XML-Search: XML-basiertes Protokoll für Suchanfragen via Webservices	Konvention xml-sw 1.0.0 Empfehlung

Kurzbeschreibung	Das vorliegende Dokument beschreibt ein generisches Protokoll für Suchanfragen via Webservices im E-Government. Diese Empfehlung identifiziert gemeinsame XML-Elemente und definiert eine allgemeine XML-Struktur für Suchanfragen. Diese Empfehlung ist Teil eines Gesamtpakets aus W3C XML Schema, WSDL-Dokument und Beispielanfragen.		
Autor(en):	Franz-Josef Herpers	Projektteam / Arbeitsgruppe	KommArch Stabsstelle IKT-Strategie

Stelle	Vorgelegt am	Angenommen am	Abgelehnt am
Bund Länder Gemeindebund Städtebund	18.04.2005 18.04.2005 18.04.2005 18.04.2005	09.05.2005 09.05.2005 09.05.2005 09.05.2005	

XML-Search

Inhaltsverzeichnis

(1) Einleitung

Das vorliegende Dokument spezifiziert ein generisches Protokoll für Suchanfragen via Webservices im E-Government. Dazu gehört insbesondere:

- Identifikation und Strukturierung gemeinsamer XML-Elemente für alle Suchanfragen die Use Cases *SearchById* und *SearchByExample*.
- Bereitstellung eines Mechanismus zum segmentweisen Abfragen von großen Ergebnismengen.
- Konventionen für das Wildcarding
- Vorgabe von MessageCodes und eine Empfehlung zur Definition eigener MessageCodes

Der Transport der Suchanfrage erfolgt typischerweise über SOAP (s. **[soap-1.1]**) es sind aber auch andere Transportprotokolle denkbar. Die normative Spezifikation wird als ein W3C XML Schema (s. **[wxs-1]** und **[wxs-2]**) **[wxs-1]** zur Verfügung gestellt, das in ein WSDL-Dokument (s. **[wsdl-1.1]**), welches die konkrete Schnittstelle beschreibt, integriert bzw. von diesem referenziert werden kann. Ein solches WSDL-Dokument wird vom Gesamtpaket dieser Spezifikation ebenfalls zur Verfügung gestellt. Das Schema ist an vielen Stellen offen gehalten für notwendige Erweiterungen bei konkreten Implementierungen von Suchanfragen.

Neben einer verbalen Beschreibung der Schemakomponenten werden Richtlinien zur Verwendung erläutert. Beispiele zur Schemaanwendung ergänzen die Erklärungen der Schemakomponenten und sind der Spezifikation als XML-Dateien beigefügt. Darüber hinaus legt dieses Dokument Konventionen für das Wildcarding und grundlegende Message-Codes und -Texte fest.

Es wurde versucht, sich soweit möglich an bestehenden Implementierungen und Entwürfe konkreter Registerabfragen zu orientieren. Zudem wurde darauf geachtet, möglichst viele bereits bestehende Standards im Rahmen des E-Government zu nutzen und bei der Erstellung des WSDL-Dokuments auf Interoperabilität Rücksicht zu nehmen.

Im Anhang finden sich eine Beschreibung der zu dieser Spezifikation mitgelieferten Dateien sowie komplette Listings des Schemas und des WSDL-Dokuments.

(2) Grundlegende Design-Prinzipien

(2.1) Sprache

Als einheitliche Sprache für die Bezeichnung der Schemakomponenten wurde Englisch gewählt, da die meisten Begriffe im vorliegenden Kontext in ihrer englischen Variante gängiger als in ihrer deutschen sind. Auch die vordefinierten Texte der Fehler- bzw. Statusmeldungen sind um der Einheitlichkeit willen in Englisch gehalten.

(2.2) Schreibweisen

Die Namen der Elementtypen beginnen ausnahmslos mit einem Großbuchstaben.

Die Schreibweise zusammengesetzter Namen folgt der so genannten *CamelCase*-Methode, d.h. einer Schreibweise von zusammengesetzten Worten, bei der die einzelnen Worte mit einem Großbuchstaben am Anfang geschrieben werden (z.B.: `ResultRecords`).

(2.3) Erweiterbarkeit/Offenheit

Das Schema ist an relevanten Stellen durch Platzhalter (*Wildcards*) bzw. Erweiterungselemente, die Wildcards enthalten, offen gehalten, so dass Implementierungen eigene Erweiterungen integrieren können.

