens ein Feld auswählen kann.
MultiItemChooserControl	Ermöglicht dem Benutzer, mindestens ein Element aus einer Werteliste auszuwählen.
PasswordBoxControl	Einzeilige Textzeile, in der die eingegebenen Zeichen ausgeblendet werden.
PropertyControl	Ein benutzerdefinierbares Steuerelement für eine Eigenschaft.
RadioButtonGroupControl	Satz von Optionsfeldern, die gleichzeitig ausgewählt werden können. Für enum-Eigenschaften gibt es ein Optionsfeld pro enum-Wert; für boolesche Eigenschaften werden zwei Optionsfelder angezeigt.
ServerDirectoryChooserControl	Einzeiliges Textfeld und zugehörige Schaltfläche mit der der Benutzer ein Verzeichnis auf dem Server auswählen kann.
ServerFileChooserControl	Einzeiliges Textfeld und zugehörige Schaltfläche mit der der Benutzer eine Datei auf dem Server auswählen kann.
SingleFieldChooserControl	(Nur Knoten) Liste mit Feldnamen, aus der der Benutzer ein einzelnes Feld auswählen kann.
SingleItemChooserControl	Ermöglicht dem Benutzer, ein einzelnes Element aus einer Werteliste auszuwählen.
SpinnerControl	Drehfeld (numerisches Feld mit aufwärts und abwärts weisenden Pfeilen, mit denen der Wert geändert wird).
TableControl	Fügt eine Tabelle zu einem Dialogfeld oder Fenster hinzu.
TextAreaControl	Mehrzeiliges Textfeld.
TextBoxControl	Einzeiliges Textfeld.

Controller-Attribute

Controller-Spezifikationen können folgende Attribute enthalten:

```
property="Wert" showLabel="true_false" label="Anzeigebeschriftung" labelKey="Beschriftungsschlüssel"
labelWidth="Beschriftungsbreite" labelAbove="true_false" description="Beschreibung"
descriptionKey="Beschreibungsschlüssel" mnemonic="mnemonisches_Zeichen" mnemonicKey="mnemonische_Taste"
```

Dabei gilt Folgendes:

property (erforderlich) ist die eindeutige ID des Eigenschaftssteuerelements.

showLabel gibt an, ob die Anzeigebeschriftung des Eigenschaftssteuerelements angezeigt (true) oder ausgeblendet (false) werden soll. Der Standardwert ist true.

label ist der Anzeigename für das Eigenschaftssteuerelement, der auf der Benutzerschnittstelle angezeigt wird. Dieser Wert wird auch als kurze aufrufbare Beschreibung des Steuerelements verwendet. Weitere Informationen finden Sie im Thema „Zugriffsmöglichkeiten” auf Seite 175.

labelKey gibt die Beschriftung für Lokalisierungszwecke an.

labelWidth ist die Anzahl der anzuzeigenden Rasterspalten, über die sich die Beschriftung erstreckt. Der Standardwert ist 1.

labelAbove legt fest, ob die Beschriftung für das Steuerelement über (true) oder neben (false) dem Steuerelement angezeigt wird. Der Standardwert ist false.

description ist der Text der QuickInfo, die angezeigt wird, wenn der Mauszeiger auf das Steuerelement bewegt wird. Dieser Wert wird auch als lange aufrufbare Beschreibung des Eigenschaftssteuerelements verwendet. Weitere Informationen finden Sie im Thema „Zugriffsmöglichkeiten" auf Seite 175.

descriptionKey gibt die Beschreibung für Lokalisierungszwecke an.

mnemonic ist der Buchstabe, der zusammen mit der Taste Alt gedrückt wird, um dieses Steuerelement zu aktivieren (wenn z. B. der Wert S angegeben ist, kann der Benutzer dieses Steuerelement über Alt+S aktivieren).

mnemonicKey gibt das mnemonische Zeichen für Lokalisierungszwecke an. Wenn weder mnemonic noch mnemonicKey verwendet wird, ist kein Direktaufruf dieser Aktion verfügbar. Weitere Informationen finden Sie im Thema „Zugriffstasten und Tastenkombinationen" auf Seite 115.

Steuerelement für Kontrollkästchen

Definiert ein Kontrollkästchen.

Abbildung 51. Kontrollkästchen

Format

```
<CheckBoxControl Controller-Attribute invert="true_false" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</CheckBoxControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

invert wird selten verwendet. Wenn es auf true gesetzt ist, kehrt es den Effekt des Aktivierens bzw. Inaktivierens des Kontrollkästchens um. Der Standardwert ist false.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

Das folgende Beispiel zeigt den Code, der verwendet wird, um die oben dargestellten Kontrollkästchen zu gestalten (die Kontrollkästchenbeschriftungen werden an anderer Stelle der Spezifikationsdatei definiert). Eine Beschreibung des Layout-Elements finden Sie in „Erweitertes benutzerdefiniertes Layout" auf Seite 153

```
<CheckBoxControl property="means">
     <Layout rowIncrement="0" gridWidth="1" />
</CheckBoxControl>
<CheckBoxControl property="within_groups_correlation">
     <Layout gridColumn="2" />
</CheckBoxControl>
<CheckBoxControl property="univariate_anovas">
     <Layout gridWidth="1" rowIncrement="0" />
```

```
</CheckBoxControl>
<CheckBoxControl property="within_group_covariance">
     <Layout gridColumn="2" />
</CheckBoxControl>
<CheckBoxControl property="box_m">
     <Layout gridWidth="1" rowIncrement="0" />
</CheckBoxControl>
<CheckBoxControl property="separate_groups_covariance">
     <Layout gridColumn="2" />
</CheckBoxControl>
```

Steuerelement für Kontrollkästchengruppe

Definiert ein Set von Kontrollkästchen, die gruppiert und als eine Einheit behandelt werden. Diese Option kann nur zusammen mit einer Aufzählungslisteneigenschaft verwendet werden, die die Mitglieder der Gruppe definiert.

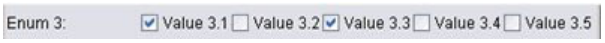

Abbildung 52. Kontrollkästchengruppe

Format

```
<CheckBoxGroupControl Controller-Attribute rows="ganze_Zahl" layoutByRow="true_false"
     useSubPanel="true_false" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</CheckBoxGroupControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`rows` ist eine positive ganze Zahl, die die Anzahl der Zeilen auf dem Bildschirm angibt, die von der Kontrollkästchengruppe belegt werden. Der Standardwert ist 1.

`layoutByRow` gibt an, ob die Kontrollkästchen zuerst innerhalb der Zeile (`true`) oder in der Spalte (`false`) angeordnet werden. Der Standardwert ist `true`. Informationen zur Verwendung von `layoutByRow` mit einer Optionsfeldgruppe finden Sie in „Ändern der Reihenfolge der Steuerelemente" auf Seite 152.

`useSubPanel` gibt an, ob die Kontrollkästchen als Unterfenster angeordnet (`true`) werden oder nicht (`false`). Der Standardwert ist `true`.

Kontrollkästchengruppen werden normalerweise als Unterfenster angeordnet, in denen alle Kästchen der Gruppe enthalten sind. Dies kann jedoch zu Ausrichtungsproblemen führen, wenn das Kontrollkästchen zu einem daneben liegenden Textfeld gehört. Wenn `useSubPanel` auf `false` gesetzt ist, wird dieses Problem vermieden.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

Der Code für die oben dargestellte Kontrollkästchengruppe lautet:

```
<CheckBoxGroupControl property="enum3" label="Enum 3" labelKey="enum3.LABEL"/>
```

Die den einzelnen Kontrollkästchen zugeordneten Beschriftungen und Werte sind im Abschnitt 'Properties' des entsprechenden Knotens definiert:

```
<Property name="enum3" valueType="enum" isList="true" defaultValue="[value1 value3]">
     <Enumeration>
          <Enum value="value1" label="Value 3.1" labelKey="enum3.value1.LABEL"/>
          <Enum value="value2" label="Value 3.2" labelKey="enum3.value2.LABEL"/>
          <Enum value="value3" label="Value 3.3" labelKey="enum3.value3.LABEL"/>
          <Enum value="value4" label="Value 3.4" labelKey="enum3.value4.LABEL"/>
          <Enum value="value5" label="Value 3.5" labelKey="enum3.value5.LABEL"/>
     </Enumeration>
</Property>
```

Steuerelement für Clientverzeichnisauswahl

Definiert ein einzeiliges Textfeld und die zugehörige Schaltfläche, mit der der Benutzer ein Verzeichnis auf dem Client auswählen kann. Das Verzeichnis muss bereits vorhanden sein. Benutzer können in diesem Verzeichnis entweder eine Datei öffnen oder speichern. Dies hängt von der Moduseinstellung ab.

Abbildung 53. Steuerelement für Clientverzeichnisauswahl

Der Benutzer kann entweder den Verzeichnispfad und den Namen direkt in das Textfeld eingeben oder auf die daneben liegende Schaltfläche klicken, um ein Dialogfeld zu öffnen, in dem er ein Verzeichnis auswählen kann.

Format

```
<ClientDirectoryChooserControl Controller-Attribute mode="Auswahlmodus" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</ClientDirectoryChooserControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`mode` legt die Schaltfläche fest, die im Dialogfeld angezeigt wird, in dem der Benutzer ein Verzeichnis auswählt. Folgende Optionen sind verfügbar:

- `open` (Standard) zeigt die Schaltfläche **Öffnen** an.
- `save` zeigt die Schaltfläche **Speichern** an.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

```
<ClientDirectoryChooserControl property="directory2" label="Client Directory"
     labelKey="directory2.LABEL"/>
```

Steuerelement für Clientdateiauswahl

Definiert ein einzeiliges Textfeld und die zugehörige Schaltfläche, mit der der Benutzer eine Datei auf dem Client auswählen kann. Die Datei muss bereits vorhanden sein. Benutzer können die Datei entweder öffnen oder speichern. Dies hängt von der Moduseinstellung ab.

Abbildung 54. Element für die Clientdateiauswahl

Der Benutzer kann entweder den Dateipfad und den Namen direkt in das Textfeld eingeben oder auf die daneben liegende Schaltfläche klicken, um ein Dialogfeld zu öffnen, in dem er eine Datei auswählen kann.

Format

```
<ClientFileChooserControl Controller-Attribute mode="Auswahlmodus" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</ClientFileChooserControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`mode` legt die Schaltfläche fest, die im Dialogfeld angezeigt wird, in dem der Benutzer eine Datei auswählt. Folgende Optionen sind verfügbar:

- `open` (Standard) zeigt die Schaltfläche **Öffnen** an.
- `save` zeigt die Schaltfläche **Speichern** an.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

```
<ClientFileChooserControl property="file2" label="Client File" labelKey="file2.LABEL"/>
```

Steuerelement für Kombinationsfelder

Definiert eine Dropdown-Liste für ein Kombinationsfeld.

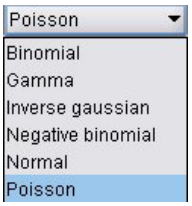

Abbildung 55. Kombinationsfeld

Format

```
<ComboBoxControl Controller-Attribute >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</ComboBoxControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

Im folgenden Beispiel ist der Code aufgeführt, der für die Anordnung der in der vorigen Abbildung dargestellten Dropdown-Liste des Kombinationsfelds verwendet wird:

```
<ComboBoxControl property="distribution" >
     <Layout rowIncrement="0" gridWidth="1" fill="none"/>
</ComboBoxControl>
```

Eine Beschreibung des Layout-Elements finden Sie in „Erweitertes benutzerdefiniertes Layout" auf Seite 153

Hinweis: Die eigentlichen Listeneinträge werden im Abschnitt 'Properties' des entsprechenden Knotens definiert; in diesem Fall als Aufzählungsliste in der Deklaration für die distribution-Eigenschaft:

```
<Property name="distribution" valueType="enum" label="Distribution" labelKey="distribution.
LABEL" defaultValue="NORMAL">
     <Enumeration>
          <Enum value="BINOMIAL" label="Binomial" labelKey="distribution.BINOMIAL.LABEL"/>
          <Enum value="GAMMA" label="Gamma" labelKey="distribution.GAMMA.LABEL"/>
          <Enum value="IGAUSS" label="Inverse gaussian" labelKey="distribution.IGAUSS.LABEL"/>
          <Enum value="NEGBIN" label="Negative binomial" labelKey="distribution.NEGBIN.LABEL"/>
          <Enum value="NORMAL" label="Normal" labelKey="distribution.NORMAL.LABEL"/>
          <Enum value="POISSON" label="Poisson" labelKey="distribution.POISSON.LABEL"/>
     </Enumeration>
</Property>
```

Steuerelement für die Auswahl der Datenbankverbindung

Definiert ein Steuerelement, das es dem Benutzer ermöglicht, eine Datenquelle auszuwählen und eine Verbindung zur Datenbank herzustellen.

Abbildung 56. Steuerelement für die Auswahl der Datenbankverbindung

Benutzer können keinen Text in das Textfeld eingeben, sondern müssen stattdessen auf die Schaltfläche klicken, damit das Standarddialogfeld von IBM SPSS Modeler für Datenbankverbindungen angezeigt wird.

Bei einer erfolgreichen Verbindung werden die Details der Verbindung im Textfeld des Steuerelements für die Datenbankauswahl angezeigt.

Format

```
<DBConnectionChooserControl Controller-Attribute >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</DBConnectionChooserControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

Das folgende Beispiel illustriert, wie das Steuerelement die Definition einer Zeichenfolgeneigenschaft benötigt, die für die Verbindungszeichenfolge verwendet werden kann.

```
<Node ... >
     <Properties>
      ...
```

```
          <Property name="dbconnect" valueType="databaseConnection" />
     </Properties>
      ...
     <UserInterface>
      ...
          <Tabs>
               <Tab label="Database">
                    <PropertiesPanel>
                         <DBConnectionChooserControl property="dbconnect" label="Database
                          Connection"/>
                     ...
                    </PropertiesPanel>
                     ...
     </UserInterface>
</Node>
```

Steuerelement für die Auswahl der Datenbanktabelle

Definiert ein Textfeld und die zugehörige Schaltfläche, mit der ein Benutzer im Anschluss an eine erfolgreiche Datenbankverbindung eine Datenbanktabelle auswählen kann.

Abbildung 57. Steuerelement für die Auswahl der Datenbanktabelle

Benutzer können den Tabellennamen entweder direkt in das Textfeld eingeben oder auf die Schaltfläche klicken und ihn aus einer Liste auswählen.

Format

```
<DBTableChooserControl connectionProperty="Datenbankverbindungseigenschaft" Controller-Attribute >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</DBTableChooserControl>
```

Dabei gilt Folgendes:

`connectionProperty` ist der Name einer bereits definierten Datenbankverbindungseigenschaft. Dies ist der Wert des `property`-Attributs des `DBConnectionChooserControl`-Elements, das zuvor für den Knoten definiert wurde.

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

Dieses Beispiel schließt an das für `DBConnectionChooserControl` an und zeigt wie Sie außerdem ein `DBTableChooserControl`-Element aufnehmen können, um nach dem erfolgreichen Herstellen einer Datenbankverbindung eine Datenbanktabelle auszuwählen.

```
<Node ... >
     <Properties>
      ...
          <Property name="dbconnect" valueType="databaseConnection" />
          <Property name="dbtable" valueType="string" />
     </Properties>
       ...
     <UserInterface>
```

```
      ...
          <Tabs>
               <Tab label="Database">
                    <PropertiesPanel>
                         <DBConnectionChooserControl property="dbconnect" label="Database
                          connection"/>
                         <DBTableChooserControl property="dbtable" connectionProperty=
                          "dbconnect" label="Database Table" />
                     ...
                    </PropertiesPanel>
      ...
     </UserInterface>
</Node>
```

Steuerelement für die Auswahl mehrerer Felder

Definiert ein Steuerelement, mit dem der Benutzer einen oder mehrere Feldnamen aus einer Liste auswählen kann.

Abbildung 58. Steuerelement für die Auswahl mehrerer Felder

Wenn der Benutzer auf dieses Steuerelement klickt, wird eine Liste mit Feldern angezeigt, die der Benutzer auswählen kann.

Das Set besteht aus allen Feldern, die an diesem Knoten sichtbar sind. Wenn Felder oberhalb dieses Knotens gefiltert wurden, sind nur die Felder sichtbar, die den Filter durchlaufen haben. Die Liste kann noch weiter eingeschränkt werden, indem angegeben wird, dass nur Felder mit bestimmten Speicher- und Datentypen zur Auswahl stehen sollen.

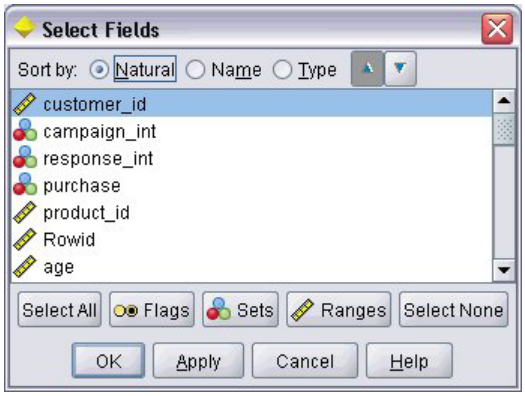

Abbildung 59. Liste mit mehreren Feldern

Alle Steuerelemente für die Auswahl mehrerer Felder legen ein Eigenschaftsattribut fest, das an anderer Stelle in der Datei deklariert ist und definiert wie die Liste im Knotendialogfeld angezeigt wird.

Format

```
<MultiFieldChooserControl Controller-Attribute storage="Speichertypen" onlyNumeric="true_false"
     onlySymbolic="true_false" onlyDatetime="true_false" types="Datentypen"
     onlyRanges="true_false" onlyDiscrete="true_false" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</MultiFieldChooserControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

Sie können die Feldliste weiter einschränken, indem Sie außerdem zwei weitere Attribute angeben, von denen eines aus der folgenden Liste stammen muss:

- `storage` ist eine Listeneigenschaft, die den Speichertyp der Felder angibt, die in der Liste zulässig sein sollen. So bedeutet `storage="[integer real]"`, dass nur Felder mit diesen Speichertypen aufgelistet werden. Die möglichen Speichertypen können Sie der Tabelle unter „Daten- und Speichertypen" auf Seite 186 entnehmen.
- `onlyNumeric` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit numerischem Speichertyp aufgelistet werden sollen.
- `onlySymbolic` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit symbolischem Speichertyp (also Zeichenfolgen) aufgelistet werden sollen.
- `onlyDatetime` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit einem Speichertyp für Datum und Uhrzeit aufgelistet werden sollen.

Das zweite angegebene Attribut muss aus dieser Liste stammen:

- `types` ist eine Listeneigenschaft, die den Datentyp der Felder angibt, die in der Liste zulässig sein sollen. So bedeutet `types="[range flag]"`, dass nur Felder mit diesen Speichertypen aufgelistet werden. Folgende Datentypen sind möglich:

  `range`

  `flag`

  `set`

  `orderedSet`

  `numeric`

  `discrete`

  `typeless`
- `onlyRanges` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit dem Datentyp "range" aufgelistet werden sollen.
- `onlyDiscrete` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit einem Datentyp 'discrete' (also 'flag', 'set' oder 'typeless') aufgelistet werden sollen.

So stellt beispielsweise ein Steuerelement mit der Angabe `storage="[integer]"` und `types="[flag]"` sicher, dass nur ganzzahlige Felder, bei denen es sich um Flags handelt, in der Liste angezeigt werden.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Hinweis: Dieses Steuerelement wird nur in Knotenelementdefinitionen verwendet. Verwenden Sie folgendes Format, wenn Sie ein Auswahlelement für mehrere Felder ohne Ausgabedatenmodelldefinition festlegen möchten:

```
<OutputDataModel mode="Modus">
 ...
    <ForEach var="field" inProperty="Eigenschaftsname">
         <AddField name="${Feldname}_NEW" fieldRef="${Feldname}" />
    </ForEach>
 ...
</OutputDataModel>
```

Weitere Informationen finden Sie im Thema „Ausgabedatenmodell" auf Seite 59. Eine Beschreibung des Elements `ForEach` finden Sie in „Iteration mit dem ForEach-Element" auf Seite 68. Eine Beschreibung zu `AddField` finden Sie in „Hinzufügen eines Felds" auf Seite 65.

Beispiel

Im folgenden Beispiel ist der Code aufgeführt, mit dem das Steuerelement für die Auswahl mehrerer Felder aus der vorigen Abbildung angegeben wird.

```
<MultiFieldChooserControl property="inputs" >
     <Enabled>
          <Condition control="custom_fields" op="equals" value="true"/>
     </Enabled>
</MultiFieldChooserControl>
```

Der Abschnitt `Enabled` sorgt dafür, dass das Steuerelement nur dann aktiviert ist, wenn das Steuerelement `custom_fields` ausgewählt ist.

Hinweis: Der Inhalt dieser Liste wird durch die Deklaration für die Eigenschaft `inputs` im Abschnitt 'Properties' des entsprechenden Knotens bestimmt:

```
<Property name="inputs" valueType="string" isList="true" label="Inputs" labelKey="inputs.
LABEL"/>
```

Steuerelement für die Auswahl mehrerer Elemente

Definiert ein Steuerelement, mit dem der Benutzer mindestens ein Element aus einer Werteliste auswählen kann. Es ordnet eine Eigenschaft einem Katalog zu, der eine Werteliste enthält. Weitere Informationen finden Sie im Thema „Kataloge" auf Seite 41.

Abbildung 60. Steuerelement für die Auswahl mehrerer Elemente

Format

```
<MultiItemChooserControl Controller-Attribute catalog="Katalogname" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</MultiItemChooserControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`catalog` (erforderlich) ist der Name für den zuzuordnenden Katalog. Die Bibliothek, aus der der Katalog bezogen wird, wird im Element `Module` des Abschnitts "Execution" angegeben. Weitere Informationen finden Sie im Thema „Module" auf Seite 57.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

```
<MultiItemChooserControl property="selection2" catalog="cat2" />
```

Die vom Attribut `property` (`selection2` in diesem Fall) referenzierte Eigenschaft muss über ein Attribut `isList="true"` verfügen. Eine Erklärung und ein Beispiel für die Verwendung von `MultiItemChooserControl` finden Sie in „Kataloge" auf Seite 41.

Steuerelement für das Kennwortfeld

Definiert ein einzeiliges Textfeld, in dem Zeichen bei der Eingabe ausgeblendet werden.

Abbildung 61. Steuerelement für das Kennwortfeld

Format

```
<PasswordBoxControl Controller-Attribute columns="ganze_Zahl" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</PasswordBoxControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`columns` ist eine positive ganze Zahl, die die Anzahl der Zeichenspalten definiert, die das Kennwortfeld belegt. Der Standardwert ist 20.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

```
<PasswordBoxControl property="encrypted_string1" label="Encrypted string 1" labelKey=
"encryptedString1.LABEL"/>
```

Das Textfeld wird verschlüsselt, indem es einer Eigenschaft zugeordnet wird, die im Abschnitt 'Properties' des zugehörigen Knotens als verschlüsselte Zeichenfolge definiert ist:

```
<Property name="encrypted_string1" valueType="encryptedString"/>
```

Eigenschaftssteuerelement

Ein Eigenschaftssteuerelement ist ein vollständig benutzerdefinierbares Steuerelement, mit dem Benutzer Eigenschaften für den Knoten eingeben können. Die Verarbeitung wird durch eine vom Benutzer geschriebene Java-Klasse durchgeführt. Die folgende Abbildung ist ein Beispiel für ein Eigenschaftssteuerelement.

Abbildung 62. Dialogfeldabschnitt mit hervorgehobenem Beispiel für ein Eigenschaftssteuerelement

Format

```
<PropertyControl Controller-Attribute controlClass="Java-Klasse" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</PropertyControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`controlClass` (erforderlich) ist der Pfad innerhalb einer *JAR*-Datei zu der Java-Klasse, die das Eigenschaftssteuerelement implementiert. *Hinweis*: Die *JAR*-Datei ist in einem `JarFile`-Element im Abschnitt 'Resources' deklariert. Weitere Informationen finden Sie im Thema „JAR-Dateien" auf Seite 35.)

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

```
<PropertyControl property="target_field_values_specify" labelAbove="true"
     controlClass="com.spss.clef.selflearning.propertycontrols.list.CustomListControl" label=""
     labelKey="target_field_values_specify.LABEL">
     <Enabled>
          <Condition control="target_field_values" op="equals" value="specify"/>
     </Enabled>
     <Layout rowIncrement="2" />
</PropertyControl>
```

Das Eigenschaftssteuerelement ist einer Eigenschaft zugeordnet, die im Abschnitt 'Properties' für den entsprechenden Knoten definiert ist:

```
<Property name="target_field_values_specify" valueType="string" isList="true" label=""
     labelKey="target_field_values_specify.LABEL"/>
```

Steuerelement für eine Optionsfeldgruppe

Definiert ein Set von Optionsfeldern, die gleichzeitig ausgewählt werden können.

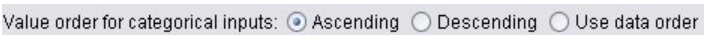

Abbildung 63. Steuerelement für eine Optionsfeldgruppe

Jedes Steuerelement für eine Optionsfeldgruppe hat ein Eigenschaftsattribut, das die Gruppe einer bestimmten Eigenschaft zuordnet. Diese Eigenschaft wird an anderer Stelle in der Datei definiert und gibt die Schaltflächen an, aus denen die Gruppe besteht.

Die zugeordnete Eigenschaft kann entweder eine Aufzählungsliste oder eine boolesche Eigenschaft sein. Für Aufzählungslisten (mit dem Eigenschaftsattribut `valueType="enum"`), wird für jeden `enum`-Wert ein Optionsfeld angezeigt. Für boolesche Eigenschaften (mit `valueType="boolean"`) werden immer zwei Optionsfelder angezeigt.

Format

```
<RadioButtonGroupControl Controller-Attribute
     rows="ganze_Zahl" layoutByRow="true_false" useSubPanel="true_false"
     falseLabel="Schaltflächenbeschriftung" falseLabelKey="Beschriftungsschlüssel"
     trueLabel="?Schaltflächenbeschriftung"
     trueLabelKey="Beschriftungsschlüssel" trueFirst="true_false" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</RadioButtonGroupControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`rows` ist eine positive ganze Zahl, die die Anzahl der Bildschirmzeilen festlegt, in denen die Gruppe angezeigt wird. Der Standardwert ist 1.

layoutByRow gibt an, ob die Optionsfelder zuerst innerhalb der Zeile (true) oder in der Spalte (false) angeordnet werden. Der Standardwert ist true. Ein Beispiel zur Verwendung von layoutByRow mit einer Optionsfeldgruppe finden Sie in „Ändern der Reihenfolge der Steuerelemente” auf Seite 152.

useSubPanel gibt an, ob die Optionsfelder als Unterfenster angeordnet (true) werden oder nicht (false). Der Standardwert ist true.

Optionsfeldgruppen werden normalerweise als Unterfenster angeordnet, in denen alle Schaltflächen der Gruppe enthalten sind. Dies kann jedoch zu Ausrichtungsproblemen führen, wenn die Optionsfeldgruppe zu einem daneben liegenden Textfeld gehört. Wenn useSubPanel auf false gesetzt ist, wird dieses Problem vermieden.

falseLabel ist die Beschriftung für den Wert "false" einer booleschen Eigenschaft (siehe unten das zweite Beispiel). Wird nur mit booleschen Eigenschaften verwendet, sofern es benötigt wird.

falseLabelKey gibt die Beschriftung für "false" für Lokalisierungszwecke an.

trueLabel ist die Beschriftung für den Wert "true" einer booleschen Eigenschaft (siehe unten das zweite Beispiel). Wird nur mit booleschen Eigenschaften verwendet, sofern es benötigt wird.

trueLabelKey gibt die Beschriftung für "true" für Lokalisierungszwecke an.

trueFirst sorgt in der Einstellung true dafür, dass die Anzeigereihenfolge der Schaltflächen für eine boolesche Eigenschaft umgekehrt wird, sodass die Schaltfläche für den Wert "true" zuerst angezeigt wird. Der Standardwert ist false, was bedeutet, dass die Schaltfläche für den "false"-Wert zuerst angezeigt wird.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout” auf Seite 153.

Beispiele

Das erste Beispiel zeigt den Code für die oben dargestellte Optionsfeldgruppe.

```
<RadioButtonGroupControl property="value_order" labelWidth="2">
     <Layout gridWidth="4"/>
</RadioButtonGroupControl>
```

Eine Beschreibung des Layout-Elements finden Sie in „Erweitertes benutzerdefiniertes Layout” auf Seite 153.

Hinweis: Die Anzahl der Schaltflächen und deren Beschriftungen werden im Abschnitt 'Properties' des entsprechenden Knotens definiert; in diesem Fall als Aufzählungsliste in der Deklaration für die Eigenschaft value_order. Diese Deklaration enthält außerdem die Beschriftung für die Gruppe als solche:

```
<Property name="value_order" valueType="enum" label="Value order for categorical
   inputs" labelKey="value_order.LABEL">
     <Enumeration>
          <Enum value="Ascending" label="Ascending" labelKey="value_order.Ascending.LABEL"/>
          <Enum value="Descending" label="Descending" labelKey="value_order.Descending.
           LABEL"/>
          <Enum value="DataOrder" label="Use data order" labelKey="value_order.UseDataOrder.
           LABEL"/>
     </Enumeration>
</Property>
```

Das zweite Beispiel veranschaulicht die Verwendung von falseLabel und trueLabel für eine Optionsfeldgruppe, die eine boolesche Eigenschaft steuert, wie etwa, ob Standardeinstellungen oder benutzerdefinier-

te Einstellungen aktiviert sind.

Abbildung 64. Eine Optionsfeldgruppe, die eine boolesche Eigenschaft steuert

Der Code, um dies zu erreichen:

```
<RadioButtonGroupControl property="boolean5" label="Boolean 5" labelKey="boolean5.LABEL"
     falseLabel="Standard" falseLabelKey="boolean5.false.LABEL" trueLabel="Custom"
     trueLabelKey="boolean5.true.LABEL" />
```

In diesem Fall werden die Schaltflächen- und Gruppenbeschriftungen im Element `RadioButtonGroupControl` selbst definiert. Die Eigenschaft, der die Gruppe zugeordnet ist, wird im Abschnitt "Properties" für den Knoten definiert.

```
<Property name="boolean5" valueType="boolean" defaultValue="false"/>
```

Steuerelement für die Serververzeichnisauswahl

Definiert ein einzeiliges Textfeld und die zugehörige Schaltfläche, mit der der Benutzer ein Verzeichnis auf dem Server auswählen kann. Das Verzeichnis muss bereits vorhanden sein. Benutzer können in diesem Verzeichnis entweder eine Datei öffnen oder speichern. Dies hängt von der Moduseinstellung ab.

Abbildung 65. Steuerelement für die Serververzeichnisauswahl

Der Benutzer kann entweder den Verzeichnispfad und den Namen direkt in das Textfeld eingeben oder auf die daneben liegende Schaltfläche klicken, um ein Dialogfeld zu öffnen, in dem er ein Verzeichnis auswählen kann.

Format

```
<ServerDirectoryChooserControl Controller-Attribute mode="Auswahlmodus" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</ServerDirectoryChooserControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`mode` legt die Schaltfläche fest, die im Dialogfeld angezeigt wird, in dem der Benutzer ein Verzeichnis auswählt. Folgende Optionen sind verfügbar:

- `open` (Standard) zeigt die Schaltfläche **Öffnen** an.
- `save` zeigt die Schaltfläche **Speichern** an.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

```
<ServerDirectoryChooserControl property="directory1" label="Server Directory"
     labelKey="directory1.LABEL"/>
```

Steuerelement für die Serverdateiauswahl

Definiert ein einzeiliges Textfeld und die zugehörige Schaltfläche, mit der der Benutzer eine Datei auf dem Server auswählen kann. Die Datei muss bereits vorhanden sein. Benutzer können die Datei entweder

öffnen oder speichern. Dies hängt von der Moduseinstellung ab.

Abbildung 66. Element für die Serverdateiauswahl

Der Benutzer kann entweder den Dateipfad und den Namen direkt in das Textfeld eingeben oder auf die daneben liegende Schaltfläche klicken, um ein Dialogfeld zu öffnen, in dem er eine Datei auswählen kann.

Format

```
<ServerFileChooserControl Controller-Attribute mode="Auswahlmodus" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</ServerFileChooserControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`mode` legt die Schaltfläche fest, die im Dialogfeld angezeigt wird, in dem der Benutzer eine Datei auswählt. Folgende Optionen sind verfügbar:

- `open` (Standard) zeigt die Schaltfläche **Öffnen** an.
- `save` zeigt die Schaltfläche **Speichern** an.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

```
<ServerFileChooserControl property="file1" label="Server File" labelKey="file1.LABEL"/>
```

Steuerelement für die Auswahl eines einzelnen Felds

Definiert ein Steuerelement, mit dem der Benutzer ein einzelnes Feld aus einer Liste auswählen kann.

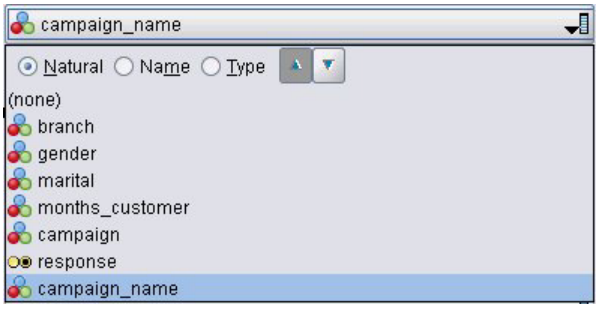

Abbildung 67. Steuerelement für die Auswahl eines einzelnen Felds

Wenn der Benutzer auf dieses Steuerelement klickt, wird eine Feldliste angezeigt, aus der ein einzelnes Feld ausgewählt werden kann.

Das Set besteht aus allen Feldern, die an diesem Knoten sichtbar sind. Wenn Felder oberhalb dieses Knotens gefiltert wurden, sind nur die Felder sichtbar, die den Filter durchlaufen haben. Die Liste kann noch weiter eingeschränkt werden, indem angegeben wird, dass nur Felder mit bestimmten Speicher- und Datentypen zur Auswahl stehen sollen.

Format

```
<SingleFieldChooserControl Controller-Attribute storage="Speichertypen" onlyNumeric="true_
     false" onlySymbolic="true_false" onlyDatetime="true_false" types="Datentypen"
     onlyRanges="true_false" onlyDiscrete="true_false" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</SingleFieldChooserControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

Sie können die Feldliste weiter einschränken, indem Sie außerdem zwei weitere Attribute angeben, von denen eines aus der folgenden Liste stammen muss:

- `storage` ist eine Listeneigenschaft, die den Speichertyp der Felder angibt, die in der Liste zulässig sein sollen. So bedeutet `storage="[integer real]"`, dass nur Felder mit diesen Speichertypen aufgelistet werden. Die möglichen Speichertypen können Sie der Tabelle unter „Daten- und Speichertypen" auf Seite 186 entnehmen.
- `onlyNumeric` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit numerischem Speichertyp aufgelistet werden sollen.
- `onlySymbolic` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit symbolischem Speichertyp (also Zeichenfolgen) aufgelistet werden sollen.
- `onlyDatetime` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit einem Speichertyp für Datum und Uhrzeit aufgelistet werden sollen.

Das zweite angegebene Attribut muss aus dieser Liste stammen:

- `types` ist eine Listeneigenschaft, die den Datentyp der Felder angibt, die in der Liste zulässig sein sollen. So bedeutet `types="[range flag]"`, dass nur Felder mit diesen Speichertypen aufgelistet werden. Folgende Datentypen sind möglich:

  `range`

  `flag`

  `set`

  `orderedSet`

  `numeric`

  `discrete`

  `typeless`
- `onlyRanges` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit dem Datentyp "range" aufgelistet werden sollen.
- `onlyDiscrete` (sofern auf `true` (wahr) gesetzt) gibt an, dass nur Felder mit einem Datentyp 'discrete' (also 'flag', 'set' oder 'typeless') aufgelistet werden sollen.

So stellt beispielsweise ein Steuerelement mit der Angabe `storage="[integer]"` und `types="[flag]"` sicher, dass nur ganzzahlige Felder, bei denen es sich um Flags handelt, in der Liste angezeigt werden.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Hinweis: Dieses Steuerelement wird nur in Knotendefinitionen verwendet. Verwenden Sie folgendes Format, wenn Sie ein Auswahlelement für mehrere Felder ohne Ausgabedatenmodelldefinition festlegen möchten:

```
<OutputDataModel mode="Modus">
 ...
    <ForEach var="field" from="1" to="${ganze_Zahl}">
         <AddField name="${Zeichenfolge}_${Feld}" fieldRef="${Feldreferenz}" />
    </ForEach>
 ...
</OutputDataModel>
```

Weitere Informationen finden Sie im Thema „Ausgabedatenmodell" auf Seite 59. Eine Beschreibung des Elements `ForEach` finden Sie in „Iteration mit dem ForEach-Element" auf Seite 68. Eine Beschreibung zu `AddField` finden Sie in „Hinzufügen eines Felds" auf Seite 65.

Beispiel

Im folgenden Beispiel ist der Code aufgeführt, mit dem das Steuerelement für die Auswahl eines einzelnen Felds aus der vorigen Abbildung angegeben wird.

```
<SingleFieldChooserControl property="target"  storage="string" onlyDiscrete="true"/>
```

Inhalt: Der Inhalt der Liste wird im Abschnitt 'Properties' des entsprechenden Knotens definiert; in diesem Fall in der Deklaration für die Eigenschaft `target`:

```
<Property name="target" valueType="string" label="Target field" labelKey="target.LABEL"/>
```

Steuerelement für die Auswahl eines einzelnen Elements

Definiert ein Steuerelement, mit dem der Benutzer ein einzelnes Element aus einer Werteliste auswählen kann. Es ordnet eine Eigenschaft einem Katalog zu, der eine Werteliste enthält. Weitere Informationen finden Sie im Thema „Kataloge" auf Seite 41.

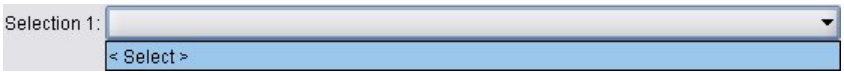

Abbildung 68. Steuerelement für die Auswahl eines einzelnen Elements

Format

```
<SingleItemChooserControl Controller-Attribute catalog="Katalogname" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</MultiItemChooserControl
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`catalog` (erforderlich) ist der Name für den zuzuordnenden Katalog. Die Bibliothek, aus der der Katalog bezogen wird, wird im Element `Module` des Abschnitts "Execution" angegeben. Weitere Informationen finden Sie im Thema „Module" auf Seite 57.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

```
<SingleItemChooserControl property="selection1" catalog="cat1" />
```

Eine Erklärung und ein Beispiel für die Verwendung dieses Steuerelements finden Sie in „Kataloge" auf Seite 41.

Drehfeldsteuerelement

Definiert ein Drehfeld (ein numerisches Feld mit aufwärts und abwärts weisenden Pfeilen, mit denen der Feldwert geändert wird).

Abbildung 69. Drehfeld

Format

```
<SpinnerControl Controller-Attribute columns="ganze_Zahl" stepSize="Inkrement"
     minDecimalDigits="Anzahl" maxDecimalDigits="Anzahl" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</SpinnerControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`columns` ist eine positive ganze Zahl, die die Anzahl der Zeichenfolgenspalten festlegt, über die sich das Steuerelement erstreckt. Der Standardwert ist 5.

`stepSize` ist eine Dezimalzahl, die angibt, um welchen Wert der Feldwert verändert wird, wenn der Benutzer eine der Pfeiltasten anklickt. Der Standardwert ist 1,0.

`minDecimalDigits` ist der Mindestwert, der für den Feldwert angezeigten Dezimalstellen. Der Standardwert ist 1.

`maxDecimalDigits` ist der Höchstwert, der für den Feldwert angezeigten Dezimalstellen.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

Im folgenden Beispiel ist der Code aufgeführt, mit dem das Steuerelement für das Drehfeld aus der vorigen Abbildung angegeben wird.

```
<SpinnerControl property="double1" label="Double 1" labelKey="double1.LABEL"/>
```

Die Präzision und der Gültigkeitsbereich für die numerischen Feldinhalte werden im Eigenschaftsabschnitt des entsprechenden Knotens definiert; in diesem Fall in der Deklaration für die Eigenschaft `double1`:

```
<Property name="double1" valueType="double" min="0" max="100"/>
```

Tabellensteuerelement

Definiert ein Tabellenlayoutelement, das in einem Knotendialogfeld oder einem Knotenausgabefenster angezeigt wird.

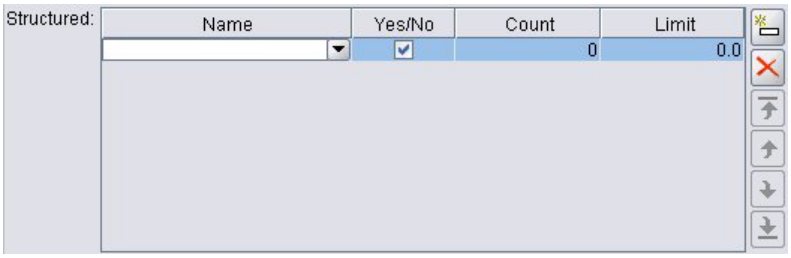

Abbildung 70. Tabellensteuerelement

Format

```
<TableControl Controller-Attribute rows="ganze_Zahl" columns="ganze_Zahl" columnWidths="Liste" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</TableControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`rows` ist eine positive ganze Zahl, die die Anzahl der Tabellenzeilen angibt, die auf dem Bildschirm angezeigt werden. Der Standardwert ist 8.

`columns` ist eine positive ganze Zahl, die die Anzahl der Zeichenfolgenspalten festlegt, über die sich die Tabelle erstreckt. Der Standardwert ist 20.

`columnwidths` ist eine Liste von Werten, mit der die relative Breite der Spalten angegeben wird. Beispiel: Die Werte `[30 5 10]` geben an, dass Spalte 1 dreimal so breit ist wie Spalte 3.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Das Attribut *ColumnControl* im folgenden Beispiel wird in „Spaltensteuerelement" auf Seite 149 beschrieben.

Beispiel

Der Code, mit dem das in der vorigen Abbildung dargestellte Tabellensteuerelement angegeben wird, lautet:

```
<TableControl property="structure1" allowReorder="true" label="Structured"
  labelKey="structure1.LABEL" columnWidths="[20 6 10 10]">
     <ColumnControl column="0" editor="fieldValue" fieldControl="field1"/>
</TableControl>
```

Die Struktur des Tabellensteuerelements wird im Abschnitt für gemeinsame Objekte der Spezifikationsdatei als Eigenschaftstyp festgelegt:

```
<PropertyType id="shared_structure1" valueType="structure" isList="true">
     <Structure>
          <Attribute name="id" valueType="string" label="Name" labelKey="structure1.id.LABEL"/>
          <Attribute name="yesno" valueType="boolean" label="Yes/No" labelKey="structure1.
           yesno.LABEL" defaultValue="true"/>
          <Attribute name="count" valueType="integer" label="Count" labelKey="structure1.
           count.LABEL" defaultValue="0"/>
```

```
          <Attribute name="limit" valueType="double" label="Limit" labelKey="structure1.
           limit.LABEL" defaultValue="0.0"/>
     </Structure>
</PropertyType>
```

In der Knotenspezifikation wird dann die ID dieses Eigenschaftstyps mittels einer Eigenschaftsdeklaration der ID des Tabellensteuerelements zugeordnet:

```
<Property name="structure1" type="shared_structure1"/>
```

Wenn in einem Script Bezug auf den Knoten genommen wird, können in eckigen Klammern [] die Werte der Eigenschaft für die Liste und in geschweiften Klammern {} für die Struktur angegeben werden. Beispiel: So legen Sie ein Raster aus zwei Strukturen für die Eigenschaft `structure1` fest:

```
set :Knoten-ID.structure1 = [{"hello" true 4 0.21} {"bye" false 5 0.95}]
```

Beachten Sie, dass die Reihenfolge der Werte mit der Reihenfolge übereinstimmen muss, in der die `Attribute`-Definitionen vorgenommen werden.

Spaltensteuerelement: Definiert das Layout von Spalten in Tabellen.

Alle Spalten des Tabellensteuerelements nutzen gemeinsam denselben Datentyp. Auf dieser Grundlage können Sie einen Editor für eine bestimmte Spalte für alle Zeilen angeben. Es muss für jede Spalte also nur ein Editor angegeben werden. Wenn Spalte X beispielsweise erfordert, dass der Benutzer eine ganze Zahl eingibt, können Sie für Spalte X einen Ganzzahleditor festlegen.

Das Attribut *editor* gibt den Editortyp der Spalte an. Es gibt die folgenden vier Editortypen: *default*, *field*, *fieldValue*, und *enumeration*. Jeder Editortyp ist ein bearbeitbares Kombinationsfeld.

Beim Editortyp *fieldValue* enthält die Dropdown-Liste alle Werte des Felds, das Sie in *fieldControl* angegeben haben. Daher definiert das folgende XML-Element, dass wenn Sie Spalte 0 bearbeiten, der Editor ein Kombinationsfeld ist und die Dropdown-Liste enthält alle Werte von field1:

```
<ColumnControl column="0" editor="fieldValue" fieldControl="field1" />
```

Sie können *fieldControl* durch *fieldDirection* ersetzen. Beispiel: *fieldDirecton="[in out]"* bedeutet, dass die Dropdown-Liste des Kombinationsfelds alle Werte des ersten Felds enthält, dessen Richtung *in* oder *out* ist.

Beim Editortyp *field* enthält die Dropdown-Liste alle Felder, die der Filterbedingung entsprechen. Im folgenden Beispiel wird definiert, dass Spalte 0 ein Feldkombinationsfeld als Editor verwendet und die Dropdown-Liste alle Felder für reele Zahlen und ganze Zahlen enthält:

```
<ColumnControl column="0" editor="field" storage="[real integer]" />
```

Darüber hinaus können Sie mit dem Attribut *types* die Maßtypen angeben, die die Felder in der Dropdown-Liste berücksichtigen sollen. Die entsprechenden booleschen Attribute für das Feld sind *onlyRanges*, *onlyDiscrete*, *onlyNumeric*, *onlySymbolic* und *onlyDatetime*.

Steuerelement für Textbereich

Definiert ein mehrzeiliges Texteingabefeld.

Abbildung 71. Steuerelement für Textbereich

Format

```
<TextAreaControl Controller-Attribute rows="ganze_Zahl" columns="ganze_Zahl" wrapLines="true_false" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</TextAreaControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`rows` ist eine positive ganze Zahl, die die Anzahl der Bildschirmzeilen angibt, die der Textbereich belegt. Der Standardwert ist 8.

`columns` ist eine positive ganze Zahl, die die Anzahl der Zeichenfolgenspalten festlegt, über die sich der Textbereich erstreckt. Der Standardwert ist 20.

`wrapLines` gibt an, ob lange Zeilen umgebrochen werden (`true`) oder ob ein horizontales Blättern erforderlich ist, um lange Textzeilen zu lesen (`false`). Der Standardwert ist `true`.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

Der Code für das oben dargestellte Beispiel lautet:

```
<TextAreaControl property="string2" label="String 2" labelKey="string2.LABEL"/>
```

In diesem Fall wird die Beschriftung für den Textbereich in der Deklaration für das Steuerelement für Textbereiche definiert, während der Eingabedatentyp im Abschnitt "Properties" für den relevanten Knoten definiert wird; in der Deklaration für die Eigenschaft `string2`:

```
<Property name="string2" valueType="string"/>
```

Textfeldsteuerelement

Definiert ein einzeiliges Texteingabefeld.

Abbildung 72. Textfeldsteuerelement

Format

```
<TextBoxControl Controller-Attribute columns="ganze_Zahl" >
     -- erweiterte Optionen für benutzerdefiniertes Layout --
</TextBoxControl>
```

Dabei gilt Folgendes:

Controller-Attribute werden unter „Controller-Attribute" auf Seite 130 beschrieben.

`columns` ist eine positive ganze Zahl, die die Anzahl der Zeichenfolgenspalten festlegt, über die sich das Textfeld erstreckt. Der Standardwert ist 20.

Die erweiterten Optionen für das benutzerdefinierte Layout ermöglichen eine Feinsteuerung für Positionierung und Anzeige von Anzeigenkomponenten. Weitere Informationen finden Sie im Thema „Erweitertes benutzerdefiniertes Layout" auf Seite 153.

Beispiel

Der Code für das oben dargestellte Textfeld lautet:

```
<TextBoxControl property="string1" label="String 1" labelKey="string1.LABEL"/>
```

Der Eingabedatentyp für das Textfeld wird im Abschnitt "Properties" des entsprechenden Knotens definiert; in diesem Fall in der Deklaration für die Eigenschaft `string1`:

```
<Property name="string1" valueType="string"/>
```

Layouts für Eigenschaftssteuerelemente

In diesem Abschnitt werden die Methoden für Standardlayouts beschrieben, die für Dialogfenster und Fenster verwendet werden, sowie Verfahren, mit denen diese verändert und eigene benutzerdefinierte Layouts definiert werden können.

Standardsteuerelementlayout

Ein Eigenschaftsfenster kann als zweidimensionales Raster von Zellen betrachtet werden. Jede Zeile kann eine andere Höhe und jede Spalte eine andere Breite haben. Die Komponenten der Benutzerschnittstelle können mehreren zusammenhängenden Zellen zugeordnet werden, auch wenn Komponenten der Benutzerschnittstelle normalerweise nur einer Zelle zugeordnet werden.

Standardmäßig wird ein Eigenschaftssteuerelement einer Zeile zugeordnet und jedes Steuerelement belegt zwei Zeilen: eine für die Beschriftung und eine für die Steuerelementkomponente(n). Die Spalte, die die Beschriftung enthält, wird an die Breite der breitesten Beschriftung angepasst. Wenn in der Spezifikationsdatei z. B. folgende Elemente angegeben sind:

```
<TextBoxControl property="string1" label="String 1"/>
<PasswordBoxControl property="encryptedString1" label="Encrypted string 1"/>
<TextAreaControl property="string2" label="String 2"/>
```

Der hieraus resultierende Bereich ist in der folgenden Abbildung dargestellt.

Abbildung 73. Einfacher Eigenschaftsbereich

Beachten Sie, dass das Zeichen ":" am Ende der Beschriftung automatisch hinzugefügt wird.

Ein Eigenschaftssteuerelement, das mehrere Komponenten der Benutzerschnittstelle enthält, erstellt seinen eigenen sichtbaren rechteckigen Bereich, in dem diese Komponenten angeordnet werden. Die Elemente `RadioButtonGroupControl` und `CheckBoxGroupControl` sind Beispiele für solche Steuerelemente.

Beachten Sie, dass die Form des rechteckigen Bereichs, in dem die Komponenten angeordnet sind, sich abhängig vom Eigenschaftensteuerelement ändern kann. Daher ist das Layout bei verschiedenen Steuerelementen nicht immer genau ausgerichtet.

Einige Eigenschaftensteuerelemente umfassen Komponenten, die die Komponentenspalte komplett füllen und sich horizontal anpassen, wenn die Fensterbreite vergrößert oder verkleinert wird. Beispiele dafür sind die Steuerelemente, die von den Elementen `TextBoxControl`, `PasswordBoxControl` und `TextAreaControl` angegeben werden. Dies gilt jedoch nicht für alle Komponenten. Kontrollkästchen und Drehfelder nehmen z. B. einen fest definierten horizontalen Raum ein, selbst wenn sich die Fensterbreite ändert:

Benutzerdefiniertes Steuerelementlayout

Das Standardlayout für Steuerelemente kann nach verschiedenen Verfahren geändert werden, von denen einige einfach und andere komplexer sind.

Einfaches benutzerdefiniertes Layout

Drei einfache Verfahren zur Anpassung des Steuerelementlayouts:

- Positionieren einer Beschriftung über der zugehörigen Komponente
- Änderung der Anzahl der Zeilen, über die hinweg die Steuerelemente angeordnet sind
- Änderung der Reihenfolge, in der die Steuerelemente angeordnet sind

Positionieren einer Beschriftung über der zugehörigen Komponente: Sie können eine Beschriftung in einer separaten Zeile über der zugehörigen Komponente positionieren, indem Sie das `labelAbove`-Attribut des Steuerelements auf `true` setzen. Beispiel:

```
<TextBoxControl property="string0" label="String 0" labelAbove="true"/>
<TextBoxControl property="string1" label="String 1"/>
<PasswordBoxControl property="encryptedString1" label="Encrypted string 1"/>
```

Neben der Positionierung der Beschriftung über der Komponente werden die Benutzerschnittstellenkomponenten der Beschriftungsspalte der Anzeige zugeordnet. Dies ergibt folgendes Fenster mit der Beschriftung **String 0**, das über dem zugehörigen Feld angezeigt wird.

Abbildung 74. Fenster mit Feldbeschriftung in separater Zeile

Ändern der Zeilenzahl: Standardmäßig werden Optionsfeld- und Kontrollkästchengruppen in einer Reihe angeordnet, und die Breite des Dialogfelds wird entsprechend angepasst. Wenn eine Optionsfeld- und Kontrollkästchengruppe eine große Anzahl an Optionen umfasst, kann dies zu einem sehr breiten Dialogfeld führen. Sie können dies vermeiden, indem Sie die Anzahl der Zeilen für das Layout des Steuerelements ändern. Dies ist möglich, indem Sie das Attribut `rows` der Steuerelementdefinition auf den gewünschten Wert einstellen. Beispiel:

```
<RadioButtonGroupControl property="enum1" label="Enum 1" rows="2"/>
```

Dies führt dazu, dass sich die Optionsfeldgruppe im Fenster über zwei Zeilen erstreckt.

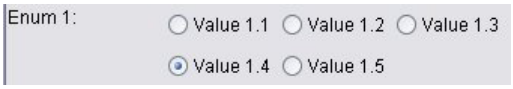

Abbildung 75. Fenster mit über zwei Zeilen angeordneter Optionsfeldgruppe

Ändern der Reihenfolge der Steuerelemente: Für Optionsfeld- und Kontrollkästchengruppen können Sie auch die Reihenfolge ändern, in der die Steuerelemente für die einzelnen `enum`-Werte zum Fenster hinzugefügt werden.

Standardmäßig werden die Steuerelemente in der Zeilenreihenfolge hinzugefügt, wie im vorherigen Beispiel, wo der erste, der zweite und der dritte Wert in die erste Zeile und der vierte und fünfte Wert in die zweite Zeile eingefügt werden. Sie können die Steuerelemente stattdessen innerhalb der angegebenen Zeilenzahl in der Spaltenreihenfolge einfügen, indem Sie für `layoutByRow` den Wert `false` festlegen. Beispiel:

```
<RadioButtonGroupControl property="enum1" label="Enum 1" rows="2" layoutByRow="false"/>
```

Die Werte werden weiterhin in zwei Zeilen angezeigt, der erste und zweite Wert werden jetzt jedoch in der ersten Spalte angezeigt, der dritte und vierte in der zweiten und der fünfte in der dritten.

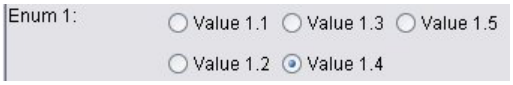

Abbildung 76. Fenster mit Optionsfeldgruppe, nach Spaltenreihenfolge angeordnet

Boolesche Eigenschaften, die als zwei Optionsfelder angezeigt werden, werden standardmäßig so angeordnet, dass die Schaltfläche für "False" vor der Schaltfläche für "True" angezeigt wird. Sie können diese Reihenfolge umkehren, indem Sie das Attribut `trueFirst` auf `true` einstellen.

Sie können auch verhindern, dass Optionsfeld- und Kontrollkästchengruppen ein Unterfenster verwenden, indem Sie das Attribut `useSubPanel` auf `false` einstellen. Dies kann jedoch zu einem unerwünschten Layoutverhalten führen, wenn diese Option nicht zusammen mit dem Element `Layout` verwendet wird (siehe „Festlegen präziser Steuerelementpositionen mit dem Element "Layout"“).

Erweitertes benutzerdefiniertes Layout

Innerhalb der einzelnen Steuerelementdeklarationen können Sie komplexe Steuerelementlayouts festlegen, die verschiedene Elemente verwenden. Sie verfügen über folgende Möglichkeiten:

- Festlegen präziser Steuerelementpositionen auf dem Bildschirm mit dem Element `Layout`
- Anzeigeeigenschaften des Steuerelements mit dem Element `Enabled` festlegen
- Sichtbarkeit des Steuerelements auf dem Bildschirm mit dem Element `Visible` steuern

Festlegen präziser Steuerelementpositionen mit dem Element "Layout": Eine präzise Layoutpositionierung kann durch die explizite Angabe eines `Layout`-Elements erreicht werden, das dem Steuerelement zugeordnet wird.

Format

```
<Eigenschaftssteuerelement ... >
     <Layout Attribute
      --- Zellenspezifikation ---
          ...
</Eigenschaftssteuerelement>
```

Dabei gilt Folgendes:

Eigenschaftssteuerelement ist eines der Eigenschaftssteuerelemente (siehe „Spezifikationen von Eigenschaftssteuerelementen“ auf Seite 123).

Attribute steht für eines der in der folgenden Tabelle aufgeführten Attribute.

Tabelle 39. Layoutattribute.

Attribut	Werte	Beschreibung

`anchor`	`north`
	`northeast`
	`east`
	`southeast`
	`south`
	`southwest`
	`west`
	`northwest`
	`center`
		Definiert den Verankerungspunkt für das Steuerelement.

Tabelle 39. Layoutattribute (Forts.).

Attribut	Werte	Beschreibung
`columnWeight`	0,0 - 1,0	Definiert, wie sich die Änderung der horizontalen Größe des Fensters auf die Breite des Steuerelements auswirkt. Die Summe aller `columnWeight`-Attribute in einem Fenster sollten den Wert 1,0 nicht überschreiten.

`fill`	`none`
	`horizontal`
	`vertical`
	`both`
		Definiert, ob und wie das Steuerelement die ihm zugeordneten Zellen ausfüllt.

`gridColumn`	ganze Zahl >= 0	Definiert die erste Spalte, in der das Layout des Steuerelements beginnt.
`gridHeight`	ganze Zahl	Definiert die Anzahl der Zeilen, über die sich das Layout des Steuerelements erstreckt. Der Wert 0 (Standard) ordnet dem Steuerelement alle verbleibenden Zeilen zu.
`gridRow`	ganze Zahl >= 0	Definiert die erste Zeile, in das Layout des Steuerelements verwendet wird. Standardmäßig wird der Rasterzeilenindex automatisch inkrementiert.
`gridWidth`	ganze Zahl	Definiert die Anzahl der Spalten, über die sich das Layout des Steuerelements erstreckt. Der Wert 0 (Standard) ordnet dem Steuerelement alle verbleibenden Spalten zu.
`leftIndent`	ganze Zahl	Definiert die Anzahl Pixel, um die das Steuerelement in Bezug auf seine Standardposition eingerückt wird.
`rowWeight`	0,0 - 1,0	Definiert, wie sich die Änderung der vertikalen Größe des Fensters auf die Höhe des Steuerelements auswirkt. Die Summe aller `rowWeight`-Attribute in einem Fenster sollten den Wert 1,0 nicht überschreiten.

Mit einer **Zellenspezifikation** können Sie die exakte Position eines Steuerelements auf dem Bildschirm definieren. Das Format lautet wie folgt:

```
<Cell row="ganze_Zahl" column="ganze_Zahl" width="ganze_Zahl" />
```

Dabei gilt Folgendes:

`row` (erforderlich) ist eine nicht negative ganze Zahl, die die Zeilenposition für den Start des Steuerzeichens angibt.

`column` (erforderlich) ist eine nicht negative ganze Zahl, die die Spaltenposition für den Start des Steuerzeichens angibt.

`width` (erforderlich) ist eine nicht negative ganze Zahl, die die Anzahl der Bildschirmrasterspalten angibt, die das Steuerelement belegt.

Wenn ein Bildschirmraster z. B. drei Spalten und drei Zeilen enthält, ist ein benutzerdefiniertes Steuerelementlayout im folgenden Format möglich.

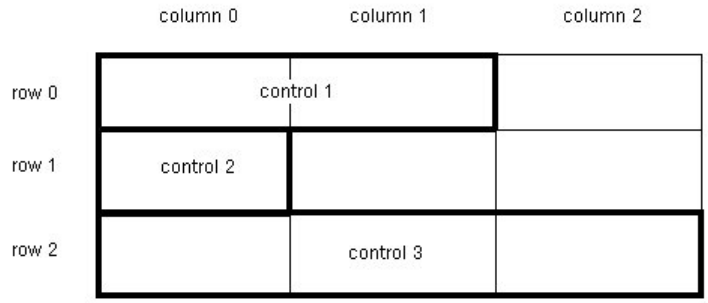

Abbildung 77. Beispiel für Steuerelementlayout, das Zellen verwendet

Dieses Format benötigt ein `Layout`-Element mit folgenden Zellenspezifikationen:

```
<Layout ... >
     <Cell row="0" column="0" width="2">
<Cell row="1" column="0" width="1">
<Cell row="2" column="0" width="3"> </Layout>
```

Es folgen einige detailliertere Beispiele, die die Verwendung des `Layout`-Elements weiter darstellen.

Beispiel: Kontrollkästchen, das ein Textfeld aktiviert: Dieses Beispiel zeigt die Verwendung eines Kontrollkästchens, um ein Textfeld in derselben Zeile der Anzeige zu aktivieren.

Wenn ein Kontrollkästchen verwendet wird, um ein anderes Steuerelement in derselben Zeile zu aktivieren, wird ein einfaches `Layout`-Element benötigt, damit die Steuerelemente richtig angezeigt werden. (*Hinweis*: Eine Beschreibung des Mechanismus zur Aktivierung bzw. Inaktivierung von Steuerelementen finden Sie in „Steuerung der Anzeigeeigenschaften mit dem Enabled-Element" auf Seite 160).

Angenommen, wir möchten folgende Anzeige erhalten.

Abbildung 78. Kontrollkästchen, das ein Textfeld aktiviert

Es gibt zwei Steuerelemente:

- Ein Kontrollkästchen mit einer Beschriftung, die als Beschriftung eines Textfelds verwendet wird
- Das Textfeld

Ausgangspunkt ist eine normale Deklaration der beiden Steuerelemente:

```
<CheckBoxControl property="boolean3" label="Check box 3"/>
<TextBoxControl property="string3" label="String 3"/>
```

Die daraus resultierende Anzeige sieht wie folgt aus.

Abbildung 79. Kontrollkästchen und Textfeld in separaten Zeilen

Zuerst soll die Anzeige der Textfeldbeschriftung **String 3** verhindert werden. Dies wird erreicht, indem das Attribut `showLabel` des Steuerelements für Textfelder auf `false` gesetzt wird:

```
<CheckBoxControl property="boolean3" label="Check box 3"/>
<TextBoxControl property="string3" label="String 3" showLabel="false"/>
```

Das Textfeld wird vergrößert und füllt jetzt den zuvor von der Beschriftung belegten Raum aus.

Abbildung 80. Kontrollkästchen und Textfeld ohne Beschriftung

Jetzt möchten wir erreichen, dass das Textfeld in derselben Zeile angezeigt wird, wie das Kontrollkästchen. Hierzu fügen wir ein `Layout`-Element innerhalb des `CheckBoxControl`-Elements hinzu, um die Inkrementierung der Zeile auf 0 zu setzen (standardmäßig wird die Zeile für jedes Steuerelement um 1 inkrementiert):

```
<CheckBoxControl property="boolean3" label="Check box 3">
     <Layout rowIncrement="0"/>
</CheckBoxControl>
<TextBoxControl property="string3" label="String 3" showLabel="false"/>
```

Die daraus resultierende Anzeige sieht jedoch wie folgt aus.

Abbildung 81. Textfeld überschreibt Kontrollkästchen

Das Textfeld wurde um eine Zeile nach oben verschoben, es belegt aber immer noch die gesamte Zeile und überschreibt daher das Kontrollkästchen.

Hinweis: Wenn die Anzeige wie folgt aussieht, wurde das Kontrollkästchen nach dem Textfeld angelegt.

Abbildung 82. Kontrollkästchen überschreibt Textfeld

Die ersten Zeichen des Textfelds sind teilweise überlagert.

Unabhängig davon, welches Objekt zuerst angelegt wird, verursacht die Zuordnung mehrerer Benutzerschnittstellenkomponenten zu derselben Zelle ein unerwünschtes bzw. undefiniertes Verhalten und sollte vermieden werden. Um dieses Problem zu beseitigen, müssen wir ein zweites `Layout`-Element hinzufügen, das jetzt innerhalb des `TextBoxControl`-Elements angegeben wird, um zu erzwingen, dass das Textfeld in der zweiten Spalte der Anzeige beginnt:

```
<CheckBoxControl property="boolean3" label="Check box 3">
     <Layout rowIncrement="0"/>
</CheckBoxControl>
<TextBoxControl property="string3" label="String 3" showLabel="false">
     <Layout gridColumn="1"/>
</TextBoxControl>
```

Dies ist jedoch nur eine Teillösung. Beide Steuerelemente sind richtig platziert, das Textfeld ist aber zu kurz, wie in der folgenden Anzeige zu sehen ist.

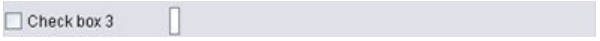

Abbildung 83. Richtig platziertes, aber sehr kurzes Textfeld

Das Problem ist, dass die Zuordnung eines benutzerdefinierten Layouts zu einem Steuerelement dazu führt, dass die jedem Steuerelementtyp zugeordneten Standardwerte überschrieben werden. In diesem Fall ist das Standardverhalten des `Layout`-Elements hinsichtlich des Auffüllens (d. h., wie die Komponente

die verfügbaren Zellen auffüllt) so festgelegt, dass die verfügbaren Zellen belegt werden und so wenig Bildschirmplatz wie möglich eingenommen wird. Damit sich dies ändert, weisen wir das Textfeld an, den horizontalen Raum zu füllen:

```
<CheckBoxControl property="boolean3" label="Check box 3">
     <Layout rowIncrement="0"/>
</CheckBoxControl>
<TextBoxControl property="string3" label="String 3" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
```

Es ist erforderlich, einen kleinen Wert für `columnWeight` hinzuzufügen, damit Java den ausgefüllten Bereich richtig zuordnet.

So erhalten wir das gewünschte Layout.

Abbildung 84. Kontrollkästchen, das ein Textfeld aktiviert

Dies sieht schon gut aus, es gibt aber immer noch ein Problem. Das Kontrollkästchen versucht jetzt, die gesamte Zeile zu besetzen, obwohl wir jetzt ein weiteres Steuerelement in derselben Zeile platzieren. Das Problem ist zurzeit nicht sichtbar, weil die Beschriftung des Kontrollkästchens relativ kurz ist und weil andere Beschriftungen im Fenster (hier nicht dargestellt) die zweite Anzeigespalte vergrößert haben, weswegen keine Überlappung vorliegt. Das Problem wird sichtbar, wenn die Beschriftung des Kontrollkästchens länger wird:

```
<CheckBoxControl property="boolean3" label="Check box 3 with a much longer label than we had">
     <Layout rowIncrement="0"/>
</CheckBoxControl>
<TextBoxControl property="string3" label="String 3" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
```

Dadurch wird die folgende Anzeige erzeugt.

Abbildung 85. Lange Kontrollkästchenbeschriftung durch Textfeld überschrieben

Wir müssen das Kontrollkästchen anweisen, seine Breite auf eine einzige Spalte einzuschränken:

```
<CheckBoxControl property="boolean3" label="Check box 3 with a much longer label than we had">
     <Layout rowIncrement="0" gridWidth="1"/>
</CheckBoxControl>
<TextBoxControl property="string3" label="String 3" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
```

So erhalten wir das gewünschte Ergebnis.

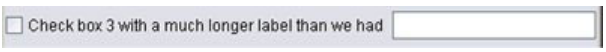

Abbildung 86. Lange Kontrollkästchenbeschriftung wird vollständig angezeigt

Beispiel: Optionsfeldgruppe und Textfelder: Dieses Beispiel zeigt ein Verfahren, mit dem jeder neuen Schaltfläche einer Optionsfeldgruppe ein eigenes Textfeld zugeordnet wird.

Wir möchten ein Fenster definieren, das wie die folgende Anzeige aussieht.

Abbildung 87. Optionsfeldgruppe mit Textfeldern

Dieses Mal haben wir vier Steuerelemente:

- Eine Optionsfeldgruppe für eine Aufzählungsliste mit drei Werten
- Drei Textfelder, für jeden Wert eins

Wie im vorherigen Beispiel beginnen wir mit einer einfachen Deklaration der Steuerelemente:

```
<RadioButtonGroupControl property="enum4" label="Enum 4"/>
<TextBoxControl property="string4" label="String 4"/>
<TextBoxControl property="string5" label="String 5"/>
<TextBoxControl property="string6" label="String 6"/>
```

Dadurch wird die folgende Anzeige erzeugt.

Abbildung 88. Optionsfeldgruppe mit Textfeldern und Beschriftungen

Wir möchten die Optionsfeldbeschriftungen verwenden, um die Textfelder zu identifizieren, daher besteht unsere erste Aufgabe darin, die Optionsfelder in einer einzigen von drei Spalten anzuordnen und die Textfeldbeschriftungen auszublenden:

```
<RadioButtonGroupControl property="enum4" label="Enum 4" rows="3"/>
<TextBoxControl property="string4" label="String 4" showLabel="false"/>
<TextBoxControl property="string5" label="String 5" showLabel="false"/>
<TextBoxControl property="string6" label="String 6" showLabel="false"/>
```

Die daraus resultierende Anzeige sieht wie folgt aus.

Abbildung 89. Optionsfelder in einer einzigen Spalte und Textfelder

Wir können hier bereits ein kleines Problem erkennen. Die Beschriftung der Optionsfeldgruppe ist nicht mit dem ersten Optionsfeld ausgerichtet. Wir werden uns später darum kümmern. Jetzt werden wir zuerst unsere Textfelder in Bezug auf die zugehörigen Optionsfelder ausrichten.

Das Verfahren ähnelt dem, das wir im 1. Beispiel angewendet haben. Wir müssen:

- Den Inkrementwert der Optionsfeldgruppen auf 0 setzen.
- Die Rasterbreite so einschränken, dass sich die nächsten Textfelder und Optionsfelder nicht überlappen.

- Alle Textfelder in derselben Zeile anordnen wie das zugehörige Optionsfeld.

Hierzu fügen wir einige Layout-Elemente hinzu, genau so wie im vorherigen Beispiel. In diesem Fall ändern wir die Spezifikationsdatei wie folgt:

```
<RadioButtonGroupControl property="enum4" label="Enum 4" rows="3">
     <Layout rowIncrement="0" gridWidth="1"/>
</RadioButtonGroupControl>
<TextBoxControl property="string4" label="String 4" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
<TextBoxControl property="string5" label="String 5" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
<TextBoxControl property="string6" label="String 6" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
```

Allerdings resultiert daraus die folgende Anzeige.

Abbildung 90. Textfelder überschreiben die Optionsfelder

Wir haben genau dieselben Layout-Elemente verwendet, wie im 1. Beispiel. Was ist passiert?

Die Antwort lautet, dass die Optionsfeldgruppe (wie die meisten Steuerelemente), anders als das Kontrollkästchen im vorherigen Beispiel, über eine separate Beschriftung und das tatsächliche Steuerelement verfügt. Dies bedeutet, dass für die Optionsfeldgruppe eine gesonderte Spalte erforderlich ist, weswegen die Textfelder eine Spalte später beginnen müssen, d. h. in Spalte 2 anstatt in Spalte 1. Daher setzen wir in den Layout-Elementen für die Textfelder die gridColumn-Werte auf 2:

```
<RadioButtonGroupControl property="enum4" label="Enum 4" rows="3">
     <Layout rowIncrement="0" gridWidth="1"/>
</RadioButtonGroupControl>
<TextBoxControl property="string4" label="String 4" showLabel="false">
     <Layout gridColumn="2" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
<TextBoxControl property="string5" label="String 5" showLabel="false">
     <Layout gridColumn="2" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
<TextBoxControl property="string6" label="String 6" showLabel="false">
     <Layout gridColumn="2" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
```

Beachten Sie, dass die Rasterbreite für die Optionsfeldgruppe unverändert bei 1 bleibt, obwohl wir die Textfeldrasterspalte mit 2 inkrementieren. Dies hat den Grund, dass sich für die Eigenschaftssteuerelemente die meisten Layout-Attribute nur auf die Benutzerschnittstellenkomponenten auswirken, die den editierbaren Bereich des Steuerelements ausmachen, und nicht die Beschriftung.

Die daraus resultierende Anzeige sieht wie folgt aus.

Abbildung 91. Textfelder überschreiben die Optionsfelder nicht mehr

Dies sieht schon viel eher wie das gewünschten Ergebnis aus. Es gibt aber immer noch ein paar Ausrichtungsprobleme zwischen den Optionsfeldern und den Textfeldern.

Das Hauptproblem ist, dass die Optionsfelder in separaten Unterfenstern angeordnet werden, weswegen keine wirkliche Beziehung zwischen einem Optionsfeld und seinem Textfeld besteht. Wir müssen also lediglich dafür sorgen, dass die Optionsfeldgruppe kein Unterfenster verwendet:

```
<RadioButtonGroupControl property="enum4" label="Enum 4" rows="3" useSubPanel="false">
     <Layout rowIncrement="0" gridWidth="1"/>
</RadioButtonGroupControl>
<TextBoxControl property="string4" label="String 4" showLabel="false">
     <Layout gridColumn="2" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
<TextBoxControl property="string5" label="String 5" showLabel="false">
     <Layout gridColumn="2" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
<TextBoxControl property="string6" label="String 6" showLabel="false">
     <Layout gridColumn="2" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
```

Jetzt wurde das gewünschte Layout definiert.

Abbildung 92. Optionsfeldgruppe mit Textfeldern

Steuerung der Anzeigeeigenschaften mit dem Enabled-Element: Sie können mit dem Element `Enabled` dafür sorgen, dass ein Steuerelement aktiviert bzw. inaktiviert wird, in der Regel unabhängig von einer bestimmten Bedingung.

Fenster- und Eigenschaftssteuerelemente können Bedingungen zugeordnet werden, über die verschiedene Anzeigeeigenschaften festgelegt werden. Beispiel: Ein Kontrollkästchen kann verwendet werden, um ein zugehöriges Textfeld zu aktivieren. Oder ein Optionsfeld kann dafür sorgen, dass eine Gruppe ausgeblendeter Felder angezeigt wird.

Bedingungen für die Benutzerschnittstelle basieren in der Regel auf dem Wert eines anderen Steuerelements, anstatt auf einer Eigenschaft. Bedingungen, die auf Eigenschaften basieren, treten nur in Kraft, wenn Änderungen am zugrunde liegenden Objekt durchgeführt wurden (z. B. ein Knoten, eine Modellausgabe oder eine Dokumentausgabe). Auf der Benutzerschnittstelle müssen Steuerelemente aktiviert werden, sobald ein zugehöriges Steuerelement geändert wurde.

Format

```
<Enabled>
     <Condition .../>
     <And ... />
     <Or ... />
     <Not ... />
</Enabled>
```

Das Element `Condition` gibt eine Bedingung an, die getestet wird, um zu ermitteln, ob das Steuerelement aktiviert ist.

Mit den Elementen `And`, `Or` und `Not` können zusammengesetzte Bedingungen festgelegt werden.

Weitere Informationen finden Sie im Thema „Bedingungen" auf Seite 72.

Beispiel: Aktivieren von Steuerelementen mit einer einfachen Bedingung: In „Beispiel: Kontrollkästchen, das ein Textfeld aktiviert" auf Seite 155 haben wir ein Kontrollkästchen entworfen, das ein Textfeld aktiviert, wenn das Kontrollkästchen ausgewählt wird.

Wir möchten, dass das Textfeld aktiviert wird, sobald das Kontrollkästchen markiert wird und nicht erst, wenn die Eigenschaft des zugrunde liegenden Objekts geändert wird. Hierzu müssen wir eine Bedingung für `Enabled` hinzufügen:

```
<CheckBoxControl property="boolean3" label="Check box 3 with a much longer label than we had">
     <Layout rowIncrement="0" gridWidth="1"/>
</CheckBoxControl>
<TextBoxControl property="string3" label="String 3" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
          <Enabled>
               <Condition control="boolean3" op="equals" value="true"/>
          </Enabled>
</TextBoxControl>
```

Diese Anweisungen sorgen dafür, dass das Textfeld nur dann aktiviert wird, wenn der dem Textfeld zugeordnete boolesche Wert wahr ist.

Beispiel: Aktivieren von Steuerelementen mit einer komplexen Bedingung: Um die Codierung komplexer Bedingungen zu veranschaulichen, werden wir uns eine der Dialogfeldregisterkarten für den verallgemeinerten linearen Knoten ansehen, der mit CLEF entwickelt wurde.

Das Knotendialogfeld verfügt über eine Registerkarte 'Experten', die Optionen enthält, die für Benutzer mit detaillierten Kenntnissen solcher Modelle gedacht sind. Alle auf der Registerkarte vorhandenen Optionen sind anfangs inaktiviert.

Wenn für das Kontrollkästchen **Modus** die Option **Experten** festgelegt wird, werden mehrere Optionen aktiviert.

Einige sind jedoch immer noch inaktiviert, wie z. B. das Steuerelement **Iterationen** am unteren Rand des Dialogfelds. Dieses Steuerelement ist nur dann inaktiviert, wenn die **beiden** folgenden Bedingungen wahr sind:

- **Verteilung** ist auf **Normal** gesetzt
- **Linkfunktion** ist auf **Identität** gesetzt

Diese Kombination entspricht der Standardeinstellung für diese Registerkarte im Expertenmodus. Eine Änderung der Einstellung eines dieser Kombinationsfelder sorgt dafür, dass **Iterationen** aktiviert wird.

Der entsprechende Code befindet sich in einer `PropertiesSubPanel`-Deklaration für die Schaltfläche **Iterationen**:

```
<PropertiesSubPanel buttonLabel="Iterations..." buttonLabelKey= ...
     <Enabled>
          <And>
               <Condition control="mode" op="equals" value="Expert"/>
                    <Not>
                         <And>
                              <Condition control="distribution" op="equals" value="NORMAL"/>
```

```
                              <Condition control="link_function" op="equals" value="IDENTITY"/>
                         </And>
                    </Not>
          </And>
     </Enabled>
      ...
</PropertiesSubPanel>
```

Das Element `Condition` im äußeren Abschnitt `And` legt fest, dass der **Modus** auf **Experten** eingestellt sein muss, damit eine Änderung erfolgt. Wenn diese Bedingung wahr ist, ist im Abschnitt `Not` festgelegt, dass die Schaltfläche nicht aktiviert ist (d. h. inaktiviert ist), außer wenn *beide* Bedingungen des inneren Abschnitts `And` erfüllt sind. Das heißt, im Expertenmodus wird **Iterationen** aktiviert, wenn entweder **Verteilung** oder **Linkfunktion** einen anderen Wert als den jeweiligen Standardwert haben.

Steuern der Anzeigeeigenschaften mit dem Visible-Element: Sie können auch Bedingungen verwenden, um dafür zu sorgen, dass Steuerelemente unter bestimmten Umständen angezeigt oder ausgeblendet werden. Hierzu wird das Element `Visible` verwendet.

Format

```
<Visible>
     <Condition .../>
     <And ... />
     <Or ... />
     <Not ... />
</Visible>
```

Das Element `Condition` gibt eine Bedingung an, die getestet wird, um zu ermitteln, ob das Steuerelement sichtbar ist.

Mit den Elementen `And`, `Or` und `Not` können zusammengesetzte Bedingungen festgelegt werden.

Weitere Informationen finden Sie im Thema „Bedingungen” auf Seite 72.

Beispiel

Das folgende Beispiel sorgt dafür, dass das angegebene Eigenschaftsfenster nur angezeigt wird, wenn die `source_language`-Bedingung erfüllt ist:

```
<PropertiesPanel>
     <Visible>
          <Condition control="source_language" op="equals" value="eng" />
     </Visible>
       ...
</PropertiesPanel>
```

Benutzerdefinierte Ausgabefenster

Für Modellausgabe-, Dokumentausgabe- und interaktive Ausgabeobjekte (nicht aber für Knoten) ist es möglich, dass eine Erweiterung das Standardausgabefenster vollständig durch ein benutzerdefiniertes Fenster ersetzt. Dies ist als `java.awt.Frame`-Standardklasse implementiert.

Um ein benutzerdefiniertes Fenster zu erstellen, müssen Sie eine Java-Klasse als `frameClass`-Attribut des `UserInterface`-Elements angeben:

```
<DocumentOutput id="my_modelling_node" type="modelBuilder" ...>
     <Properties>
          <Property name="use_custom_type" valueType="boolean" .../>
           ...
```

```
    </Properties>
    <UserInterface frameClass="com.myextension.MyOutputFrame"/>
     ...
</DocumentOutput>
```

Die angegebene Klasse muss die `ExtensionObjectFrame`-Schnittstelle implementieren, die durch die clientseitige API von CLEF definiert ist. Während der Lebensdauer des Fensters:

- Zugriff auf den zugrunde liegenden `java.awt.Frame`
- Fensterinitialisierung, einschließlich Zugriff auf das Ausgabeobjekt und die Sitzung
- Synchronisierung des Fensters und des zugrunde liegenden Objekts, wenn das Objekt gespeichert oder gelöscht wird
- Fensteranordnung

Weitere Informationen finden Sie im Thema „Clientseitige API-Klassen” auf Seite 178.

Kapitel 7. Hinzufügen eines Hilfesystems

Hilfesystemtypen

Wenn Sie eine CLEF-Erweiterung entwickeln, möchten Sie vermutlich auch ein Online-Hilfesystem hinzufügen, in der die Verwendung der Erweiterung beschrieben wird. CLEF unterstützt die folgenden Hilfesystemtypen:

- HTML Help
- JavaHelp

HTML Help

HTML Help ist ein proprietäres Hilfeformat von Microsoft, das nur auf Windows-Plattformen ausgeführt werden kann. Ein HTML Help-System besteht aus mehreren `.htm`- oder `.html`-Dateien, die in eine komprimierte Datei mit der Erweiterung `.chm` kompiliert werden. Auch das Hilfesystem von IBM SPSS Modeler wird im HTML Help-Format bereitgestellt.

HTML Help unterstützt Inhaltsverzeichnis, Index, Volltextsuche sowie Glossarbegriffe in Pop-up-Fenstern. Die `.htm`- oder `.html`-Quellendateien mit den einzelnen Themen können in einem HTML-Editor oder mit einem professionellen Hilfe-Authoringtool erstellt werden. Die `.chm`-Datei kann mit HTML Help Workshop kompiliert werden. Dieses Tool erhalten Sie als kostenloses Download auf der Microsoft Download Centre-Website (Informationen zur Erstellung von `.chm`-Dateien finden Sie im Hilfesystem von HTML Help Workshop). Alternativ können Sie zur Kompilierung Ihrer Themendateien und der zugehörigen Grafikdateien in eine `.chm`-Datei jedes Hilfe-Authoringtool verwenden, das das HTML Help-Format unterstützt.

JavaHelp

JavaHelp ist ein von Sun Microsystems entwickeltes Open Source-Hilfeformat, das auf jeder Plattform ausgeführt werden kann, die Java unterstützt. Ein JavaHelp-System besteht aus den folgenden Dateien:

- Die `.htm`- oder `.html`-Quellendateien mit den einzelnen Themen
- Die in den Themen verwendeten Grafikdateien
- Eine Helpset-Datei (mit der Erweiterung `.hs`), die das Hilfesystem steuert
- Die Datei `map.xml`, die die Themen-IDs und die Themendateien einander zuordnet und das Fenster für die Anzeige der Hilfethemen definiert
- Die Datei `index.xml`, die die Indexeinträge enthält
- Die Datei `toc.xml`, die die Einträge des Inhaltsverzeichnisses enthält

JavaHelp unterstützt Inhaltsverzeichnis, Index, Volltextsuche und Glossar. Die `.htm`- oder `.html`-Quellendateien können in einem HTML-Editor oder mit einem professionellen Hilfe-Authoringtool erstellt werden. Zudem benötigen Sie die JavaHelp-Software, die Sie als kostenloses Download auf der Sun Developer Network-Website erhalten (weitere Informationen finden Sie in *JavaHelp System User's Guide*, das Sie ebenfalls von dieser Website herunterladen können).

Implementieren eines Hilfesystems

In diesem Abschnitt wird beschrieben, wie die relevanten Komponenten des Hilfesystems in der Spezifikationsdatei festgelegt werden.

Festlegen von Speicherort und Typ des Hilfesystems

Sofern einer Erweiterung ein Hilfesystem hinzugefügt wird, wird dessen Typ im Abschnitt "Resources" der Spezifikationsdatei der Erweiterung im Element `HelpInfo` angegeben.

Format

```
<Resources>
 ...
     <HelpInfo id="Name" type="Hilfetyp" path="Hilfepfad" helpset="Hilfetextgruppenposition"
      default="ID_des_Standardthemas" />
 ...
</Resources>
```

Dabei gilt Folgendes:

`id` (erforderlich) ist die Kennung für die Hilfeinformationen dieser Erweiterung.

`type` (erforderlich) gibt einen der folgenden Hilfetypen an:

- `htmlhelp` - die Hilfe hat das Format HTML und ist in der durch das Attribut `path` angegebenen `.chm`-Datei enthalten.
- `javahelp` - die Hilfe hat das Format Java und verwendet die durch das Attribut `helpset` angegebene Helpset-Datei (`.hs`) sowie die Quellen- und zugehörigen Dateien der Hilfe.

Beim Hilfetyp `htmlhelp` ist das folgende zusätzliche Attribut erforderlich:

- `path` - der Speicherort (relativ zur Spezifikationsdatei) und der Name der `.chm`-Datei mit dem Hilfesystem.

Beim Hilfetyp `javahelp` sind die folgenden zusätzlichen Attribute erforderlich:

- `helpset` - der Speicherort (relativ zur Spezifikationsdatei) und der Name der zu verwendenden `.hs`-Datei mit der Hilfetextgruppe.
- `default` - die Kennung des Standardthemas, das angezeigt wird, wenn für eine Registerkarte kein bestimmtes Thema angegeben wurde.

Wenn kein `HelpInfo`-Element angegeben ist, wird der Erweiterung keine Hilfe hinzugefügt.

Beispiele

Das erste Beispiel zeigt ein `HelpInfo`-Element für HTML Help:

```
<HelpInfo id="help" type="htmlhelp" path="help/mynode.chm" />
```

Für ein JavaHelp-System würden diese Angaben wie folgt aussehen:

```
<HelpInfo id="help" type="javahelp" helpset="help/mynode.hs"/>
```

Bei einem JavaHelp-System müssen sich die zugehörigen Dateien (Bilder sowie die Dateien map.xml, index.xml und toc.xml) in demselben Ordner befinden wie die `.hs`-Helpset-Datei.

Festlegen des anzuzeigenden Hilfethemas

Für Knotendialogfelder, Registerkarten und untergeordnete Fenster der Eigenschaftenseiten können Sie festlegen, welches Hilfethema angezeigt wird, wenn der Benutzer im jeweiligen Bereich Hilfe aufruft. Dazu verwenden Sie in der Spezifikation für den Knoten, die Registerkarte oder den Eigenschaftenbereich das Attribut `helpLink`.

Wenn kein `helpLink`-Attribut angegeben ist, wird das Standardthema des Hilfesystems angezeigt, sobald der Benutzer die Hilfe aufruft.

Weitere Informationen zum Attribut helpLink finden Sie in den Abschnitten „Knoten” auf Seite 48, „Registerkarten” auf Seite 114 und „Eigenschaftsunterfenster” auf Seite 127.

Beispiel

Dieses Beispiel geht von einem HTML Help-System aus. Es zeigt, wie in Abhängigkeit des Fensterfokus beim Aufrufen der Hilfe verschiedene kontextsensitive Themen angezeigt werden.

```
<Resources>
      ...
     <HelpInfo id="help" type="htmlhelp" path="help/mynode.chm"/>
      ...
</Resources>
 ...
<Node id="mynode" scriptName="my_node" type="dataTransformer" palette="recordOp"
  label="Sorter" description="Sorts a data file" >
  ...
     <Tabs defaultTab="1">
          <Tab label="Basic Controls" labelKey="basicControlsTab.LABEL"
            helpLink="basic_controls.htm">
               <PropertiesPanel>
                ...
                    <PropertiesSubPanel buttonLabel="Additional settings..."
                      buttonLabelKey="AdditionalOptions.LABEL" dialogTitle="Additional
                      Settings" dialogTitleKey="AdditionalOptionsDialog.LABEL" helpLink=
                      "addsettingsdlg.htm">
  ...
          </Tab>
          <Tab label="Selector Controls" labelKey="selectorControlsTab.LABEL"
           helpLink="selector_controls.htm">
  ...
          </Tab>
  ...
</Node>
```

In diesem Beispiel ist Folgendes festgelegt: Wenn sich der Fokus beim Aufrufen der Hilfe auf der Registerkarte "Basic Controls" befindet, wird das Hilfethema basic_controls.htm der Hilfedatei mynode.chm angezeigt. Wenn der Benutzer danach auf die Schaltfläche **Additional settings** klickt, um das Dialogfeld "Additional Settings" zu öffnen und darin **Help** wählt, wird das Thema addsettingsdlg.htm angezeigt. Wenn der Benutzer daraufhin das Dialogfeld "Additional Settings" schließt und die Registerkarte "Selector Controls" und dort wieder **Help** wählt, wird das Thema selector_controls.htm angezeigt.

Bei einem JavaHelp-System muss der Wert des Attributs helpLink mit dem Wert des Attributs target der Datei map.xml übereinstimmen. Angenommen, die Datei map.xml enthält Folgendes:

```
<map version="1.0">
 ...
     <mapID target="basic_controls" url="basic_controls.htm"/>
 ...
</map>
```

In diesem Fall muss das entsprechende Attribut helpLink den folgenden Wert haben:

```
helpLink="basic_controls"
```

Diese Zuordnung ist wichtig, da JavaHelp beim Aufrufen der Hilfe den Wert des Attributs target liest und ihm den zugehörigen url-Wert zuordnet, um die anzuzeigende Hilfedatei zu finden.

Kapitel 8. Lokalisierung und Eingabehilfen

Einführung

Mit **Lokalisierung** bezeichnet man die Anpassung von Software, Hilfe und Dokumentation an die Sprache und die Eigenheiten eines bestimmten Landes. Sie umfasst die Übersetzung der Benutzerschnittstelle, der Hilfe und der Dokumentation sowie das Testen des Systems mit der entsprechenden Ländereinstellung. Wenn Sie Ihre Erweiterung für ein internationales Publikum entwickeln, können Sie lokalisierte Versionen der Erweiterung bereitstellen.

Eingabehilfen sind Funktionen der Benutzerschnittstelle, die den Zugriff auf das System für Personen mit bestimmten Behinderungen, beispielsweise einer Sehbehinderung oder eingeschränkten feinmotorischen Fähigkeiten, erleichtern.