Im Rahmen von *W3C XML Schema* (WXS) (s. **[wxs-1]**) sind bei der Nutzung von Wildcards einige Regeln zu beachten - insbesondere für Element-Wildcards (`xs:any`). Diese hängen damit zusammen, dass nicht-deterministische Inhaltsmodelle in WXS unzulässig sind. So ist z.B. eine Wildcard, die beliebige Elemente aus beliebigen Namensräumen erlaubt (`<xs:any namespace="##any">`), nach einem optionalen Elementtyp unzulässig, weil ein solches Inhaltsmodell nicht-deterministisch ist. Alle in XML-Search gesetzten Wildcards, die Erweiterungspunkte für Implementierer darstellen, vertreten daher nur Elemente aus anderen Namensräumen als dem Zielnamensraum, wodurch dieses Problem umgangen wird.

Durch eine solche Form des Erweiterungsprozesses wird aber auch sicher gestellt, dass dezentrale Erweiterungen durch Serverbetreiber nicht den Kernstandard selbst erweitern können, also zu Inkompatibilitäten führen. Somit sind Erweiterungen durch Serverbetreiber immer kompatibel zur jeweilig verwendeten Version von XML-Search. Zentrale Erweiterungen am Standard selbst stellen dann eine neue Version dar.

Wildcards, die auch von den Designern von XML-Search genutzt werden können, um den Standard zu erweitern, ohne eine zur Vorgängerversion inkompatible Version zu erstellen, werden in einem Elementtyp gekapselt, der einen Erweiterungspunkt darstellt. Angezeigt werden solche Elementtypen durch das Präfix "Extra" im Namen des Elementtyps (z.B. `ExtraResultCriteria`). Dies ist ebenfalls wegen der Einschränkung in WXS, nur deterministische Inhaltsmodelle zu unterstützen, notwendig.

(2.4) Wiederverwendbarkeit

Das Schema ist auf maximale Wiederverwendbarkeit ausgelegt und daher nach dem so genannten *Venetian Blind Model* aufgebaut. D.h. Elemente und komplexe

Typen sind in der Regel global deklariert und somit in anderen Schemata wieder verwendbar.

(2.5) Namensraum

Der Namenraumbezeichner (s. **[nsp-1.0]**) für den Zielnamensraum von XML-Search ist nach dem gleichen Prinzip wie viele andere im Rahmen des E-Government gewählten Namensraum-Bezeichner aufgebaut. Er stellt eine URI nach dem http-Schema dar (s. **[uri]** und **[url]**). Domain ist der Reference-Server (`reference.e-government.gv.at`). Es folgt das Schlüsselwort `namespace` und das Kürzel des Standards (`xml-sw`) als Pfadangaben. Bei XML-Search wird aufgrund der vorgeschlagenen Versionierungsstrategie (s. Kapitel (2.6)) die erste Stelle der Versionsnummer (*Major Version*) angehängt. Der Namensraumbezeichner endet auf einem so genannten Fragment-Identifier (#). Dieses Muster ermöglicht es, einen zusätzlichen Identifier für Ableitungen des Schemas (z.B. konkrete Erweiterungen für spezifische Registerabfragen) anzuhängen. Dabei bleibt der optische Bezug zum allgemeinen Schema im Namensraum-Bezeichner erhalten. Die aktuelle Version hat somit den Namensraumbezeichner:

`http://reference.e-government.gv.at/namespace/xml-sw/1#`

Alle Elementtypen, inkl. der lokal deklarierten, sind qualifiziert, Attribute nicht.

(2.6) Versionierung

Die Versionierung des Schemas erfolgt über das `version`-Attribut des `xs:schema`-Elements. Die Versionsnummer ist dieselbe wie die des vorliegenden Dokuments, aktuell also 1.0.0.

Änderungen in der ersten Stelle der Versionsnummer sollten eine neue Version anzeigen, die nicht kompatibel zur alten Version sein muss *(Major Version*). Änderungen der zweiten und dritten Stelle der Versionsnummer zeigen neue Versionen an, die kompatibel zur Vorgängerversion sind (*Minor Version*). Dies entspricht im wesentlichen dem in **[rm]** vorgeschlagenen Vorgehen.

Der Namensraumbezeichner enthält die Nummer der *Major Version*, also lediglich die erste Stelle der vollständigen Versionsnummer. Die aktuelle Version hat den Namensraum-Bezeichner:
`http://reference.e-government.gv.at/namespace/xml-sw/1#`

Neue Komponenten, die im Rahmen einer *Minor Version* hinzukommen, müssen zum bestehenden Zielnamensraum hinzugefügt werden. Eine *Major Version* zieht eine Änderung des Namensraumbezeichners und damit zwangsläufig eine Imkompatibilität zur Vorgängerversion nach sich.

Im Umgang mit Schemakomponenten höherer/tieferer Versionen, die von einem Server oder Client nicht verstanden werden (inkompatibel zur unterstützen Version sind), gilt das *Must-Ignore-Prinzip*: XML-Elemente und Attribute in einem gültigen Dokument, die nicht verstanden werden, müssen ignoriert werden. Die Verarbeitung wird dabei nicht durch einen Fehler abgebrochen. Dieses Prinzip ist wohlbekannt von HTTP 1.1 (s. **[http-1.1]**) und wurde für XML publiziert in **[fxpp]**.