Lokalisierung

IBM SPSS Modeler wurde für verschiedene Länder und Sprachen lokalisiert. Wenn der Benutzer die Windows-Ländereinstellung (Regions- und Sprachoptionen) auf seine eigene Sprache eingestellt und diese Sprache unterstützt wird, werden die Standardkomponenten der Benutzerschnittstelle von IBM SPSS Modeler in dieser Sprache angezeigt. Zu den lokalisierten Komponenten zählen unter anderem:

- Systemmenüs und Menüelemente
- Systemschaltflächen (Generieren, OK, Ausführen, Abbrechen, Anwenden, Zurücksetzen)
- Registerkarten in Standarddialogfeldern (Anmerkungen und Debuggen, sofern verwendet)
- Fehler- und Systemnachrichten (z. B. "Dieses Objekt wurde nicht gespeichert.")

Diese Standardkomponenten von IBM SPSS Modeler werden auch in Ihrer Erweiterung automatisch in der ausgewählten Sprache angezeigt, sofern diese unterstützt wird.

Für alle anderen Komponenten Ihrer Erweiterung stellt CLEF eine Lokalisierungsfunktion bereit. Folgende Komponenten können lokalisiert werden:

- Knotennamen (auf Palette und Erstellungsbereich)
- Modellnamen (auf der Registerkarte "Modelle" des Managerfensters)
- Dokumentnamen (auf der Registerkarte "Ausgaben" des Managerfensters)
- Speicherort des Symbolbilds, das einer Aktion zugeordnet ist
- QuickInfo-Text
- Hilfesysteme
- Knotendialogfelder:
  - Titelleistentext
  - Benutzerdefinierte Menüs und Menüeinträge
  - Beschriftungen von Feldern, Eigenschaften, Schaltflächen und Registerkarten
  - Statischer Text
- System- und Fehlernachrichten

Textzeichenfolgen sollten relativ kurz gehalten werden, damit auch bei Übersetzungen genügend Platz vorhanden ist.

System- und Fehlernachrichten können durch eine Kombination aus Spezifikationsdatei, Eigenschaftendateien und der serverseitigen API lokalisiert werden. Weitere Informationen finden Sie im Thema „Statusdetaildokument” auf Seite 196.

Eigenschaftendateien

Die Textzeichenfolgen der lokalisierbaren Komponenten werden in sogenannten **Eigenschaftendateien** gespeichert. Diese verwenden zum Speichern der Lokalisierungsressourcenbundles ein Java-Standardformat. Jede Eigenschaftendatei enthält jeweils einen Datensatz für jede lokalisierte Komponente der Erweiterung. Da in jedem Datensatz ein Feld dem `labelKey`-Attribut der Spezifikationsdatei entspricht, kann CLEF die jeweils passende lokalisierte Textzeichenfolge aus der Eigenschaftendatei einlesen und sie an der richtigen Stelle auf dem Bildschirm anzeigen.

Eigenschaftendateien müssen die Dateinamenerweiterung `.properties` haben und in demselben Verzeichnis gespeichert sein, in dem sich auch die Spezifikationsdatei des Knotens befindet, zu der die Datei gehört. IBM SPSS Modeler sucht zunächst nach der Standardeigenschaftendatei, die folgenden Namen hat:

Pfad`.properties`

Dabei ist *Pfad* der Wert des `path`-Attributs des `Bundle`-Elements (im Abschnitt "Resources"), das das Eigenschaftenbündel definiert. Beispiel:

```
<Bundle id="bundle" path="my_resources"/>
```

Wenn keine Standardeigenschaftendatei vorliegt, ruft IBM SPSS Modeler die Textzeichenfolgen aus den Definitionen der Spezifikationsdatei ab.

Für jede von der Lokalisierung unterstützte Sprache muss eine eigene Eigenschaftendatei vorliegen. Die Dateien für alle Sprachen, mit Ausnahme der Standardsprache, werden durch ein Suffix im Dateinamen unterschieden. Beispiel:

```
my_ressources.properties
my_ressources_de.properties
my_ressources_fr.properties
```

Die Suffixe entsprechen den zweistelligen ISO 639-1-Standardsprachcodes.

Jeder Datensatz einer Eigenschaftendatei hat das folgende Format:

id=Textzeichenfolge

Dabei gilt Folgendes:

id ist die Kennung eines `buttonLabelKey`-, `descriptionKey`-, `dialogTitleKey`-, `falseLabelKey`-, `imagePathKey`-, `labelKey`-, `messageKey`-, `textKey`- oder `trueLabelKey`-Attributs in der Spezifikationsdatei. Diese Kennung hat in der Regel das Suffix `.LABEL`, damit sie leichter erkannt wird. Sie kann aber auch ein beliebiges Suffix oder gar kein Suffix aufweisen, je nachdem, wie sie in der Spezifikationsdatei angegeben ist.

Textzeichenfolge ist der Text für die Komponente.

Beispiel: Lokalisieren einer Registerkarte

In diesem Beispiel, das die Lokalisierung einer Registerkarte eines Knotendialogfelds zeigt, werden zwei Eigenschaftendateien verwendet: die englische Standardversion und die französische Version. Die Lokalisierung sieht wie folgt aus:

Erweiterungsordner`\`**`my_resources`**`.properties`
Erweiterungsordner`\`**`my_resources`**`_fr.properties`

Dabei ist *Erweiterungsordner* der Ordner, in dem sich die Spezifikationsdatei der Erweiterung befindet.

In der Spezifikationsdatei werden die Eigenschaftendateien durch ein `Bundle`-Element im Abschnitt `Resources` referenziert:

```
<Resources>
     <Bundle id="bundle" type="properties" path="my_resources"/>
</Resources>
```

Das path-Attribut darf weder Spracherweiterungen noch das Suffix .properties enthalten.

Die restlichen relevanten Abschnitte der Spezifikationsdateien sehen wie folgt aus:

```
<Node id="uiTest" scriptName="ui_test" type="dataTransformer" palette="recordOp" label=
"UI Test" ... >
     <Properties>
          <Property name="enum1" valueType="enum" defaultValue="value4">
               <Enumeration>
                    <Enum value="value1" label="Value 1.1" labelKey="enum1.value1.LABEL"/>
                    <Enum value="value2" label="Value 1.2" labelKey="enum1.value2.LABEL"/>
                    <Enum value="value3" label="Value 1.3" labelKey="enum1.value3.LABEL"/>
                    <Enum value="value4" label="Value 1.4" labelKey="enum1.value4.LABEL"/>
                    <Enum value="value5" label="Value 1.5" labelKey="enum1.value5.LABEL"/>
               </Enumeration>
          </Property>
     </Properties>
 ...
     <UserInterface ... >
          <Tabs defaultTab="1">
               <Tab label="Basic Controls" labelKey="basicControlsTab.LABEL" ... >
      ...
     </UserInterface>
 ...
</Node>
```

Die englische Version der Eigenschaftendatei enthält die folgenden Datensätze:

```
basicControlsTab.LABEL=Basic Controls
enum1.value1.LABEL=Value 1.1
enum1.value2.LABEL=Value 1.2
enum1.value3.LABEL=Value 1.3
enum1.value4.LABEL=Value 1.4
enum1.value5.LABEL=Value 1.5
```

In der folgenden Abbildung sind die Teile des Dialogfelds hervorgehoben, auf die sich diese Datensätze auswirken:

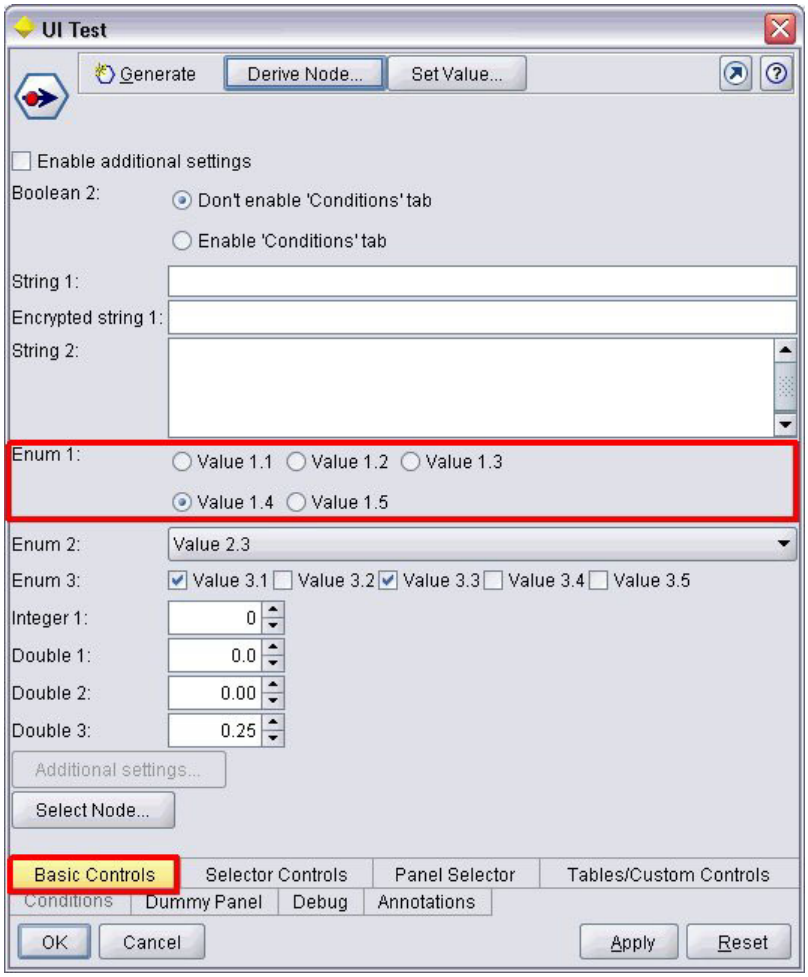

Abbildung 93. Nicht lokalisierte Registerkarte

Der entsprechende Abschnitt der französischen Version der Eigenschaftendatei (`my_resources_fr.properties`) sieht wie folgt aus:

```
basicControlsTab.LABEL=Contrôles de Base
enum1.value1.LABEL=Valeur 1,1
enum1.value2.LABEL=Valeur 1,2
enum1.value3.LABEL=Valeur 1,3
enum1.value4.LABEL=Valeur 1,4
enum1.value5.LABEL=Valeur 1,5
```

Diese Datensätze bewirken, dass die entsprechenden Anzeigenkomponenten (wie in der folgenden Abbildung dargestellt) in übersetzter Form angezeigt werden.

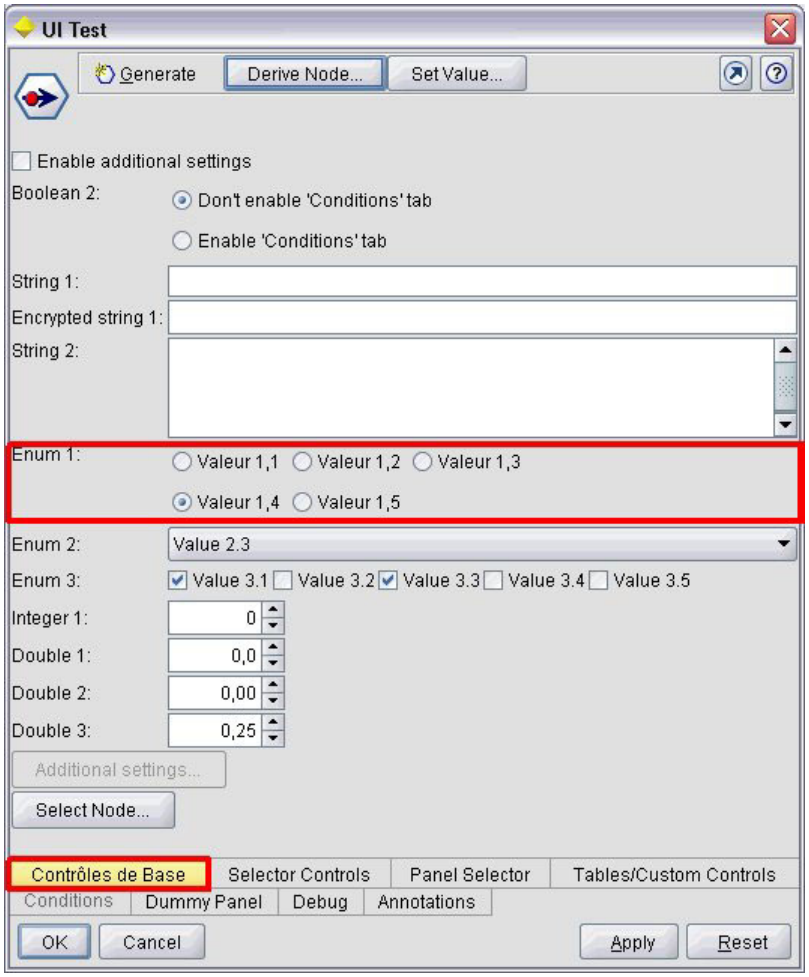

Abbildung 94. Lokalisierte Registerkarte

Die vier Schaltflächen am unteren Rand des Bildschirms werden durch die Standardfunktion von IBM SPSS Modeler und nicht durch CLEF lokalisiert.

Beispiel: Verwendung von Sonderzeichen

In der Eigenschaftendatei müssen Sie für Sonderzeichen in Standard-ASCII-Textzeichenfolgen Unicode-Escape-Sequenzen verwenden. Nachfolgend sehen Sie einen Ausschnitt einer Eigenschaftendatei, die in die französische Sprache lokalisiert wurde:

```
Genlinnode.LABEL=Lin\u00e9aire g\u00e9n\u00e9ralis\u00e9

Fields.LABEL=Champs
Model.LABEL=Mod\u00e8le
Expert.LABEL=Expert

inputFields.LABEL=Entr\u00e9es
targetField.LABEL=Cible
 ...
```

Für Sprachen wie Japanisch und Chinesisch, die keine lateinischen Zeichen verwenden, dürfen Sie für die Textzeichenfolgen nur Unicode-Escape-Sequenzen verwenden. Nachfolgend sehen Sie dieselben Datensätze in der japanischen Lokalisierung:

```
Genlinnode.LABEL=\u4e00\u822c\u5316\u7dda\u578b

Fields.LABEL=\u30d5\u30a3\u30fc\u30eb\u30c9
```

```
Model.LABEL=\u30e2\u30c7\u30eb
Expert.LABEL=\u30a8\u30ad\u30b9\u30d1\u30fc\u30c8

inputFields.LABEL=\u5165\u529b
targetField.LABEL=\u5bfe\u8c61
 ...
```

Hilfedateien

Wenn Sie eine Erweiterung lokalisieren, die über ein Hilfesystem verfügt, sollten Sie auch eine lokalisierte Version des Hilfesystems bereitstellen. Normalerweise wird das Hilfesystem in jeder Sprache lokalisiert, in der die Erweiterung zur Verfügung steht.

Lokalisieren eines HTML Help-Systems

Wenn die Erweiterung, die Sie lokalisieren, über eine HTML Help-Datei verfügt (eine Datei mit dem Suffix `.chm`), ersetzen Sie einfach die Standard-`.chm`-Datei durch die lokalisierte Version. Weitere Informationen zu HTML Help-Systemen finden Sie in „HTML Help" auf Seite 165.

So erstellen Sie eine lokalisierte `.chm`-Datei:

1. Übersetzen Sie die einzelnen Hilfethemen (`.htm`- oder `.html`-Dateien), aus denen das Hilfesystem besteht. Die Dateinamen dürfen Sie nicht ändern.
2. Lokalisieren Sie gegebenenfalls auch die im Hilfesystem enthaltenen Grafiken (erstellen Sie z. B. lokalisierte Screenshots).
3. Kompilieren Sie die übersetzten Dateien mit Microsoft HTML Help Workshop oder einem anderen Hilfe-Authoringtool in eine lokalisierte `.chm`-Datei.
4. Testen Sie das Hilfesystem im lokalisierten Knoten. Weitere Informationen finden Sie im Thema „Testen eines lokalisierten CLEF-Knotens".

Lokalisieren eines JavaHelp-Systems

Wenn die Erweiterung, die Sie lokalisieren, über ein JavaHelp-System verfügt, müssen Sie für jede unterstützte Sprache eine lokalisierte Version der Quellendateien der Hilfe bereitstellen. Die Anzeige der richtigen lokalisierten Version erfolgt bei JavaHelp automatisch, sofern eine Lokalisierung vorliegt. Weitere Informationen finden Sie im Thema „JavaHelp" auf Seite 165.

So erstellen Sie ein lokalisiertes JavaHelp-System:

1. Übersetzen Sie die einzelnen Hilfethemen (`.htm`- oder `.html`-Dateien), aus denen das Hilfesystem besteht. Die Dateinamen dürfen Sie nicht ändern.
2. Lokalisieren Sie gegebenenfalls auch die im Hilfesystem enthaltenen Grafiken (erstellen Sie z. B. lokalisierte Screenshots).
3. Generieren Sie die Helpset-Datei und andere erforderliche Dateien (map.xml, index.xml und toc.xml).
4. Testen Sie das Hilfesystem im lokalisierten Knoten. Weitere Informationen finden Sie im Thema „Testen eines lokalisierten CLEF-Knotens".

Testen eines lokalisierten CLEF-Knotens

So testen Sie einen lokalisierten Knoten und das zugehörige Hilfesystem:

1. Gehen Sie zum Abschnitt "Resources" der Spezifikationsdatei des lokalisierten Knotens und setzen Sie dort das `path`-Attribut des `HelpInfo`-Elements auf die lokalisierte `.chm`- bzw. `.hs`-Datei. Beispiel für HTML Help:

   ```
   <Resources>
    ...
        <HelpInfo id="help" type="HTMLHelp" path="help/mynode_fr.chm "/>
   </Resources>
   ```

   Beispiel für JavaHelp:

```
<Resources>
 ...
     <HelpInfo id="help" type="javahelp" helpset="help/mynode_fr.hs "/>
</Resources>
```

2. Kopieren Sie die lokalisierte `.chm`- oder `JAR`-Datei in das im `path`-Attribut angegebene Verzeichnis.
3. Legen Sie die Windows-Region für die gewünschte Ländereinstellung fest:

   Systemsteuerung > **Regions- und Sprachoptionen** > **Regionale Einstellungen** > **Standards und Formate** > **<Sprache>**
4. Starten Sie IBM SPSS Modeler und stellen Sie sicher, dass es in der gewünschten Sprache angezeigt wird.
5. Fügen Sie den lokalisierten Knoten zu IBM SPSS Modeler hinzu. Weitere Informationen finden Sie im Thema „Testen einer CLEF-Erweiterung" auf Seite 203.
6. Ziehen Sie eine Kopie des Knotens in den Erstellungsbereich.

   Öffnen Sie das Knotendialogfeld und prüfen Sie, ob es korrekt in der gewünschten Sprache angezeigt wird.
7. Klicken Sie auf die Hilfeschaltfläche des Dialogfelds und stellen Sie sicher, dass das richtige Hilfethema in der gewünschten Sprache angezeigt wird.

Zugriffsmöglichkeiten

CLEF stellt sämtliche Standardeingabehilfen von IBM SPSS Modeler wie Tastenkombinationen für Mausaktionen und Unterstützung für Sprachausgabeprogramme bereit.

Darüber hinaus können Sie für einen CLEF-Knoten als Eingabehilfe benutzerdefinierte QuickInfos bereitstellen.

Sie können auch Tastenkombinationen angeben, um Endbenutzern einen alternativen Zugriff auf verschiedene Funktionen der Benutzerschnittstelle zu bieten, die Sie in CLEF hinzugefügt haben. Weitere Informationen finden Sie im Thema „Zugriffstasten und Tastenkombinationen" auf Seite 115.

Für Aktionsschaltflächen und für alle Anzeigenkomponenten, die als Steuerelemente fungieren (z. B. Kontrollkästchen und Optionsfelder), können Sie folgende Texte definieren:

- Beschriftung
- Beschreibung

Die Beschriftung (**Label**) ist der Bildschirmtext, der als Name einer Komponente angezeigt wird und von Sprachausgabeprogrammen gelesen werden kann. Für Benutzer mit Sehbehinderung kann die Größe der Beschriftung auf der Registerkarte "Anzeige" des Dialogfelds "Benutzeroptionen" geändert werden. Dieses Dialogfeld öffnen Sie mit folgender Befehlsfolge:

Tools > **Benutzeroptionen**

Die Beschreibung (**Description**) ist der QuickInfo-Text, der angezeigt wird, wenn Sie mit dem Mauszeiger auf eine Anzeigenkomponente zeigen. QuickInfos bieten Raum für weitere Informationen und beschreiben eine Anzeigenkomponente ausführlicher als die Beschriftung. Auch QuickInfos können von Sprachausgabeprogrammen gelesen werden, sofern diese entsprechend konfiguriert sind.

Beschriftungen und Beschreibungen werden in dem Element, das die Komponente in der Spezifikationsdatei definiert, mit den Attributen `label` und `description` festgelegt. Beide können mittels der Attribute `labelKey` bzw. `descriptionKey` lokalisiert werden.

Beispiel

Nachfolgend wird die Verwendung der Beschriftungs- und Beschreibungsfunktion am Beispiel einer Aktionsschaltfläche veranschaulicht.

```
<Action id="setValue" label="Set Value..." labelKey="setValue.LABEL"
     description="Sets a value" descriptionKey="setValue.TOOLTIP"/>
```

Kapitel 9. Programmierung

Informationen zur Programmierung von CLEF-Knoten

Ein Knoten kann auch Verarbeitungsschritte durchführen, die sich nicht in der Spezifikationsdatei festlegen lassen. Hierfür stellt CLEF die folgenden APIs (Anwendungsprogrammierschnittstellen) zur Verfügung, die aus Ihren Programmen aufgerufen werden können:

- **Clientseitige API**: Eine Java-API, die von Erweiterungen verwendet werden kann, für die Steuerelemente, Benutzerschnittstellenkomponenten oder Interaktivitätselemente erforderlich sind, die nicht in der Spezifikationsdatei bereitgestellt werden können.
- **Predictive Server-API (PSAPI)**: Eine Java-API, die Funktionalität von IBM SPSS Modeler bereitstellt für Anwendungen, die Data-Mining-Funktionen und Mechanismen für die Vorhersageanalyse benötigen. Der PSAPI und der Data-Mining-Workbench von IBM SPSS Modeler liegt die gleiche Technologie zugrunde.
- **Serverseitige API**: Eine C-basierte API, die Aspekte wie die Festlegung und das Abrufen von Ausführungseinstellungen, die Persistenz dieser Einstellungen, Ausführungsfeedback, die Jobsteuerung (z. B. Unterbrechung der Ausführung), die SQL-Generierung und die zurückgegebenen Objekte definiert.

Dokumentation zu den CLEF-APIs

Die nachfolgenden Abschnitte bieten einen Überblick über die client- und serverseitigen APIs. Eine ausführlichere Dokumentation zu den APIs ist jeder IBM SPSS Modeler-Installation in einer ZIP-Datei beigefügt, die vor der Verwendung extrahiert werden muss.

So extrahieren Sie die API-Dokumentation:

1. Suchen Sie auf der DVD des Produkts im Ordner *\Documentation\en* nach der Datei *clef_apidoc.zip*.
2. Extrahieren Sie den Inhalt der ZIP-Datei mit WinZip oder einem ähnlichen Dekomprimierungstool in ein beliebiges Verzeichnis. Durch die Extraktion wird in diesem Verzeichnis der Unterordner *clef_apidoc* mit der gesamten API-Dokumentation erstellt.

So zeigen Sie die API-Dokumentation an:

1. Wechseln Sie in den Unterordner *clef_apidoc* und öffnen Sie dort die Datei *clef_apidoc.htm*.
2. Wählen Sie die gewünschte API-Dokumentation aus (clientseitige, serverseitige oder PSAPI-API).

Clientseitige API

CLEF enthält eine Reihe von Java-Klassen mit Methoden, die Sie für die clientseitige Verarbeitung verwenden können. Die Klasse `DataModelProvider` ermöglicht beispielsweise die Berechnung eines Ausgabedatenmodells, wenn Änderungen am Eingabedatenmodell zu komplex für die durch die Spezifikationsdatei bereitgestellten Funktionen sind.

Clientseitige API-Klassen

Im Folgenden sind die clientseitigen Klassen aufgeführt.

Tabelle 40. Clientseitige API-Klassen

Klasse	Beschreibung
`NodeDelegate`	Definiert die Methoden, die von einem Knotendelegate unterstützt werden. Für jede `ExtensionProcessor`-Instanz wird eine Knotendelegate-Instanz erstellt. Der Pfad der Implementierungsklasse wird im Attribut "delegate" des relevanten Elements `Node` in der Datei extension.xml angegeben.
`OutputDelegate`	Definiert die Methoden, die von einem Ausgabedelegate unterstützt werden. Für jede `ExtensionOutput`-Instanz wird eine Ausgabedelegate-Instanz erstellt. Der Pfad der Implementierungsklasse wird im Attribut "delegate" des relevanten Elements `DocumentOutput` oder `ModelOutput` in der Datei `extension.xml` angegeben.
`ExtensionObjectUIDelegate`	Definiert die Methoden, die von einem Benutzerschnittstellendelegate für Erweiterungsobjekte unterstützt werden. Für jede Dialog- oder Ausgabefensterinstanz für Erweiterungsknoten wird eine Benutzerschnittstellendelegate-Instanz erstellt. Der Pfad der Implementierungsklasse wird im Attribut "uiDelegate" des relevanten Elements `UserInterface` in der Datei `extension.xml` angegeben.
`ExtensionDelegate`	Definiert die Methoden, die von einem Erweiterungsdelegate unterstützt werden. Pro Erweiterung, die in einem Prozess gemeinsam verwendet wird, wird eine Erweiterungsdelegate-Instanz erstellt. Da ein einzelner Prozess Sitzungen in verschiedenen Ländereinstellungen unterstützen kann, sind keine Ländereinstellungsinformationen für den Erweiterungsdelegate verfügbar. Der Pfad der Implementierungsklasse wird im Attribut "extensionDelegate" des Elements `CommonObjects` in der Datei `extension.xml` angegeben.
`ActionHandler`	Ermöglicht der Erweiterung die Verwaltung von Aktionen, die vom Benutzer über Menüoptionen und Symbolleistenschaltflächen angefordert werden.
`DataModelProvider`	Ermöglicht Knoten, die komplexe Änderungen am Datenmodell vornehmen, die Verwendung von Java zur Berechnung des Ausgabedatenmodells.
`ExtensionObjectFrame`	Legt die Funktionalität von Fenstern fest, die Modell- oder Dokumentausgaben enthalten.
`ExtensionObjectPanel`	Legt die Funktionalität von Erweiterungsobjektfenstern fest.
`PropertyControl`	Legt die Funktionalität benutzerdefinierter Eigenschaftssteuerelemente in einem Eigenschaftenfenster fest.

Ausführliche Informationen zu diesen Klassen erhalten Sie in der Dokumentation zur clientseitigen API. Weitere Informationen finden Sie im Thema „Dokumentation zu den CLEF-APIs” auf Seite 177.

Verwenden der clientseitigen API

So fügen Sie einem CLEF-Knoten clientseitige Funktionsaufrufe hinzu:

1. Erstellen Sie *.java*-Quellendateien mit den gewünschten Funktionsaufrufen.
2. Kompilieren Sie die Quellendateien in *.class*-Dateien.
3. Sie können mehrere *.class*-Dateien in einer *JAR*-Datei zusammenfassen und in der Spezifikationsdatei auf diese *JAR*-Datei verweisen. Beispiel:

```
<Resources>
 ...
     <JarFile id="selfjar" path="selflearning.jar"/>
 ...
</Resources>
```

In einigen Elementen von CLEF können Sie direkt auf eine Klasse verweisen. Beispielsweise können Sie im Attribut `controlClass` eines `PropertyControl`-Elements der Spezifikationsdatei, wie nachfolgend gezeigt, einen Verweis auf eine Klasse einfügen:

```
<PropertyControl property="target_field_values_specify" ...
     controlClass="com.spss.clef.selflearning.propertycontrols.list.CustomListControl" ... />
```

Dabei ist `CustomListControl` der Name der Klasse, die das Steuerelement für eine Eigenschaft implementiert, und `com.spss.clef.selflearning.propertycontrols.list` ist der Pfad der Klasse innerhalb der im Element `JarFile` deklarierten *JAR*-Datei.

Weitere Informationen finden Sie im Thema „Abschnitt 'Resources'" auf Seite 34.

Sehen Sie sich hierzu auch den Quellcode der in dieser Version bereitgestellten Beispielknoten an. Sie finden dort eventuell einige nützliche Anwendungsbeispiele. Weitere Informationen finden Sie im Thema „Untersuchen des Quellcodes" auf Seite 29.

Predictive Server-API (PSAPI)

Die PSAPI ist eine Programmierschnittstelle für die diesem Produkt zugrunde liegende Predictive Server-Technologie. Die wichtigsten Elemente der PSAPI sind als Java-Schnittstellen definiert. Die meisten dieser Schnittstellen sind mittels interner Klassen implementiert, die von der PSAPI bereitgestellt werden, aber nicht Bestandteil der PSAPI-Spezifikation sind. Der PSAPI-Benutzer soll dadurch vor Änderungen an der Predictive Server-Technologie geschützt werden (z. B. vor Architekturänderungen oder vor Änderungen am privaten Client-/Serverprotokoll).

Ausführliche Informationen zu diesen Klassen erhalten Sie in der PSAPI-Dokumentation. Weitere Informationen finden Sie im Thema „Dokumentation zu den CLEF-APIs" auf Seite 177.

Serverseitige API

Die serverseitige API ist in der Programmiersprache C definiert, unterstützt aber auch Implementierungen in der Programmiersprache C++. Erweiterungsmodule können in C oder C++ direkt für die C-basierte API programmiert werden. Darüber hinaus können auch andere Programmiersprachen verwendet werden, sofern der Entwickler über eine Methode der Bindung an die CAPI verfügt. CLEF bietet zudem verschiedene C++-Helper-Quellendateien, die als Wrapper für einen Teil der CAPI fungieren können.

Architektur

Der **Knoten** einer Erweiterung auf dem Client wird auf dem Server durch einen **Peer** der Erweiterung ergänzt. Ein Peer wird durch ein **Erweiterungsmodul** definiert, das als gemeinsame Bibliothek auf dem Server implementiert wird. Die Kommunikation zwischen einem Knoten und seinem Peer wird über eine vom Server verwaltete Erweiterungs**ressource** vermittelt. Eine Ressource ruft zur Erstellung und Manipulation seines Peers die im Erweiterungsmodul definierten **Servicefunktionen** auf, während der Peer **Callback-Funktionen** verwendet, um Informationen und Services von seinem Host anzufordern.

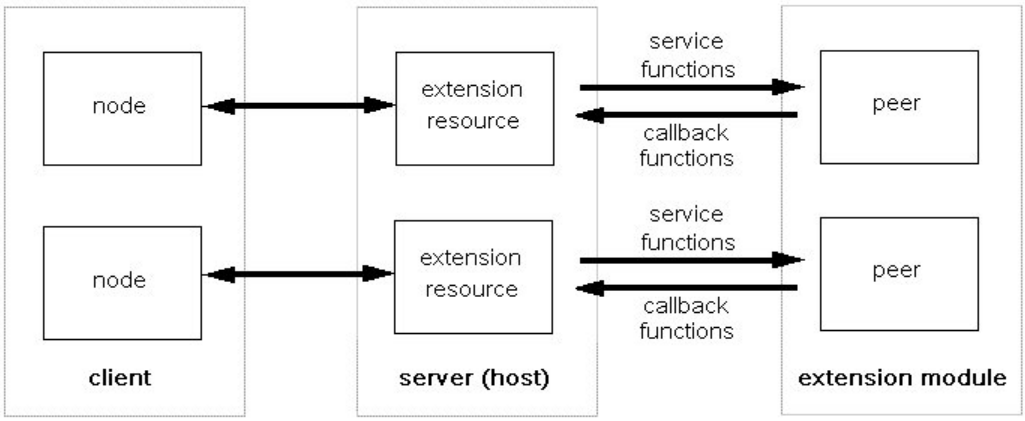

Abbildung 95. CLEF API-Architektur

Servicefunktionen

Servicefunktionen werden durch das Erweiterungsmodul implementiert. Ein Erweiterungsmodul muss alle Funktionen implementieren, die als erforderlich gekennzeichnet sind. Die als nicht erforderlich gekennzeichneten Funktionen kann, muss es aber nicht implementieren.

Es gibt zwei Arten von Servicefunktionen:

- Modulfunktionen
- Peerfunktionen

Die nachfolgenden Abschnitte bieten einen Überblick über die Servicefunktionen. Ausführlichere Beschreibungen dieser Funktionen finden Sie in der Dokumentation der serverseitigen API, die Sie wie folgt aufrufen:

1. Wählen Sie im API-Dokumentationsfenster von CLEF die Option **Server-side API overview** (Überblick über die serverseitige API) aus.
2. Klicken Sie auf die Registerkarte **Module**.
3. Wählen Sie **API Service Functions to be implemented by the extension module** (Vom Erweiterungsmodul zu implementierende API-Servicefunktionen) aus.

Informationen zum Zugriff auf die API-Dokumentation von CLEF finden Sie im Abschnitt „Dokumentation zu den CLEF-APIs" auf Seite 177.

Modulfunktionen

Modulfunktionen werden von nur einem Thread aufgerufen.

Tabelle 41. Modulfunktionen

Funktion	Erforderlich?	Beschreibung
`clemext_initialise`	Ja	Initialisiert ein Erweiterungsmodul
`clemext_cleanup`	Ja	Gibt die durch ein Erweiterungsmodul zugeordneten Ressourcen wieder frei
`clemext_getModuleInformation`	Nein	Ruft Informationen über ein Erweiterungsmodul ab
`clemext_create_peer`	Ja	Erstellt eine Peerinstanz für ein Erweiterungsmodul
`clemext_destroy_peer`	Ja	Entfernt eine Peerinstanz

Peerfunktionen

Peerfunktionen werden auf das Handle einer Peerinstanz angewendet, das von einem früheren `clemext_create_peer`-Aufruf zurückgegeben wurde. Peerfunktionen können nur dann gleichzeitig von verschiedenen Threads aufgerufen werden, wenn sich die Peer-Handles der Funktionen unterscheiden. Die einzige Ausnahme hiervon ist die Funktion `clemext_peer_cancelExecution`. Diese kann, sofern sie definiert ist, jederzeit von jedem Thread aufgerufen werden, um die Ausführung eines anderen Threads, die zu lange dauert, abzubrechen.

Tabelle 42. Peerfunktionen

Funktion	Erforderlich?	Beschreibung
`clemext_peer_configure`	Ja	Weist eine Peerinstanz an, ihre Parameter und ihr Datenmodell auszuführen
`clemext_peer_getDataModel`	Ja	Ruft das Ausgabedatenmodell einer Peerinstanz ab
`clemext_peer_getCatalogueInformation`	Nein	Ruft die Kataloginformationen eines Moduls ab
`clemext_peer_getExecutionRequirements`	Nein	Ruft die Ausführungsanforderungen einer Peerinstanz ab
`clemext_peer_beginExecution`	Ja	Startet die Ausführung einer Peerinstanz
`clemext_peer_nextRecord`	Ja	Bewirkt einen Wechsel zum nächsten Datensatz im Ergebnisset eines Peers
`clemext_peer_getRecordValue`	Ja	Gibt den Wert eines bestimmten Felds aus dem zuletzt abgerufenen Ergebnisdatensatz zurück
`clemext_peer_endExecution`	Ja	Zeigt einer Peerinstanz an, dass die Ausführung abgeschlossen ist
`clemext_peer_cancelExecution`	Nein	Wird von einem anderen Thread aufgerufen, um einen `beginExecution`- oder `nextRecord`-Funktionsaufruf, der zu lange dauert, abzubrechen
`clemext_peer_getSQLGeneration`	Nein	Generiert aus einer Peerinstanz SQL-Anweisungen
`clemext_peer_getErrorDetail`	Ja	Ruft zusätzliche Informationen über den letzten modulspezifischen Fehler eines Peers ab

Die in der folgenden Tabelle aufgeführten Peerfunktionen sind für die interaktive Modellerstellung vorgesehen.

Tabelle 43. Für die interaktive Modellerstellung vorgesehene Peerfunktionen

Funktion	Erforderlich?	Beschreibung
`clemext_peer_beginInteraction`	Nein	Beginnt die Interaktion mit einer Peerinstanz
`clemext_peer_request`	Nein	Führt eine interaktive Anforderung an einem Peer aus
`clemext_peer_getRequestReply`	Nein	Ruft die Antwort der letzten, erfolgreich ausgeführten Anforderung ab
`clemext_peer_endInteraction`	Nein	Zeigt einer Peerinstanz an, dass die Interaktion abgeschlossen ist

Callback-Funktionen

Wenn ein Erweiterungsmodul Informationen oder Services vom Hostprozess benötigt, muss es diese über ein **Callback** anfordern. Ein Callback wird auf ein **Handle** angewendet, also auf einen Zeiger, der das Ziel der Anforderung angibt.

Der Aufruf eines Callbacks erfolgt durch Übergabe des Handles desjenigen IBM SPSS Modeler-Objekts, an das der Aufruf gerichtet ist. Die Übergabe eines Handles in ein Erweiterungsmodul erfolgt in Form eines Parameters in einer Servicefunktion.

Falls eine Callback-Funktion fehlschlägt, sollte sie im zugehörigen modulspezifischen Fehlercode (gekennzeichnet durch `CLEMEXTErrorCode`) Informationen über den Fehler zurückgeben. Das Erweiterungsmodul kann seinerseits einen Callback-Fehler zurückgeben, indem es diese Informationen weiterleitet, damit diese vom Host untersucht werden können.

Folgende Typen von Callback-Funktionen stehen zur Verfügung:

- Hostfunktionen
- Knotenfunktionen
- Iteratorfunktionen
- Fortschrittsfunktionen
- Kanalfunktionen (nur für interaktive Modelle)

Die nachfolgenden Abschnitte bieten einen Überblick über die Callback-Funktionen. Ausführlichere Beschreibungen dieser Funktionen finden Sie in der Dokumentation der serverseitigen API, die Sie wie folgt aufrufen:

1. Wählen Sie im API-Dokumentationsfenster von CLEF die Option **Server-side API overview** (Überblick über die serverseitige API) aus.
2. Klicken Sie auf die Registerkarte **Module**.
3. Wählen Sie **General callbacks** (Allgemeine Callbacks) aus.

Informationen zum Zugriff auf die API-Dokumentation von CLEF finden Sie im Abschnitt „Dokumentation zu den CLEF-APIs" auf Seite 177.

Hostfunktionen

Hostfunktionen werden für ein Host-Handle definiert, das mittels `clemext_initialise` übergeben wird.

Tabelle 44. Hostfunktionen

Funktion	Beschreibung
`clemext_host_getHostInformation`	Ruft statische Informationen zur Hostumgebung ab

Knotenfunktionen

Knotenfunktionen werden für ein Knotenhandle definiert, das mittels `clemext_create_peer` übergeben wird.

Tabelle 45. Knotenfunktionen

Funktion	Beschreibung
`clemext_node_getNodeInformation`	Ruft statische Informationen zu einem Knoten ab
`clemext_node_getParameters`	Ruft die Parameter eines Knotens ab
`clemext_node_getDataModel`	Ruft das Eingabedatenmodell eines Knotens ab
`clemext_node_getOutputDataModel`	Ruft das Ausgabedatenmodell eines Knotens ab
`clemext_node_getSQLGeneration`	Ruft die vorgeordneten SQL-Generierungsinformationen eines Knotens ab
`clemext_node_getPassword`	Ruft die Klartextversion eines verschlüsselten Kennworts ab
`clemext_node_getFilePath`	Ruft den Pfad einer Datei ab, die während der Ausführung zwischen Client und Server ausgetauscht wird

Iteratorfunktionen

Iteratorfunktionen werden für ein Iteratorhandle definiert, das mittels `clemext_peer_beginExecution` übergeben wird. Ein Iterator stellt einem Erweiterungsmodul ein Eingabedataset bereit.

Tabelle 46. Iteratorfunktionen

Funktion	Beschreibung
`clemext_iterator_nextRecord`	Geht zum nächsten Datensatz im Eingabedataset
`clemext_iterator_getRecordValue`	Gibt den Wert eines bestimmten Felds aus dem zuletzt abgerufenen Eingabedatensatz zurück
`clemext_iterator_rewind`	Setzt den Status des Eingabedatasets zurück, damit der nächste `nextRecord`-Aufruf beim ersten Datensatz des Eingabedatasets beginnt

Fortschrittsfunktionen

Fortschrittsfunktionen werden für ein Fortschrittshandle definiert, das mittels `clemext_peer_beginExecution` übergeben wird.

Tabelle 47. Fortschrittsfunktionen

Funktion	Beschreibung
`clemext_progress_report`	Meldet den Fortschritt an den Host

Kanalfunktionen

Kanalfunktionen werden nur bei interaktiven Modellen verwendet. Sie werden für ein Kanalhandle definiert, das mittels `clemext_peer_beginInteraction` übergeben wird.

Tabelle 48. Kanalfunktionen

Funktion	Beschreibung
`clemext_channel_send`	Sendet eine asynchrone Nachricht an den Client, über die ein Peer-Thread Fortschritts- und Statusinformationen meldet

Prozessfluss

Ein Erweiterungsmodul ruft zur Ausführung seiner Verarbeitungsaufgaben verschiedene Service- und Callback-Funktionen auf. Welche Funktionen aufgerufen werden, richtet sich nach den Verarbeitungsschritten, die das Modul ausführen muss.

Beispiel

In der folgenden Abbildung wird das Ablaufdiagramm einer typischen Modulausführung dargestellt.

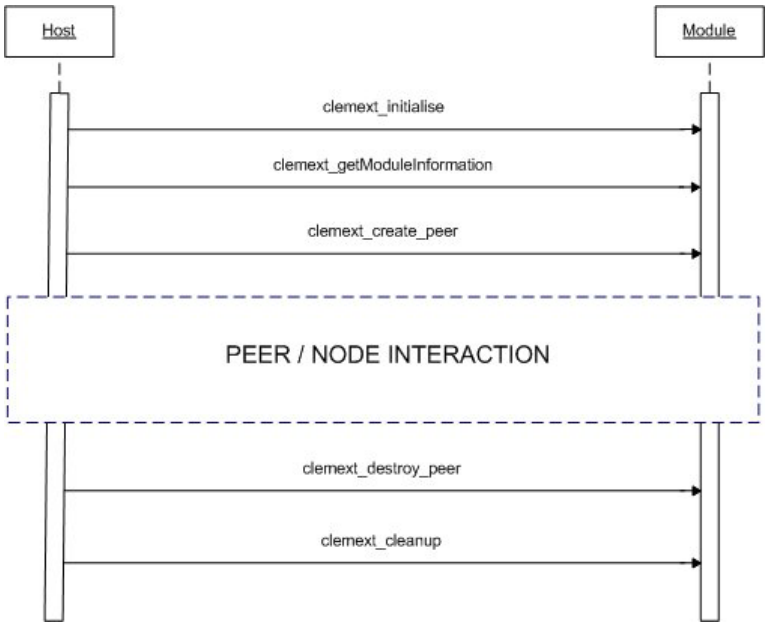

Abbildung 96. Typischer Prozessfluss

In der folgenden Abbildung wird der Peer/Knoten-Interaktionsblock dargestellt.

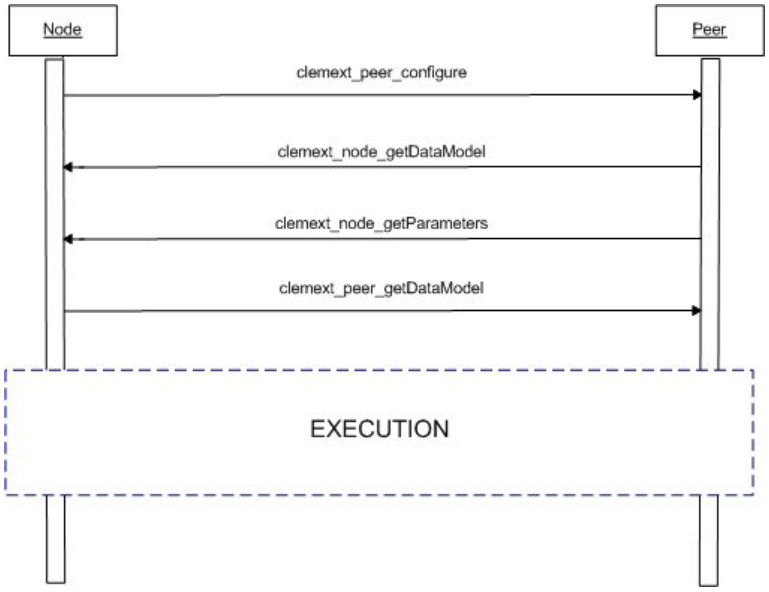

Abbildung 97. Typischer Peer/Knoten-Interaktionsblock

In der folgenden Abbildung wird ein typischer Ausführungsblock dargestellt.

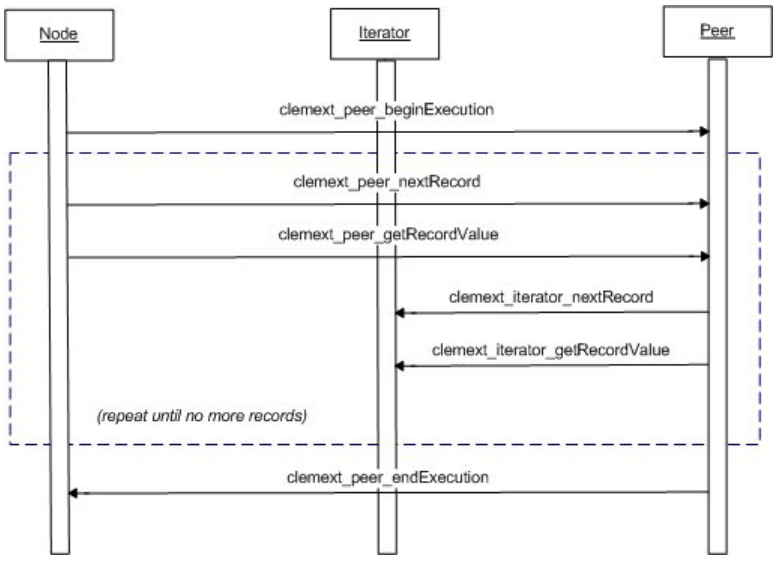

Abbildung 98. Typischer Ausführungsblock

Anmerkungen:

- Ein Erweiterungsmodul kann bereits beim Starten des Servers in den Serverprozess von IBM SPSS Modeler geladen werden. Es kann aber auch erst später geladen werden, wenn seine Services zum ersten Mal benötigt werden.
- Nach dem Laden des Moduls ruft der Host einmal die Servicefunktion `clemext_initialise` auf.
- Nach dem Laden und der Initialisierung des Moduls kann der Host das Modul anhand der Servicefunktion `clemext_getModuleInformation` abfragen.
- Nachdem das Modul geladen wurde, werden seine Services anhand der vom Modul bereitgestellten Peerobjekte aufgerufen. Im Modul selbst wird das Peerobjekt durch die Servicefunktion `clemext_create_peer` als Gegenstück zum Knotenobjekt des Hosts erstellt. Über dieses Peerobjekt wird die Ausführung einer Aufgabe nach Angabe der Hostanwendung verwaltet. Es können mehrere Peerobjekte des gleichen Typs vorhanden und gleichzeitig innerhalb eines Prozesses ausgeführt werden.
- Nach der Erstellung eines Peerobjekts kann dieses durch die Servicefunktion `clemext_peer_configure` konfiguriert werden.
- Nach der Konfiguration kann der Peer Callback-Funktionen ausführen (z. B. `clemext_node_getDataModel` und `clemext_node_getParameters`), um Informationen vom Client abzurufen.
- IBM SPSS Modeler ruft das Ausgabedatenmodell einer Peerinstanz anhand der Servicefunktion `clemext_peer_getDataModel` ab.
- Die Ausführung einer Peerinstanz beginnt mit dem Aufruf der Servicefunktion `clemext_peer_beginExecution`.
- Die Servicefunktion `clemext_peer_nextRecord` verschiebt den Fokus zum nächsten Datensatz im Ergebnisset des Peers (bzw. zum ersten Datensatz, wenn die Funktion zum ersten Mal aufgerufen wird). Danach wird die Servicefunktion `clemext_peer_getRecordValue` ausgeführt, die den Wert eines bestimmten Felds aus dem aktuellen Datensatz abruft.
- Die Iterator-Callback-Funktionen `clemext_iterator_nextRecord` und `clemext_iterator_getRecordValue` können vom CLEF-Modul aufgerufen werden, um die einzelnen Eingabedatensätze abzufragen und die Werte bestimmter Felder zurückzugeben.
- Die Ausführung einer Peerinstanz endet mit dem Aufruf der Servicefunktion `clemext_peer_endExecution`.
- Danach wird die Peerinstanz durch einen `clemext_destroy_peer`-Aufruf entfernt.
- Vor dem Entladen des Moduls ruft der Host die Servicefunktion `clemext_cleanup` auf.

- Das Entladen eines Moduls erfolgt entweder beim Beenden des Serverprozesses oder bereits früher, wenn die Services des Moduls nicht mehr benötigt werden.

Serverseitige API-Funktionen

In diesem Abschnitt werden einige Konzepte der serverseitigen API beschrieben:

- Informationen zum Knotentyp
- Datentypen für verschiedene Datenspeichertypen
- Serverseitige gemeinsam verwendete Bibliotheken
- Filespaces und temporäre Dateien
- SQL-Pushback zur Ausführung von SQL-Anweisungen in der Datenbank
- Austausch von Datenmodellinformationen zwischen IBM SPSS Modeler und der Erweiterung
- Ausgabedokumente
- C++-Helper

Knotentypen

In der Spezifikationsdatei hat eine Knotendefinition folgendes Format:

```
<Node id="ID" type="Knotentyp" .../>
```

Das Attribut `id` ist eine Zeichenfolge, die den Knoten eindeutig angibt.

Das Attribut `type` gibt den Typ des Knotens an. Folgende Typen sind möglich:

- Data reader (Datenleser)
- Data writer (Datenschreiber)
- Data transformer (Datentransformer)
- Model builder (Modellerstellung)
- Model applier (Modellanwender)
- Document builder (Dokumenterstellung)

Weitere Informationen finden Sie im Thema „Überblick über die Knoten” auf Seite 9.

Die Funktion `clemext_create_peer` enthält als Argumente sowohl die Werte des `id`- als auch des `type`-Attributs des `Node`-Elements.

Ein Erweiterungsmodul kann mehrere Knoten verschiedener Typen implementieren, von denen jeder unterschiedliche Funktionen ausführt. Von einem Modul können beispielsweise die folgenden Typen implementiert werden:

- Ein Datenleser- und ein Datenschreiberknoten für eine Datenquelle
- Modellerstellungs- und Modellanwendungsknoten für verschiedene Modellierungsalgorithmen
- Dokumenterstellungsknoten für verschiedene Diagrammtypen

Daten- und Speichertypen

Eine Peerinstanz ruft Eingabedaten mittels eines `clemext_iterator_getRecordValue`-Aufrufs mit dem Iterator ab, der der Instanz zu Beginn der Ausführung bereitgestellt wird. Die Ausgabedaten stellt die Peerinstanz als Antwort auf eine `clemext_peer_getRecordValue`-Anforderung des Hosts bereit. Die Daten werden in binärem Format im Speicher übertragen, wobei sich der Peer und der Host über den Datentyp abstimmen müssen.

Der binäre Datentyp wird vom Datenmodell bestimmt. Er richtet sich nach dem Speicherattribut eines Felds.

In folgender Tabelle sind die möglichen Speichertypen sowie die Datentypen aufgelistet, die zu deren Darstellung verwendet werden.

Tabelle 49. Speichertypen

Speichertyp	Dargestellt durch	Hinweise
string	`char *`	Zeichenfolgen sind immer UTF-8-codiert
real	`CLEMEXTReal`	Gleitkommawert mit doppelter Genauigkeit
integer	`CLEMEXTInteger`	Ganze Zahl mit Vorzeichen (64 Bit)
Datum	`CLEMEXTDate`	Ganze Zahl mit Vorzeichen (64 Bit), die die Anzahl der Tage seit dem 1. Januar 1970 angibt
time	`CLEMEXTTime`	Ganze Zahl mit Vorzeichen (64 Bit), die die Anzahl der Sekunden seit Mitternacht angibt
timestamp	`CLEMEXTTimestamp`	Ganze Zahl mit Vorzeichen (64 Bit), die die Anzahl der Sekunden seit dem 1. Januar 1970 (Mitternacht) angibt
unknown	-	Unbekannter Datentyp; die Werte können nicht dargestellt werden

Bibliotheken

Zur Unterstützung der Knotenausführung kann in der Spezifikationsdatei eine serverseitige gemeinsam verwendete Bibliothek deklariert werden. Der Pfad der gemeinsam verwendeten Bibliothek wird zur Lokalisierung der Bibliothek benötigt, um sie dynamisch in den Hostprozess laden zu können. Die gemeinsam verwendete Bibliothek muss alle erforderlichen API-Funktionen definieren. Weitere Informationen finden Sie im Thema „Freigegebene Bibliotheken" auf Seite 36.

Wenn in der Spezifikationsdatei ein Modulname angegeben ist (im Abschnitt `Execution` der Knotendefinition), wird dieser Name an den Parameter `nodeId` der Servicefunktion `clemext_create_peer` zur Erstellung des Peerobjekts übergeben. Auf diese Weise kann die Erweiterung das benötigte Peermodul erstellen. Der Wert des Parameters `nodeType` bestimmt darüber hinaus den Typ des erstellten Peers. Der Modulname kann auch leer bleiben, da eine gemeinsam verwendete Bibliothek jeweils nur ein Modul eines Typs implementieren kann.

Eventuell sind für eine gemeinsam verwendete Bibliothek, die ein Erweiterungsmodul implementiert, abhängige Bibliotheken erforderlich. Diese sollten sich im gleichen Verzeichnis befinden wie die gemeinsam verwendete Bibliothek der Erweiterung.

Temporäre Dateien

In der Clientspezifikationsdatei und dem Server-Erweiterungsmodul können Pfadnamen relativ zu einem **Dateibereich** (Filespace) angegeben werden. Es handelt sich hier um einen temporären, privaten Speicherbereich, in dem ein Peer während seiner Ausführung Dateien erstellen kann. Ein Dateibereich ist ein Unterverzeichnis des für einen Peer erstellten temporären Verzeichnisses auf dem Server. Er wird bei Bedarf erstellt und wieder gelöscht, wenn der Peer entfernt wird.

Solange der Filespace existiert, hat der Peer vollständige Kontrolle über diesen Speicherbereich. Der vollständige Pfadname des Filespace ist in einem **Knoteninformationsdokument** angegeben. Dieses Dokument enthält Informationen im XML-Format, die als Ergebnis der Ausführung einer `clemext_node_getNodeInformation`-Callback-Funktion zurückgegeben werden. Weitere Informationen finden Sie im Thema „Knoteninformationsdokument" auf Seite 194.

SQL-Pushback

Ein IBM SPSS Modeler-Stream liest die Daten einer SQL-Datenbank ein und verarbeitet die Daten anschließend. Erfahrene Benutzer können diesen Vorgang anhand eines Pushbacks der SQL-Anweisungen effizienter gestalten, wodurch die Anweisungen direkt in der Datenbank ausgeführt werden.

SQL-Pushback wird von einigen IBM SPSS Modeler-Standardknoten unterstützt und die serverseitige API enthält Funktionsaufrufe, mit der ein SQL-Pushback auch in CLEF-Knoten implementiert werden kann.

Die Servicefunktion `clemext_peer_getSQLGeneration` generiert aus einer Peerinstanz SQL-Anweisungen und verlegt die SQL-Ausführung in die Datenbank. Bei einem Datenleserknoten müssen die generierten SQL-Anweisungen in sich abgeschlossen sein, damit das Peer-Ergebnisset erstellt werden kann. Bei allen anderen Knotentypen hängen die generierten SQL-Anweisungen in der Regel von den für vorgeordnete Knoten generierten SQL-Anweisungen ab, die die Eingabedaten für den Peer bereitstellen. Ein Peer kann das SQL der vorgeordneten Knoten durch einen Aufruf der `clemext_node_getSQLGeneration`-Callback-Funktion mit dem ihm zugeordneten Knotenhandle abrufen.

Behandlung des Datenmodells

Einige serverseitige API-Aufrufe bewirken den Austausch von Datenmodellinformationen zwischen IBM SPSS Modeler und dem Erweiterungsmodul:

- `clemext_node_getDataModel` ruft das Eingabedatenmodell eines Knotens ab
- `clemext_peer_getDataModel` ruft das Ausgabedatenmodell einer Peerinstanz ab
- `clemext_node_getOutputDataModel` ruft das Ausgabedatenmodell eines Knotens ab

Andere Aufrufe beziehen sich auf die Methoden zur Übertragung der Daten in und aus dem Modul. Das Datenmodell bestimmt den Index, der für das Look-up der Feldwerte in den folgenden Funktionen verwendet wird (diese Funktionen geben den Wert eines bestimmten Felds aus dem zuletzt abgerufenen Eingabedatensatz zurück):

- `clemext_peer_getRecordValue`
- `clemext_iterator_getRecordValue`

IBM SPSS Modeler ruft `clemext_node_getDataModel` auf, um Informationen über die Felder des Eingabedatenmodells abzurufen. Die Informationen werden im XML-Format zurückgegeben. Beispiel:

```
<DataModel>
     <Fields>
          <Field name="abc" storage="string" type="set" />
          <Field name="uvw" storage="integer" type="range" />
          <Field name="xyz" storage="real" type="range" />
     <Fields>
</DataModel>
```

Anhand dieser Informationen kann ein Modul den Feldindex bereitstellen, wenn es mittels der `clemext_iterator_getRecordValue`-Funktion Werte aus einem Eingabedatensatz abruft.

Auf welche Weise das Modul das Eingabedatenmodell modifiziert, wird durch den Wert des `mode`-Attributs des Elements `OutputDataModel` der Spezifikationsdatei bestimmt. Das Modul kann folgende Modifizierungen durchführen:

- Es kann das Modell durch Hinzufügen neuer Felder erweitern.
- Es kann das Modell durch Entfernen oder Umbenennen vorhandener Felder ändern.
- Es kann das vorhandene Modell durch neue Felder ersetzen.
- Es kann das Modell unverändert beibehalten.

Die folgenden Beispiele zeigen, wie ein Eingabedatenmodell erweitert bzw. ersetzt wird.

Beispiel - Erweitern des Eingabedatenmodells: Dies ist der einfachste Fall: Ein Modul wird so eingerichtet, dass es neue Felder hinzufügt und deren Werte einstellt, jedoch die Werte vorhandener Felder nicht entfernt bzw. unverändert lässt.

In diesem Beispiel wird davon ausgegangen, dass die Knotendefinition der Spezifikationsdatei die folgenden Anweisungen enthält:

```
<OutputDataModel mode="extend">
     <AddField name="field1" storage="string" ... />
     <AddField name="field2" storage="real" ... />
     ...
</OutputDataModel>
```

Die Definition des Ausgabedatenmodells legt in diesem Fall fest, dass das Modell alle Felder des Eingabedatenmodells sowie zusätzlich die beiden im Element `OutputDataModel` angegebenen Felder enthalten soll. Das Ausgabedatenmodell besteht also aus fünf Feldern.

Die Funktion `clemext_peer_getDataModel` gibt nur die Informationen der hinzugefügten Felder zurück. Beispiel:

```
<DataModel>
     <Fields>
          <Field name="field1" storage="string" ... />
          <Field name="field2" storage="real" ... />
     <Fields>
</DataModel>
```

Die zurückgegebenen Informationen müssen mit den "type"- und "number"-Werten (jedoch nicht mit dem "name"-Wert) des Elements `<AddField>` der Spezifikationsdatei übereinstimmen.

Ein Modul kann mit der Callback-Funktion `clemext_node_getOutputDataModel` die Informationen der Felder abrufen, von denen IBM SPSS Modeler erwartet, dass sie hinzugefügt werden. Diese Informationen können als Antwort auf einen `clemext_peer_getDataModel`-Aufruf direkt an IBM SPSS Modeler zurückgegeben werden. Diese Vorgehensweise empfiehlt sich vor allem dann, wenn die Logik der Spezifikationsdatei für die Erstellung und Benennung der Ausgabefelder kompliziert ist.

Das Modul stellt die neuen Werte für jeden Ausgabedatensatz bereit, wenn IBM SPSS Modeler die Funktion `clemext_peer_getRecordValue` aufruft. Die Feldindizes der neuen Felder beginnen nach dem letzten Index der Eingabefelder. In diesem Beispiel enthält das Eingabedatenmodell drei Felder (mit den Indexpositionen 0, 1 und 2). Den beiden neuen Ausgabefeldern werden daher die Feldindizes 3 und 4 zugewiesen. IBM SPSS Modeler ruft die Funktion `clemext_peer_getRecordValue` nicht mit den Feldindizes der Eingabefelder auf, da diese Felder vom Modul nicht geändert werden können.

Beispiel - Ersetzen des Eingabedatenmodells (1): In diesem Beispiel entfernt das Erweiterungsmodul in seiner Ausgabe sämtliche Felder des Eingabedatenmodells und ersetzt sie durch neue Felder.

Die Spezifikationsdatei enthält folgende Definition:

```
<OutputDataModel mode="modify">
     <AddField name="key" storage="integer" ... />
     <AddField name="field1" storage="real" ... />
     <AddField name="field2" storage="real" ... />
     ...
</OutputDataModel>
```

In diesem Fall beschreiben die von einem `clemext_peer_getDataModel`-Aufruf zurückgegebenen XML-Daten alle Felder des Ausgabedatenmodells:

```
<DataModel>
     <Fields>
          <Field name="key" storage="integer" ... />
          <Field name="field1" storage="real" ... />
          <Field name="field2" storage="real" ... />
     <Fields>
</DataModel>
```

Das erste Ausgabefeld (key) eines clemext_peer_getRecordValue-Aufrufs erhält den Feldindex 0, das zweite Ausgabefeld (field1) erhält den Feldindex 1 usw.

Beispiel - Ersetzen des Eingabedatenmodells (2): Auch in diesem Beispiel wird das Eingabedatenmodell durch das von der Erweiterung bereitgestellte Ausgabedatenmodell ersetzt. Im Gegensatz zum vorherigen Beispiel ist das Ausgabedatenmodell jedoch nicht in der Spezifikationsdatei definiert, sondern es wird während der Laufzeit vom Erweiterungsmodul auf dem Server berechnet. Die Spezifikationsdatei enthält folgende Definition:

```
<OutputDataModel mode="modify" method="sharedLibrary" libraryId="myLibraryId" />
```

Zur Berechnung des Ausgabedatenmodells ruft IBM SPSS Modeler zunächst clemext_peer_configure und danach clemext_peer_getDataModel auf. Wie im vorherigen Beispiel wird keines der Felder des Eingabedatenmodells automatisch auch im Ausgabedatenmodell übernommen. Dieses wird vollständig durch die vom clemext_peer_getDataModel-Aufruf zurückgegebene Antwort definiert.

Note: Wenn das Erweiterungsmodul das Ausgabedatenmodell wie in diesem Beispiel auf dem Server definiert, kann das Modul das Ausgabedatenmodell nicht mit der Funktion clemext_node_getOutputDataModel abrufen, da dies zu einem Fehler aufgrund einer ungültigen Operation führen würde.

XML-Ausgabedokumente

Einige Service- und Callback-Funktionen verwenden XML-Ausgabedokumente zur Übergabe der Informationen zwischen dem Host und dem Erweiterungsmodul. Hierzu stehen mehrere verschiedene Dokumente zur Verfügung, die in der folgenden Tabelle aufgeführt sind.

Tabelle 50. XML-Ausgabedokumente

XML-Ausgabedokument	Hinweise	Zurückgegeben von Aufruf
Katalogdokument	Enthält die Liste der Werte für ein Steuerelement, das einem Katalog zugeordnet ist	clemext_peer_getCatalogInformation
Datenmodelldokument	Beschreibt den ein- bzw. ausgehenden Feldsatz eines Knotens	clemext_peer_getDataModel clemext_node_getDataModel clemext_node_getOutputDataModel
Fehlerdetaildokument	Enthält Informationen zu einem Fehler oder einer anderen Bedingung	clemext_peer_getErrorDetail
Ausführungsanforderungsdokument	Beschreibt die für eine Peerinstanz erforderlichen Ausführungsvoraussetzungen, beispielsweise einen Datencache oder erforderliche Eingabefelder	clemext_peer_getExecutionRequirements
Hostinformationsdokument	Stellt Informationen zur Hostumgebung bereit, darunter die ID, die Beschreibung, die Version und der Anbieter des Produkts sowie Copyright- und Plattformangaben	clemext_host_getHostInformation
Modulinformationsdokument	Stellt Informationen zum Erweiterungsmodul bereit, darunter die ID, die Beschreibung, die Version und der Anbieter des Moduls sowie Copyright- und Lizenzangaben	clemext_getModuleInformation

Tabelle 50. XML-Ausgabedokumente (Forts.)

XML-Ausgabedokument	Hinweise	Zurückgegeben von Aufruf
Knoteninformationsdokument	Stellt Informationen zu dem einer Peerinstanz zugeordneten Knoten bereit, darunter die ID und der Typ des Knotens sowie die Dateibereichsangaben	`clemext_node_getNodeInformation`
Parameterdokument	Enthält die Konfigurationsparameter des Clientknotens; der Inhalt wird durch die Erweiterung bestimmt	`clemext_node_getParameters`
SQL-Generierungsdokument	Beschreibt, wie die Ausführung eines Peers in SQL-Anweisungen umgesetzt wird	`clemext_peer_getSQLGeneration` `clemext_node_getSQLGeneration`
Statusdetaildokument	Enthält ergänzende Informationen über den Fortschritt, über Warnungen oder über andere Bedingungen, die während der Ausführung auftreten	`clemext_progress_report`

Katalogdokument: Ein Katalogdokument beschreibt den Inhalt eines Katalogs, der eine Liste von Werten enthält, die über ein Steuerelement der Benutzerschnittstelle angezeigt werden kann.

Das Modul CLEF implementiert wie folgt einen Aufruf an `getCatalogInformation`:

```
CLEMEXTStatus
getCatalogInformation(
          const char *catalogId,
            char* buffer,
            size_t buffer_size,
            size_t* data_size,
            CLEMEXTErrorCode* errorCode) {
      ...
}
```

Dabei ist `catalogId` die ID eines bestimmten Katalogs, wie im Element `Catalog` in der Spezifikationsdatei definiert.

Diese Funktion gibt das Katalogdokument zurück.

Beispiel

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<CatalogInformation>
     <CatalogEntry>
          <CatalogValue>apples</CatalogValue>
          <CatalogValue>0</CatalogValue>
     </CatalogEntry>
     <CatalogEntry>
          <CatalogValue>oranges</CatalogValue>
          <CatalogValue>1</CatalogValue>
     </CatalogEntry>
     <CatalogEntry>
          <CatalogValue>bananas</CatalogValue>
          <CatalogValue>2</CatalogValue>
     </CatalogEntry>
</CatalogInformation>
```

Datenmodelldokument: Ein Datenmodelldokument beschreibt das **Datenmodell**, d. h. das Set der in einen Knoten eingehenden bzw. von einem Knoten abgehenden Felder mit ihren Namen, Typen und zugehörigen Informationen. Es gekapselt die in einem Typknoten verfügbaren Informationen.

Ein Peer, der keine Eingabe annimmt (ein Quellenknoten), verfügt über ein leeres Eingabemodell, während ein Peer, der keine Ausgabe generiert (ein Endknoten), über ein leeres Ausgabemodell verfügt. Ein Peer, der Eingaben annimmt und Ausgaben generiert (ein Prozessknoten), muss wissen, wie er aus der Eingabe sein Ausgabemodell berechnet.

Ein Peer ruft sein Eingabedatenmodell ab, indem er `clemext_node_getDataModel` mit dem zugehörigen Knotenhandle aufruft. Sein Ausgabedatenmodell stellt er als Antwort auf eine `clemext_peer_getDataModel`-Anforderung des Hosts bereit.

Jedes Datenmodell kann als Datenwörterbuch dargestellt werden, das alle Felder des Datenmodells mit ihren Eigenschaften auflistet. Ein Knoten stellt das Eingabedatenmodell für einen Peer immer in dieser Form bereit. Das von einem Peer generierte Ausgabedatenmodell kann in der Form eines Datenwörterbuchs übergeben oder als Operationssequenz dargestellt werden (Feld hinzufügen, Feld entfernen, Feld ändern), die auf das Eingabemodell angewendet werden muss. Bei einigen Knoten vereinfacht sich dadurch das Ausgabemodell erheblich.

Die Reihenfolge der Felder in einem Datenmodelldokument ist wichtig. Sie legt die Reihenfolge fest, in der die Daten des entsprechenden Eingabe- oder Ausgabedatasets übergeben werden.

Ein Datenmodell kann unvollständig sein und nur eine Teilspezifikation der Daten bereitstellen. Ein Eingabemodell, das alle erforderlichen Spezifikationen für den Ausführungsplan des Peers enthält, wird für diesen Peer als **ausführbar** bezeichnet. Ein ausführbares Datenmodell muss immer den binären Typ jedes Felds enthalten, damit die Eingabe- und Ausgabedaten korrekt zusammengestellt werden können.

Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<DataModel>
  <Fields>
    <Field name="Age" type="range" storage="integer" direction="in">
      <Range minValue="15" maxValue="74"/>
    </Field>
    <Field name="Sex" type="flag" storage="string">
      <Values>
        <Value value="F" flagValue="false" displayLabel="Female"/>
        <Value value="M" flagValue="true" displayLabel="Male"/>
      </Values>
    </Field>
    <Field name="BP" type="orderedSet" storage="integer">
      <Values>
        <Value value="-1" />
        <Value value="0" />
        <Value value="1" />
      </Values>
    </Field>
    <Field name="Cholesterol" type="flag" storage="string">
      <Values>
        <Value value="NORMAL" flagValue="false"/>
        <Value value="HIGH" flagValue="true"/>
      </Values>
    </Field>
    <Field name="Na" type="range" storage="real" displayLabel="Blood sodium">
      <Range minValue="0.500517" maxValue="0.899774"/>
    </Field>
    <Field name="K" type="range" storage="real" displayLabel="Potassium concentration">
```

```
      <Range minValue="0.020152" maxValue="0.079925"/>
    </Field>
    <Field name="Drug" type="set" storage="string" direction="out">
      <Values>
        <Value value="drugA"/>
        <Value value="drugB"/>
        <Value value="drugC"/>
        <Value value="drugX"/>
        <Value value="drugY"/>
      </Values>
    </Field>
  </Fields>
</DataModel>
```

Fehlerdetaildokument: In einem Fehlerdetaildokument werden Nachrichten (Fehlernachrichten, Warnungen, Informationen) zurück an IBM SPSS Modeler gesendet. Es enthält Informationen über einen Fehler oder eine andere Bedingung. Ein Erweiterungsmodul kann als Antwort auf eine `clemext_peer_getErrorDetail`-Anforderung des Hosts ein Fehlerdetaildokument mit Informationen zu einem modulspezifischen Fehler bereitstellen.

Die Fehlerinformationen umfassen ein oder mehrere `Diagnostic`-Elemente, wobei jedes dieser Elemente mindestens einen Fehlercode, eine Nachricht sowie ein oder mehrere Parameter mit weiteren Informationen für die Nachricht enthält. Die Fehlercodes entsprechen den Werten der `StatusCode`-Elemente der Spezifikationsdatei.

Die Nachricht kann in verschiedenen Sprachvarianten vorliegen oder der Client kann die lokalisierte Nachricht mithilfe des Fehlercodes aus einem Ressourcenbundle abrufen. Eine Abfolge mehrerer Diagnostic-Elemente beschreibt eine kausale Fehlerkette.

Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<ErrorDetail>
  <Diagnostic code="123" severity="error">
    <Message>You can't do that ({0})</Message>
    <Parameter>Permission denied</Parameter>
  </Diagnostic>
  <Diagnostic code="456" severity="warning">
    <Message>That was silly!</Message>
    <Message lang="fr">Quel idiot!</Message>
  </Diagnostic>
</ErrorDetail>
```

Ausführungsanforderungsdokument: Ein Ausführungsanforderungsdokument beschreibt die für eine Peerinstanz erforderlichen Ausführungsvoraussetzungen. Eine Peerinstanz kann ein Ausführungsanforderungsdokument als Antwort auf eine `clemext_peer_getExecutionRequirements`-Anforderung vom Host bereitstellen. Vor einem `clemext_peer_beginExecution`-Aufruf an den Peer konsultiert der Host das Ausführungsanforderungsdokument, um die erforderliche Ausführungsumgebung bereitzustellen.

Der Host kann einen Datencache-Service bereitstellen, um sicherzustellen, dass das Modul die Eingabedaten mittels der `clemext_iterator_rewind`-Funktion mehrmals übergeben kann.

Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<ExecutionRequirements> <Cache/><!-- this ensures that the
CLEF module can make multiple passes
over the input data --> </ExecutionRequirements>
```

Hostinformationsdokument: Ein Hostinformationsdokument beschreibt die Hostumgebung. Ein Erweiterungsmodul kann mittels eines Aufrufs von `clemext_host_getHostInformation` mit dem entsprechenden Host-Handle Informationen über einen Host abrufen.

Die zurückgegebenen Informationen umfassen die Kennung, die Beschreibung, die Version und den Hersteller des Produkts sowie Copyright- und Plattformangaben.

Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<HostInformation>
  <Host name="clemlocal" externalEncoding="cp1252" language="english_us"
        locale="English_United Kingdom.1252" provider="IBM" version="17.1" platform=
        "Windows XP SP2" copyright="Copyright 1995-2011 IBM Corp. All rights reserved.">
    <VersionDetail major="12" minor="0"/>
    <PlatformDetail osType="windows" osName="WindowsNT" osMajor="5" osMinor="1"/>
    <LibraryDetail path="C:\Programme\IBM\SPSS\Modeler\17.1\ext\bin\my.module\myModule.dll"/>
  </Host>
</HostInformation>
```

Modulinformationsdokument: Ein Modulinformationsdokument beschreibt ein Erweiterungsmodul. Ein Erweiterungsmodul muss als Antwort auf eine `clemext_getModuleInformation`-Anforderung des Hosts ein Modulinformationsdokument bereitstellen.

Die zurückgegebenen Informationen umfassen die Kennung, die Beschreibung, die Version und den Hersteller des Moduls sowie Copyright- und Lizenzangaben.

Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<ModuleInformation>
  <Module name="MyModule" provider="My Company Inc." version="10.1.0.329"
          copyright="Copyright 2006 My Company Inc. All rights reserved.">
    <VersionDetail major="10" minor="1" release="0" build="329"/>
    <Licence code="1234" type="mandatory"/>
    <Description>Provides a thorough test of the new extensions framework.</Description>
  </Module>
</ModuleInformation>
```

Knoteninformationsdokument: Ein Knoteninformationsdokument beschreibt den mit einer Peerinstanz verbundenen Knoten. Eine Peerinstanz kann mittels eines `clemext_node_getNodeInformation`-Aufrufs mit dem entsprechenden Knotenhandle Informationen über einen Knoten abrufen. Knoteninformationen umfassen die Kennung, den Typ und den Filespace des Knotens.

Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<NodeInformation>
  <Node name="databaseImport" type="dataReader">
    <FileSpace path="C:\Programme\IBM SPSS Modeler Server
17.1\tmp\ext-8005-6711-01"/>
  </Node>
</NodeInformation>
```

Parameterdokument: Ein Parameterdokument enthält Informationen über alle `Property`-Elemente einer Spezifikationsdatei. Die Informationen werden in Form von Konfigurationsparametern zurückgegeben, die ein Peer mittels eines `clemext_node_getParameters`-Aufrufs mit dem entsprechenden Knotenhandle abrufen kann.

Ein Parameter hat einen Namen und einen Wert, wobei der Wert Folgendes sein kann:

- Einfacher Wert (Zeichenfolge)
- Verschlüsselter Wert (Schlüssel und Wert)
- Strukturierter Wert (Satz mit benannten Werten)
- Werteliste

Der Inhalt eines Parameterdokuments wird durch das Erweiterungspaket bestimmt. Die Parameter sind in der clientseitigen Spezifikationsdatei definiert und werden vom Erweiterungsmodul auf dem Server interpretiert.

Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<Parameters>
  <Parameter name="linesToScan" value="50"/>
  <Parameter name="useCaption" value="true"/>
  <Parameter name="caption" value="My Caption"/>
  <Parameter name="captionPosition" value="north"/>
  <Parameter name="defaultAggregation">
    <ListValue>
      <Value value="min"/>
      <Value value="max"/>
      <Value value="mean"/>
      <Value value="stddev"/>
    </ListValue>
  </Parameter>
</Parameters>
```

SQL-Generierungsdokument: Ein SQL-Generierungsdokument beschreibt, wie die Ausführung eines Peers in SQL-Anweisungen umgesetzt wird.

Ein Peer kann ein SQL-Generierungsdokument als Antwort auf eine `clemext_peer_getSQLGeneration`-Anforderung vom Host bereitstellen. Wenn dem Host ein SQL-Generierungsdokument vorliegt, zieht er die Ausführung der SQL-Anweisungen der internen Ausführung des Peers vor.

Ein Peer, der Eingabedaten annimmt, kann seine Eingabe-SQL-Anweisungen abrufen, indem er `clemext_node_getSQLGeneration` mit dem zugehörigen KnotenHandle aufruft.

Der wichtigste Teil eines SQL-Generierungsdokuments ist eine SQL-Anweisung, die das Ausführungsverhalten eines Knotens oder eines Streamfragments repliziert. Bei einem Knoten, der Daten generiert, also einem Datenleser- oder Datentransformationsknoten, muss es sich um eine `SELECT`-Anweisung handeln. Dieser muss ein Wörterbuch beigefügt sein, das die Feldnamen des Datenmodells den Spaltennamen der `SELECT`-Anweisung zuordnet.

Darüber hinaus kann ein SQL-Generierungsdokument auch die Eigenschaften der Datenbankverbindung enthalten, über die die Anweisung ausgeführt wird (z. B. den Datenquellen- und den Produktnamen). Diese Eigenschaften helfen dem Peer bei der Ermittlung der zu generierenden SQL-Anweisungen.

Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<SqlGeneration>
  <Properties
    datasourceName="SQL Server"
    databaseName="DataMining"
    serverName="GB1-RDUNCAN1"
    passwordKey="PW0"
    userName="fred"
    dbmsName="Microsoft SQL Server"
```

```
    dbmsVersion="09.00.1399"/>
  <Statement>
    <Bindings>
      <Binding columnName="C0" fieldName="ID"/>
      <Binding columnName="C1" fieldName="START_DATE"/>
    </Bindings>
    <TableParameters>
  <TableParameter name="${TABLE26}" value="dbo.DRUG4N"/>
    </TableParameters>
    <Sql>
      SELECT
      T0.ID AS C0,T0."START_DATE" AS C1
      FROM ${TABLE26} T0
      WHERE (T0."START_DATE" > '2003-01-01')
      ORDER BY 2 ASC
    </Sql>
  </Statement>
</SqlGeneration>
```

Statusdetaildokument: Ein Statusdetaildokument stellt Informationen über den Fortschritt sowie Informationen über unkritische Warnungen und andere Bedingungen bereit, die während der Ausführung auftreten. Statusdetaildokumente können vom Erweiterungsmodul mithilfe der Callback-Funktion `clemext_progress_report` asynchron versendet werden.

Ein Statusdetaildokument enthält mindestens ein `Diagnostic`-Element, wobei jedes Element mindestens einen Bedingungscode, eine Nachricht (außer diese ist in einer Eigenschaftendatei angegeben) sowie einen oder mehrere Parameter mit weiteren Informationen für die Nachricht enthält. Das Element `StatusDetail` kann ebenfalls ein optionales `destination`-Attribut enthalten, um die Nachricht an eine der folgenden Stellen weiterzugeben:

- Eine lokale, von IBM SPSS Modeler verwaltete Tracedatei
- Der Client (für Nachrichten an den Benutzer)
- Alle (die Nachricht wird an alle möglichen Ziele gesendet)

Das Format des Elements `Diagnostic` sieht wie folgt aus:

```
<Diagnostic code="ganze_Zahl" severity="Schweregrad">
     <Message>Nachrichtentext</Message>
     <Parameter>Wert</Parameter>
</Diagnostic>
```

Dabei gilt Folgendes:

`code` (erforderlich) ist eine ganze Zahl, die den Bedingungscode angibt.

`severity` gibt den Schweregrad der Bedingung an und ist auf einen der folgenden Werte gesetzt: `unknown`, `information`, `warning`, `error` oder `fatal`

Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<StatusDetail destination="client">
  <Diagnostic code="654" severity="information">
    <Message>Processed {0} records</Message>
    <Parameter>10000</Parameter>
  </Diagnostic>
</StatusDetail>
```

Verwenden lokalisierter Nachrichten

Wenn Sie lokalisierte Nachrichten aus einer Eigenschaftendatei verwenden möchten, lassen Sie das Element `Message` aus dem Statusdetaildokument weg und verwenden in der Spezifikationsdatei Nachrichtenschlüssel wie im folgenden Beispiel:

```
...
<Execution ...>
...
    <StatusCodes>
        ...
        <StatusCode code="21" status="warning" messageKey="fieldIgnoredMsg.LABEL"/>
        ...
    </StatusCodes>
</Execution>
...
```

Die Datei "messages.properties" würde dann Folgendes enthalten:

```
fieldIgnoredMsg.LABEL=Field "{0}" cannot be used for model building and was ignored
```

Im Statusdetaildokument können Sie dann einen Parameter (wie einen Feldnamen) verwenden, der in der lokalisierten Nachricht ersetzt werden soll, z. B.:

```
<?xml version="1.0" encoding="utf-8"?>
<StatusDetail>
    <Diagnostic code="21">
        <Parameter>BP</Parameter>
    </Diagnostic>
</StatusDetail>
```

C++-Helper

Einige der Beispielknoten von CLEF enthalten vordefinierte C++-Quellendateien. Diese Dateien werden auch als **Helper** bezeichnet. C++-Helper-Dateien fungieren als Wrapper für einen Teil der C-basierten, serverseitigen API und lassen sich problemlos in ein C++ CLEF kompilieren.

Tabelle 51. C++-Helper

Helper	Beschreibung
BufferHelper	Verwaltet die in der CAPI verwendeten, skalierbaren Speicherpuffer
DataHelper	Hilft bei der Aufnahme von Datenlese- und Datenschreiboperationen
HostHelper	Nimmt das CLEMEXT HostHandle-Objekt auf
XMLHelper	Nimmt die XML-Verarbeitung auf

Ein Helper ist immer ein Dateienpaar bestehend aus einer *.cpp*- und einer *.h*-Datei (z. B. *BufferHelper.cpp* und *BufferHelper.h*).

Informationen zum Anzeigen dieser Helperdateien finden Sie im Abschnitt „Untersuchen des Quellcodes" auf Seite 29.

Ausführlichere Beschreibungen dieser Dateien finden Sie in der Dokumentation der serverseitigen API, die Sie wie folgt aufrufen:

1. Wählen Sie im API-Dokumentationsfenster von CLEF die Option **Server-side API overview** (Überblick über die serverseitige API) aus.
2. Klicken Sie auf die Registerkarte **Dateien**.
3. Klicken Sie auf den Namen der `.h`-Datei des Helpers, dessen Informationen Sie anzeigen möchten.
4. Klicken Sie unter **Data Structures** (Datenstrukturen) auf den zugehörigen Klassennamen, um die Dokumentation anzuzeigen.

Informationen zum Zugriff auf die API-Dokumentation von CLEF finden Sie im Abschnitt „Dokumentation zu den CLEF-APIs" auf Seite 177.

Fehlerbehandlung

Jeder API-Funktionsaufruf gibt einen Statuscode (`CLEMEXTStatus`) und optional einen modulspezifischen Fehlercode (`CLEMEXTErrorCode`) zurück. Der Statuscode kann `success` (kein Fehler) lauten oder einen der für die API-Funktion definierten Fehlercodes enthalten (bei diesen handelt es sich fast immer um modulspezifische Fehler). Wenn es sich nicht um einen modulspezifischen Fehler handelt, lautet der modulspezifische Fehlercode 0.

Die Statuscodenachrichten werden von IBM SPSS Modeler bereitgestellt. Beschreibungen der allgemeinen Statuscodes finden Sie in der Dokumentation der serverseitigen API, die Sie wie folgt aufrufen:

1. Wählen Sie im API-Dokumentationsfenster von CLEF die Option **Server-side API overview** (Überblick über die serverseitige API) aus.
2. Klicken Sie auf die Registerkarte **Module**.
3. Wählen Sie **Common Status Codes** (Allgemeine Statuscodes) aus.

Informationen zum Zugriff auf die API-Dokumentation von CLEF finden Sie im Abschnitt „Dokumentation zu den CLEF-APIs" auf Seite 177.

Modulspezifische Nachrichten können wie folgt bereitgestellt werden:

- In der Spezifikationsdatei (im Element `StatusCodes` des Abschnitts "Module")
- In einem Ressourcenbundle, auf das die Spezifikationsdatei verweist
- Durch das Erweiterungsmodul

Für einen modulspezifischen Fehlercode kann das Modul zusätzliche Fehlerinformationen bereitstellen. Diese bestehen aus einer Standardfehlernachricht (bei Fehlern, die nicht in der Spezifikationsdatei oder im Ressourcenbundle definiert sind) und Parametern, die in die Nachricht eingefügt werden. Kausale Fehlerketten können in mehreren Fehlernachrichten beschrieben werden.

Ein Fehlerbericht für den Client hat folgendes Format:

Knotenbeschriftung`:`*Nachricht*

Dabei gilt Folgendes:

- *Knotenbeschriftung* ist der Wert des `label`-Attributs des `Node`-Elements, in dem das Modul definiert ist.
- *Nachricht* ist der Nachrichtentext. Dieser kann entweder vom Server bereitgestellt oder in der Spezifikationsdatei definiert werden. Bei lokalisierten Modulen wird der Nachrichtentext in einer *.properties*-Datei bereitgestellt.

XML-Parser-API

IBM SPSS Modeler enthält einen Xerces-C-XML-Parser von Apache, der verschiedene Callbacks bereitstellt, mit denen ein Modul XML-Daten lesen und schreiben kann. Sie können statt dieses Parsers auch Ihren eigenen XML-Parser verwenden.

Verwenden der serverseitigen API

So fügen Sie einem Knoten serverseitige Funktionsaufrufe hinzu:

1. Erstellen Sie in C++ die *.cpp*- und *.h*-Quellendateien mit den gewünschten Funktionsaufrufen.
2. Kompilieren Sie die Quellendateien in eine Dynamic Link Library (*.dll*-Datei).
3. Fügen Sie in der Spezifikationsdatei einen Verweis auf die *.dll*-Datei ein. Beispiel:

```
<Resources>
.
     <SharedLibrary id="mylib1" path="mycorp.mynode/mylib" />
.
</Resources>
```

Weitere Informationen finden Sie im Thema „Freigegebene Bibliotheken” auf Seite 36.

Sehen Sie sich hierzu auch den Quellcode der in dieser Version bereitgestellten Beispielknoten an. Sie finden dort eventuell einige nützliche Anwendungsbeispiele. Weitere Informationen finden Sie im Thema „Untersuchen des Quellcodes” auf Seite 29.

Richtlinien für die serverseitige Programmierung

Der serverseitige DLL-Teil (Dynamic Link Library) eines CLEF-Moduls muss eine Reihe von Richtlinien einhalten, damit das Modul korrekt arbeitet und der Betrieb von IBM SPSS Modeler nicht beeinträchtigt wird. Ein CLEF-Modul muss:

- selbstständige Peerausführung sicherstellen.
- mehrere Peerinstanzen in einem einzigen Prozess unterstützen.
- Threadsicherheit garantieren.
- Ändern der Thread- oder Prozessumgebung vermeiden.
- Threadnutzung im Modul beschränken.
- Anforderungen zum Abbrechen der Ausführung korrekt behandeln.
- unterbrochene Systemaufrufe neu starten (UNIX).
- Aufruf von CoInitialize oder CoUninitialize (Windows) sorgfältig abwickeln.
- Annahmen über den Zeitpunkt vermeiden, an dem das Modul entladen wird.
- beim Start von Unterprozessen sorgfältig vorgehen.
- Schreiben an Standardausgabe oder Standardfehler vermeiden.

Die folgenden Abschnitte behandeln jedes dieser Themen ausführlicher.

Sicherstellen der selbstständigen Peerausführung

Peerinstanzen sollten das Vorhandensein anderer Peerinstanzen innerhalb des IBM SPSS Modeler-Serverprozesses nicht voraussetzen. IBM SPSS Modeler kann die Ausführung so planen, dass Peerinstanzen, die direkt benachbarten Knoten in einem Stream entsprechen, in anderen Phasen ausgeführt werden und sich Existenz und Ausführung der Instanzen nicht überlagern.

Peerinstanzen sollten daher eigenständig sein und nicht versuchen, direkt mit anderen Peerinstanzen zu kommunizieren, etwa durch Pipes oder Sockets. Sämtliche Kommunikation zwischen verschiedenen Peerinstanzen muss direkt durch Lesen und Schreiben von Daten in den Stream oder indirekt durch einen externen Agenten erfolgen (z. B. durch einen Datenbankserver, der gemeinsam genutzte Daten zwischen Peers verwaltet).

Unterstützen mehrerer Peerinstanzen in einem einzigen Prozess

Endbenutzer können beim Ausführen eines Streams mehrere Peerinstanzen eines bestimmten CLEF-Moduls (d. h. mehrere Knoten desselben Typs) in einem Serverprozess erstellen. So werden alle statischen Daten im CLEF-Modul von mehreren Peerinstanzen gemeinsam genutzt und sollten nicht zum Speichern von privaten Daten eines Peerobjekts verwendet werden. Beispiele für statische Daten sind statische Mitglieder von C++-Klassen sowie globale oder statische Variablen in C-Kompilierungseinheiten.

API-Funktionen des CLEF-Moduls müssen ablaufinvariant sein und Systemaufrufe vermeiden, die nicht ablaufinvariant sind. Beispiel: Wenn eine Peerinstanz Eingabedaten mithilfe von `clemext_iterator_nextRecord` von seinem Eingabe-Iterator abruft, kann dieser wiederum

clemext_peer_nextRecord von einer zweiten Peerinstanz abrufen, die dem ersten Peer vorgeordnet istund die Daten erzeugt, die der erste Peer schließlich verwendet.

Systemaufrufe wie strtok sind nicht ablaufinvariant und dürfen nicht verwendet werden. Einzelheiten zu Alternativen, die ablaufinvariant sind, können Sie der Betriebssystemdokumentation für Ihre verwendete Plattform entnehmen.

Garantieren der Threadsicherheit

IBM SPSS Modeler kann die Ausführung von mehreren Peerinstanzen von verschiedenen Ausführungs-Threads verschachteln. Zugriff auf alle Ressourcen, die zwischen Peerobjekten gemeinsam genutzt werden, müssen daher geschützt werden, z. B. durch Synchronsisierung mit Mutexes (Mutual Exclusion Objects - sich gegenseitig ausschließende Objekte) oder ähnliche Threading-Bibliotheksdienste.

CLEF-Module müssen Systemaufrufe vermeiden, die nicht threadsicher sind. Konsultieren Sie für weitere Informationen die Dokumentation zu Ihrem Betriebssystem oder UNIX-man-Seiten.

Vermeiden der Änderung der Thread- oder Prozessumgebung

Vermeiden Sie Systemaufrufe, mit denen die Umgebung des Aufruf-Threads oder -Prozesses geändert werden könnte.

Nachfolgend sind einige solche Aufrufe aufgeführt, die Liste ist jedoch bei Weitem nicht vollständig:

- setlocale, wenn zum Ändern und nicht zum einfachen Lesen einer Ländereinstellung verwendet
- SetCurrentDirectory (Windows) oder chdir (UNIX)
- LogonUser (Windows) oder seteuid (UNIX)
- putenv
- exit
- signal

Hinweis: Ein Aufruf unter Windows, der die Umgebung eines Threads ändert, aber erforderlich sein kann, ist CoInitialize. Weitere Informationen finden Sie im Thema „Sorgfältige Abwicklung des Aufrufs von CoInitialize oder CoUninitialize (Windows)" auf Seite 201.

Beschränken der Threadnutzung im Modul

Im Allgemeinen können Module intern beliebige Threads verwenden. Jedoch sollten Callback-Funktionen nur für den Thread aufgerufen werden, den IBM SPSS Modeler für den Aufruf der Funktionen des CLEF-Moduls verwendet hat (mit Ausnahme von clemext_peer_cancelExecution).

Die folgenden Callback-Funktionen können asynchron von einem beliebigen innerhalb des Moduls ausgeführten Thread aufgerufen werden:

- clemext_progress_report
- clemext_channel_send

Eine Peerinstanz muss sicherstellen, dass mehrere Threads keinen dieser Aufrufe simultan auslösen.

Korrekte Behandlung von Anforderungen zum Abbrechen der Ausführung

Wenn ein Endbenutzer den Abbruch der Ausführung einer Peerinstanz anfordert, führt IBM SPSS Modeler einen asynchronen Aufruf an die Funktion clemext_peer_cancelExecution des Moduls durch. Entwickler sollten versuchen, diesen Aufruf zu implementieren. Beachten Sie, dass er für diese Funktion als asynchroner Aufruf vorgesehen ist und aufgerufen wird, während ein anderer CLEF-API-Funktionsaufruf vom Modul ausgeführt wird.

Erneutes Starten unterbrochener Systemaufrufe (UNIX)

Unter UNIX verwendet die IBM SPSS Modeler-Anwendung Signale und Signalhandler. Einige UNIX-Systemaufrufe können den Code EINTR zurückgeben, falls der Prozess bei der Ausführung des Aufrufs ein Signal empfängt. Prüfen Sie die Man-Pages für den Systemaufruf auf Ihrer verwendeten UNIX-Plattform.

Wenn dieses Ereignis eintritt, sollte der aufrufende Code das Vorhandensein des Rückgabecodes EINTR prüfen und den Aufruf neu starten. Eine Möglichkeit dafür ist, eine einfache Wrapperfunktion zu erstellen (open_safe) und diesen Wrapper von Ihrer Anwendung aufrufen zu lassen:

```
int
open_safe(const char* path, int oflag, mode_t mode) {
  int res;
  while ((res = ::open(path, oflag, mode)) == -1
                                     && errno == EINTR) {
  }
  return res;
}
```

Sorgfältige Abwicklung des Aufrufs von CoInitialize oder CoUninitialize (Windows)

Unter Windows sollten Threads, die Windows-COM-Bibliotheksdienste (Component Object Model) verwenden müssen, die System-API-Funktion CoInitialize vor dem Verwenden von COM-Diensten und am Ende CoUninitialize aufrufen. Die Threads, aus denen IBM SPSS Modeler das CLEF-API für ein Modul aufruft, hat eventuell bereits CoInitialize aufgerufen.

Ein CLEF-Modul, das COM-Dienste aus diesen Threads nutzen möchte, sollte CoInitialize (gewöhnlich in einer clemext_create_peer- oder clemext_peer_beginExecution-Funktion) aufrufen. Falls dieser Aufruf erfolgreich ist, muss das Modul später ebenfalls CoUninitialize aufrufen, wenn der Thread die Ausführung beendet, gewöhnlich in clemext_destroy_peer bzw. clemext_peer_endExecution.

Weitere Informationen zum Aufruf CoInitiaize finden Sie in der Dokumentation auf der MSDN-Website (Microsoft Developer Network) unter http://msdn.microsoft.com.

Vermeiden von Annahmen über den Zeitpunkt, an dem das Modul entladen wird

Derzeit bleibt ein CLEF-Modul bis zum Sitzungsende geladen (d. h., Module können nicht nach Anforderung entladen und neu geladen werden). Die Funktion clemext_cleanup wird nie aufgerufen, nicht einmal beim Beenden des IBM SPSS Modeler-Serverprozesses, in den das Modul geladen wurde. Entwickler sollten also zu keinem Zeitpunkt davon ausgehen, dass ein Modul entladen wird und seine Ressourcen freigegeben werden.

Sorgfältige Vorgehensweise beim Start von Unterprozessen

Das Starten eines Unterprozesses durch CreateProcess (Windows) bzw. fork (UNIX) kann eine Reihe von Komplikationen bei der Interaktion zwischen übergeordneten und untergeordneten Prozessen verursachen sowie dabei, wie untergeordnete Prozesse Ressourcen erben, die im übergeordneten Prozess geöffnet sind.

Wenn ein CLEF-Modul "out-of-process"-Ausführung aufrufen muss, erwägen Sie die Verwendung einer geeigneten alternativen Architektur. Beispielsweise könnte das CLEF-Modul die Dienste nutzen, die von einem Anwendungsserver angeboten werden, um die entsprechende Aufgabe zu erledigen.

Insbesondere sollten Windows-Prozesse keine Unterprozesse mithilfe der Funktion CreateProcess mit dem Parameter bInheritHandles auf TRUE starten. Denn damit erbt der Unterprozess alle Dateideskriptoren, die im übergeordneten Prozess (IBM SPSS Modeler-Server) geöffnet sind.

Vermeiden des Schreibens in Standardausgabe oder Standardfehlerausgabe

Wenn ein CLEF-Modul an die Standardausgabe oder den Standardfehlerstream eines Prozesses schreibt (z. B. für Debugging-Zwecke), ist dies im Allgemeinen für den Endbenutzer nicht sichtbar. Wenn jedoch

ein Stream, der CLEF-Knoten enthält, mit IBM SPSS Modeler Solution Publisher bereitgestellt und von einer Befehlszeilenshell (Windows oder UNIX) ausgeführt wird, wird diese Ausgabe sichtbar und kann Benutzer verwirren.

CLEF-Module können stattdessen einen Verfolgungsdienst starten, indem Sie die Host-Callback-Funktion `clemext_host_trace` aufrufen und die anzuzeigende Nachricht als Textzeichenfolge angeben. Die Verfolgung muss ebenfalls in der IBM SPSS Modeler-Installation über die folgende Einstellung in der IBM SPSS Modeler Server-Konfigurationsoptionendatei (*/config/options.cfg* im IBM SPSS Modeler-Installationsverzeichnis) aktiviert werden:

```
trace_extension, 1
```

Verfolgte Nachrichten werden in die Datei */log/trace-<Prozess-ID>-<Prozess-ID>.log* im IBM SPSS Modeler-Installationsverzeichnis ausgegeben, wobei *Prozess-ID* die Kennung des IBM SPSS Modeler Server-Prozesses ist. Vermeiden Sie die gleichzeitige Verfolgung von mehreren Sitzungen, da sie alle dieselbe Protokolldatei verwenden.

Kapitel 10. Test und Verteilung

Testen von CLEF-Erweiterungen

Eine neue Erweiterung sollte unbedingt getestet werden, bevor sie an andere Benutzer verteilt wird.

Nach dem Anlegen einer Spezifikationsdatei und den zugehörigen Ressourcenbundles, JAR-Dateien, freigegebenen Bibliotheken und Benutzerhilfedateien können Sie die Erweiterung testen, indem Sie die Dateien in der erforderlichen Dateistruktur anordnen und in Ihre lokale IBM SPSS Modeler-Installation kopieren. Beim nächsten Start von IBM SPSS Modeler sollten Sie die neue Erweiterung an der IBM SPSS Modeler-Benutzerschnittstelle sehen.

Testen einer CLEF-Erweiterung

1. Schließen Sie IBM SPSS Modeler, falls die Software geöffnet ist.
2. Wenn die Erweiterung einen CLEF-Knoten oder eine Ausgabe definiert, wird empfohlen, dass Sie die Registerkarte "Debuggen" im Dialogfeld "Erweiterung" aktivieren, bis Sie sicher sind, dass die Erweiterung korrekt funktioniert. Weitere Informationen finden Sie im Thema „Verwenden der Registerkarte "Debuggen"" auf Seite 204.
3. Ordnen Sie die clientseitigen und serverseitigen Dateien in der erforderlichen Struktur an. Stellen Sie sicher, dass die Spezifikationsdatei und alle für den Knoten erforderlichen Ressourcen (z. B. `JAR`- oder `.dll`-Dateien) in die richtigen Verzeichnisse kopiert wurden. Weitere Informationen finden Sie im Thema „Dateistruktur" auf Seite 5.
4. Kopieren Sie das clientseitige Verzeichnis in den Ordner *\ext\lib* im Installationsverzeichnis von IBM SPSS Modeler.
5. Kopieren Sie das serverseitige Verzeichnis in den Ordner *\ext\bin* im Installationsverzeichnis von IBM SPSS Modeler.
6. Starten Sie IBM SPSS Modeler.
7. Wenn die Erweiterung ein Menü oder ein Menüelement definiert, überprüfen Sie, ob dieses korrekt im Menüsystem angezeigt wird. Wenn die Erweiterung einen neuen Knoten definiert, stellen Sie sicher, dass der Knoten an der gewünschten Position auf der korrekten Knotenpalette angezeigt wird, wie in der Spezifikationsdatei festgelegt.
8. Testen Sie die Erweiterung gründlich.

   Stellen Sie z. B. Folgendes sicher:

   - Dass die Knotenleistung nicht nachlässt, auch wenn weitere Felder und Datensätze hinzukommen
   - Dass Nullwerte konsistent behandelt werden
   - Dass gegebenenfalls verschiedene Ländereinstellungen unterstützt werden (z. B. Europa, Ostasien)
9. Nachdem Sie eine Erweiterung hinzugefügt haben, können Sie immer noch Änderungen an ihrer Spezifikationsdatei vornehmen. Jedoch werden diese Änderungen erst nach einem Neustart von IBM SPSS Modeler wirksam.

Fehlersuche in einer CLEF-Erweiterung

Die folgenden Funktionen von CLEF helfen Ihnen bei der Fehlersuche in einer Erweiterung:

- XML-Syntaxfehlernachrichten
- Externe Ausführung
- Registerkarte "Debuggen"

XML-Syntaxfehler

Bei einem XML-Syntaxfehler in der Spezifikationsdatei gibt der XML-Parser eine Fehlernachricht zurück.

Die Nachricht versucht, die Zeilennummer des Fehlers sowie die Fehlerursache möglichst genau zu bestimmen.

Führen Sie in diesem Fall die folgenden Schritte aus:

1. Korrigieren Sie den Fehler in der Spezifikationsdatei.
2. Testen Sie die Datei erneut, indem Sie das unter „Testen einer CLEF-Erweiterung” auf Seite 203 beschriebene Testverfahren nochmals ausführen.
3. Wiederholen Sie dieses Verfahren so oft, bis in der Spezifikationsdatei keine Syntaxfehler mehr gefunden werden.

Externe Ausführung

Eine benutzerdefinierte CLEF-Erweiterung wird normalerweise in einem eigenen Prozess, getrennt vom IBM SPSS Modeler-Prozess, ausgeführt. Diese Trennung kann bei der Fehlersuche hilfreich sein. Dadurch fällt bei einem Fehler im Erweiterungsprozess nicht der gesamte IBM SPSS Modeler Server-Prozess aus.

Hinweis: Diese Standardeinstellung kann überschrieben werden. Weitere Informationen finden Sie im Thema „Ändern der Ausführungsoptionen” auf Seite 205.

Verwenden der Registerkarte "Debuggen"

Sie können für jedes Dialogfeld (bzw. jeden Rahmen), das einem CLEF-Knoten oder einer -Ausgabe zugeordnet ist, die Registerkarte "Debuggen" aktivieren. Auf dieser Registerkarte können Sie die Eigenschafteneinstellungen des Objekts untersuchen. Auch die Inhalte der in der Erweiterung definierten Container können Sie anzeigen und für eine nähere Untersuchung in eine Datei speichern. Weitere Informationen finden Sie im Thema „Container” auf Seite 53.

Die Registerkarte "Debuggen" aktivieren Sie, indem Sie den Wert des Attributs `debug` des Elements `Extension` der Spezifikationsdatei auf `true` setzen. Weitere Informationen finden Sie im Thema „Extension-Element” auf Seite 33.

Die Registerkarte enthält folgende Felder:

Element-ID: Die eindeutige Kennung der Erweiterung (der Wert des Attributs `id` des Elements `ExtensionDetails` der Spezifikationsdatei).

Scriptname: Die eindeutige Kennung des Knotens in einem Script (der Wert des Attributs `scriptName` des Elements `Node`).

Erweiterungs-ID: Der Name des Erweiterungsordners, in dem sich die Datei- und Verzeichnisressourcen der Erweiterung befinden. Der Wert ergibt sich aus der Verkettung der Attribute `providerTag` und `id` (getrennt durch einen Punkt (`.`)) des Elements `ExtensionDetails`. Die für die interne Verwendung reservierte Zeichenfolge `spss` darf im `providerTag`-Teil der Kennung nicht enthalten sein.

Eigenschaften: Diese Tabelle enthält ausgewählte Details der `Property`-Deklarationen des Knotens:

- **Eigenschaft**: Die eindeutige Kennung der Eigenschaft (der Wert des Felds `name` des Elements `Property`).
- **Scriptname**: Die eindeutige Kennung der Eigenschaft in einem Script (der Wert des Attributs `scriptName` des Elements `Property`).
- **Wertetyp**: Der Wertetyp, den diese Eigenschaft annehmen kann (der Wert des Attributs `valueType` des Elements `Property`).
- **Liste?** Gibt an, ob die Eigenschaft eine Liste von Werten mit dem angegebenen Wertetyp ist (der Wert des Attributs `isList` des Elements `Property`).
- **Gemeinsam?** Wenn aktiviert, wird diese Eigenschaft an mehreren Stellen der Erweiterung verwendet (z. B. vom Modellerstellungs-, Modellausgabe- und Modellanwendungsknoten).
- **Wert**: Der Standardwert der Eigenschaft, sofern festgelegt.

Container: Zeigt den Inhalt des ausgewählten Containers an (z. B. Modelldaten). Wenn Sie auf dieses Feld klicken, wird eine Liste aller anderen, für die Erweiterung definierten Container angezeigt. Sie können dort den gewünschten Container auswählen, um dessen Inhalt anzuzeigen. Mit der nebenstehenden Schaltfläche **Container speichern** können Sie den Inhalt des ausgewählten Containers zur näheren Untersuchung in einer XML-Datei speichern.

Verfolgung: Öffnet bei der Ausführung des Knotens ein Dialogfeld mit der Trace-Ausgabe.

Ändern der Ausführungsoptionen

Ein benutzerdefiniertes CLEF-Erweiterungsmodul wird standardmäßig in einem eigenen Prozess, getrennt vom IBM SPSS Modeler-Hauptprozess, ausgeführt. Ein Fehler im Erweiterungsprozess führt auf diese Weise nicht zu einem Ausfall des IBM SPSS Modeler-Prozesses. Die von IBM bereitgestellten Module werden hingegen standardmäßig im Hauptprozess ausgeführt.

Diese Standardeinstellung kann der Systemadministrator mithilfe von zwei Konfigurationsoptionen für ein oder mehrere benannte Module in beide Richtungen ändern. Für beide Optionen werden die Modulkennungen der von der Änderung betroffenen Module in einer durch Kommas getrennten Liste angegeben.

Hinweis: Änderungen an diesen Optionen sollten nur auf ausdrückliche Anweisung eines Kundendienstmitarbeiters vorgenommen werden.

Es handelt sich hier um die folgenden Optionen:

Ausführungsoption "inprocess_execution" (Ausführung im Prozess)

Diese Option bewirkt, dass Erweiterungsmodule, die normalerweise in einem externen Prozess geladen werden (also in der Regel benutzerdefinierte Module), direkt in IBM SPSS Modeler geladen werden. Format:

```
clef_inprocess_execution, "Modul-ID1[,Modul-ID2[,...Modul-IDn]]"
```

Dabei ist *Modul-ID* der Wert des `id`-Attributs im `ExtensionDetails`-Element der relevanten Spezifikationsdatei. Beispiel:

```
clef_inprocess_execution, "test.example_filereader"
```

Ausführungsoption "external_execution" (externe Ausführung)

Diese Option bewirkt, dass Erweiterungsmodule, die normalerweise direkt in IBM SPSS Modeler geladen werden (also in der Regel die von IBM bereitgestellten Module), in einem externen Prozess geladen werden. Format:

```
clef_external_execution, "Modul-ID1[,Modul-ID2[,...Modul-IDn]]"
```

Modul-ID ist dabei ebenso wie bei der Option `clef_inprocess_execution` die Modulkennung. Hier ein fiktives Beispiel:

```
clef_external_execution, "spss.naivebayes,spss.terminator"
```

Hinzufügen oder Ändern einer Ausführungsoption

Zum Hinzufügen oder Ändern einer Ausführungsoption folgen Sie der Prozedur im Abschnitt "Verwenden der Datei options.cfg" im Verwaltungs- und Leistungshandbuch zu IBM SPSS Modeler Server17.1.

Verteilen von CLEF-Erweiterungen

Wenn die neue Erweiterung gründlich getestet und bereit zur Verteilung ist, führen Sie die folgenden Schritte aus:

1. Inaktivieren Sie die Registerkarte "Debuggen", falls diese aktiviert wurde. Weitere Informationen finden Sie im Thema „Verwenden der Registerkarte "Debuggen"" auf Seite 204.
2. Erstellen Sie eine Dateistruktur, die exakt der Struktur entspricht, in der die Erweiterungsdateien installiert werden sollen. Weitere Informationen finden Sie im Thema „Dateistruktur" auf Seite 5.
3. Komprimieren Sie die Dateistruktur in eine `ZIP`-Datei. Es ist einfacher, wenn Sie für die clientseitige und die serverseitige Installation jeweils separate `ZIP`-Dateien anlegen.
4. Verteilen Sie die `ZIP`-Dateien an die Endbenutzer.

Installieren von CLEF-Erweiterungen

So installieren Sie eine CLEF-Erweiterung:

1. Wenn Sie eine `.zip`-Datei mit der Dateistruktur einer Erweiterung erhalten, extrahieren Sie die clientseitigen Dateien in den Ordner `\ext\lib` im Installationsverzeichnis von IBM SPSS Modeler.
2. Extrahieren Sie die serverseitigen Dateien in den Ordner `\ext\bin` im Installationsverzeichnis von IBM SPSS Modeler (bzw. im Installationsverzeichnis von IBM SPSS Modeler Server, wenn Sie IBM SPSS Modeler Server verwenden).
3. Starten Sie IBM SPSS Modeler und überprüfen Sie, ob die neuen Knoten an den erwarteten Stellen der Knotenpalette angezeigt werden.
4. Wenn Sie eine Bereitstellungsumgebung wie IBM SPSS Collaboration and Deployment Services oder IBM SPSS Modeler Solution Publisher verwenden, wiederholen Sie die Schritte 1 und 2, um die Dateien den Ordnern `\ext\lib` und `\ext\bin` in diesen Produkten hinzuzufügen und SPSS Modeler erneut zu starten.

Deinstallieren von CLEF-Erweiterungen

So deinstallieren Sie eine CLEF-Erweiterung:

1. Suchen Sie den Erweiterungsordner im Ordner `\ext\lib` im Installationsverzeichnis von IBM SPSS Modeler.

   Falls durch die Erweiterung auch ein serverseitiger Erweiterungsordner installiert wurde, suchen Sie diesen Ordner im Ordner `\ext\bin` im Installationsverzeichnis von IBM SPSS Modeler bzw. IBM SPSS Modeler Server.
2. Löschen Sie den bzw. die Erweiterungsordner.
3. Wenn Sie eine Bereitstellungsumgebung wie IBM SPSS Collaboration and Deployment Services oder IBM SPSS Modeler Solution Publisher verwenden, wiederholen Sie die Schritte 1 und 2, um die Dateien aus den Ordnern `\ext\lib` und `\ext\bin` in diesen Produkten zu entfernen.

Die Änderung wird wirksam, sobald Sie IBM SPSS Modeler das nächste Mal starten.

Anhang. CLEF-XML-Schema

CLEF-Elementreferenz

Dieser Abschnitt enthält eine Referenz für alle Elemente von CLEF.

Jedes Thema listet die gültigen Attribute eines Elements sowie seine über- und untergeordneten Elemente auf. Das Diagramm zeigt alle untergeordneten Elemente des Elements an. Die Pfeile im Diagramm verweisen auf Elemente, die mit anderen Elementen gemeinsam verwendet werden können. Diese Elemente werden im Inhaltsverzeichnis direkt unter diesem Thema ("CLEF-Elementreferenz"), nicht unter dem übergeordneten Element aufgeführt.

Elemente

Action-Element

Tabelle 52. Attribute für Action

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
id	**erforderlich**		*Zeichenfolge*
imagePath	optional		*Zeichenfolge*
imagePathKey	optional		*Zeichenfolge*
label	**erforderlich**		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
shortcut	optional		*Zeichenfolge*
shortcutKey	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Action">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="imagePath" type="xs:string" use="optional"/>
  <xs:attribute name="imagePathKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="shortcut" type="xs:string" use="optional"/>
  <xs:attribute name="shortcutKey" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Actions

ActionButton-Element

Definiert eine Schaltfläche, mit der eine Aktion aufgerufen werden kann. Die Aktion wird normalerweise von einem Benutzerschnittstellendelegate oder einem Aktionslistener implementiert.

Tabelle 53. Attribute für ActionButton

Attribut	Verwendung	Beschreibung	Gültige Werte
action	**erforderlich**		*Zeichenfolge*
showIcon	optional		*boolean*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="ActionButton">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="action" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="showIcon" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

ComboBoxControl, ExtensionObjectPanel, FieldAllocationList, ModelViewerPanel, SelectorPanel, StaticText, SystemControls, TabbedPanel, TextBrowserPanel

Actions-Element

XML-Darstellung

```
<xs:element name="Actions">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="Action"/>
    </xs:choice>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

CommonObjects

Untergeordnete Elemente

Action

AddField-Element

Tabelle 54. Attribute für AddField

Attribut	Verwendung	Beschreibung	Gültige Werte
depth	optional		*Ganzzahl*

Tabelle 54. Attribute für AddField (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
direction	optional		**in** **out** **both** **none** **partition**
directionRef	optional		
fieldRef	optional		
group	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
missingValuesRef	optional		
name	**erforderlich**		
prefix	optional		*Zeichenfolge*
role	optional		**unknown** **predictedValue** **predictedDisplayValue** **probability** **residual** **standardError** **entityId** **entityAffinity** **upperConfidenceLimit** **lowerConfidenceLimit** **propensity** **value** **supplementary**
storage	optional		**unknown** **integer** **real** **string** **date** **time** **timestamp** **list**
storageRef	optional		
tag	optional		*Zeichenfolge*
targetField	optional		*Zeichenfolge*
type	optional		**auto** **range** **discrete** **set** **orderedSet** **flag** **typeless** **collection** **geospatial**
typeRef	optional		
value	optional		*Zeichenfolge*

Tabelle 54. Attribute für AddField (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
valueStorage	optional		**unknown** **integer** **real** **string** **date** **time** **timestamp** **list**

XML-Darstellung

```
<xs:element name="AddField">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Range" minOccurs="0"/>
      <xs:element ref="Values" minOccurs="0"/>
      <xs:element ref="NumericInfo" minOccurs="0"/>
      <xs:element name="MissingValues" minOccurs="0">
        <xs:sequence>
          <xs:element ref="Values" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="Range" minOccurs="0"/>
        </xs:sequence>
      </xs:element>
      <xs:element name="ModelField" type="MODEL-FIELD-INFORMATION" minOccurs="0">
      </xs:element>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-NAME" use="required"/>
  <xs:attribute name="storage" type="FIELD-STORAGE">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="real"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="time"/>
    <xs:enumeration value="timestamp"/>
    <xs:enumeration value="list"/>
  </xs:attribute>
  <xs:attribute name="type" type="FIELD-TYPE">
    <xs:enumeration value="auto"/>
    <xs:enumeration value="range"/>
    <xs:enumeration value="discrete"/>
    <xs:enumeration value="set"/>
    <xs:enumeration value="orderedSet"/>
    <xs:enumeration value="flag"/>
    <xs:enumeration value="typeless"/>
    <xs:enumeration value="collection"/>
    <xs:enumeration value="geospatial"/>
  </xs:attribute>
  <xs:attribute name="direction" type="FIELD-DIRECTION">
    <xs:enumeration value="in"/>
    <xs:enumeration value="out"/>
    <xs:enumeration value="both"/>
    <xs:enumeration value="none"/>
    <xs:enumeration value="partition"/>
  </xs:attribute>
  <xs:attribute name="label" type="xs:string"/>
  <xs:attribute name="depth" type="xs:integer" use="optional" default="-1"/>
  <xs:attribute name="valueStorage" type="FIELD-STORAGE" use="optional">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="real"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="time"/>
    <xs:enumeration value="timestamp"/>
    <xs:enumeration value="list"/>
  </xs:attribute>
  <xs:attribute name="fieldRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="storageRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="typeRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="directionRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="missingValuesRef" type="EVALUATED-STRING" use="optional"/>
```

```
  <xs:attribute name="role" type="MODEL-FIELD-ROLE" use="optional">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="predictedValue"/>
    <xs:enumeration value="predictedDisplayValue"/>
    <xs:enumeration value="probability"/>
    <xs:enumeration value="residual"/>
    <xs:enumeration value="standardError"/>
    <xs:enumeration value="entityId"/>
    <xs:enumeration value="entityAffinity"/>
    <xs:enumeration value="upperConfidenceLimit"/>
    <xs:enumeration value="lowerConfidenceLimit"/>
    <xs:enumeration value="propensity"/>
    <xs:enumeration value="value"/>
    <xs:enumeration value="supplementary"/>
  </xs:attribute>
  <xs:attribute name="targetField" type="xs:string" use="optional"/>
  <xs:attribute name="value" type="xs:string" use="optional"/>
  <xs:attribute name="group" type="xs:string" use="optional"/>
  <xs:attribute name="tag" type="xs:string" use="optional"/>
  <xs:attribute name="prefix" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

ForEach, ModelFields

Untergeordnete Elemente

MissingValues, ModelField, NumericInfo, Range, Range, Values, Values

Zugehörige Elemente

ChangeField

MissingValues-Element:

Tabelle 55. Attribute für MissingValues

Attribut	Verwendung	Beschreibung	Gültige Werte
treatNullAsMissing	optional		*boolean*
treatWhitespaceAsMissing	optional		*boolean*

XML-Darstellung

```
<xs:element name="MissingValues" minOccurs="0">
  <xs:sequence>
    <xs:element ref="Values" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Range" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="treatWhitespaceAsMissing" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="treatNullAsMissing" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Übergeordnete Elemente

AddField

Untergeordnete Elemente

Range, Range, Values, Values

ModelField-Element:

Tabelle 56. Attribute für ModelField

Attribut	Verwendung	Beschreibung	Gültige Werte
group	optional		
role	**erforderlich**		**unknown** **predictedValue** **predictedDisplayValue** **probability** **residual** **standardError** **entityId** **entityAffinity** **upperConfidenceLimit** **lowerConfidenceLimit** **propensity** **value** **supplementary**
tag	optional		*Zeichenfolge*
targetField	optional		*Zeichenfolge*
value	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="ModelField" type="MODEL-FIELD-INFORMATION" minOccurs="0">
  <xs:attribute name="role" type="MODEL-FIELD-ROLE" use="required">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="predictedValue"/>
    <xs:enumeration value="predictedDisplayValue"/>
    <xs:enumeration value="probability"/>
    <xs:enumeration value="residual"/>
    <xs:enumeration value="standardError"/>
    <xs:enumeration value="entityId"/>
    <xs:enumeration value="entityAffinity"/>
    <xs:enumeration value="upperConfidenceLimit"/>
    <xs:enumeration value="lowerConfidenceLimit"/>
    <xs:enumeration value="propensity"/>
    <xs:enumeration value="value"/>
    <xs:enumeration value="supplementary"/>
  </xs:attribute>
  <xs:attribute name="targetField" type="xs:string"/>
  <xs:attribute name="value" type="xs:string"/>
  <xs:attribute name="group" type="MODEL-FIELD-GROUP"/>
  <xs:attribute name="tag" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

AddField

And-Element

XML-Darstellung

```
<xs:element name="And">
  <xs:sequence minOccurs="2" maxOccurs="unbounded">
    <xs:group ref="CONDITION-EXPRESSION">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

And, Command, Constraint, CreateDocument, CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModel, CreateModelApplier, CreateModelOutput, Enabled, Not, Option, Or, Run, Visible

Untergeordnete Elemente

And, Condition, Not, Or

Attribute-Element

Tabelle 57. Attribute für Attribute

Attribut	Verwendung	Beschreibung	Gültige Werte
defaultValue	optional		
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
isList	optional		*boolean*
label	**erforderlich**		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
name	**erforderlich**		*Zeichenfolge*
valueType	optional		**string** **encryptedString** **integer** **double** **boolean** **date** **enum**

XML-Darstellung

```
<xs:element name="Attribute">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="valueType" type="ATTRIBUTE-VALUE-TYPE">
    <xs:enumeration value="string"/>
    <xs:enumeration value="encryptedString"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="double"/>
    <xs:enumeration value="boolean"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="enum"/>
  </xs:attribute>
  <xs:attribute name="defaultValue" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="isList" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Übergeordnete Elemente

Catalog, Structure

BinaryFormat-Element

XML-Darstellung

```
<xs:element name="BinaryFormat"/>
```

Übergeordnete Elemente

FileFormatType

Catalog-Element

Tabelle 58. Attribute für Catalog

Attribut	Verwendung	Beschreibung	Gültige Werte
id	**erforderlich**		*Zeichenfolge*
valueColumn	**erforderlich**		*Ganzzahl*

XML-Darstellung

```
<xs:element name="Catalog">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element ref="Attribute"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="valueColumn" type="xs:integer" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Catalogs

Untergeordnete Elemente

Attribute

Catalogs-Element

XML-Darstellung

```
<xs:element name="Catalogs">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="Catalog"/>
    </xs:choice>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

CommonObjects

Untergeordnete Elemente

Catalog

ChangeField-Element

Tabelle 59. Attribute für ChangeField

Attribut	Verwendung	Beschreibung	Gültige Werte
depth	optional		*Ganzzahl*
direction	optional		**in** **out** **both** **none** **partition**

Tabelle 59. Attribute für ChangeField (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
directionRef	optional		
fieldRef	**erforderlich**		
label	optional		*Zeichenfolge*
missingValuesRef	optional		
name	**erforderlich**		
storage	optional		**unknown** **integer** **real** **string** **date** **time** **timestamp** **list**
storageRef	optional		
type	optional		**auto** **range** **discrete** **set** **orderedSet** **flag** **typeless** **collection** **geospatial**
typeRef	optional		
valueStorage	optional		**unknown** **integer** **real** **string** **date** **time** **timestamp** **list**

XML-Darstellung

```
<xs:element name="ChangeField">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Range" minOccurs="0"/>
      <xs:element ref="Values" minOccurs="0"/>
      <xs:element ref="NumericInfo" minOccurs="0"/>
      <xs:element name="MissingValues" minOccurs="0">
        <xs:sequence>
          <xs:element ref="Values" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="Range" minOccurs="0"/>
        </xs:sequence>
      </xs:element>
      <xs:element name="ModelField" type="MODEL-FIELD-INFORMATION" minOccurs="0">
      </xs:element>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-NAME" use="required"/>
  <xs:attribute name="storage" type="FIELD-STORAGE">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="real"/>
```

```
    <xs:enumeration value="string"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="time"/>
    <xs:enumeration value="timestamp"/>
    <xs:enumeration value="list"/>
  </xs:attribute>
  <xs:attribute name="type" type="FIELD-TYPE">
    <xs:enumeration value="auto"/>
    <xs:enumeration value="range"/>
    <xs:enumeration value="discrete"/>
    <xs:enumeration value="set"/>
    <xs:enumeration value="orderedSet"/>
    <xs:enumeration value="flag"/>
    <xs:enumeration value="typeless"/>
    <xs:enumeration value="collection"/>
    <xs:enumeration value="geospatial"/>
  </xs:attribute>
  <xs:attribute name="direction" type="FIELD-DIRECTION">
    <xs:enumeration value="in"/>
    <xs:enumeration value="out"/>
    <xs:enumeration value="both"/>
    <xs:enumeration value="none"/>
    <xs:enumeration value="partition"/>
  </xs:attribute>
  <xs:attribute name="label" type="xs:string"/>
  <xs:attribute name="depth" type="xs:integer" use="optional" default="-1"/>
  <xs:attribute name="valueStorage" type="FIELD-STORAGE" use="optional">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="real"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="time"/>
    <xs:enumeration value="timestamp"/>
    <xs:enumeration value="list"/>
  </xs:attribute>
  <xs:attribute name="fieldRef" type="EVALUATED-STRING" use="required"/>
  <xs:attribute name="storageRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="typeRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="directionRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="missingValuesRef" type="EVALUATED-STRING" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

ForEach, ModelFields

Untergeordnete Elemente

MissingValues, ModelField, NumericInfo, Range, Range, Values, Values

Zugehörige Elemente

AddField

MissingValues-Element:

Tabelle 60. Attribute für MissingValues

Attribut	Verwendung	Beschreibung	Gültige Werte
treatNullAsMissing	optional		*boolean*
treatWhitespaceAsMissing	optional		*boolean*

XML-Darstellung

```
<xs:element name="MissingValues" minOccurs="0">
  <xs:sequence>
    <xs:element ref="Values" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Range" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="treatWhitespaceAsMissing" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="treatNullAsMissing" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Übergeordnete Elemente

AddField

Untergeordnete Elemente

Range, Range, Values, Values

ModelField-Element:

Tabelle 61. Attribute für ModelField

Attribut	Verwendung	Beschreibung	Gültige Werte
group	optional		
role	**erforderlich**		**unknown** **predictedValue** **predictedDisplayValue** **probability** **residual** **standardError** **entityId** **entityAffinity** **upperConfidenceLimit** **lowerConfidenceLimit** **propensity** **value** **supplementary**
tag	optional		*Zeichenfolge*
targetField	optional		*Zeichenfolge*
value	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="ModelField" type="MODEL-FIELD-INFORMATION" minOccurs="0">
  <xs:attribute name="role" type="MODEL-FIELD-ROLE" use="required">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="predictedValue"/>
    <xs:enumeration value="predictedDisplayValue"/>
    <xs:enumeration value="probability"/>
    <xs:enumeration value="residual"/>
    <xs:enumeration value="standardError"/>
    <xs:enumeration value="entityId"/>
    <xs:enumeration value="entityAffinity"/>
    <xs:enumeration value="upperConfidenceLimit"/>
    <xs:enumeration value="lowerConfidenceLimit"/>
    <xs:enumeration value="propensity"/>
    <xs:enumeration value="value"/>
    <xs:enumeration value="supplementary"/>
  </xs:attribute>
  <xs:attribute name="targetField" type="xs:string"/>
  <xs:attribute name="value" type="xs:string"/>
  <xs:attribute name="group" type="MODEL-FIELD-GROUP"/>
  <xs:attribute name="tag" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

AddField

CheckBoxControl-Element

Definiert ein Steuerelement für Kontrollkästchen, mit dem ein boolescher Wert geändert werden kann.

Tabelle 62. Attribute für CheckBoxControl

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
invert	optional		*boolean*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="CheckBoxControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="invert" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

CheckBoxGroupControl-Element

Definiert eine Gruppe von Steuerelementen für Kontrollkästchen, mit denen eine Auswahl von Werten aus einem Aufzählungslistentyp angegeben werden kann.

Tabelle 63. Attribute für CheckBoxGroupControl

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
layoutByRow	optional		*boolean*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
rows	optional		*positiveGanzzahl*
showLabel	optional		*boolean*
useSubPanel	optional		*boolean*

XML-Darstellung

```
<xs:element name="CheckBoxGroupControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="rows" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="layoutByRow" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="useSubPanel" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

ClientDirectoryChooserControl-Element

Definiert ein Steuerelement, mit dem ein Verzeichnis auf dem Client ausgewählt werden kann.

Tabelle 64. Attribute für ClientDirectoryChooserControl

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
mode	**erforderlich**		**open** **save** **import** **export**
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="ClientDirectoryChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="mode" type="FILE-CHOOSER-MODE" use="required">
    <xs:enumeration value="open"/>
    <xs:enumeration value="save"/>
    <xs:enumeration value="import"/>
    <xs:enumeration value="export"/>
  </xs:attribute>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl,

PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

ClientFileChooserControl-Element

Definiert ein Steuerelement, mit dem eine Datei auf dem Client ausgewählt werden kann.

Tabelle 65. Attribute für ClientFileChooserControl

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
mode	**erforderlich**		**open** **save** **import** **export**
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="ClientFileChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="mode" type="FILE-CHOOSER-MODE" use="required">
    <xs:enumeration value="open"/>
    <xs:enumeration value="save"/>
    <xs:enumeration value="import"/>
    <xs:enumeration value="export"/>
  </xs:attribute>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

ComboBoxControl-Element

Tabelle 66. Attribute für ComboBoxControl

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="ComboBoxControl" type="CONTROLLER">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
</xs:element>
```

Tabelle 67. Erweiterte Typen

Typ	Beschreibung
ItemChooserControl	

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

ActionButton, ExtensionObjectPanel, FieldAllocationList, ModelViewerPanel, SelectorPanel, StaticText, SystemControls, TabbedPanel, TextBrowserPanel

Command-Element

Tabelle 68. Attribute für Command

Attribut	Verwendung	Beschreibung	Gültige Werte
path	**erforderlich**		

XML-Darstellung

```
<xs:element name="Command">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
  <xs:attribute name="path" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Run

Untergeordnete Elemente

And, Condition, Not, Or

CommonObjects-Element

Stellt eine Position für Definitionen bereit, die für die Erweiterung global sind.

Tabelle 69. Attribute für CommonObjects

Attribut	Verwendung	Beschreibung	Gültige Werte
extensionDelegate	optional		*Zeichenfolge*
extensionListenerClass	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="CommonObjects">
  <xs:all>
    <xs:element ref="PropertyTypes" minOccurs="0"/>
    <xs:element ref="PropertySets" minOccurs="0"/>
    <xs:element ref="FileFormatTypes" minOccurs="0"/>
    <xs:element ref="ContainerTypes" minOccurs="0"/>
    <xs:element ref="Actions" minOccurs="0"/>
    <xs:element ref="Catalogs" minOccurs="0"/>
  </xs:all>
  <xs:attribute name="extensionDelegate" type="xs:string" use="optional"/>
  <xs:attribute name="extensionListenerClass" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Extension

Untergeordnete Elemente

Actions, Catalogs, ContainerTypes, FileFormatTypes, PropertySets, PropertyTypes

Condition-Element

Tabelle 70. Attribute für Condition

Attribut	Verwendung	Beschreibung	Gültige Werte
container	optional		*Zeichenfolge*
control	optional		*Zeichenfolge*
expression	optional		
listMode	optional		**all** **any**

Tabelle 70. Attribute für Condition (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
op	**erforderlich**		**equals** **notEquals** **isEmpty** **isNotEmpty** **lessThan** **lessOrEquals** **greaterThan** **greaterOrEquals** **equalsIgnoreCase** **isSubstring** **startsWith** **endsWith** **startsWithIgnoreCase** **endsWithIgnoreCase** **isSubstring** **hasSubstring** **isSubstringIgnoreCase** **hasSubstringIgnoreCase** **in** **countEquals** **countLessThan** **countLessOrEquals** **countGreaterThan** **countGreaterOrEquals** **contains** **storageEquals** **typeEquals** **directionEquals** **isMeasureDiscrete** **isMeasureContinuous** **isMeasureCollection** **isMeasureGeospatial** **isMeasureTypeless** **isMeasureUnknown** **isStorageString** **isStorageNumeric** **isStorageDatetime** **isStorageList** **isStorageUnknown** **isModelOutput** **modelOutputRoleEquals** **modelOutputTarget-FieldEquals** **modelOutputHasValue** **modelOutputTagEquals** **enumRestriction**
property	optional		*Zeichenfolge*
value	optional		

XML-Darstellung

```
<xs:element name="Condition">
  <xs:attribute name="expression" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="control" type="xs:string" use="optional"/>
  <xs:attribute name="property" type="xs:string" use="optional"/>
  <xs:attribute name="container" type="xs:string" use="optional"/>
  <xs:attribute name="op" type="CONDITION-TEST" use="required">
    <xs:enumeration value="equals"/>
    <xs:enumeration value="notEquals"/>
```

```
    <xs:enumeration value="isEmpty"/>
    <xs:enumeration value="isNotEmpty"/>
    <xs:enumeration value="lessThan"/>
    <xs:enumeration value="lessOrEquals"/>
    <xs:enumeration value="greaterThan"/>
    <xs:enumeration value="greaterOrEquals"/>
    <xs:enumeration value="equalsIgnoreCase"/>
    <xs:enumeration value="isSubstring"/>
    <xs:enumeration value="startsWith"/>
    <xs:enumeration value="endsWith"/>
    <xs:enumeration value="startsWithIgnoreCase"/>
    <xs:enumeration value="endsWithIgnoreCase"/>
    <xs:enumeration value="isSubstring"/>
    <xs:enumeration value="hasSubstring"/>
    <xs:enumeration value="isSubstringIgnoreCase"/>
    <xs:enumeration value="hasSubstringIgnoreCase"/>
    <xs:enumeration value="in"/>
    <xs:enumeration value="countEquals"/>
    <xs:enumeration value="countLessThan"/>
    <xs:enumeration value="countLessOrEquals"/>
    <xs:enumeration value="countGreaterThan"/>
    <xs:enumeration value="countGreaterOrEquals"/>
    <xs:enumeration value="contains"/>
    <xs:enumeration value="storageEquals"/>
    <xs:enumeration value="typeEquals"/>
    <xs:enumeration value="directionEquals"/>
    <xs:enumeration value="isMeasureDiscrete"/>
    <xs:enumeration value="isMeasureContinuous"/>
    <xs:enumeration value="isMeasureCollection"/>
    <xs:enumeration value="isMeasureGeospatial"/>
    <xs:enumeration value="isMeasureTypeless"/>
    <xs:enumeration value="isMeasureUnknown"/>
    <xs:enumeration value="isStorageString"/>
    <xs:enumeration value="isStorageNumeric"/>
    <xs:enumeration value="isStorageDatetime"/>
    <xs:enumeration value="isStorageList"/>
    <xs:enumeration value="isStorageUnknown"/>
    <xs:enumeration value="isModelOutput"/>
    <xs:enumeration value="modelOutputRoleEquals"/>
    <xs:enumeration value="modelOutputTargetFieldEquals"/>
    <xs:enumeration value="modelOutputHasValue"/>
    <xs:enumeration value="modelOutputTagEquals"/>
    <xs:enumeration value="enumRestriction"/>
  </xs:attribute>
  <xs:attribute name="value" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="listMode" use="optional" default="all">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="all"/>
        <xs:enumeration value="any"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>
```

Übergeordnete Elemente

And, Command, Constraint, CreateDocument, CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModel, CreateModelApplier, CreateModelOutput, Enabled, ExpertSettings, Not, Option, Or, Run, Visible

Constraint-Element

Tabelle 71. Attribute für Constraint

Attribut	Verwendung	Beschreibung	Gültige Werte
property	**erforderlich**		*Zeichenfolge*
singleSelection	optional		*boolean*

XML-Darstellung

```
<xs:element name="Constraint">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION">
      <xs:choice>
        <xs:element ref="Condition"/>
```

```
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="singleSelection" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Übergeordnete Elemente

AutoModeling

Untergeordnete Elemente

And, Condition, Not, Or

Constructors-Element

XML-Darstellung

```
<xs:element name="Constructors">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="CreateModelOutput"/>
      <xs:element ref="CreateDocumentOutput"/>
      <xs:element ref="CreateInteractiveModelBuilder"/>
      <xs:element ref="CreateInteractiveDocumentBuilder"/>
      <xs:element ref="CreateModelApplier"/>
    </xs:choice>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

DocumentOutput, Execution, InteractiveDocumentBuilder, InteractiveModelBuilder, ModelOutput, Node

Untergeordnete Elemente

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

Container-Element

Tabelle 72. Attribute für Container

Attribut	Verwendung	Beschreibung	Gültige Werte
name	**erforderlich**		*Zeichenfolge*
type	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Container">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="type" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Containers, Containers, Containers, Containers, Containers

ContainerFile-Element

Tabelle 73. Attribute für ContainerFile

Attribut	Verwendung	Beschreibung	Gültige Werte
container	optional		
containerType	optional		
path	**erforderlich**		

XML-Darstellung

```
<xs:element name="ContainerFile" type="SERVER-CONTAINER-FILE">
  <xs:attribute name="path" type="EVALUATED-STRING" use="required"/>
  <xs:attribute name="container" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="containerType" type="EVALUATED-STRING" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

InputFiles, OutputFiles

ContainerTypes-Element

XML-Darstellung

```
<xs:element name="ContainerTypes">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="DocumentType"/>
      <xs:element ref="ModelType"/>
    </xs:choice>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

CommonObjects

Untergeordnete Elemente

DocumentType, ModelType

Controls-Element

Definiert die Steuerelemente, die einem Benutzerschnittstellenobjekt hinzugefügt werden können, wie Menüs und Symbolleistenelemente.

XML-Darstellung

```
<xs:element name="Controls">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="Menu"/>
      <xs:element ref="MenuItem"/>
      <xs:element ref="ToolbarItem"/>
    </xs:choice>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

UserInterface

Untergeordnete Elemente

Menu, MenuItem, ToolbarItem

CreateDocument-Element

Tabelle 74. Attribute für CreateDocument

Attribut	Verwendung	Beschreibung	Gültige Werte
sourceFile	**erforderlich**		*Zeichenfolge*
target	**erforderlich**		*Zeichenfolge*
type	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="CreateDocument">
  <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
    <xs:choice>
      <xs:element ref="Condition"/>
      <xs:element ref="And"/>
      <xs:element ref="Or"/>
      <xs:element ref="Not"/>
    </xs:choice>
  </xs:group>
  <xs:attribute name="sourceFile" type="xs:string" use="required"/>
  <xs:attribute name="target" type="xs:string" use="required"/>
  <xs:attribute name="type" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

Untergeordnete Elemente

And, Condition, Not, Or

Zugehörige Elemente

CreateModel

CreateDocumentOutput-Element

Tabelle 75. Attribute für CreateDocumentOutput

Attribut	Verwendung	Beschreibung	Gültige Werte
type	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="CreateDocumentOutput">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="SetProperty"/>
        <xs:element ref="SetContainer"/>
        <xs:element ref="CreateModel"/>
        <xs:element ref="CreateDocument"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Constructors

Untergeordnete Elemente

And, Condition, CreateDocument, CreateModel, Not, Or, SetContainer, SetProperty

Zugehörige Elemente

CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

CreateInteractiveDocumentBuilder-Element

Tabelle 76. Attribute für CreateInteractiveDocumentBuilder

Attribut	Verwendung	Beschreibung	Gültige Werte
type	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="CreateInteractiveDocumentBuilder">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="SetProperty"/>
        <xs:element ref="SetContainer"/>
        <xs:element ref="CreateModel"/>
        <xs:element ref="CreateDocument"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Constructors

Untergeordnete Elemente

And, Condition, CreateDocument, CreateModel, Not, Or, SetContainer, SetProperty

Zugehörige Elemente

CreateDocumentOutput, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

CreateInteractiveModelBuilder-Element

Tabelle 77. Attribute für CreateInteractiveModelBuilder

Attribut	Verwendung	Beschreibung	Gültige Werte
type	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="CreateInteractiveModelBuilder">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="SetProperty"/>
        <xs:element ref="SetContainer"/>
        <xs:element ref="CreateModel"/>
        <xs:element ref="CreateDocument"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Constructors

Untergeordnete Elemente

And, Condition, CreateDocument, CreateModel, Not, Or, SetContainer, SetProperty

Zugehörige Elemente

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateModelApplier, CreateModelOutput

CreateModel-Element

Tabelle 78. Attribute für CreateModel

Attribut	Verwendung	Beschreibung	Gültige Werte
signatureFile	optional		*Zeichenfolge*
sourceFile	**erforderlich**		*Zeichenfolge*
target	**erforderlich**		*Zeichenfolge*
type	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="CreateModel">
  <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
    <xs:choice>
      <xs:element ref="Condition"/>
      <xs:element ref="And"/>
      <xs:element ref="Or"/>
      <xs:element ref="Not"/>
    </xs:choice>
  </xs:group>
  <xs:attribute name="sourceFile" type="xs:string" use="required"/>
  <xs:attribute name="target" type="xs:string" use="required"/>
  <xs:attribute name="type" type="xs:string" use="optional"/>
  <xs:sequence>
    <xs:element name="ModelDetail" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="signatureFile" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

Untergeordnete Elemente

And, Condition, ModelDetail, Not, Or

Zugehörige Elemente

CreateDocument

ModelDetail-Element:

Tabelle 79. Attribute für ModelDetail

Attribut	Verwendung	Beschreibung	Gültige Werte
algorithm	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="ModelDetail" maxOccurs="unbounded">
  <xs:attribute name="algorithm" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

CreateModel

CreateModelApplier-Element

Tabelle 80. Attribute für CreateModelApplier

Attribut	Verwendung	Beschreibung	Gültige Werte
type	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="CreateModelApplier">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="SetProperty"/>
        <xs:element ref="SetContainer"/>
        <xs:element ref="CreateModel"/>
        <xs:element ref="CreateDocument"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Constructors

Untergeordnete Elemente

And, Condition, CreateDocument, CreateModel, Not, Or, SetContainer, SetProperty

Zugehörige Elemente

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelOutput

CreateModelOutput-Element

Tabelle 81. Attribute für CreateModelOutput

Attribut	Verwendung	Beschreibung	Gültige Werte
type	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="CreateModelOutput">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="SetProperty"/>
        <xs:element ref="SetContainer"/>
        <xs:element ref="CreateModel"/>
        <xs:element ref="CreateDocument"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Constructors

Untergeordnete Elemente

And, Condition, CreateDocument, CreateModel, Not, Or, SetContainer, SetProperty

Zugehörige Elemente

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier

DBConnectionChooserControl-Element

Definiert ein Steuerelement, mit dem eine Datenbankverbindung ausgewählt werden kann.

Tabelle 82. Attribute für DBConnectionChooserControl

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*

Tabelle 82. Attribute für DBConnectionChooserControl (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="DBConnectionChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

DBTableChooserControl-Element

Definiert ein Steuerelement, mit dem eine Datenbanktabelle ausgewählt werden kann.

Tabelle 83. Attribute für DBTableChooserControl

Attribut	Verwendung	Beschreibung	Gültige Werte
connectionProperty	**erforderlich**		*Zeichenfolge*
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*

Tabelle 83. Attribute für DBTableChooserControl (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="DBTableChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="connectionProperty" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DB-ConnectionChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

DataFile-Element

Tabelle 84. Attribute für DataFile

Attribut	Verwendung	Beschreibung	Gültige Werte
path	**erforderlich**		

XML-Darstellung

```
<xs:element name="DataFile" type="SERVER-DATA-FILE">
  <xs:attribute name="path" type="EVALUATED-STRING" use="required"/>
  <xs:choice>
    <xs:element ref="DelimitedDataFormat"/>
  </xs:choice>
</xs:element>
```

Übergeordnete Elemente

InputFiles, OutputFiles

Untergeordnete Elemente

DelimitedDataFormat

DataFormat-Element

XML-Darstellung

```
<xs:element name="DataFormat">
  <xs:group ref="DATA-FORMAT-TYPE">
    <xs:choice>
      <xs:element ref="DelimitedDataFormat"/>
      <xs:element ref="SPSSDataFormat"/>
    </xs:choice>
  </xs:group>
</xs:element>
```

Übergeordnete Elemente

FileFormatType

Untergeordnete Elemente

DelimitedDataFormat, SPSSDataFormat

DataModel-Element

Das in einen Knoten eingehende bzw. von einem Knoten abgehende Datenmodell. Ein Eingabe-/Ausgabedatenmodell ist ein Set von Feldern.

XML-Darstellung

```
<xs:element name="DataModel" type="DATA-MODEL">
  <xs:sequence>
    <xs:element name="FieldFormats" type="FIELD-FORMATS" minOccurs="0">
      <xs:sequence>
        <xs:element name="NumberFormat" type="NUMBER-FORMAT-DECLARATION" minOccurs="0" maxOccurs="unbounded">
        </xs:element>
      </xs:sequence>
    </xs:element>
    <xs:element name="FieldGroups" type="FIELD-GROUPS" minOccurs="0">
      <xs:sequence>
        <xs:element name="FieldGroup" type="FIELD-GROUP-DECLARATION" minOccurs="0" maxOccurs="unbounded">
          <xs:sequence>
            <xs:element name="FieldName">
            </xs:element>
          </xs:sequence>
        </xs:element>
      </xs:sequence>
    </xs:element>
    <xs:element name="Fields" type="FIELDS">
      <xs:sequence>
        <xs:element name="Field" type="FIELD" minOccurs="0" maxOccurs="unbounded">
          <xs:group ref="FIELD-CONTENT">
            <xs:sequence>
              <xs:element ref="DisplayLabel"/>
              <xs:choice minOccurs="0">
                <xs:element ref="Range"/>
                <xs:element ref="Values"/>
              </xs:choice>
              <xs:element ref="MissingValues"/>
            </xs:sequence>
          </xs:group>
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
</xs:element>
```

Untergeordnete Elemente

FieldFormats, FieldGroups, Fields

FieldFormats-Element: Definiert die Standardfeldformate. Feldformate werden beim Anzeigen von Werten in der Ausgabe verwendet. Beispielsweise das allgemeine Format (Standardformate für Zahlen, wissenschaftliche Notation oder Währung), die Anzahl der anzuzeigenden Dezimalzeichen, Dezimaltrennzeichen usw. Derzeit werden Feldformate nur für numerische Felder verwendet. Dies kann sich aber in zukünftigen Versionen ändern.

Tabelle 85. Attribute für FieldFormats

Attribut	Verwendung	Beschreibung	Gültige Werte
count	optional		*nichtnegativeGanzzahl*
defaultNumberFormat	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="FieldFormats" type="FIELD-FORMATS" minOccurs="0">
  <xs:sequence>
    <xs:element name="NumberFormat" type="NUMBER-FORMAT-DECLARATION" minOccurs="0" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="defaultNumberFormat" type="xs:string" use="required"/>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Übergeordnete Elemente

DataModel

Untergeordnete Elemente

NumberFormat

NumberFormat-Element: Definiert Formatinformationen für ein numerisches Feld.

Tabelle 86. Attribute für NumberFormat

Attribut	Verwendung	Beschreibung	Gültige Werte
decimalPlaces	**erforderlich**		*nichtnegativeGanzzahl*
decimalSymbol	**erforderlich**		**period** **comma**
formatType	**erforderlich**		**standard** **scientific** **currency**
groupingSymbol	**erforderlich**		**none** **period** **comma** **space**
name	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="NumberFormat" type="NUMBER-FORMAT-DECLARATION" minOccurs="0" maxOccurs="unbounded">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="formatType" type="NUMBER-FORMAT-TYPE" use="required">
```

```
    <xs:enumeration value="standard"/>
    <xs:enumeration value="scientific"/>
    <xs:enumeration value="currency"/>
  </xs:attribute>
  <xs:attribute name="decimalPlaces" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="decimalSymbol" type="DECIMAL-SYMBOL" use="required">
    <xs:enumeration value="period"/>
    <xs:enumeration value="comma"/>
  </xs:attribute>
  <xs:attribute name="groupingSymbol" type="NUMBER-GROUPING-SYMBOL" use="required">
    <xs:enumeration value="none"/>
    <xs:enumeration value="period"/>
    <xs:enumeration value="comma"/>
    <xs:enumeration value="space"/>
  </xs:attribute>
</xs:element>
```

Übergeordnete Elemente

FieldFormats

FieldGroups-Element: Definiert die Feldgruppen. Feldgruppen werden verwendet, um zugehörige Felder zuzuordnen. Ein Umfragepunkt, in dem ein Befragter aufgefordert wird, aus einer Gruppe von Optionen die Orte auszuwählen, die er besucht hat, wird beispielsweise als Gruppe von Flagfeldern dargestellt. Eine Feldgruppe kann verwendet werden, um anzugeben, welche Felder dieser Umfragefrage zugeordnet werden.

Tabelle 87. Attribute für FieldGroups

Attribut	Verwendung	Beschreibung	Gültige Werte
count	optional		*nichtnegativeGanzzahl*

XML-Darstellung

```
<xs:element name="FieldGroups" type="FIELD-GROUPS" minOccurs="0">
  <xs:sequence>
    <xs:element name="FieldGroup" type="FIELD-GROUP-DECLARATION" minOccurs="0" maxOccurs="unbounded">
      <xs:sequence>
        <xs:element name="FieldName">
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Übergeordnete Elemente

DataModel

Untergeordnete Elemente

FieldGroup

FieldGroup-Element: Definiert eine Feldgruppe. Eine Feldgruppe besteht aus einer Liste von Feldnamen und Informationen zur Feldgruppe, wie dem Gruppennamen und optional einer Beschriftung, dem dem Typ der Gruppe und bei Multidichotomiegruppen dem gezählten Werte, d. h. dem Wert, der "true" (wahr) darstellt.

Tabelle 88. Attribute für FieldGroup

Attribut	Verwendung	Beschreibung	Gültige Werte
count	optional		*nichtnegativeGanzzahl*
countedValue	optional		*Zeichenfolge*
displayLabel	optional		*Zeichenfolge*

Tabelle 88. Attribute für FieldGroup (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
groupType	**erforderlich**		**fieldGroup** **multiCategorySet** **multiDichotomySet**
name	**erforderlich**		

XML-Darstellung

```
<xs:element name="FieldGroup" type="FIELD-GROUP-DECLARATION" minOccurs="0" maxOccurs="unbounded">
  <xs:sequence>
    <xs:element name="FieldName">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-GROUP-NAME" use="required"/>
  <xs:attribute name="displayLabel" type="xs:string"/>
  <xs:attribute name="groupType" type="FIELD-GROUP-TYPE" use="required">
    <xs:enumeration value="fieldGroup"/>
    <xs:enumeration value="multiCategorySet"/>
    <xs:enumeration value="multiDichotomySet"/>
  </xs:attribute>
  <xs:attribute name="countedValue" type="xs:string"/>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Übergeordnete Elemente

FieldGroups

Untergeordnete Elemente

FieldName

FieldName-Element:

Tabelle 89. Attribute für FieldName

Attribut	Verwendung	Beschreibung	Gültige Werte
name	**erforderlich**		

XML-Darstellung

```
<xs:element name="FieldName">
  <xs:attribute name="name" type="FIELD-NAME" use="required"/>
</xs:element>
```

Übergeordnete Elemente

FieldGroup

Fields-Element:

Tabelle 90. Attribute für Fields

Attribut	Verwendung	Beschreibung	Gültige Werte
count	optional		*nichtnegativeGanzzahl*

XML-Darstellung

```
<xs:element name="Fields" type="FIELDS">
  <xs:sequence>
    <xs:element name="Field" type="FIELD" minOccurs="0" maxOccurs="unbounded">
```

```
      <xs:group ref="FIELD-CONTENT">
        <xs:sequence>
          <xs:element ref="DisplayLabel"/>
          <xs:choice minOccurs="0">
            <xs:element ref="Range"/>
            <xs:element ref="Values"/>
          </xs:choice>
          <xs:element ref="MissingValues"/>
        </xs:sequence>
      </xs:group>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Übergeordnete Elemente

DataModel

Untergeordnete Elemente

Field

Field-Element:

Tabelle 91. Attribute für Field

Attribut	Verwendung	Beschreibung	Gültige Werte
direction	optional		**in** **out** **both** **none** **partition**
displayLabel	optional		*Zeichenfolge*
name	**erforderlich**		*Zeichenfolge*
storage	optional		**unknown** **integer** **real** **string** **date** **time** **timestamp** **list**
type	optional		**auto** **range** **discrete** **set** **orderedSet** **flag** **typeless** **collection** **geospatial**

XML-Darstellung

```
<xs:element name="Field" type="FIELD" minOccurs="0" maxOccurs="unbounded">
  <xs:group ref="FIELD-CONTENT">
    <xs:sequence>
      <xs:element ref="DisplayLabel"/>
      <xs:choice minOccurs="0">
        <xs:element ref="Range"/>
```

```
        <xs:element ref="Values"/>
      </xs:choice>
      <xs:element ref="MissingValues"/>
    </xs:sequence>
  </xs:group>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="type" type="FIELD-TYPE" default="auto">
    <xs:enumeration value="auto"/>
    <xs:enumeration value="range"/>
    <xs:enumeration value="discrete"/>
    <xs:enumeration value="set"/>
    <xs:enumeration value="orderedSet"/>
    <xs:enumeration value="flag"/>
    <xs:enumeration value="typeless"/>
    <xs:enumeration value="collection"/>
    <xs:enumeration value="geospatial"/>
  </xs:attribute>
  <xs:attribute name="storage" type="FIELD-STORAGE" default="unknown">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="real"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="time"/>
    <xs:enumeration value="timestamp"/>
    <xs:enumeration value="list"/>
  </xs:attribute>
  <xs:attribute name="direction" type="FIELD-DIRECTION" default="in">
    <xs:enumeration value="in"/>
    <xs:enumeration value="out"/>
    <xs:enumeration value="both"/>
    <xs:enumeration value="none"/>
    <xs:enumeration value="partition"/>
  </xs:attribute>
  <xs:attribute name="displayLabel" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

Fields

Untergeordnete Elemente

DisplayLabel, MissingValues, Range, Range, Values, Values

DatabaseConnectionValue-Element

Ein Wert, der die Details einer Datenbankverbindung angibt.

Tabelle 92. Attribute für DatabaseConnectionValue

Attribut	Verwendung	Beschreibung	Gültige Werte
connectionType	**erforderlich**		*Zeichenfolge*
datasourceName	**erforderlich**		*Zeichenfolge*
password	**erforderlich**		*Zeichenfolge*
userName	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="DatabaseConnectionValue" type="DATABASE-CONNECTION-VALUE">
  <xs:attribute name="connectionType" type="xs:string" use="required"/>
  <xs:attribute name="datasourceName" type="xs:string" use="required"/>
  <xs:attribute name="userName" type="xs:string" use="required"/>
  <xs:attribute name="password" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Attribute, Attribute, ListValue, ListValue, ListValue, Parameter

DefaultValue-Element

XML-Darstellung

```
<xs:element name="DefaultValue">
  <xs:choice>
    <xs:element name="ServerTempFile">
    </xs:element>
    <xs:element name="ServerTempDir">
    </xs:element>
    <xs:element name="Identifier">
    </xs:element>
  </xs:choice>
</xs:element>
```

Übergeordnete Elemente

Property, PropertyType

Untergeordnete Elemente

Identifier, ServerTempDir, ServerTempFile

ServerTempFile-Element:

Tabelle 93. Attribute für ServerTempFile

Attribut	Verwendung	Beschreibung	Gültige Werte
basename	**erforderlich**		

XML-Darstellung

```
<xs:element name="ServerTempFile">
  <xs:attribute name="basename" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Übergeordnete Elemente

DefaultValue

ServerTempDir-Element:

Tabelle 94. Attribute für ServerTempDir

Attribut	Verwendung	Beschreibung	Gültige Werte
basename	**erforderlich**		

XML-Darstellung

```
<xs:element name="ServerTempDir">
  <xs:attribute name="basename" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Übergeordnete Elemente

DefaultValue

Identifier-Element:

Tabelle 95. Attribute für Identifier

Attribut	Verwendung	Beschreibung	Gültige Werte
basename	**erforderlich**		

XML-Darstellung

```
<xs:element name="Identifier">
  <xs:attribute name="basename" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Übergeordnete Elemente

DefaultValue

DelimitedDataFormat-Element

Tabelle 96. Attribute für DelimitedDataFormat

Attribut	Verwendung	Beschreibung	Gültige Werte
delimeter	optional		**tab** **comma** **semicolon** **colon** **verticalBar** **other**
eol	optional		**cr** **crlf** **lf** **other**
includeFieldNames	optional		*boolean*
otherDelimiter	optional		*Zeichenfolge*
otherEol	optional		*Zeichenfolge*
quoteStrings	optional		*boolean*
stringQuote	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="DelimitedDataFormat">
  <xs:attribute name="delimeter" use="optional" default="tab">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="tab"/>
        <xs:enumeration value="comma"/>
        <xs:enumeration value="semicolon"/>
        <xs:enumeration value="colon"/>
        <xs:enumeration value="verticalBar"/>
        <xs:enumeration value="other"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="otherDelimiter" type="xs:string" use="optional"/>
  <xs:attribute name="eol" use="optional" default="cr">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="cr"/>
        <xs:enumeration value="crlf"/>
        <xs:enumeration value="lf"/>
        <xs:enumeration value="other"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="otherEol" type="xs:string" use="optional"/>
  <xs:attribute name="includeFieldNames" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="quoteStrings" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="stringQuote" type="xs:string" use="optional" default="""/>
</xs:element>
```

Übergeordnete Elemente

DataFile, DataFormat

DisplayLabel-Element

Eine Anzeigebeschriftung für ein Feld oder einen Wert in einer angegebenen Sprache. Das DisplayLabel-Attribut kann verwendet werden, wenn es keine Beschriftung für eine bestimmte Sprache gibt.

Tabelle 97. Attribute für DisplayLabel

Attribut	Verwendung	Beschreibung	Gültige Werte
lang	optional		*NMTOKEN*
value	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="lang" type="xs:NMTOKEN" default="en"/>
</xs:element>
```

Übergeordnete Elemente

Field

DocumentBuilder-Element

XML-Darstellung

```
<xs:element name="DocumentBuilder">
  <xs:sequence>
    <xs:element name="DocumentGeneration">
    </xs:element>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

Node

Untergeordnete Elemente

DocumentGeneration

DocumentGeneration-Element:

Tabelle 98. Attribute für DocumentGeneration

Attribut	Verwendung	Beschreibung	Gültige Werte
controlsId	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="DocumentGeneration">
  <xs:attribute name="controlsId" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

DocumentBuilder

DocumentOutput-Element

Tabelle 99. Attribute für DocumentOutput

Attribut	Verwendung	Beschreibung	Gültige Werte
delegate	optional		*Zeichenfolge*
deprecatedScriptNames	optional		*Zeichenfolge*
id	**erforderlich**		*Zeichenfolge*
scriptName	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="DocumentOutput">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="Properties"/>
      <xs:element name="Containers" minOccurs="0">
        <xs:sequence maxOccurs="unbounded">
          <xs:element ref="Container"/>
        </xs:sequence>
      </xs:element>
      <xs:element ref="UserInterface"/>
      <xs:element ref="Constructors" minOccurs="0"/>
      <xs:element ref="ModelProvider" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="scriptName" type="xs:string" use="optional"/>
  <xs:attribute name="deprecatedScriptNames" type="xs:string" use="optional"/>
  <xs:attribute name="delegate" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Extension

Untergeordnete Elemente

Constructors, Containers, ModelProvider, Properties, UserInterface

Zugehörige Elemente

InteractiveDocumentBuilder, InteractiveModelBuilder, ModelOutput, Node

Containers-Element:

XML-Darstellung

```
<xs:element name="Containers" minOccurs="0">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Container"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

Node

Untergeordnete Elemente

Container

DocumentType-Element

Definiert einen neuen Dokumenttyp.

Tabelle 100. Attribute für DocumentType

Attribut	Verwendung	Beschreibung	Gültige Werte
format	**erforderlich**		**utf8** **binary**
id	**erforderlich**		*Zeichenfolge*
type	optional		**unknown** **rowSet** **report** **graph**

XML-Darstellung

```
<xs:element name="DocumentType">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="format" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="utf8"/>
        <xs:enumeration value="binary"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="type" type="DOCUMENT-TYPE" use="optional">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="rowSet"/>
    <xs:enumeration value="report"/>
    <xs:enumeration value="graph"/>
  </xs:attribute>
</xs:element>
```

Übergeordnete Elemente

ContainerTypes

Zugehörige Elemente

ModelType

Enabled-Element

Definiert die Bedingung, unter der eine UI-Komponente aktiviert oder bearbeitbar sein soll.

XML-Darstellung

```
<xs:element name="Enabled">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

ActionButton, CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, ComboBoxControl, DBConnectionChooserControl, DBTableChooserControl, ExtensionObject-

Panel, FieldAllocationList, ItemChooserControl, ModelViewerPanel, MultiFieldAllocationControl, MultiFieldChooserControl, MultiItemChooserControl, PasswordBoxControl, PropertiesPanel, PropertiesSubPanel, PropertyControl, RadioButtonGroupControl, SelectorPanel, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SingleItemChooserControl, SpinnerControl, StaticText, SystemControls, TabbedPanel, TableControl, TextAreaControl, TextBoxControl, TextBrowserPanel

Untergeordnete Elemente

And, Condition, Not, Or

Enumeration-Element

XML-Darstellung

```
<xs:element name="Enumeration">
  <xs:sequence>
    <xs:element name="Enum" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

PropertyType

Untergeordnete Elemente

Enum

Enum-Element:

Tabelle 101. Attribute für Enum

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
imagePath	optional		*Zeichenfolge*
imagePathKey	optional		*Zeichenfolge*
label	**erforderlich**		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
value	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Enum" maxOccurs="unbounded">
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="imagePath" type="xs:string" use="optional"/>
  <xs:attribute name="imagePathKey" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Enumeration

ErrorDetail-Element

Ergänzende Informationen zu einem Fehler oder einer anderen Bedingung.

XML-Darstellung

```
<xs:element name="ErrorDetail" type="ERROR-DETAIL">
  <xs:sequence>
    <xs:element name="Diagnostic" type="DIAGNOSTIC" minOccurs="0" maxOccurs="unbounded">
      <xs:sequence>
        <xs:element name="Message" type="DIAGNOSTIC-MESSAGE" minOccurs="0">
        </xs:element>
        <xs:element name="Parameter" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
</xs:element>
```

Untergeordnete Elemente

Diagnostic

Diagnostic-Element:

Tabelle 102. Attribute für Diagnostic

Attribut	Verwendung	Beschreibung	Gültige Werte
code	**erforderlich**		*Ganzzahl*
severity	optional		**unknown** **information** **warning** **error** **fatal**
source	optional		*Zeichenfolge*
subCode	optional		*Ganzzahl*

XML-Darstellung

```
<xs:element name="Diagnostic" type="DIAGNOSTIC" minOccurs="0" maxOccurs="unbounded">
  <xs:sequence>
    <xs:element name="Message" type="DIAGNOSTIC-MESSAGE" minOccurs="0">
    </xs:element>
    <xs:element name="Parameter" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="code" type="xs:integer" use="required"/>
  <xs:attribute name="subCode" type="xs:integer" default="0"/>
  <xs:attribute name="severity" type="DIAGNOSTIC-SEVERITY" default="error">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="information"/>
    <xs:enumeration value="warning"/>
    <xs:enumeration value="error"/>
    <xs:enumeration value="fatal"/>
  </xs:attribute>
  <xs:attribute name="source" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

ErrorDetail

Untergeordnete Elemente

Message, Parameter

Message-Element:

Tabelle 103. Attribute für Message

Attribut	Verwendung	Beschreibung	Gültige Werte
lang	optional		*NMTOKEN*

XML-Darstellung

```
<xs:element name="Message" type="DIAGNOSTIC-MESSAGE" minOccurs="0">
  <xs:attribute name="lang" type="xs:NMTOKEN"/>
</xs:element>
```

Übergeordnete Elemente

Diagnostic

Parameter-Element:

XML-Darstellung

```
<xs:element name="Parameter" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
```

Übergeordnete Elemente

Diagnostic

Executable-Element

XML-Darstellung

```
<xs:element name="Executable">
  <xs:sequence>
    <xs:element ref="Run" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

Execution

Untergeordnete Elemente

Run

Execution-Element

XML-Darstellung

```
<xs:element name="Execution">
  <xs:sequence>
    <xs:element ref="Properties" minOccurs="0"/>
    <xs:element ref="InputFiles"/>
    <xs:element ref="OutputFiles"/>
    <xs:choice>
      <xs:element ref="Executable"/>
      <xs:element ref="Module"/>
    </xs:choice>
    <xs:element ref="Constructors" minOccurs="0"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

Node

Untergeordnete Elemente

Constructors, Executable, InputFiles, Module, OutputFiles, Properties

Extension-Element

Definiert die Erweiterungscontainer der höchsten Ebene.

Tabelle 104. Attribute für Extension

Attribut	Verwendung	Beschreibung	Gültige Werte
debug	optional		*boolean*
version	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Extension">
  <xs:sequence>
    <xs:element ref="ExtensionDetails"/>
    <xs:element ref="Resources"/>
    <xs:element ref="License" minOccurs="0"/>
    <xs:element ref="CommonObjects"/>
    <xs:element ref="UserInterface" minOccurs="0"/>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="Node"/>
        <xs:element ref="ModelOutput"/>
        <xs:element ref="DocumentOutput"/>
        <xs:element ref="InteractiveModelBuilder"/>
        <xs:element ref="InteractiveDocumentBuilder"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="version" type="xs:string" use="required"/>
  <xs:attribute name="debug" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Untergeordnete Elemente

CommonObjects, DocumentOutput, ExtensionDetails, InteractiveDocumentBuilder, InteractiveModelBuilder, License, ModelOutput, Node, Resources, UserInterface

ExtensionDetails-Element

Definiert Informationen zur Erweiterung wie z. B. die Erweiterungs-ID, den Erweiterungsprovider und die Versionsinformation.

Tabelle 105. Attribute für ExtensionDetails

Attribut	Verwendung	Beschreibung	Gültige Werte
copyright	optional		*Zeichenfolge*
description	optional		*Zeichenfolge*
id	**erforderlich**		*Zeichenfolge*
label	**erforderlich**		*Zeichenfolge*
provider	optional		*Zeichenfolge*
providerTag	**erforderlich**		*Zeichenfolge*
version	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="ExtensionDetails">
  <xs:attribute name="providerTag" type="xs:string" use="required"/>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="version" type="xs:string"/>
```

```
  <xs:attribute name="provider" type="xs:string" use="optional" default="(unknown)"/>
  <xs:attribute name="copyright" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Extension

ExtensionObjectPanel-Element

Definiert ein benutzerdefiniertes Fenster. Die vom Attribut panelClass angegebene Klasse muss die ExtensionObjectPanel-Schnittstelle implementieren.

Tabelle 106. Attribute für ExtensionObjectPanel

Attribut	Verwendung	Beschreibung	Gültige Werte
id	optional		*Zeichenfolge*
panelClass	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="ExtensionObjectPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="panelClass" type="xs:string" use="required"/>
  <xs:attribute name="id" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel, Tab

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

ActionButton, ComboBoxControl, FieldAllocationList, ModelViewerPanel, SelectorPanel, StaticText, SystemControls, TabbedPanel, TextBrowserPanel

Field-Element

Tabelle 107. Attribute für Field

Attribut	Verwendung	Beschreibung	Gültige Werte
depth	optional		*Ganzzahl*
direction	optional		**in** **out** **both** **none** **partition**
label	optional		*Zeichenfolge*
name	**erforderlich**		

Tabelle 107. Attribute für Field (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
storage	optional		**unknown** **integer** **real** **string** **date** **time** **timestamp** **list**
type	optional		**auto** **range** **discrete** **set** **orderedSet** **flag** **typeless** **collection** **geospatial**
valueStorage	optional		**unknown** **integer** **real** **string** **date** **time** **timestamp** **list**

XML-Darstellung

```
<xs:element name="Field" type="FIELD-DECLARATION">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Range" minOccurs="0"/>
      <xs:element ref="Values" minOccurs="0"/>
      <xs:element ref="NumericInfo" minOccurs="0"/>
      <xs:element name="MissingValues" minOccurs="0">
        <xs:sequence>
          <xs:element ref="Values" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="Range" minOccurs="0"/>
        </xs:sequence>
      </xs:element>
      <xs:element name="ModelField" type="MODEL-FIELD-INFORMATION" minOccurs="0">
      </xs:element>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-NAME" use="required"/>
  <xs:attribute name="storage" type="FIELD-STORAGE">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="real"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="time"/>
    <xs:enumeration value="timestamp"/>
    <xs:enumeration value="list"/>
  </xs:attribute>
  <xs:attribute name="type" type="FIELD-TYPE">
    <xs:enumeration value="auto"/>
    <xs:enumeration value="range"/>
    <xs:enumeration value="discrete"/>
    <xs:enumeration value="set"/>
    <xs:enumeration value="orderedSet"/>
    <xs:enumeration value="flag"/>
    <xs:enumeration value="typeless"/>
```

```
    <xs:enumeration value="collection"/>
    <xs:enumeration value="geospatial"/>
  </xs:attribute>
  <xs:attribute name="direction" type="FIELD-DIRECTION">
    <xs:enumeration value="in"/>
    <xs:enumeration value="out"/>
    <xs:enumeration value="both"/>
    <xs:enumeration value="none"/>
    <xs:enumeration value="partition"/>
  </xs:attribute>
  <xs:attribute name="label" type="xs:string"/>
  <xs:attribute name="depth" type="xs:integer" use="optional" default="-1"/>
  <xs:attribute name="valueStorage" type="FIELD-STORAGE" use="optional">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="real"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="time"/>
    <xs:enumeration value="timestamp"/>
    <xs:enumeration value="list"/>
  </xs:attribute>
</xs:element>
```

Untergeordnete Elemente

MissingValues, ModelField, NumericInfo, Range, Range, Values, Values

MissingValues-Element:

Tabelle 108. Attribute für MissingValues

Attribut	Verwendung	Beschreibung	Gültige Werte
treatNullAsMissing	optional		*boolean*
treatWhitespaceAsMissing	optional		*boolean*

XML-Darstellung

```
<xs:element name="MissingValues" minOccurs="0">
  <xs:sequence>
    <xs:element ref="Values" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Range" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="treatWhitespaceAsMissing" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="treatNullAsMissing" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Übergeordnete Elemente

AddField

Untergeordnete Elemente

Range, Range, Values, Values

ModelField-Element:

Tabelle 109. Attribute für ModelField

Attribut	Verwendung	Beschreibung	Gültige Werte
group	optional		

Tabelle 109. Attribute für ModelField (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
role	**erforderlich**		**unknown** **predictedValue** **predictedDisplayValue** **probability** **residual** **standardError** **entityId** **entityAffinity** **upperConfidenceLimit** **lowerConfidenceLimit** **propensity** **value** **supplementary**
tag	optional		*Zeichenfolge*
targetField	optional		*Zeichenfolge*
value	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="ModelField" type="MODEL-FIELD-INFORMATION" minOccurs="0">
  <xs:attribute name="role" type="MODEL-FIELD-ROLE" use="required">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="predictedValue"/>
    <xs:enumeration value="predictedDisplayValue"/>
    <xs:enumeration value="probability"/>
    <xs:enumeration value="residual"/>
    <xs:enumeration value="standardError"/>
    <xs:enumeration value="entityId"/>
    <xs:enumeration value="entityAffinity"/>
    <xs:enumeration value="upperConfidenceLimit"/>
    <xs:enumeration value="lowerConfidenceLimit"/>
    <xs:enumeration value="propensity"/>
    <xs:enumeration value="value"/>
    <xs:enumeration value="supplementary"/>
  </xs:attribute>
  <xs:attribute name="targetField" type="xs:string"/>
  <xs:attribute name="value" type="xs:string"/>
  <xs:attribute name="group" type="MODEL-FIELD-GROUP"/>
  <xs:attribute name="tag" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

AddField

FieldAllocationList-Element

Definiert ein Fenster, das eine Liste verfügbarer Felder enthält. Steuerelemente für die Zuordnung einzelner und mehrerer Felder können Felder aus diesem Fenster zuordnen.

Tabelle 110. Attribute für FieldAllocationList

Attribut	Verwendung	Beschreibung	Gültige Werte
enabledProperty	optional		*Zeichenfolge*
enabledValue	optional		*Zeichenfolge*
id	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="FieldAllocationList">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="enabledProperty" type="xs:string" use="optional"/>
  <xs:attribute name="enabledValue" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

ActionButton, ComboBoxControl, ExtensionObjectPanel, ModelViewerPanel, SelectorPanel, StaticText, SystemControls, TabbedPanel, TextBrowserPanel

FieldFormats-Element

Definiert die Standardfeldformate. Feldformate werden beim Anzeigen von Werten in der Ausgabe verwendet. Beispielsweise das allgemeine Format (Standardformate für Zahlen, wissenschaftliche Notation oder Währung), die Anzahl der anzuzeigenden Dezimalzeichen, Dezimaltrennzeichen usw. Derzeit werden Feldformate nur für numerische Felder verwendet. Dies kann sich aber in zukünftigen Versionen ändern.

Tabelle 111. Attribute für FieldFormats

Attribut	Verwendung	Beschreibung	Gültige Werte
count	optional		*nichtnegativeGanzzahl*
defaultNumberFormat	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="FieldFormats" type="FIELD-FORMATS">
  <xs:sequence>
    <xs:element name="NumberFormat" type="NUMBER-FORMAT-DECLARATION" minOccurs="0" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="defaultNumberFormat" type="xs:string" use="required"/>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Untergeordnete Elemente

NumberFormat

NumberFormat-Element: Definiert Formatinformationen für ein numerisches Feld.

Tabelle 112. Attribute für NumberFormat

Attribut	Verwendung	Beschreibung	Gültige Werte
decimalPlaces	**erforderlich**		*nichtnegativeGanzzahl*

Tabelle 112. Attribute für NumberFormat (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
decimalSymbol	**erforderlich**		**period** **comma**
formatType	**erforderlich**		**standard** **scientific** **currency**
groupingSymbol	**erforderlich**		**none** **period** **comma** **space**
name	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="NumberFormat" type="NUMBER-FORMAT-DECLARATION" minOccurs="0" maxOccurs="unbounded">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="formatType" type="NUMBER-FORMAT-TYPE" use="required">
    <xs:enumeration value="standard"/>
    <xs:enumeration value="scientific"/>
    <xs:enumeration value="currency"/>
  </xs:attribute>
  <xs:attribute name="decimalPlaces" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="decimalSymbol" type="DECIMAL-SYMBOL" use="required">
    <xs:enumeration value="period"/>
    <xs:enumeration value="comma"/>
  </xs:attribute>
  <xs:attribute name="groupingSymbol" type="NUMBER-GROUPING-SYMBOL" use="required">
    <xs:enumeration value="none"/>
    <xs:enumeration value="period"/>
    <xs:enumeration value="comma"/>
    <xs:enumeration value="space"/>
  </xs:attribute>
</xs:element>
```

Übergeordnete Elemente

FieldFormats

FieldGroup-Element

Definiert eine Feldgruppe. Eine Feldgruppe besteht aus einer Liste von Feldnamen und Informationen zur Feldgruppe, wie dem Gruppennamen und optional einer Beschriftung, dem dem Typ der Gruppe und bei Multidichotomiegruppen dem gezählten Werte, d. h. dem Wert, der "true" (wahr) darstellt.

Tabelle 113. Attribute für FieldGroup

Attribut	Verwendung	Beschreibung	Gültige Werte
count	optional		*nichtnegativeGanzzahl*
countedValue	optional		*Zeichenfolge*
displayLabel	optional		*Zeichenfolge*
groupType	**erforderlich**		**fieldGroup** **multiCategorySet** **multiDichotomySet**
name	**erforderlich**		

XML-Darstellung

```
<xs:element name="FieldGroup" type="FIELD-GROUP-DECLARATION">
  <xs:sequence>
    <xs:element name="FieldName">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-GROUP-NAME" use="required"/>
  <xs:attribute name="displayLabel" type="xs:string"/>
  <xs:attribute name="groupType" type="FIELD-GROUP-TYPE" use="required">
    <xs:enumeration value="fieldGroup"/>
    <xs:enumeration value="multiCategorySet"/>
    <xs:enumeration value="multiDichotomySet"/>
  </xs:attribute>
  <xs:attribute name="countedValue" type="xs:string"/>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Untergeordnete Elemente

FieldName

FieldName-Element:

Tabelle 114. Attribute für FieldName

Attribut	Verwendung	Beschreibung	Gültige Werte
name	**erforderlich**		

XML-Darstellung

```
<xs:element name="FieldName">
  <xs:attribute name="name" type="FIELD-NAME" use="required"/>
</xs:element>
```

Übergeordnete Elemente

FieldGroup

FieldGroups-Element

Definiert die Feldgruppen. Feldgruppen werden verwendet, um zugehörige Felder zuzuordnen. Ein Umfragepunkt, in dem ein Befragter aufgefordert wird, aus einer Gruppe von Optionen die Orte auszuwählen, die er besucht hat, wird beispielsweise als Gruppe von Flagfeldern dargestellt. Eine Feldgruppe kann verwendet werden, um anzugeben, welche Felder dieser Umfragefrage zugeordnet werden.

Tabelle 115. Attribute für FieldGroups

Attribut	Verwendung	Beschreibung	Gültige Werte
count	optional		*nichtnegativeGanzzahl*

XML-Darstellung

```
<xs:element name="FieldGroups" type="FIELD-GROUPS">
  <xs:sequence>
    <xs:element name="FieldGroup" type="FIELD-GROUP-DECLARATION" minOccurs="0" maxOccurs="unbounded">
      <xs:sequence>
        <xs:element name="FieldName">
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Untergeordnete Elemente

FieldGroup

FieldGroup-Element: Definiert eine Feldgruppe. Eine Feldgruppe besteht aus einer Liste von Feldnamen und Informationen zur Feldgruppe, wie dem Gruppennamen und optional einer Beschriftung, dem dem Typ der Gruppe und bei Multidichotomiegruppen dem gezählten Werte, d. h. dem Wert, der "true" (wahr) darstellt.

Tabelle 116. Attribute für FieldGroup

Attribut	Verwendung	Beschreibung	Gültige Werte
count	optional		*nichtnegativeGanzzahl*
countedValue	optional		*Zeichenfolge*
displayLabel	optional		*Zeichenfolge*
groupType	**erforderlich**		**fieldGroup** **multiCategorySet** **multiDichotomySet**
name	**erforderlich**		

XML-Darstellung

```
<xs:element name="FieldGroup" type="FIELD-GROUP-DECLARATION" minOccurs="0" maxOccurs="unbounded">
  <xs:sequence>
    <xs:element name="FieldName">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-GROUP-NAME" use="required"/>
  <xs:attribute name="displayLabel" type="xs:string"/>
  <xs:attribute name="groupType" type="FIELD-GROUP-TYPE" use="required">
    <xs:enumeration value="fieldGroup"/>
    <xs:enumeration value="multiCategorySet"/>
    <xs:enumeration value="multiDichotomySet"/>
  </xs:attribute>
  <xs:attribute name="countedValue" type="xs:string"/>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Übergeordnete Elemente

FieldGroups

Untergeordnete Elemente

FieldName

FieldName-Element:

Tabelle 117. Attribute für FieldName

Attribut	Verwendung	Beschreibung	Gültige Werte
name	**erforderlich**		

XML-Darstellung

```
<xs:element name="FieldName">
  <xs:attribute name="name" type="FIELD-NAME" use="required"/>
</xs:element>
```

Übergeordnete Elemente

FieldGroup

FileFormatType-Element

Tabelle 118. Attribute für FileFormatType

Attribut	Verwendung	Beschreibung	Gültige Werte
name	optional		

XML-Darstellung

```
<xs:element name="FileFormatType">
  <xs:sequence>
    <xs:group ref="FILE-FORMAT">
      <xs:choice>
        <xs:element ref="UTF8Format"/>
        <xs:element ref="BinaryFormat"/>
        <xs:element ref="DataFormat"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
  <xs:attribute name="name" type="EVALUATED-STRING" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

FileFormatTypes

Untergeordnete Elemente

BinaryFormat, DataFormat, UTF8Format

FileFormatTypes-Element

XML-Darstellung

```
<xs:element name="FileFormatTypes">
  <xs:sequence>
    <xs:element ref="FileFormatType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

CommonObjects

Untergeordnete Elemente

FileFormatType

ForEach-Element

Tabelle 119. Attribute für ForEach

Attribut	Verwendung	Beschreibung	Gültige Werte
container	optional		*Zeichenfolge*
from	optional		*Zeichenfolge*
inFieldValues	optional		*Zeichenfolge*
inFields	optional		*Zeichenfolge*
inProperty	optional		*Zeichenfolge*
step	optional		*Zeichenfolge*
to	optional		*Zeichenfolge*
var	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="ForEach">
  <xs:sequence maxOccurs="unbounded">
    <xs:group ref="DATA-MODEL-EXPRESSION">
      <xs:choice>
        <xs:element ref="ForEach"/>
        <xs:element ref="AddField"/>
        <xs:element ref="ChangeField"/>
        <xs:element ref="RemoveField"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
  <xs:attribute name="var" type="xs:string" use="required"/>
  <xs:attribute name="inProperty" type="xs:string" use="optional"/>
  <xs:attribute name="inFields" type="xs:string" use="optional"/>
  <xs:attribute name="inFieldValues" type="xs:string" use="optional"/>
  <xs:attribute name="from" type="xs:string" use="optional"/>
  <xs:attribute name="to" type="xs:string" use="optional"/>
  <xs:attribute name="step" type="xs:string" use="optional"/>
  <xs:attribute name="container" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

ForEach, ModelFields

Untergeordnete Elemente

AddField, ChangeField, ForEach, RemoveField

Icon-Element

Tabelle 120. Attribute für Icon

Attribut	Verwendung	Beschreibung	Gültige Werte
imagePath	**erforderlich**		*Zeichenfolge*
resourceID	optional		*Zeichenfolge*
type	**erforderlich**		**standardNode** **smallNode** **standardWindow**

XML-Darstellung

```
<xs:element name="Icon">
  <xs:attribute name="type" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="standardNode"/>
        <xs:enumeration value="smallNode"/>
        <xs:enumeration value="standardWindow"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="imagePath" type="xs:string" use="required"/>
  <xs:attribute name="resourceID" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Icons, Palette

Icons-Element

XML-Darstellung

```
<xs:element name="Icons">
  <xs:sequence>
    <xs:element ref="Icon" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

UserInterface

Untergeordnete Elemente

Icon

InputFiles-Element

XML-Darstellung

```
<xs:element name="InputFiles">
  <xs:group ref="RUNTIME-FILES">
    <xs:sequence>
      <xs:element ref="DataFile"/>
      <xs:element ref="ContainerFile" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:group>
</xs:element>
```

Übergeordnete Elemente

Execution, Module

Untergeordnete Elemente

ContainerFile, DataFile

InteractiveDocumentBuilder-Element

Tabelle 121. Attribute für InteractiveDocumentBuilder

Attribut	Verwendung	Beschreibung	Gültige Werte
delegate	optional		*Zeichenfolge*
deprecatedScriptNames	optional		*Zeichenfolge*
id	**erforderlich**		*Zeichenfolge*
scriptName	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="InteractiveDocumentBuilder">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="Properties"/>
      <xs:element name="Containers" minOccurs="0">
        <xs:sequence maxOccurs="unbounded">
          <xs:element ref="Container"/>
        </xs:sequence>
      </xs:element>
      <xs:element ref="UserInterface"/>
      <xs:element ref="Constructors" minOccurs="0"/>
      <xs:element ref="ModelProvider" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
```

```
  <xs:attribute name="scriptName" type="xs:string" use="optional"/>
  <xs:attribute name="deprecatedScriptNames" type="xs:string" use="optional"/>
  <xs:attribute name="delegate" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Extension

Untergeordnete Elemente

Constructors, Containers, ModelProvider, Properties, UserInterface

Zugehörige Elemente

DocumentOutput, InteractiveModelBuilder, ModelOutput, Node

Containers-Element:

XML-Darstellung

```
<xs:element name="Containers" minOccurs="0">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Container"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

Node

Untergeordnete Elemente

Container

InteractiveModelBuilder-Element

Tabelle 122. Attribute für InteractiveModelBuilder

Attribut	Verwendung	Beschreibung	Gültige Werte
delegate	optional		*Zeichenfolge*
deprecatedScriptNames	optional		*Zeichenfolge*
id	**erforderlich**		*Zeichenfolge*
scriptName	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="InteractiveModelBuilder">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="Properties"/>
      <xs:element name="Containers" minOccurs="0">
        <xs:sequence maxOccurs="unbounded">
          <xs:element ref="Container"/>
        </xs:sequence>
      </xs:element>
      <xs:element ref="UserInterface"/>
      <xs:element ref="Constructors" minOccurs="0"/>
      <xs:element ref="ModelProvider" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="scriptName" type="xs:string" use="optional"/>
  <xs:attribute name="deprecatedScriptNames" type="xs:string" use="optional"/>
  <xs:attribute name="delegate" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Extension

Untergeordnete Elemente

Constructors, Containers, ModelProvider, Properties, UserInterface

Zugehörige Elemente

DocumentOutput, InteractiveDocumentBuilder, ModelOutput, Node

Containers-Element:

XML-Darstellung

```
<xs:element name="Containers" minOccurs="0">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Container"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

Node

Untergeordnete Elemente

Container

Layout-Element

Definiert das Layout einer UI-Komponente in einem Fenster. Das Layout basiert auf einem Grid-Bag-Layout, bei dem jedes Steuerelement mindestens eine Zelle im Raster belegt. Das Layout ermöglicht die Angabe des Größenänderungs-, Füll-, Anker- und Einrückungsverhaltens.

Tabelle 123. Attribute für Layout

Attribut	Verwendung	Beschreibung	Gültige Werte
anchor	optional		**north** **northeast** **east** **southeast** **south** **southwest** **west** **northwest** **center**
columnWeight	optional		*double*
fill	optional		**horizontal** **vertical** **both** **none**
gridColumn	optional		*nichtnegativeGanzzahl*
gridHeight	optional		*nichtnegativeGanzzahl*
gridRow	optional		*nichtnegativeGanzzahl*
gridWidth	optional		*nichtnegativeGanzzahl*

Tabelle 123. Attribute für Layout (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
leftIndent	optional		*nichtnegativeGanzzahl*
rowIncrement	optional		*nichtnegativeGanzzahl*
rowWeight	optional		*double*

XML-Darstellung

```
<xs:element name="Layout">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="Cell">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="gridRow" type="xs:nonNegativeInteger" use="optional"/>
  <xs:attribute name="gridColumn" type="xs:nonNegativeInteger" use="optional"/>
  <xs:attribute name="rowIncrement" type="xs:nonNegativeInteger" use="optional"/>
  <xs:attribute name="gridWidth" type="xs:nonNegativeInteger" use="optional" default="1"/>
  <xs:attribute name="gridHeight" type="xs:nonNegativeInteger" use="optional" default="1"/>
  <xs:attribute name="rowWeight" type="xs:double" use="optional"/>
  <xs:attribute name="columnWeight" type="xs:double" use="optional"/>
  <xs:attribute name="fill" type="UI-COMPONENT-FILL" use="optional" default="none">
    <xs:enumeration value="horizontal"/>
    <xs:enumeration value="vertical"/>
    <xs:enumeration value="both"/>
    <xs:enumeration value="none"/>
  </xs:attribute>
  <xs:attribute name="anchor" type="UI-COMPONENT-ANCHOR" use="optional" default="west">
    <xs:enumeration value="north"/>
    <xs:enumeration value="northeast"/>
    <xs:enumeration value="east"/>
    <xs:enumeration value="southeast"/>
    <xs:enumeration value="south"/>
    <xs:enumeration value="southwest"/>
    <xs:enumeration value="west"/>
    <xs:enumeration value="northwest"/>
    <xs:enumeration value="center"/>
  </xs:attribute>
  <xs:attribute name="leftIndent" type="xs:nonNegativeInteger" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

ActionButton, CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, ComboBoxControl, DBConnectionChooserControl, DBTableChooserControl, ExtensionObjectPanel, FieldAllocationList, ItemChooserControl, ModelViewerPanel, MultiFieldAllocationControl, MultiFieldChooserControl, MultiItemChooserControl, PasswordBoxControl, PropertiesPanel, PropertiesSubPanel, PropertyControl, RadioButtonGroupControl, SelectorPanel, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SingleItemChooserControl, SpinnerControl, StaticText, SystemControls, TabbedPanel, TableControl, TextAreaControl, TextBoxControl, TextBrowserPanel

Untergeordnete Elemente

Cell

Cell-Element:

Tabelle 124. Attribute für Cell

Attribut	Verwendung	Beschreibung	Gültige Werte
column	**erforderlich**		*nichtnegativeGanzzahl*
row	**erforderlich**		*nichtnegativeGanzzahl*
width	**erforderlich**		*nichtnegativeGanzzahl*

XML-Darstellung

```
<xs:element name="Cell">
  <xs:attribute name="row" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="column" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="width" type="xs:nonNegativeInteger" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Layout

License-Element

Für die Verwendung durch das System reserviert.

XML-Darstellung

```
<xs:element name="License">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="OptionCode"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

Extension

Untergeordnete Elemente

OptionCode

ListValue-Element

Eine Folge von Werten. Alle Werte müssen denselben Inhaltstyp haben, aber dies wird nicht geprüft.

XML-Darstellung

```
<xs:element name="ListValue" type="LIST-VALUE">
  <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="MapValue"/>
      <xs:element ref="StructuredValue"/>
      <xs:element ref="ListValue"/>
      <xs:element ref="Value"/>
      <xs:element ref="DatabaseConnectionValue"/>
    </xs:choice>
  </xs:group>
</xs:element>
```

Übergeordnete Elemente

Attribute, Attribute, ListValue, ListValue, ListValue, Parameter

Untergeordnete Elemente

DatabaseConnectionValue, ListValue, MapValue, StructuredValue, Value

MapValue-Element

Ein Set von Zuordnungseinträgen, die jeweils aus einem Schlüssel und einem Wert bestehen.

XML-Darstellung

```
<xs:element name="MapValue" type="MAP-VALUE">
  <xs:sequence>
    <xs:element name="MapEntry" type="MAP-ENTRY" maxOccurs="unbounded">
      <xs:sequence>
        <xs:element name="KeyValue" type="KEY-VALUE">
        </xs:element>
        <xs:element name="StructuredValue" type="STRUCTURED-VALUE">
```

```
        <xs:sequence>
          <xs:element name="Attribute" type="ATTRIBUTE" maxOccurs="unbounded">
            <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
              <xs:choice>
                <xs:element ref="MapValue"/>
                <xs:element ref="StructuredValue"/>
                <xs:element ref="ListValue"/>
                <xs:element ref="Value"/>
                <xs:element ref="DatabaseConnectionValue"/>
              </xs:choice>
            </xs:group>
            <xs:sequence>
              <xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
                <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
                  <xs:choice>
                    <xs:element ref="MapValue"/>
                    <xs:element ref="StructuredValue"/>
                    <xs:element ref="ListValue"/>
                    <xs:element ref="Value"/>
                    <xs:element ref="DatabaseConnectionValue"/>
                  </xs:choice>
                </xs:group>
              </xs:element>
            </xs:sequence>
          </xs:element>
        </xs:sequence>
      </xs:element>
    </xs:sequence>
  </xs:element>
</xs:element>
```

Übergeordnete Elemente

Attribute, Attribute, ListValue, ListValue, ListValue, Parameter

Untergeordnete Elemente

MapEntry

MapEntry-Element: Ein Eintrag in einer verschlüsselten Eigenschaftszuordnung. Jeder Eintrag besteht aus einem Schlüssel und einem zugehörigen Wert.

XML-Darstellung

```
<xs:element name="MapEntry" type="MAP-ENTRY" maxOccurs="unbounded">
  <xs:sequence>
    <xs:element name="KeyValue" type="KEY-VALUE">
    </xs:element>
    <xs:element name="StructuredValue" type="STRUCTURED-VALUE">
      <xs:sequence>
        <xs:element name="Attribute" type="ATTRIBUTE" maxOccurs="unbounded">
          <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
            <xs:choice>
              <xs:element ref="MapValue"/>
              <xs:element ref="StructuredValue"/>
              <xs:element ref="ListValue"/>
              <xs:element ref="Value"/>
              <xs:element ref="DatabaseConnectionValue"/>
            </xs:choice>
          </xs:group>
          <xs:sequence>
            <xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
              <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
                <xs:choice>
                  <xs:element ref="MapValue"/>
                  <xs:element ref="StructuredValue"/>
                  <xs:element ref="ListValue"/>
                  <xs:element ref="Value"/>
                  <xs:element ref="DatabaseConnectionValue"/>
                </xs:choice>
              </xs:group>
            </xs:element>
          </xs:sequence>
        </xs:element>
      </xs:sequence>
    </xs:element>
```

```
      </xs:sequence>
    </xs:element>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

MapValue

Untergeordnete Elemente

KeyValue, StructuredValue

KeyValue-Element: Der Schlüsselwert in einem Zuordnungseintrag.

Tabelle 125. Attribute für KeyValue

Attribut	Verwendung	Beschreibung	Gültige Werte
value	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="KeyValue" type="KEY-VALUE">
  <xs:attribute name="value" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

MapEntry

StructuredValue-Element: Eine Folge von benannten Werten ("Attribute").

XML-Darstellung

```
<xs:element name="StructuredValue" type="STRUCTURED-VALUE">
  <xs:sequence>
    <xs:element name="Attribute" type="ATTRIBUTE" maxOccurs="unbounded">
      <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
        <xs:choice>
          <xs:element ref="MapValue"/>
          <xs:element ref="StructuredValue"/>
          <xs:element ref="ListValue"/>
          <xs:element ref="Value"/>
          <xs:element ref="DatabaseConnectionValue"/>
        </xs:choice>
      </xs:group>
      <xs:sequence>
        <xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
          <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
            <xs:choice>
              <xs:element ref="MapValue"/>
              <xs:element ref="StructuredValue"/>
              <xs:element ref="ListValue"/>
              <xs:element ref="Value"/>
              <xs:element ref="DatabaseConnectionValue"/>
            </xs:choice>
          </xs:group>
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

MapEntry

Untergeordnete Elemente

Attribute

Attribute-Element:

Tabelle 126. Attribute für Attribute

Attribut	Verwendung	Beschreibung	Gültige Werte
name	**erforderlich**		*Zeichenfolge*
value	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Attribute" type="ATTRIBUTE" maxOccurs="unbounded">
  <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
    <xs:choice>
      <xs:element ref="MapValue"/>
      <xs:element ref="StructuredValue"/>
      <xs:element ref="ListValue"/>
      <xs:element ref="Value"/>
      <xs:element ref="DatabaseConnectionValue"/>
    </xs:choice>
  </xs:group>
  <xs:sequence>
    <xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
      <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
        <xs:choice>
          <xs:element ref="MapValue"/>
          <xs:element ref="StructuredValue"/>
          <xs:element ref="ListValue"/>
          <xs:element ref="Value"/>
          <xs:element ref="DatabaseConnectionValue"/>
        </xs:choice>
      </xs:group>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="value" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

StructuredValue

Untergeordnete Elemente

DatabaseConnectionValue, ListValue, ListValue, MapValue, StructuredValue, Value

ListValue-Element: Eine Folge von Werten. Alle Werte müssen denselben Inhaltstyp haben, aber dies wird nicht geprüft.

XML-Darstellung

```
<xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
  <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="MapValue"/>
      <xs:element ref="StructuredValue"/>
      <xs:element ref="ListValue"/>
      <xs:element ref="Value"/>
      <xs:element ref="DatabaseConnectionValue"/>
    </xs:choice>
  </xs:group>
</xs:element>
```

Übergeordnete Elemente

Attribute

Untergeordnete Elemente

DatabaseConnectionValue, ListValue, MapValue, StructuredValue, Value

Menu-Element

Definiert ein Menü. Dies kann ein neues Menü höchster Ebene in einer Menüleiste oder ein Untermenü in einem vorhandenen Menü sein.

Tabelle 127. Attribute für Menu

Attribut	Verwendung	Beschreibung	Gültige Werte
id	**erforderlich**		*Zeichenfolge*
label	**erforderlich**		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
offset	optional		*nichtnegativeGanzzahl*
separatorAfter	optional		*boolean*
separatorBefore	optional		*boolean*
showIcon	optional		*boolean*
showLabel	optional		*boolean*
systemMenu	**erforderlich**		**file** **edit** **insert** **view** **tools** **window** **help** **generate** **file.project** **file.outputs** **file.models** **edit.stream** **edit.node** **edit.outputs** **edit.models** **edit.project** **tools.repository** **tools.options** **tools.streamProperties**

XML-Darstellung

```
<xs:element name="Menu">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="systemMenu" type="STANDARD-MENU" use="required">
    <xs:enumeration value="file"/>
    <xs:enumeration value="edit"/>
    <xs:enumeration value="insert"/>
    <xs:enumeration value="view"/>
    <xs:enumeration value="tools"/>
    <xs:enumeration value="window"/>
    <xs:enumeration value="help"/>
    <xs:enumeration value="generate"/>
    <xs:enumeration value="file.project"/>
```

```
    <xs:enumeration value="file.outputs"/>
    <xs:enumeration value="file.models"/>
    <xs:enumeration value="edit.stream"/>
    <xs:enumeration value="edit.node"/>
    <xs:enumeration value="edit.outputs"/>
    <xs:enumeration value="edit.models"/>
    <xs:enumeration value="edit.project"/>
    <xs:enumeration value="tools.repository"/>
    <xs:enumeration value="tools.options"/>
    <xs:enumeration value="tools.streamProperties"/>
  </xs:attribute>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="showIcon" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="separatorBefore" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="separatorAfter" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="offset" type="xs:nonNegativeInteger" use="optional" default="0"/>
</xs:element>
```

Übergeordnete Elemente

Controls

MenuItem-Element

Definiert ein Element, das einem Menü hinzugefügt werden kann.

Tabelle 128. Attribute für MenuItem

Attribut	Verwendung	Beschreibung	Gültige Werte
action	**erforderlich**		*Zeichenfolge*
customMenu	optional		*Zeichenfolge*
offset	optional		*nichtnegativeGanzzahl*
separatorAfter	optional		*boolean*
separatorBefore	optional		*boolean*
showIcon	optional		*boolean*
showLabel	optional		*boolean*
systemMenu	optional		**file** **edit** **insert** **view** **tools** **window** **help** **generate** **file.project** **file.outputs** **file.models** **edit.stream** **edit.node** **edit.outputs** **edit.models** **edit.project** **tools.repository** **tools.options** **tools.streamProperties**

XML-Darstellung

```
<xs:element name="MenuItem">
  <xs:attribute name="action" type="xs:string" use="required"/>
  <xs:attribute name="systemMenu" type="STANDARD-MENU" use="optional">
    <xs:enumeration value="file"/>
    <xs:enumeration value="edit"/>
```

```
    <xs:enumeration value="insert"/>
    <xs:enumeration value="view"/>
    <xs:enumeration value="tools"/>
    <xs:enumeration value="window"/>
    <xs:enumeration value="help"/>
    <xs:enumeration value="generate"/>
    <xs:enumeration value="file.project"/>
    <xs:enumeration value="file.outputs"/>
    <xs:enumeration value="file.models"/>
    <xs:enumeration value="edit.stream"/>
    <xs:enumeration value="edit.node"/>
    <xs:enumeration value="edit.outputs"/>
    <xs:enumeration value="edit.models"/>
    <xs:enumeration value="edit.project"/>
    <xs:enumeration value="tools.repository"/>
    <xs:enumeration value="tools.options"/>
    <xs:enumeration value="tools.streamProperties"/>
  </xs:attribute>
  <xs:attribute name="customMenu" type="xs:string" use="optional"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="showIcon" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="separatorBefore" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="separatorAfter" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="offset" type="xs:nonNegativeInteger" use="optional" default="0"/>
</xs:element>
```

Übergeordnete Elemente

Controls

MissingValues-Element

Tabelle 129. Attribute für MissingValues

Attribut	Verwendung	Beschreibung	Gültige Werte
treatNullAsMissing	optional		*boolean*
treatWhitespaceAsMissing	optional		*boolean*

XML-Darstellung

```
<xs:element name="MissingValues" type="MISSING-VALUES" minOccurs="0">
  <xs:sequence>
    <xs:element name="Range" type="RANGE">
    </xs:element>
    <xs:element name="Values" type="FIELD-VALUES">
      <xs:sequence>
        <xs:element name="Value" type="FIELD-VALUE" minOccurs="0" maxOccurs="unbounded">
          <xs:sequence>
            <xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
            </xs:element>
          </xs:sequence>
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="treatNullAsMissing" type="xs:boolean" default="true"/>
  <xs:attribute name="treatWhitespaceAsMissing" type="xs:boolean" default="false"/>
</xs:element>
```

Übergeordnete Elemente

Field

Untergeordnete Elemente

Range, Values

Range-Element:

Tabelle 130. Attribute für Range

Attribut	Verwendung	Beschreibung	Gültige Werte
maxValue	**erforderlich**		*Zeichenfolge*
minValue	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Range" type="RANGE">
  <xs:attribute name="minValue" type="xs:string" use="required"/>
  <xs:attribute name="maxValue" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

MissingValues

Values-Element:

Tabelle 131. Attribute für Values

Attribut	Verwendung	Beschreibung	Gültige Werte
count	optional		*nichtnegativeGanzzahl*

XML-Darstellung

```
<xs:element name="Values" type="FIELD-VALUES">
  <xs:sequence>
    <xs:element name="Value" type="FIELD-VALUE" minOccurs="0" maxOccurs="unbounded">
      <xs:sequence>
        <xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Übergeordnete Elemente

MissingValues

Untergeordnete Elemente

Value

Value-Element:

Tabelle 132. Attribute für Value

Attribut	Verwendung	Beschreibung	Gültige Werte
code	**erforderlich**		*Ganzzahl*
displayLabel	optional		*Zeichenfolge*
flagValue	optional		*boolean*
value	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Value" type="FIELD-VALUE" minOccurs="0" maxOccurs="unbounded">
  <xs:sequence>
    <xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
    </xs:element>
```

```
  </xs:sequence>
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="code" type="xs:integer" use="required"/>
  <xs:attribute name="flagValue" type="xs:boolean"/>
  <xs:attribute name="displayLabel" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

Values

Untergeordnete Elemente

DisplayLabel

DisplayLabel-Element: Eine Anzeigebeschriftung für ein Feld oder einen Wert in einer angegebenen Sprache. Das DisplayLabel-Attribut kann verwendet werden, wenn es keine Beschriftung für eine bestimmte Sprache gibt.

Tabelle 133. Attribute für DisplayLabel

Attribut	Verwendung	Beschreibung	Gültige Werte
lang	optional		*NMTOKEN*
value	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="lang" type="xs:NMTOKEN" default="en"/>
</xs:element>
```

Übergeordnete Elemente

Values

ModelBuilder-Element

Tabelle 134. Attribute für ModelBuilder

Attribut	Verwendung	Beschreibung	Gültige Werte
allowNoInputs	optional		*boolean*
allowNoOutputs	optional		*boolean*
miningFunctions	**erforderlich**		*beliebig*
nullifyBlanks	optional		*boolean*

XML-Darstellung

```
<xs:element name="ModelBuilder">
  <xs:sequence>
    <xs:element name="Algorithm">
    </xs:element>
    <xs:element name="ModelingFields" minOccurs="0">
      <xs:sequence>
        <xs:element name="InputFields" minOccurs="0">
        </xs:element>
        <xs:element name="OutputFields" minOccurs="0">
        </xs:element>
      </xs:sequence>
    </xs:element>
    <xs:element name="ModelGeneration">
    </xs:element>
    <xs:element name="ModelFields">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:group ref="DATA-MODEL-EXPRESSION">
```

```
          <xs:choice>
            <xs:element ref="ForEach"/>
            <xs:element ref="AddField"/>
            <xs:element ref="ChangeField"/>
            <xs:element ref="RemoveField"/>
          </xs:choice>
        </xs:group>
      </xs:sequence>
    </xs:element>
    <xs:element name="ModelEvaluation" minOccurs="0">
      <xs:sequence>
        <xs:element name="RawPropensity" minOccurs="0">
        </xs:element>
        <xs:element name="AdjustedPropensity" minOccurs="0">
        </xs:element>
        <xs:element name="VariableImportance" minOccurs="0">
        </xs:element>
      </xs:sequence>
    </xs:element>
    <xs:element name="AutoModeling" minOccurs="0">
      <xs:sequence>
        <xs:element name="SimpleSettings">
          <xs:sequence>
            <xs:element ref="PropertyGroup" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:element>
        <xs:element name="ExpertSettings" minOccurs="0">
          <xs:sequence>
            <xs:element ref="Condition" minOccurs="0"/>
            <xs:element ref="PropertyGroup" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:element>
        <xs:element name="PropertyMap" minOccurs="0">
          <xs:sequence>
            <xs:element name="PropertyMapping" maxOccurs="unbounded">
            </xs:element>
          </xs:sequence>
        </xs:element>
        <xs:element ref="Constraint" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="miningFunctions" use="required"/>
  <xs:attribute name="allowNoInputs" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="allowNoOutputs" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="nullifyBlanks" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Übergeordnete Elemente

Node

Untergeordnete Elemente

Algorithm, AutoModeling, ModelEvaluation, ModelFields, ModelGeneration, ModelingFields

Algorithm-Element:

Tabelle 135. Attribute für Algorithm

Attribut	Verwendung	Beschreibung	Gültige Werte
label	**erforderlich**		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
value	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Algorithm">
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

ModelBuilder

ModelingFields-Element:

Tabelle 136. Attribute für ModelingFields

Attribut	Verwendung	Beschreibung	Gültige Werte
controlsId	**erforderlich**		*Zeichenfolge*
ignoreBOTH	optional		*boolean*

XML-Darstellung

```
<xs:element name="ModelingFields" minOccurs="0">
  <xs:attribute name="controlsId" type="xs:string" use="required"/>
  <xs:sequence>
    <xs:element name="InputFields" minOccurs="0">
    </xs:element>
    <xs:element name="OutputFields" minOccurs="0">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="ignoreBOTH" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Übergeordnete Elemente

ModelBuilder

Untergeordnete Elemente

InputFields, OutputFields

InputFields-Element:

Tabelle 137. Attribute für InputFields

Attribut	Verwendung	Beschreibung	Gültige Werte
label	**erforderlich**		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
multiple	**erforderlich**		*boolean*
onlyDatetime	optional		*boolean*
onlyDiscrete	optional		*boolean*
onlyNumeric	optional		*boolean*
onlyRanges	optional		*boolean*
onlySymbolic	optional		*boolean*
property	**erforderlich**		*Zeichenfolge*
storage	optional		*Zeichenfolge*
types	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="InputFields" minOccurs="0">
  <xs:attribute name="storage" type="xs:string" use="optional"/>
  <xs:attribute name="onlyNumeric" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlySymbolic" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDatetime" type="xs:boolean" use="optional"/>
  <xs:attribute name="types" type="xs:string" use="optional"/>
  <xs:attribute name="onlyRanges" type="xs:boolean" use="optional"/>
```

```
  <xs:attribute name="onlyDiscrete" type="xs:boolean" use="optional"/>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="multiple" type="xs:boolean" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

ModelingFields

OutputFields-Element:

Tabelle 138. Attribute für OutputFields

Attribut	Verwendung	Beschreibung	Gültige Werte
label	**erforderlich**		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
multiple	**erforderlich**		*boolean*
onlyDatetime	optional		*boolean*
onlyDiscrete	optional		*boolean*
onlyNumeric	optional		*boolean*
onlyRanges	optional		*boolean*
onlySymbolic	optional		*boolean*
property	**erforderlich**		*Zeichenfolge*
storage	optional		*Zeichenfolge*
types	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="OutputFields" minOccurs="0">
  <xs:attribute name="storage" type="xs:string" use="optional"/>
  <xs:attribute name="onlyNumeric" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlySymbolic" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDatetime" type="xs:boolean" use="optional"/>
  <xs:attribute name="types" type="xs:string" use="optional"/>
  <xs:attribute name="onlyRanges" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDiscrete" type="xs:boolean" use="optional"/>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="multiple" type="xs:boolean" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

ModelingFields

ModelGeneration-Element:

Tabelle 139. Attribute für ModelGeneration

Attribut	Verwendung	Beschreibung	Gültige Werte
controlsId	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="ModelGeneration">
  <xs:attribute name="controlsId" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

ModelBuilder

ModelFields-Element:

XML-Darstellung

```
<xs:element name="ModelFields">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:group ref="DATA-MODEL-EXPRESSION">
      <xs:choice>
        <xs:element ref="ForEach"/>
        <xs:element ref="AddField"/>
        <xs:element ref="ChangeField"/>
        <xs:element ref="RemoveField"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

ModelBuilder

Untergeordnete Elemente

AddField, ChangeField, ForEach, RemoveField

ModelEvaluation-Element:

Tabelle 140. Attribute für ModelEvaluation

Attribut	Verwendung	Beschreibung	Gültige Werte
container	**erforderlich**		*beliebig*
controlsId	**erforderlich**		*Zeichenfolge*
outputContainer	optional		*beliebig*

XML-Darstellung

```
<xs:element name="ModelEvaluation" minOccurs="0">
  <xs:attribute name="controlsId" type="xs:string" use="required"/>
  <xs:sequence>
    <xs:element name="RawPropensity" minOccurs="0">
    </xs:element>
    <xs:element name="AdjustedPropensity" minOccurs="0">
    </xs:element>
    <xs:element name="VariableImportance" minOccurs="0">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="container" use="required"/>
  <xs:attribute name="outputContainer" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

ModelBuilder

Untergeordnete Elemente

AdjustedPropensity, RawPropensity, VariableImportance

RawPropensity-Element:

Tabelle 141. Attribute für RawPropensity

Attribut	Verwendung	Beschreibung	Gültige Werte
availabilityProperty	optional		*Zeichenfolge*
defaultValue	optional		*boolean*
property	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="RawPropensity" minOccurs="0">
  <xs:attribute name="property" type="xs:string" use="optional"/>
  <xs:attribute name="availabilityProperty" type="xs:string" use="optional"/>
  <xs:attribute name="defaultValue" type="xs:boolean" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

ModelEvaluation

AdjustedPropensity-Element:

Tabelle 142. Attribute für AdjustedPropensity

Attribut	Verwendung	Beschreibung	Gültige Werte
availabilityProperty	optional		*Zeichenfolge*
defaultValue	optional		*boolean*
property	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="AdjustedPropensity" minOccurs="0">
  <xs:attribute name="property" type="xs:string" use="optional"/>
  <xs:attribute name="availabilityProperty" type="xs:string" use="optional"/>
  <xs:attribute name="defaultValue" type="xs:boolean" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

ModelEvaluation

VariableImportance-Element:

Tabelle 143. Attribute für VariableImportance

Attribut	Verwendung	Beschreibung	Gültige Werte
availabilityProperty	optional		*Zeichenfolge*
defaultValue	optional		*boolean*
property	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="VariableImportance" minOccurs="0">
  <xs:attribute name="property" type="xs:string" use="optional"/>
  <xs:attribute name="availabilityProperty" type="xs:string" use="optional"/>
  <xs:attribute name="defaultValue" type="xs:boolean" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

ModelEvaluation

AutoModeling-Element:

Tabelle 144. Attribute für AutoModeling

Attribut	Verwendung	Beschreibung	Gültige Werte
enabledByDefault	optional		*beliebig*

XML-Darstellung

```
<xs:element name="AutoModeling" minOccurs="0">
  <xs:sequence>
    <xs:element name="SimpleSettings">
      <xs:sequence>
        <xs:element ref="PropertyGroup" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:element>
    <xs:element name="ExpertSettings" minOccurs="0">
      <xs:sequence>
        <xs:element ref="Condition" minOccurs="0"/>
        <xs:element ref="PropertyGroup" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:element>
    <xs:element name="PropertyMap" minOccurs="0">
      <xs:sequence>
        <xs:element name="PropertyMapping" maxOccurs="unbounded">
        </xs:element>
      </xs:sequence>
    </xs:element>
    <xs:element ref="Constraint" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="enabledByDefault" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

ModelBuilder

Untergeordnete Elemente

Constraint, ExpertSettings, PropertyMap, SimpleSettings

SimpleSettings-Element:

XML-Darstellung

```
<xs:element name="SimpleSettings">
  <xs:sequence>
    <xs:element ref="PropertyGroup" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

AutoModeling

Untergeordnete Elemente

PropertyGroup

ExpertSettings-Element:

XML-Darstellung

```
<xs:element name="ExpertSettings" minOccurs="0">
  <xs:sequence>
    <xs:element ref="Condition" minOccurs="0"/>
    <xs:element ref="PropertyGroup" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

AutoModeling

Untergeordnete Elemente

Condition, PropertyGroup

PropertyMap-Element:

XML-Darstellung

```
<xs:element name="PropertyMap" minOccurs="0">
  <xs:sequence>
    <xs:element name="PropertyMapping" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

AutoModeling

Untergeordnete Elemente

PropertyMapping

PropertyMapping-Element:

Tabelle 145. Attribute für PropertyMapping

Attribut	Verwendung	Beschreibung	Gültige Werte
property	**erforderlich**		*Zeichenfolge*
systemProperty	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="PropertyMapping" maxOccurs="unbounded">
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="systemProperty" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

PropertyMap

ModelOutput-Element

Tabelle 146. Attribute für ModelOutput

Attribut	Verwendung	Beschreibung	Gültige Werte
delegate	optional		*Zeichenfolge*
deprecatedScriptNames	optional		*Zeichenfolge*
helpLink	optional		*Zeichenfolge*

Tabelle 146. Attribute für ModelOutput (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
id	**erforderlich**		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
scriptName	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="ModelOutput">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="Properties"/>
      <xs:element name="Containers" minOccurs="0">
        <xs:sequence maxOccurs="unbounded">
          <xs:element ref="Container"/>
        </xs:sequence>
      </xs:element>
      <xs:element ref="UserInterface"/>
      <xs:element ref="Constructors" minOccurs="0"/>
      <xs:element ref="ModelProvider" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="scriptName" type="xs:string" use="optional"/>
  <xs:attribute name="deprecatedScriptNames" type="xs:string" use="optional"/>
  <xs:attribute name="delegate" type="xs:string" use="optional"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="helpLink" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Extension

Untergeordnete Elemente

Constructors, Containers, ModelProvider, Properties, UserInterface

Zugehörige Elemente

DocumentOutput, InteractiveDocumentBuilder, InteractiveModelBuilder, Node

Containers-Element:

XML-Darstellung

```
<xs:element name="Containers" minOccurs="0">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Container"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

Node

Untergeordnete Elemente

Container

ModelProvider-Element

Tabelle 147. Attribute für ModelProvider

Attribut	Verwendung	Beschreibung	Gültige Werte
container	**erforderlich**		*Zeichenfolge*
isPMML	optional		*boolean*

XML-Darstellung

```
<xs:element name="ModelProvider">
  <xs:attribute name="container" type="xs:string" use="required"/>
  <xs:attribute name="isPMML" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Übergeordnete Elemente

DocumentOutput, InteractiveDocumentBuilder, InteractiveModelBuilder, ModelOutput, Node

ModelType-Element

Definiert einen neuen Modelltyp.

Tabelle 148. Attribute für ModelType

Attribut	Verwendung	Beschreibung	Gültige Werte
format	**erforderlich**		**utf8** **binary**
id	**erforderlich**		*Zeichenfolge*
inputDirection	optional		*Zeichenfolge*
outputDirection	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="ModelType">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="format" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="utf8"/>
        <xs:enumeration value="binary"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="inputDirection" type="xs:string" use="optional" default="[in]"/>
  <xs:attribute name="outputDirection" type="xs:string" use="optional" default="[out]"/>
</xs:element>
```

Übergeordnete Elemente

ContainerTypes

Zugehörige Elemente

DocumentType

ModelViewerPanel-Element

Tabelle 149. Attribute für ModelViewerPanel

Attribut	Verwendung	Beschreibung	Gültige Werte
container	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="ModelViewerPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="container" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Tab

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

ActionButton, ComboBoxControl, ExtensionObjectPanel, FieldAllocationList, SelectorPanel, StaticText, SystemControls, TabbedPanel, TextBrowserPanel

Module-Element

Tabelle 150. Attribute für Module

Attribut	Verwendung	Beschreibung	Gültige Werte
libraryId	optional		*Zeichenfolge*
name	optional		*Zeichenfolge*
systemModule	optional		**SmartScore**

XML-Darstellung

```
<xs:element name="Module">
  <xs:sequence>
    <xs:element ref="InputFiles"/>
    <xs:element ref="OutputFiles"/>
    <xs:element ref="StatusCodes" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="systemModule" use="optional" default="SmartScore">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="SmartScore"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="libraryId" type="xs:string" use="optional"/>
  <xs:attribute name="name" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Execution

Untergeordnete Elemente

InputFiles, OutputFiles, StatusCodes

MultiFieldAllocationControl-Element

Definiert ein Steuerelement, mit dem Felder im Steuerelement für Feldzuordnungslisten ausgewählt werden können, die vom Zuordnungsfunktionsattribut angegeben wird.

Tabelle 151. Attribute für MultiFieldAllocationControl

Attribut	Verwendung	Beschreibung	Gültige Werte
allocator	**erforderlich**		*Zeichenfolge*
buttonColumn	**erforderlich**		*Ganzzahl*
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
onlyDatetime	optional		*boolean*
onlyDiscrete	optional		*boolean*
onlyNumeric	optional		*boolean*
onlyRanges	optional		*boolean*
onlySymbolic	optional		*boolean*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*
storage	optional		*Zeichenfolge*
types	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="MultiFieldAllocationControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="allocator" type="xs:string" use="required"/>
  <xs:attribute name="buttonColumn" type="xs:integer" use="required"/>
  <xs:attribute name="storage" type="xs:string" use="optional"/>
  <xs:attribute name="onlyNumeric" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlySymbolic" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDatetime" type="xs:boolean" use="optional"/>
  <xs:attribute name="types" type="xs:string" use="optional"/>
  <xs:attribute name="onlyRanges" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDiscrete" type="xs:boolean" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DB-ConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

MultiFieldChooserControl-Element

Definiert ein Steuerelement, mit dem Felder im aktuellen Datenmodell ausgewählt werden können.

Tabelle 152. Attribute für MultiFieldChooserControl

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
onlyDatetime	optional		*boolean*
onlyDiscrete	optional		*boolean*
onlyNumeric	optional		*boolean*
onlyRanges	optional		*boolean*
onlySymbolic	optional		*boolean*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*
storage	optional		*Zeichenfolge*
types	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="MultiFieldChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
```

```
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="storage" type="xs:string" use="optional"/>
  <xs:attribute name="onlyNumeric" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlySymbolic" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDatetime" type="xs:boolean" use="optional"/>
  <xs:attribute name="types" type="xs:string" use="optional"/>
  <xs:attribute name="onlyRanges" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDiscrete" type="xs:boolean" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

MultiItemChooserControl-Element

Definiert ein Steuerelement, mit dem mehrere Werte aus einer Auswahl ausgewählt werden können.

Tabelle 153. Attribute für MultiItemChooserControl

Attribut	Verwendung	Beschreibung	Gültige Werte
catalog	**erforderlich**		*Zeichenfolge*
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="MultiItemChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
```

```
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="catalog" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

SingleItemChooserControl

Node-Element

Tabelle 154. Attribute für Node

Attribut	Verwendung	Beschreibung	Gültige Werte
delegate	optional		*Zeichenfolge*
deprecatedScriptNames	optional		*Zeichenfolge*
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
helpLink	optional		*Zeichenfolge*
id	**erforderlich**		*Zeichenfolge*
label	**erforderlich**		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
palette	optional		**import** **fieldOp** **recordOp** **modeling** **dbModeling** **graph** **output** **export** **modeling.classification** **modeling.association** **modeling.segmentation** **modeling.auto**
relativePosition	optional		*Zeichenfolge*
relativeTo	optional		*Zeichenfolge*
scriptName	optional		*Zeichenfolge*
type	**erforderlich**		**dataReader** **dataWriter** **dataTransformer** **modelApplier** **modelBuilder** **documentBuilder**

XML-Darstellung

```
<xs:element name="Node">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="Properties"/>
      <xs:element name="Containers" minOccurs="0">
        <xs:sequence maxOccurs="unbounded">
          <xs:element ref="Container"/>
        </xs:sequence>
      </xs:element>
      <xs:element ref="UserInterface"/>
      <xs:element ref="Constructors" minOccurs="0"/>
      <xs:element ref="ModelProvider" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="scriptName" type="xs:string" use="optional"/>
  <xs:attribute name="deprecatedScriptNames" type="xs:string" use="optional"/>
  <xs:attribute name="delegate" type="xs:string" use="optional"/>
  <xs:sequence>
    <xs:element ref="ModelBuilder" minOccurs="0"/>
    <xs:element ref="DocumentBuilder" minOccurs="0"/>
    <xs:element ref="Execution"/>
    <xs:element ref="OutputDataModel" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="type" type="NODE-TYPE" use="required">
    <xs:enumeration value="dataReader"/>
    <xs:enumeration value="dataWriter"/>
    <xs:enumeration value="dataTransformer"/>
    <xs:enumeration value="modelApplier"/>
    <xs:enumeration value="modelBuilder"/>
    <xs:enumeration value="documentBuilder"/>
  </xs:attribute>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="palette" type="SYSTEM-PALETTE" use="optional">
    <xs:enumeration value="import"/>
    <xs:enumeration value="fieldOp"/>
    <xs:enumeration value="recordOp"/>
    <xs:enumeration value="modeling"/>
    <xs:enumeration value="dbModeling"/>
    <xs:enumeration value="graph"/>
    <xs:enumeration value="output"/>
    <xs:enumeration value="export"/>
    <xs:enumeration value="modeling.classification"/>
    <xs:enumeration value="modeling.association"/>
    <xs:enumeration value="modeling.segmentation"/>
    <xs:enumeration value="modeling.auto"/>
  </xs:attribute>
  <xs:attribute name="helpLink" type="xs:string" use="optional"/>
  <xs:attribute name="relativeTo" type="xs:string" use="optional"/>
  <xs:attribute name="relativePosition" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Extension

Untergeordnete Elemente

Constructors, Containers, DocumentBuilder, Execution, ModelBuilder, ModelProvider, OutputDataModel, Properties, UserInterface

Zugehörige Elemente

DocumentOutput, InteractiveDocumentBuilder, InteractiveModelBuilder, ModelOutput

Containers-Element:

XML-Darstellung

```
<xs:element name="Containers" minOccurs="0">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Container"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

Node

Untergeordnete Elemente

Container

Not-Element

XML-Darstellung

```
<xs:element name="Not">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

And, Command, Constraint, CreateDocument, CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModel, CreateModelApplier, CreateModelOutput, Enabled, Not, Option, Or, Run, Visible

Untergeordnete Elemente

And, Condition, Not, Or

NumberFormat-Element

Definiert Formatinformationen für ein numerisches Feld.

Tabelle 155. Attribute für NumberFormat

Attribut	Verwendung	Beschreibung	Gültige Werte
decimalPlaces	**erforderlich**		*nichtnegativeGanzzahl*
decimalSymbol	**erforderlich**		**period** **comma**
formatType	**erforderlich**		**standard** **scientific** **currency**
groupingSymbol	**erforderlich**		**none** **period** **comma** **space**
name	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="NumberFormat" type="NUMBER-FORMAT-DECLARATION">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="formatType" type="NUMBER-FORMAT-TYPE" use="required">
    <xs:enumeration value="standard"/>
    <xs:enumeration value="scientific"/>
    <xs:enumeration value="currency"/>
  </xs:attribute>
  <xs:attribute name="decimalPlaces" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="decimalSymbol" type="DECIMAL-SYMBOL" use="required">
    <xs:enumeration value="period"/>
    <xs:enumeration value="comma"/>
  </xs:attribute>
  <xs:attribute name="groupingSymbol" type="NUMBER-GROUPING-SYMBOL" use="required">
    <xs:enumeration value="none"/>
    <xs:enumeration value="period"/>
    <xs:enumeration value="comma"/>
    <xs:enumeration value="space"/>
  </xs:attribute>
</xs:element>
```

NumericInfo-Element

Tabelle 156. Attribute für NumericInfo

Attribut	Verwendung	Beschreibung	Gültige Werte
mean	optional		*double*
standardDeviation	optional		*double*

XML-Darstellung

```
<xs:element name="NumericInfo">
  <xs:attribute name="mean" type="xs:double"/>
  <xs:attribute name="standardDeviation" type="xs:double"/>
</xs:element>
```

Übergeordnete Elemente

AddField, ChangeField, Field

Option-Element

Tabelle 157. Attribute für Option

Attribut	Verwendung	Beschreibung	Gültige Werte
ifProperty	optional		*Zeichenfolge*
unlessProperty	optional		*Zeichenfolge*
value	**erforderlich**		

XML-Darstellung

```
<xs:element name="Option">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
  <xs:attribute name="value" type="EVALUATED-STRING" use="required"/>
  <xs:attribute name="ifProperty" type="xs:string" use="optional"/>
  <xs:attribute name="unlessProperty" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Run

Untergeordnete Elemente

And, Condition, Not, Or

OptionCode-Element

Tabelle 158. Attribute für OptionCode

Attribut	Verwendung	Beschreibung	Gültige Werte
code	optional		*lang*
description	optional		*Zeichenfolge*
type	optional		**mandatory** **optional**

XML-Darstellung

```
<xs:element name="OptionCode">
  <xs:attribute name="code" type="xs:long"/>
  <xs:attribute name="type" type="LicenseType">
    <xs:enumeration value="mandatory"/>
    <xs:enumeration value="optional"/>
  </xs:attribute>
  <xs:attribute name="description" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

License

Or-Element

XML-Darstellung

```
<xs:element name="Or">
  <xs:sequence minOccurs="2" maxOccurs="unbounded">
    <xs:group ref="CONDITION-EXPRESSION">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

And, Command, Constraint, CreateDocument, CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModel, CreateModelApplier, CreateModelOutput, Enabled, Not, Option, Or, Run, Visible

Untergeordnete Elemente

And, Condition, Not, Or

OutputDataModel-Element

Tabelle 159. Attribute für OutputDataModel

Attribut	Verwendung	Beschreibung	Gültige Werte
libraryId	optional		*Zeichenfolge*
method	optional		**xml** **delegate** **dataModelProvider** **sharedLibrary**
mode	optional		**fixed** **modify** **extend** **replace**
providerClass	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="OutputDataModel">
  <xs:attribute name="mode" use="optional" default="fixed">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="fixed"/>
        <xs:enumeration value="modify"/>
        <xs:enumeration value="extend"/>
        <xs:enumeration value="replace"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="method" use="optional" default="xml">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="xml"/>
        <xs:enumeration value="delegate"/>
        <xs:enumeration value="dataModelProvider"/>
        <xs:enumeration value="sharedLibrary"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="providerClass" type="xs:string" use="optional"/>
  <xs:attribute name="libraryId" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Node

OutputFiles-Element

XML-Darstellung

```
<xs:element name="OutputFiles">
  <xs:group ref="RUNTIME-FILES">
    <xs:sequence>
      <xs:element ref="DataFile"/>
      <xs:element ref="ContainerFile" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:group>
</xs:element>
```

Übergeordnete Elemente

Execution, Module

Untergeordnete Elemente

ContainerFile, DataFile

Palette-Element

Tabelle 160. Attribute für Palette

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
id	**erforderlich**		*Zeichenfolge*
label	**erforderlich**		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
position	optional		**atStart** **atEnd** **before** **after**
systemPalette	optional		**import** **fieldOp** **recordOp** **modeling** **dbModeling** **graph** **output** **export** **modeling.classification** **modeling.association** **modeling.segmentation** **modeling.auto**

XML-Darstellung

```
<xs:element name="Palette">
  <xs:sequence>
    <xs:element ref="Icon"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="position" use="optional">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="atStart"/>
        <xs:enumeration value="atEnd"/>
        <xs:enumeration value="before"/>
        <xs:enumeration value="after"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="systemPalette" type="SYSTEM-PALETTE" use="optional">
    <xs:enumeration value="import"/>
    <xs:enumeration value="fieldOp"/>
    <xs:enumeration value="recordOp"/>
    <xs:enumeration value="modeling"/>
    <xs:enumeration value="dbModeling"/>
    <xs:enumeration value="graph"/>
    <xs:enumeration value="output"/>
    <xs:enumeration value="export"/>
    <xs:enumeration value="modeling.classification"/>
    <xs:enumeration value="modeling.association"/>
```

```
    <xs:enumeration value="modeling.segmentation"/>
    <xs:enumeration value="modeling.auto"/>
  </xs:attribute>
</xs:element>
```

Untergeordnete Elemente

Icon

Parameters-Element

Konfigurationsparameter vom Erweiterungsknoten.

Tabelle 161. Attribute für Parameters

Attribut	Verwendung	Beschreibung	Gültige Werte
count	optional		*nichtnegativeGanzzahl*

XML-Darstellung

```
<xs:element name="Parameters" type="PARAMETERS">
  <xs:sequence>
    <xs:element name="Parameter" type="PARAMETER" minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
        <xs:choice>
          <xs:element ref="MapValue"/>
          <xs:element ref="StructuredValue"/>
          <xs:element ref="ListValue"/>
          <xs:element ref="Value"/>
          <xs:element ref="DatabaseConnectionValue"/>
        </xs:choice>
      </xs:group>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Untergeordnete Elemente

Parameter

Parameter-Element: Ein Parameter hat einen Namen und einen Wert. Ein einfacher Wert kann mit dem Wertattribut ausgedrückt werden; ein zusammengesetzter Wert verwendet das von ParameterContent beschriebene Inhaltsmodell. Diese Kombination aus Attribut und Inhalt wird für verschachtelte Werte wiederholt.

Tabelle 162. Attribute für Parameter

Attribut	Verwendung	Beschreibung	Gültige Werte
name	**erforderlich**		*Zeichenfolge*
value	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Parameter" type="PARAMETER" minOccurs="0" maxOccurs="unbounded">
  <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
    <xs:choice>
      <xs:element ref="MapValue"/>
      <xs:element ref="StructuredValue"/>
      <xs:element ref="ListValue"/>
      <xs:element ref="Value"/>
      <xs:element ref="DatabaseConnectionValue"/>
    </xs:choice>
  </xs:group>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="value" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

Parameters

Untergeordnete Elemente

DatabaseConnectionValue, ListValue, MapValue, StructuredValue, Value

PasswordBoxControl-Element

Definiert ein einzeiliges Steuerelement für Kennwörter, mit dem Zeichenfolgewerte geändert werden können, bei denen der Wert nicht angezeigt werden soll.

Tabelle 163. Attribute für PasswordBoxControl

Attribut	Verwendung	Beschreibung	Gültige Werte
columns	optional		*positiveGanzzahl*
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="PasswordBoxControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="columns" type="xs:positiveInteger" use="optional" default="20"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

Properties-Element

XML-Darstellung

```
<xs:element name="Properties">
  <xs:sequence>
    <xs:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

DocumentOutput, Execution, InteractiveDocumentBuilder, InteractiveModelBuilder, ModelOutput, Node

Untergeordnete Elemente

Property

PropertiesPanel-Element

Definiert eine Eigenschaftsanzeige der höchste Ebene.

Tabelle 164. Attribute für PropertiesPanel

Attribut	Verwendung	Beschreibung	Gültige Werte
id	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="PropertiesPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:sequence maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="CheckBoxControl"/>
      <xs:element ref="TextBoxControl"/>
      <xs:element ref="PasswordBoxControl"/>
      <xs:element ref="TextAreaControl"/>
      <xs:element ref="RadioButtonGroupControl"/>
      <xs:element ref="CheckBoxGroupControl"/>
      <xs:element ref="ComboBoxControl"/>
      <xs:element ref="SpinnerControl"/>
      <xs:element ref="ServerFileChooserControl"/>
      <xs:element ref="ServerDirectoryChooserControl"/>
      <xs:element ref="ClientFileChooserControl"/>
      <xs:element ref="ClientDirectoryChooserControl"/>
      <xs:element ref="TableControl"/>
      <xs:element ref="SingleFieldChooserControl"/>
      <xs:element ref="SingleFieldAllocationControl"/>
      <xs:element ref="MultiFieldChooserControl"/>
      <xs:element ref="MultiFieldAllocationControl"/>
      <xs:element ref="SingleFieldValueChooserControl"/>
      <xs:element ref="SingleItemChooserControl"/>
      <xs:element ref="MultiItemChooserControl"/>
      <xs:element ref="DBConnectionChooserControl"/>
```

```
      <xs:element ref="DBTableChooserControl"/>
      <xs:element ref="PropertyControl"/>
      <xs:element ref="StaticText"/>
      <xs:element ref="SystemControls"/>
      <xs:element ref="ActionButton"/>
      <xs:element ref="FieldAllocationList"/>
      <xs:element ref="PropertiesPanel"/>
      <xs:element ref="PropertiesSubPanel"/>
      <xs:element ref="SelectorPanel"/>
      <xs:element ref="ExtensionObjectPanel"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="optional"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel, Tab

Untergeordnete Elemente

ActionButton, CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, ComboBoxControl, DBConnectionChooserControl, DBTableChooserControl, Enabled, ExtensionObjectPanel, FieldAllocationList, Layout, MultiFieldAllocationControl, MultiFieldChooserControl, MultiItemChooserControl, PasswordBoxControl, PropertiesPanel, PropertiesSubPanel, PropertyControl, RadioButtonGroupControl, SelectorPanel, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SingleItemChooserControl, SpinnerControl, StaticText, SystemControls, TableControl, TextAreaControl, TextBoxControl, Visible

Zugehörige Elemente

PropertiesSubPanel

PropertiesSubPanel-Element

Definiert ein Unterfenster, das zum Gruppieren zusammengehöriger UI-Komponenten und Steuerelemente verwendet werden kann.

Tabelle 165. Attribute für PropertiesSubPanel

Attribut	Verwendung	Beschreibung	Gültige Werte
buttonLabel	optional		*Zeichenfolge*
buttonLabelKey	optional		*Zeichenfolge*
dialogTitle	optional		*Zeichenfolge*
dialogTitleKey	optional		*Zeichenfolge*
helpLink	optional		*Zeichenfolge*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="PropertiesSubPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:sequence maxOccurs="unbounded">
    <xs:choice>
```

```
      <xs:element ref="CheckBoxControl"/>
      <xs:element ref="TextBoxControl"/>
      <xs:element ref="PasswordBoxControl"/>
      <xs:element ref="TextAreaControl"/>
      <xs:element ref="RadioButtonGroupControl"/>
      <xs:element ref="CheckBoxGroupControl"/>
      <xs:element ref="ComboBoxControl"/>
      <xs:element ref="SpinnerControl"/>
      <xs:element ref="ServerFileChooserControl"/>
      <xs:element ref="ServerDirectoryChooserControl"/>
      <xs:element ref="ClientFileChooserControl"/>
      <xs:element ref="ClientDirectoryChooserControl"/>
      <xs:element ref="TableControl"/>
      <xs:element ref="SingleFieldChooserControl"/>
      <xs:element ref="SingleFieldAllocationControl"/>
      <xs:element ref="MultiFieldChooserControl"/>
      <xs:element ref="MultiFieldAllocationControl"/>
      <xs:element ref="SingleFieldValueChooserControl"/>
      <xs:element ref="SingleItemChooserControl"/>
      <xs:element ref="MultiItemChooserControl"/>
      <xs:element ref="DBConnectionChooserControl"/>
      <xs:element ref="DBTableChooserControl"/>
      <xs:element ref="PropertyControl"/>
      <xs:element ref="StaticText"/>
      <xs:element ref="SystemControls"/>
      <xs:element ref="ActionButton"/>
      <xs:element ref="FieldAllocationList"/>
      <xs:element ref="PropertiesPanel"/>
      <xs:element ref="PropertiesSubPanel"/>
      <xs:element ref="SelectorPanel"/>
      <xs:element ref="ExtensionObjectPanel"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="buttonLabel" type="xs:string" use="optional"/>
  <xs:attribute name="buttonLabelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="dialogTitle" type="xs:string" use="optional"/>
  <xs:attribute name="dialogTitleKey" type="xs:string" use="optional"/>
  <xs:attribute name="helpLink" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

ActionButton, CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, ComboBoxControl, DBConnectionChooserControl, DBTableChooserControl, Enabled, ExtensionObjectPanel, FieldAllocationList, Layout, MultiFieldAllocationControl, MultiFieldChooserControl, MultiItemChooserControl, PasswordBoxControl, PropertiesPanel, PropertiesSubPanel, PropertyControl, RadioButtonGroupControl, SelectorPanel, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SingleItemChooserControl, SpinnerControl, StaticText, SystemControls, TableControl, TextAreaControl, TextBoxControl, Visible

Zugehörige Elemente

PropertiesPanel

Property-Element

Tabelle 166. Attribute für Property

Attribut	Verwendung	Beschreibung	Gültige Werte
defaultValue	optional		
defaultValueKey	optional		*Zeichenfolge*
deprecatedScriptNames	optional		*Zeichenfolge*

Tabelle 166. Attribute für Property (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
isList	optional		*boolean*
label	optional		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
max	optional		*Zeichenfolge*
min	optional		*Zeichenfolge*
name	**erforderlich**		*Zeichenfolge*
scriptName	optional		*Zeichenfolge*
type	optional		*Zeichenfolge*
valueType	optional		**string** **encryptedString** **fieldName** **integer** **double** **boolean** **date** **enum** **structure** **databaseConnection**

XML-Darstellung

```
<xs:element name="Property">
  <xs:choice>
    <xs:element ref="DefaultValue" minOccurs="0"/>
  </xs:choice>
  <xs:attribute name="valueType" type="PROPERTY-VALUE-TYPE">
    <xs:enumeration value="string"/>
    <xs:enumeration value="encryptedString"/>
    <xs:enumeration value="fieldName"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="double"/>
    <xs:enumeration value="boolean"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="enum"/>
    <xs:enumeration value="structure"/>
    <xs:enumeration value="databaseConnection"/>
  </xs:attribute>
  <xs:attribute name="isList" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="min" type="xs:string" use="optional"/>
  <xs:attribute name="max" type="xs:string" use="optional"/>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="scriptName" type="xs:string" use="optional"/>
  <xs:attribute name="deprecatedScriptNames" type="xs:string" use="optional"/>
  <xs:attribute name="type" type="xs:string" use="optional"/>
  <xs:attribute name="defaultValue" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="defaultValueKey" type="xs:string" use="optional"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Properties, PropertySets

Untergeordnete Elemente

DefaultValue

Zugehörige Elemente

PropertyType

PropertyControl-Element

Definiert ein Steuerelement für benutzerdefinierte Eigenschaften. Die durch das Attribut controlClass angegebene Klasse muss die PropertyControl-Schnittstelle implementieren.

Tabelle 167. Attribute für PropertyControl

Attribut	Verwendung	Beschreibung	Gültige Werte
controlClass	**erforderlich**		*Zeichenfolge*
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="PropertyControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="controlClass" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

PropertyGroup-Element

Tabelle 168. Attribute für PropertyGroup

Attribut	Verwendung	Beschreibung	Gültige Werte
label	optional		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
properties	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="PropertyGroup">
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="properties" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

ExpertSettings, SimpleSettings

PropertySets-Element

XML-Darstellung

```
<xs:element name="PropertySets">
  <xs:sequence>
    <xs:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

CommonObjects

Untergeordnete Elemente

Property

PropertyType-Element

Tabelle 169. Attribute für PropertyType

Attribut	Verwendung	Beschreibung	Gültige Werte
id	**erforderlich**		*Zeichenfolge*
isKeyed	optional		*boolean*
isList	optional		*boolean*
max	optional		*Zeichenfolge*
min	optional		*Zeichenfolge*

Tabelle 169. Attribute für PropertyType (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
valueType	optional		**string** **encryptedString** **fieldName** **integer** **double** **boolean** **date** **enum** **structure** **databaseConnection**

XML-Darstellung

```
<xs:element name="PropertyType">
  <xs:choice>
    <xs:element ref="DefaultValue" minOccurs="0"/>
  </xs:choice>
  <xs:attribute name="valueType" type="PROPERTY-VALUE-TYPE">
    <xs:enumeration value="string"/>
    <xs:enumeration value="encryptedString"/>
    <xs:enumeration value="fieldName"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="double"/>
    <xs:enumeration value="boolean"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="enum"/>
    <xs:enumeration value="structure"/>
    <xs:enumeration value="databaseConnection"/>
  </xs:attribute>
  <xs:attribute name="isList" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="min" type="xs:string" use="optional"/>
  <xs:attribute name="max" type="xs:string" use="optional"/>
  <xs:choice>
    <xs:element ref="Enumeration" minOccurs="0"/>
    <xs:element ref="Structure" minOccurs="0"/>
  </xs:choice>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="isKeyed" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Übergeordnete Elemente

PropertyTypes

Untergeordnete Elemente

DefaultValue, Enumeration, Structure

Zugehörige Elemente

Property

PropertyTypes-Element

XML-Darstellung

```
<xs:element name="PropertyTypes">
  <xs:sequence>
    <xs:element ref="PropertyType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

CommonObjects

Untergeordnete Elemente

PropertyType

RadioButtonGroupControl-Element

Definiert eine Gruppe von Steuerelementen für Optionsfelder, mit denen ein boolescher Wert (wahr/falsch) oder ein Wert aus einem aufgezählten Typ angegeben werden kann.

Tabelle 170. Attribute für RadioButtonGroupControl

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
falseLabel	optional		*Zeichenfolge*
falseLabelKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
layoutByRow	optional		*boolean*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
rows	optional		*positiveGanzzahl*
showLabel	optional		*boolean*
trueFirst	optional		*boolean*
trueLabel	optional		*Zeichenfolge*
trueLabelKey	optional		*Zeichenfolge*
useSubPanel	optional		*boolean*

XML-Darstellung

```
<xs:element name="RadioButtonGroupControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="rows" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="layoutByRow" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="useSubPanel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="falseLabel" type="xs:string" use="optional"/>
```

```
  <xs:attribute name="falseLabelKey" type="xs:string" use="optional"/>
  <xs:attribute name="trueLabel" type="xs:string" use="optional"/>
  <xs:attribute name="trueLabelKey" type="xs:string" use="optional"/>
  <xs:attribute name="trueFirst" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

Range-Element

Tabelle 171. Attribute für Range

Attribut	Verwendung	Beschreibung	Gültige Werte
max	optional		*Zeichenfolge*
min	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Range">
  <xs:attribute name="min" type="xs:string"/>
  <xs:attribute name="max" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

AddField, ChangeField, Field, Field, MissingValues, MissingValues, MissingValues

Range-Element

Tabelle 172. Attribute für Range

Attribut	Verwendung	Beschreibung	Gültige Werte
maxValue	**erforderlich**		*Zeichenfolge*
minValue	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Range" type="RANGE">
  <xs:attribute name="minValue" type="xs:string" use="required"/>
  <xs:attribute name="maxValue" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

AddField, ChangeField, Field, Field, MissingValues, MissingValues, MissingValues

RemoveField-Element

Tabelle 173. Attribute für RemoveField

Attribut	Verwendung	Beschreibung	Gültige Werte
fieldRef	**erforderlich**		

XML-Darstellung

```
<xs:element name="RemoveField">
  <xs:attribute name="fieldRef" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Übergeordnete Elemente

ForEach, ModelFields

Resources-Element

Definiert allgemeine Ressourcen wie z. B. clientseitige Bibliotheken und Ressourcenbundles und serverseitige Bibliotheken.

XML-Darstellung

```
<xs:element name="Resources">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element name="Bundle" minOccurs="0">
      </xs:element>
      <xs:element name="JarFile" minOccurs="0">
      </xs:element>
      <xs:element name="SharedLibrary" minOccurs="0">
      </xs:element>
      <xs:element name="HelpInfo" minOccurs="0">
      </xs:element>
    </xs:choice>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

Extension

Untergeordnete Elemente

Bundle, HelpInfo, JarFile, SharedLibrary

Bundle-Element:

Tabelle 174. Attribute für Bundle

Attribut	Verwendung	Beschreibung	Gültige Werte
id	**erforderlich**		*Zeichenfolge*
path	**erforderlich**		
type	**erforderlich**		**list** **properties**

XML-Darstellung

```
<xs:element name="Bundle" minOccurs="0">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="type" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="list"/>
```

```
      <xs:enumeration value="properties"/>
    </xs:restriction>
  </xs:simpleType>
 </xs:attribute>
 <xs:attribute name="path" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Resources

JarFile-Element:

Tabelle 175. Attribute für JarFile

Attribut	Verwendung	Beschreibung	Gültige Werte
id	**erforderlich**		*Zeichenfolge*
path	**erforderlich**		

XML-Darstellung

```
<xs:element name="JarFile" minOccurs="0">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="path" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Resources

SharedLibrary-Element:

Tabelle 176. Attribute für SharedLibrary

Attribut	Verwendung	Beschreibung	Gültige Werte
id	**erforderlich**		*Zeichenfolge*
path	**erforderlich**		

XML-Darstellung

```
<xs:element name="SharedLibrary" minOccurs="0">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="path" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Resources

HelpInfo-Element:

Tabelle 177. Attribute für HelpInfo

Attribut	Verwendung	Beschreibung	Gültige Werte
default	optional		*Zeichenfolge*
helpset	optional		
id	optional		*Zeichenfolge*
missing	optional		*Zeichenfolge*
path	optional		

Tabelle 177. Attribute für HelpInfo (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
type	**erforderlich**		**native** **javahelp** **html**

XML-Darstellung

```
<xs:element name="HelpInfo" minOccurs="0">
  <xs:attribute name="id" type="xs:string" use="optional"/>
  <xs:attribute name="type" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="native"/>
        <xs:enumeration value="javahelp"/>
        <xs:enumeration value="html"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="path" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="helpset" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="default" type="xs:string" use="optional"/>
  <xs:attribute name="missing" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Resources

Run-Element

XML-Darstellung

```
<xs:element name="Run">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
    <xs:element ref="Command" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Option" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="StatusCodes" minOccurs="0"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

Executable

Untergeordnete Elemente

And, Command, Condition, Not, Option, Or, StatusCodes

SPSSDataFormat-Element

XML-Darstellung

```
<xs:element name="SPSSDataFormat"/>
```

Übergeordnete Elemente

DataFormat

SelectorPanel-Element

Definiert eine UI-Komponente, die mehrere Unterfenster enthalten kann, aber in der jeweils nur eines dieser Unterfenster sichtbar ist.

Tabelle 178. Attribute für SelectorPanel

Attribut	Verwendung	Beschreibung	Gültige Werte
control	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="SelectorPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="Selector">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="control" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Selector, Visible

Zugehörige Elemente

ActionButton, ComboBoxControl, ExtensionObjectPanel, FieldAllocationList, ModelViewerPanel, StaticText, SystemControls, TabbedPanel, TextBrowserPanel

Selector-Element:

Tabelle 179. Attribute für Selector

Attribut	Verwendung	Beschreibung	Gültige Werte
controlValue	**erforderlich**		*Zeichenfolge*
panelId	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Selector">
  <xs:attribute name="panelId" type="xs:string" use="required"/>
  <xs:attribute name="controlValue" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

SelectorPanel

ServerDirectoryChooserControl-Element

Definiert ein Steuerelement, mit dem ein Verzeichnis auf dem Server ausgewählt werden kann.

Tabelle 180. Attribute für ServerDirectoryChooserControl

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
mode	**erforderlich**		**open** **save** **import** **export**
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="ServerDirectoryChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="mode" type="FILE-CHOOSER-MODE" use="required">
    <xs:enumeration value="open"/>
    <xs:enumeration value="save"/>
    <xs:enumeration value="import"/>
    <xs:enumeration value="export"/>
  </xs:attribute>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DB-ConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserCon-

trol, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

ServerFileChooserControl-Element

Definiert ein Steuerelement, mit dem eine Datei auf dem Server ausgewählt werden kann.

Tabelle 181. Attribute für ServerFileChooserControl

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
mode	**erforderlich**		**open** **save** **import** **export**
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="ServerFileChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="mode" type="FILE-CHOOSER-MODE" use="required">
    <xs:enumeration value="open"/>
    <xs:enumeration value="save"/>
    <xs:enumeration value="import"/>
    <xs:enumeration value="export"/>
  </xs:attribute>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

SetContainer-Element

Tabelle 182. Attribute für SetContainer

Attribut	Verwendung	Beschreibung	Gültige Werte
source	**erforderlich**		*Zeichenfolge*
target	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="SetContainer">
  <xs:attribute name="source" type="xs:string" use="required"/>
  <xs:attribute name="target" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

Zugehörige Elemente

SetProperty

SetProperty-Element

Tabelle 183. Attribute für SetProperty

Attribut	Verwendung	Beschreibung	Gültige Werte
source	**erforderlich**		*Zeichenfolge*
target	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="SetProperty">
  <xs:attribute name="source" type="xs:string" use="required"/>
  <xs:attribute name="target" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

Zugehörige Elemente

SetContainer

SingleFieldAllocationControl-Element

Definiert ein Steuerelement, mit dem ein Feld im Steuerelement für Feldzuordnungslisten ausgewählt werden kann, die vom Zuordnungsfunktionsattribut angegeben wird.

Tabelle 184. Attribute für SingleFieldAllocationControl

Attribut	Verwendung	Beschreibung	Gültige Werte
allocator	**erforderlich**		*Zeichenfolge*
buttonColumn	**erforderlich**		*Ganzzahl*
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
onlyDatetime	optional		*boolean*
onlyDiscrete	optional		*boolean*
onlyNumeric	optional		*boolean*
onlyRanges	optional		*boolean*
onlySymbolic	optional		*boolean*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*
storage	optional		*Zeichenfolge*
types	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="SingleFieldAllocationControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="allocator" type="xs:string" use="required"/>
  <xs:attribute name="buttonColumn" type="xs:integer" use="required"/>
  <xs:attribute name="storage" type="xs:string" use="optional"/>
  <xs:attribute name="onlyNumeric" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlySymbolic" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDatetime" type="xs:boolean" use="optional"/>
  <xs:attribute name="types" type="xs:string" use="optional"/>
  <xs:attribute name="onlyRanges" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDiscrete" type="xs:boolean" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

SingleFieldChooserControl-Element

Definiert ein Steuerelement, mit dem ein Feld im aktuellen Datenmodell ausgewählt werden kann.

Tabelle 185. Attribute für SingleFieldChooserControl

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
onlyDatetime	optional		*boolean*
onlyDiscrete	optional		*boolean*
onlyNumeric	optional		*boolean*
onlyRanges	optional		*boolean*
onlySymbolic	optional		*boolean*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*
storage	optional		*Zeichenfolge*
types	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="SingleFieldChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="storage" type="xs:string" use="optional"/>
  <xs:attribute name="onlyNumeric" type="xs:boolean" use="optional"/>
```

```
  <xs:attribute name="onlySymbolic" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDatetime" type="xs:boolean" use="optional"/>
  <xs:attribute name="types" type="xs:string" use="optional"/>
  <xs:attribute name="onlyRanges" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDiscrete" type="xs:boolean" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

SingleFieldValueChooserControl-Element

Definiert ein Steuerelement, mit dem ein Feldwert in einem Feld ausgewählt werden kann, das durch das von fieldControl angegebene Steuerelement ausgewählt wurde.

Tabelle 186. Attribute für SingleFieldValueChooserControl

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
fieldControl	optional		*Zeichenfolge*
fieldDirection	optional		**in** **out** **both** **none** **partition**
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="SingleFieldValueChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
```

```
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="fieldControl" type="xs:string" use="optional"/>
  <xs:attribute name="fieldDirection" type="FIELD-DIRECTION" use="optional">
    <xs:enumeration value="in"/>
    <xs:enumeration value="out"/>
    <xs:enumeration value="both"/>
    <xs:enumeration value="none"/>
    <xs:enumeration value="partition"/>
  </xs:attribute>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

SingleItemChooserControl-Element

Definiert ein Steuerelement, mit dem ein Wert aus einer Auswahl ausgewählt werden kann.

Tabelle 187. Attribute für SingleItemChooserControl

Attribut	Verwendung	Beschreibung	Gültige Werte
catalog	**erforderlich**		*Zeichenfolge*
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="SingleItemChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
```

```
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="catalog" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

MultiItemChooserControl

SpinnerControl-Element

Definiert ein Drehfeld, mit dem numerische Werte angegeben werden können.

Tabelle 188. Attribute für SpinnerControl

Attribut	Verwendung	Beschreibung	Gültige Werte
columns	optional		*positiveGanzzahl*
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
maxDecimalDigits	optional		*positiveGanzzahl*
minDecimalDigits	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*
stepSize	optional		*Dezimalzahl*

XML-Darstellung

```
<xs:element name="SpinnerControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
```

```
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="columns" type="xs:positiveInteger" use="optional" default="5"/>
  <xs:attribute name="stepSize" type="xs:decimal" use="optional" default="1.0"/>
  <xs:attribute name="minDecimalDigits" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="maxDecimalDigits" type="xs:positiveInteger" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, TableControl, TextAreaControl, TextBoxControl

StaticText-Element

Definiert eine UI-Komponente, die einen statischen Text enthält. Dies wird oft in Unterfenstern verwendet, um den Zweck des Fensters zu beschreiben.

Tabelle 189. Attribute für StaticText

Attribut	Verwendung	Beschreibung	Gültige Werte
Text	optional		*Zeichenfolge*
textKey	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="StaticText">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="text" type="xs:string" use="optional"/>
  <xs:attribute name="textKey" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

ActionButton, ComboBoxControl, ExtensionObjectPanel, FieldAllocationList, ModelViewerPanel, SelectorPanel, SystemControls, TabbedPanel, TextBrowserPanel

StatusCode-Element

Tabelle 190. Attribute für StatusCode

Attribut	Verwendung	Beschreibung	Gültige Werte
code	**erforderlich**		*Ganzzahl*
message	optional		*Zeichenfolge*
messageKey	optional		*Zeichenfolge*
status	optional		**success** **warning** **error**

XML-Darstellung

```
<xs:element name="StatusCode">
  <xs:attribute name="code" type="xs:integer" use="required"/>
  <xs:attribute name="status" use="optional" default="success">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="success"/>
        <xs:enumeration value="warning"/>
        <xs:enumeration value="error"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="message" type="xs:string" use="optional"/>
  <xs:attribute name="messageKey" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

StatusCodes

StatusCodes-Element

Tabelle 191. Attribute für StatusCodes

Attribut	Verwendung	Beschreibung	Gültige Werte
defaultMessage	optional		*Zeichenfolge*
defaultMessageKey	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="StatusCodes">
  <xs:sequence>
    <xs:element ref="StatusCode" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="defaultMessage" type="xs:string" use="optional"/>
  <xs:attribute name="defaultMessageKey" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Module, Run

Untergeordnete Elemente

StatusCode

StatusDetail-Element

Ergänzende Informationen zu einem Fortschritt oder zu anderen Bedingungen.

Tabelle 192. Attribute für StatusDetail

Attribut	Verwendung	Beschreibung	Gültige Werte
destination	optional		**Client** **tracefile** **console**

XML-Darstellung

```
<xs:element name="StatusDetail" type="STATUS-DETAIL">
  <xs:sequence>
    <xs:element name="Diagnostic" type="DIAGNOSTIC" minOccurs="0" maxOccurs="unbounded">
      <xs:sequence>
        <xs:element name="Message" type="DIAGNOSTIC-MESSAGE" minOccurs="0">
        </xs:element>
        <xs:element name="Parameter" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="destination" type="STATUS-DESTINATION" default="client">
    <xs:enumeration value="client"/>
    <xs:enumeration value="tracefile"/>
    <xs:enumeration value="console"/>
  </xs:attribute>
</xs:element>
```

Untergeordnete Elemente

Diagnostic

Diagnostic-Element:

Tabelle 193. Attribute für Diagnostic

Attribut	Verwendung	Beschreibung	Gültige Werte
code	**erforderlich**		*Ganzzahl*
severity	optional		**unknown** **information** **warning** **error** **fatal**
source	optional		*Zeichenfolge*
subCode	optional		*Ganzzahl*

XML-Darstellung

```
<xs:element name="Diagnostic" type="DIAGNOSTIC" minOccurs="0" maxOccurs="unbounded">
  <xs:sequence>
    <xs:element name="Message" type="DIAGNOSTIC-MESSAGE" minOccurs="0">
    </xs:element>
    <xs:element name="Parameter" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="code" type="xs:integer" use="required"/>
  <xs:attribute name="subCode" type="xs:integer" default="0"/>
  <xs:attribute name="severity" type="DIAGNOSTIC-SEVERITY" default="error">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="information"/>
```

```
    <xs:enumeration value="warning"/>
    <xs:enumeration value="error"/>
    <xs:enumeration value="fatal"/>
  </xs:attribute>
  <xs:attribute name="source" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

StatusDetail

Untergeordnete Elemente

Message, Parameter

Message-Element:

Tabelle 194. Attribute für Message

Attribut	Verwendung	Beschreibung	Gültige Werte
lang	optional		*NMTOKEN*

XML-Darstellung

```
<xs:element name="Message" type="DIAGNOSTIC-MESSAGE" minOccurs="0">
  <xs:attribute name="lang" type="xs:NMTOKEN"/>
</xs:element>
```

Übergeordnete Elemente

Diagnostic

Parameter-Element:

XML-Darstellung

```
<xs:element name="Parameter" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
```

Übergeordnete Elemente

Diagnostic

Structure-Element

XML-Darstellung

```
<xs:element name="Structure">
  <xs:sequence>
    <xs:element ref="Attribute" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

PropertyType

Untergeordnete Elemente

Attribute

StructuredValue-Element

Eine Folge von benannten Werten ("Attribute").

XML-Darstellung

```
<xs:element name="StructuredValue" type="STRUCTURED-VALUE">
  <xs:sequence>
    <xs:element name="Attribute" type="ATTRIBUTE" maxOccurs="unbounded">
      <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
        <xs:choice>
          <xs:element ref="MapValue"/>
          <xs:element ref="StructuredValue"/>
          <xs:element ref="ListValue"/>
          <xs:element ref="Value"/>
          <xs:element ref="DatabaseConnectionValue"/>
        </xs:choice>
      </xs:group>
      <xs:sequence>
        <xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
          <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
            <xs:choice>
              <xs:element ref="MapValue"/>
              <xs:element ref="StructuredValue"/>
              <xs:element ref="ListValue"/>
              <xs:element ref="Value"/>
              <xs:element ref="DatabaseConnectionValue"/>
            </xs:choice>
          </xs:group>
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

Attribute, Attribute, ListValue, ListValue, ListValue, Parameter

Untergeordnete Elemente

Attribute

Attribute-Element:

Tabelle 195. Attribute für Attribute

Attribut	Verwendung	Beschreibung	Gültige Werte
name	**erforderlich**		*Zeichenfolge*
value	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Attribute" type="ATTRIBUTE" maxOccurs="unbounded">
  <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
    <xs:choice>
      <xs:element ref="MapValue"/>
      <xs:element ref="StructuredValue"/>
      <xs:element ref="ListValue"/>
      <xs:element ref="Value"/>
      <xs:element ref="DatabaseConnectionValue"/>
    </xs:choice>
  </xs:group>
  <xs:sequence>
    <xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
      <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
        <xs:choice>
          <xs:element ref="MapValue"/>
          <xs:element ref="StructuredValue"/>
          <xs:element ref="ListValue"/>
          <xs:element ref="Value"/>
          <xs:element ref="DatabaseConnectionValue"/>
        </xs:choice>
      </xs:group>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="value" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

StructuredValue

Untergeordnete Elemente

DatabaseConnectionValue, ListValue, ListValue, MapValue, StructuredValue, Value

ListValue-Element: Eine Folge von Werten. Alle Werte müssen denselben Inhaltstyp haben, aber dies wird nicht geprüft.

XML-Darstellung

```
<xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
  <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="MapValue"/>
      <xs:element ref="StructuredValue"/>
      <xs:element ref="ListValue"/>
      <xs:element ref="Value"/>
      <xs:element ref="DatabaseConnectionValue"/>
    </xs:choice>
  </xs:group>
</xs:element>
```

Übergeordnete Elemente

Attribute

Untergeordnete Elemente

DatabaseConnectionValue, ListValue, MapValue, StructuredValue, Value

SystemControls-Element

Tabelle 196. Attribute für SystemControls

Attribut	Verwendung	Beschreibung	Gültige Werte
controlsId	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="SystemControls">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="controlsId" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

ActionButton, ComboBoxControl, ExtensionObjectPanel, FieldAllocationList, ModelViewerPanel, SelectorPanel, StaticText, TabbedPanel, TextBrowserPanel

Tab-Element

Definiert eine Registerkarte in einem Registerkartenfenster.

Tabelle 197. Attribute für Tab

Attribut	Verwendung	Beschreibung	Gültige Werte
helpLink	optional		*Zeichenfolge*
id	optional		*Zeichenfolge*
label	**erforderlich**		*Zeichenfolge*
labelKey	optional		*Zeichenfolge*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Tab">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="PropertiesPanel"/>
      <xs:element ref="ExtensionObjectPanel"/>
      <xs:element ref="TextBrowserPanel"/>
      <xs:element ref="ModelViewerPanel"/>
      <xs:element ref="TabbedPanel"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="optional"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="helpLink" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

Tabs

Untergeordnete Elemente

ExtensionObjectPanel, ModelViewerPanel, PropertiesPanel, TabbedPanel, TextBrowserPanel

TabbedPanel-Element

Definiert ein Registerkartenfenster. Den Registerkarten im Registerkartenfenster können andere Fenster hinzugefügt werden.

Tabelle 198. Attribute für TabbedPanel

Attribut	Verwendung	Beschreibung	Gültige Werte
style	optional		**standard** **sidebar**

XML-Darstellung

```
<xs:element name="TabbedPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
```

```
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Tabs"/>
  </xs:sequence>
  <xs:attribute name="style" use="optional">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="standard"/>
        <xs:enumeration value="sidebar"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>
```

Übergeordnete Elemente

Tab

Untergeordnete Elemente

Enabled, Layout, Tabs, Visible

Zugehörige Elemente

ActionButton, ComboBoxControl, ExtensionObjectPanel, FieldAllocationList, ModelViewerPanel, SelectorPanel, StaticText, SystemControls, TextBrowserPanel

TableControl-Element

Definiert ein Steuerelement für Tabellen, mit dem Werte in einer Liste von Strukturen hinzugefügt, geändert und entfernt werden können.

Tabelle 199. Attribute für TableControl

Attribut	Verwendung	Beschreibung	Gültige Werte
columnWidths	optional		*Zeichenfolge*
columns	optional		*positiveGanzzahl*
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
rows	optional		*positiveGanzzahl*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="TableControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
```

```
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="rows" type="xs:positiveInteger" use="optional" default="8"/>
  <xs:attribute name="columns" type="xs:positiveInteger" use="optional" default="20"/>
  <xs:attribute name="columnWidths" type="xs:string" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TextAreaControl, TextBoxControl

Tabs-Element

Definiert die Folge von Registerkarten in einem Registerkartenfenster.

Tabelle 200. Attribute für Tabs

Attribut	Verwendung	Beschreibung	Gültige Werte
defaultTab	optional		*nichtnegativeGanzzahl*

XML-Darstellung

```
<xs:element name="Tabs">
  <xs:sequence>
    <xs:element ref="Tab" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="defaultTab" type="xs:nonNegativeInteger" use="optional" default="0"/>
</xs:element>
```

Übergeordnete Elemente

TabbedPanel, UserInterface

Untergeordnete Elemente

Tab

TextAreaControl-Element

Definiert einen mehrzeiligen Textbereich, mit dem Zeichenfolgewerte geändert werden können.

Tabelle 201. Attribute für TextAreaControl

Attribut	Verwendung	Beschreibung	Gültige Werte
columns	optional		*positiveGanzzahl*

Tabelle 201. Attribute für TextAreaControl (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
monospaced	optional		*boolean*
property	**erforderlich**		*Zeichenfolge*
rows	optional		*positiveGanzzahl*
showLabel	optional		*boolean*
wrapLines	optional		*boolean*

XML-Darstellung

```
<xs:element name="TextAreaControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="rows" type="xs:positiveInteger" use="optional" default="8"/>
  <xs:attribute name="columns" type="xs:positiveInteger" use="optional" default="20"/>
  <xs:attribute name="wrapLines" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="monospaced" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextBoxControl

TextBoxControl-Element

Definiert ein Steuerelement für einzeiligen Text, mit dem Zeichenfolgewerte geändert werden können.

Tabelle 202. Attribute für TextBoxControl

Attribut	Verwendung	Beschreibung	Gültige Werte
columns	optional		*positiveGanzzahl*
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="TextBoxControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="columns" type="xs:positiveInteger" use="optional" default="20"/>
</xs:element>
```

Übergeordnete Elemente

PropertiesPanel, PropertiesSubPanel

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldAllocationControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl

TextBrowserPanel-Element

Tabelle 203. Attribute für TextBrowserPanel

Attribut	Verwendung	Beschreibung	Gültige Werte
columns	optional		*Zeichenfolge*
container	**erforderlich**		*Zeichenfolge*
rows	optional		*Zeichenfolge*
textFormat	**erforderlich**		**plainText** **html** **rtf**
wrapLines	optional		*boolean*

XML-Darstellung

```
<xs:element name="TextBrowserPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="container" type="xs:string" use="required"/>
  <xs:attribute name="textFormat" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="plainText"/>
        <xs:enumeration value="html"/>
        <xs:enumeration value="rtf"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="rows" type="xs:string" use="optional"/>
  <xs:attribute name="columns" type="xs:string" use="optional"/>
  <xs:attribute name="wrapLines" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Übergeordnete Elemente

Tab

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Elemente

ActionButton, ComboBoxControl, ExtensionObjectPanel, FieldAllocationList, ModelViewerPanel, SelectorPanel, StaticText, SystemControls, TabbedPanel

ToolbarItem-Element

Definiert ein Element, das einer Symbolleiste für ein Fenster hinzugefügt werden kann.

Tabelle 204. Attribute für ToolbarItem

Attribut	Verwendung	Beschreibung	Gültige Werte
action	**erforderlich**		*Zeichenfolge*
offset	optional		*nichtnegativeGanzzahl*
separatorAfter	optional		*boolean*
separatorBefore	optional		*boolean*

Tabelle 204. Attribute für ToolbarItem (Forts.)

Attribut	Verwendung	Beschreibung	Gültige Werte
showIcon	optional		*boolean*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:element name="ToolbarItem">
  <xs:attribute name="action" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="showIcon" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="separatorBefore" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="separatorAfter" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="offset" type="xs:nonNegativeInteger" use="optional" default="0"/>
</xs:element>
```

Übergeordnete Elemente

Controls

UTF8Format-Element

XML-Darstellung

```
<xs:element name="UTF8Format"/>
```

Übergeordnete Elemente

FileFormatType

UserInterface-Element

Definiert die Benutzerschnittstelle für das Erweiterungsobjekt oder das Tool.

Tabelle 205. Attribute für UserInterface

Attribut	Verwendung	Beschreibung	Gültige Werte
actionHandler	optional		*beliebig*
frameClass	optional		*beliebig*
uiDelegate	optional		*beliebig*

XML-Darstellung

```
<xs:element name="UserInterface">
  <xs:sequence>
    <xs:element ref="Icons" minOccurs="0"/>
    <xs:element ref="Controls" minOccurs="0"/>
    <xs:element ref="Tabs" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="uiDelegate" use="optional"/>
  <xs:attribute name="frameClass" use="optional"/>
  <xs:attribute name="actionHandler" use="optional"/>
</xs:element>
```

Übergeordnete Elemente

DocumentOutput, Extension, InteractiveDocumentBuilder, InteractiveModelBuilder, ModelOutput, Node

Untergeordnete Elemente

Controls, Icons, Tabs

Value-Element

Ein einfacher Wert.

Tabelle 206. Attribute für Value

Attribut	Verwendung	Beschreibung	Gültige Werte
value	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Value" type="SIMPLE-VALUE">
  <xs:attribute name="value" type="xs:string" use="required"/>
</xs:element>
```

Übergeordnete Elemente

Attribute, Attribute, ListValue, ListValue, ListValue, Parameter

Values-Element

XML-Darstellung

```
<xs:element name="Values">
  <xs:sequence>
    <xs:element name="Value" minOccurs="0" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

AddField, ChangeField, Field, Field, MissingValues, MissingValues, MissingValues

Untergeordnete Elemente

Value

Value-Element:

Tabelle 207. Attribute für Value

Attribut	Verwendung	Beschreibung	Gültige Werte
flagProperty	optional		**trueValue** **falseValue**
value	**erforderlich**		*Zeichenfolge*
valueLabel	optional		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Value" minOccurs="0" maxOccurs="unbounded">
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="valueLabel" type="xs:string" use="optional"/>
  <xs:attribute name="flagProperty">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="trueValue"/>
        <xs:enumeration value="falseValue"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>
```

Übergeordnete Elemente

Values

Values-Element

Tabelle 208. Attribute für Values

Attribut	Verwendung	Beschreibung	Gültige Werte
code	**erforderlich**		*Ganzzahl*
displayLabel	optional		*Zeichenfolge*
flagValue	optional		*boolean*
value	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="Values" type="FIELD-VALUE">
  <xs:sequence>
    <xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="code" type="xs:integer" use="required"/>
  <xs:attribute name="flagValue" type="xs:boolean"/>
  <xs:attribute name="displayLabel" type="xs:string"/>
</xs:element>
```

Übergeordnete Elemente

AddField, ChangeField, Field, Field, MissingValues, MissingValues, MissingValues

Untergeordnete Elemente

DisplayLabel

DisplayLabel-Element: Eine Anzeigebeschriftung für ein Feld oder einen Wert in einer angegebenen Sprache. Das DisplayLabel-Attribut kann verwendet werden, wenn es keine Beschriftung für eine bestimmte Sprache gibt.

Tabelle 209. Attribute für DisplayLabel

Attribut	Verwendung	Beschreibung	Gültige Werte
lang	optional		*NMTOKEN*
value	**erforderlich**		*Zeichenfolge*

XML-Darstellung

```
<xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="lang" type="xs:NMTOKEN" default="en"/>
</xs:element>
```

Übergeordnete Elemente

Values

Visible-Element

Definiert die Bedingung, unter der eine UI-Komponente sichtbar sein soll.

XML-Darstellung

```
<xs:element name="Visible">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
</xs:element>
```

Übergeordnete Elemente

ActionButton, CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, ComboBoxControl, DBConnectionChooserControl, DBTableChooserControl, ExtensionObjectPanel, FieldAllocationList, ItemChooserControl, ModelViewerPanel, MultiFieldAllocationControl, MultiFieldChooserControl, MultiItemChooserControl, PasswordBoxControl, PropertiesPanel, PropertiesSubPanel, PropertyControl, RadioButtonGroupControl, SelectorPanel, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldAllocationControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SingleItemChooserControl, SpinnerControl, StaticText, SystemControls, TabbedPanel, TableControl, TextAreaControl, TextBoxControl, TextBrowserPanel

Untergeordnete Elemente

And, Condition, Not, Or

Erweiterte Typen

Erweiterte Typen erweitern Elemente in einem XML-Dokument, indem sie Attribute und untergeordnete Elemente hinzufügen. Um einen erweiterten Typ in einem XML-Dokument zu verwenden, geben Sie den erweiterten Typ mit dem xsi:type-Attribut für das Element an. Anschließend können Sie die vom erweiterten Typ definierten Attribute und Elemente verwenden.

ItemChooserControl-Typ

Tabelle 210. Attribute für ItemChooserControl

Attribut	Verwendung	Beschreibung	Gültige Werte
catalog	**erforderlich**		*Zeichenfolge*
description	optional		*Zeichenfolge*
descriptionKey	optional		*Zeichenfolge*
label	optional		*Zeichenfolge*
labelAbove	optional		*boolean*
labelKey	optional		*Zeichenfolge*
labelWidth	optional		*positiveGanzzahl*
mnemonic	optional		*Zeichenfolge*
mnemonicKey	optional		*Zeichenfolge*
property	**erforderlich**		*Zeichenfolge*
showLabel	optional		*boolean*

XML-Darstellung

```
<xs:complexType name="ItemChooserControl" mixed="false">
  <xs:sequence>
    <xs:choice>
```

```
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

Erweitert

ComboBoxControl

Untergeordnete Elemente

Enabled, Layout, Visible

Zugehörige Typen

ItemChooserControl

Bemerkungen

Diese Informationen wurden für weltweit angebotene Produkte und Dienstleistungen erarbeitet.

Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. Anstelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder anderen Schutzrechte von IBM verletzen. Die Verantwortung für den Betrieb von Produkten, Programmen und Services anderer Anbieter liegt beim Kunden.

Für in diesem Handbuch beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing
IBM Europe, Middle East & Africa
Tour Descartes
2, avenue Gambetta
92066 Paris La Defense
France

Verweise in diesen Informationen auf Websites anderer Anbieter werden lediglich als Service für den Kunden bereitgestellt und stellen keinerlei Billigung des Inhalts dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt. Die Verwendung dieser Websites geschieht auf eigene Verantwortung.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängig voneinander erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

IBM Software Group
ATTN: Licensing
200 W. Madison St.
Chicago, IL; 60606
USA

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des in diesem Dokument beschriebenen Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung bzw. der Allgemeinen Geschäftsbedingungen von IBM, der IBM Internationalen Nutzungsbedingungen für Programmpakete oder einer äquivalenten Vereinbarung.

Alle in diesem Dokument enthaltenen Leistungsdaten stammen aus einer kontrollierten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Gewährleistung, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können davon abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichten Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Aussagen über Pläne und Absichten von IBM unterliegen Änderungen oder können zurückgenommen werden und repräsentieren nur die Ziele von IBM.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufs. Sie sollen nur die Funktionen des Lizenzprogramms illustrieren und können Namen von Personen, Firmen, Marken oder Produkten enthalten. Alle diese Namen sind frei erfunden; Ähnlichkeiten mit tatsächlichen Namen und Adressen sind rein zufällig.

Marken

IBM, das IBM Logo und ibm.com sind Marken oder eingetragene Marken der IBM Corporation in den USA und/oder anderen Ländern. Weitere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite "Copyright and trademark information" unter www.ibm.com/legal/copytrade.shtml.

Intel, das Intel-Logo, Intel Inside, das Intel Inside-Logo, Intel Centrino, das Intel Centrino-Logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium und Pentium sind Marken oder eingetragene Marken der Intel Corporation oder der Tochtergesellschaften des Unternehmens in den USA und anderen Ländern.

Linux ist eine eingetragene Marke von Linus Torvalds in den USA, anderen Ländern oder beidem.

Microsoft, Windows, Windows NT und das Windows-Logo sind Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.

Java und alle auf Java basierenden Marken und Logos sind Marken oder eingetragene Marken der Oracle Corporation und/oder ihrer verbundenen Unternehmen.

Weitere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z