(3) Beschreibung der XML-Struktur

Die zentralen Objekte im Schema von XML-Search sind die Suchanfrage eines Clients an einen Server (z.B. an ein Register) und die zurück gelieferte Antwort des Servers in Form eines Suchergebnisses. Das Suchergebnis wird durch den global deklarierten Elementtyep `SearchResponse` repräsentiert. Für die Suchanfragen unterscheidet XML-Search zwei Typen, deren Ausprägung im folgenden näher beschrieben wird.

(3.1) Typen von Suchanfragen

Generell lassen sich bei Suchanfragen an Register folgende Anwendungsfälle unterscheiden:

- Suche anhand von Beispielen (*Search by Example*)
- Folgesuche über eine eindeutige ID (*Search by Id*)

Search by Example bedeutet die Suche in einer Datenbank (z.B. Register) über die Vorgabe von Inhalten für einige der vom gesuchten Objekttyp festgelegten Datenfelder. Die Suche wird dann eingegrenzt auf diejenigen Datenobjekte, die alle vorgegebenen Informationen enthalten. Z.B. könnte man auf diese Weise gemeldete Personen suchen, deren Wohnort Wien ist und die 1978 geboren sind. Oft ist bei einer *Search by Example* auch die Verknüpfungsart zwischen den einzelnen Suchfeldern wählbar (z.B. Wien UND Geburtsjahr 1978, Wien ODER Geburtsjahr 1978). Diese Variante ist in der vorliegenden Version von XML-Search nicht vorgesehen. Standardannahme für XML-Search-Implementierungen ist eine UND-Verknüpfung.

Zusätzlich ist eine Konvention für Platzhalter (*Wildcards*) üblich, mit deren Hilfe man z.B. alle Personen suchen kann, die mit der Silbe "mann" enden. Diese Spezifikation beinhaltet eine Richtlinie für das Wildcarding (s. Kapitel (6)).

Search by Id meint die Suche über eine eindeutige ID, die einem Datenobjekt auf dem Server zugeordnet ist und die dieses Objekt eindeutig bestimmt. Einer *Search by Id* geht daher immer eine *Search by Example* voraus. Die ID, mit der in der nachfolgenden *Search by Id* gesucht wird, ist eine vom Server mit dem gewünschten Datensatz ausgelieferte ID.

Beide Anwendungsfälle werden in der XML-Struktur durch ein eigenes Wurzelelement repräsentiert. Die beiden Wurzelelemente sind:

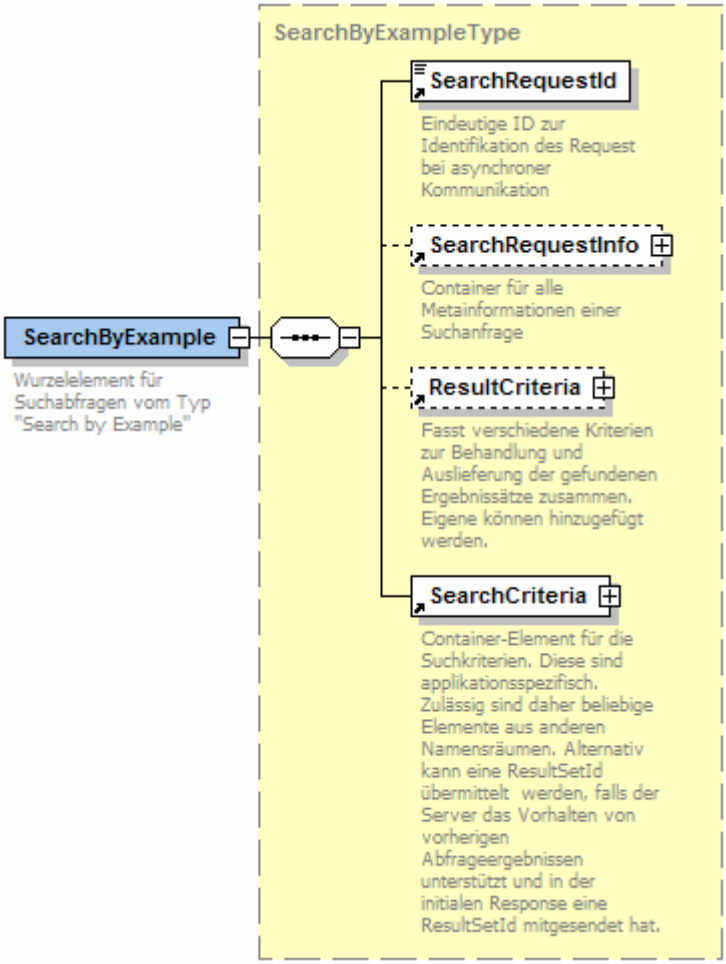
- `SearchByExample`
- `SearchById`

Diese Wurzelelemente haben jeweils die im folgenden beschriebene Struktur:

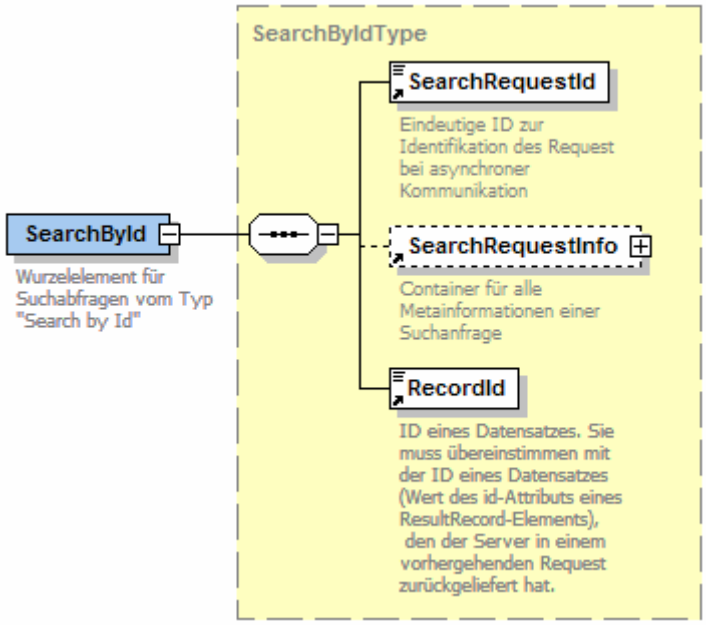
(3.2) Strukturübersichten der Elementtypen für Suchanfragen

In diesem Kapitel werden kurz die Strukturmodelle für die beiden Wurzelelemente, die die beiden Suchanfragetypen repräsentieren, abgebildet. Im Anschluss werden die Substrukturen detailliert beschrieben (Kapitel (3.3)).

3.2.1. Strukturübersicht SearchByExample



3.2.2. Strukturübersicht SearchById

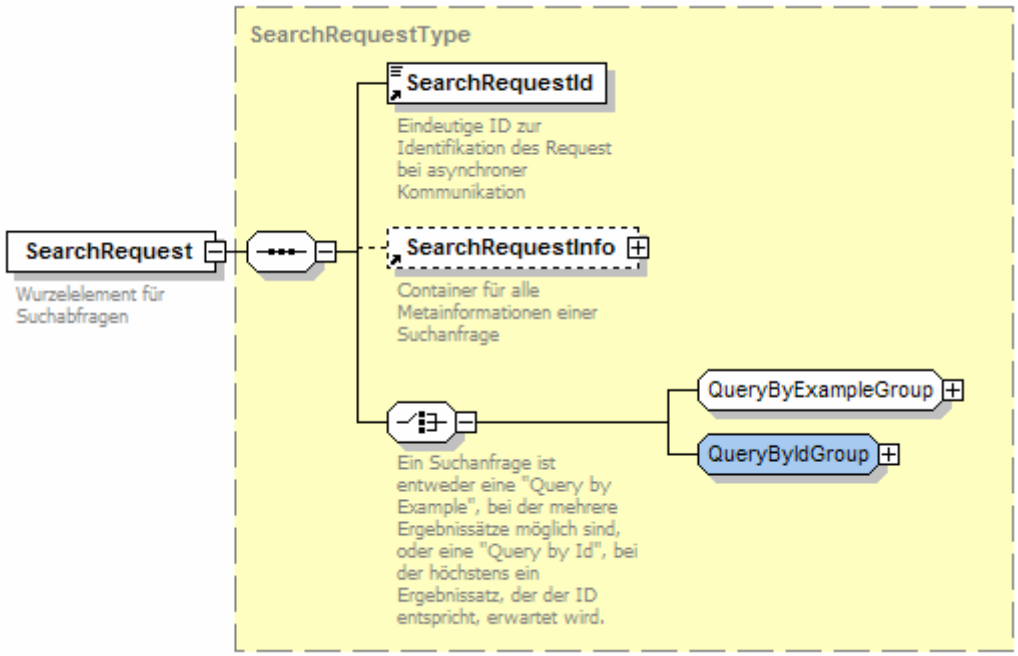


(3.3) Gemeinsame Elementtypen der Anfragetypen

Im folgenden werden die den beiden Anfragetypen gemeinsamen Kindelementtypen detailliert beschrieben.

3.3.1. SearchRequestId

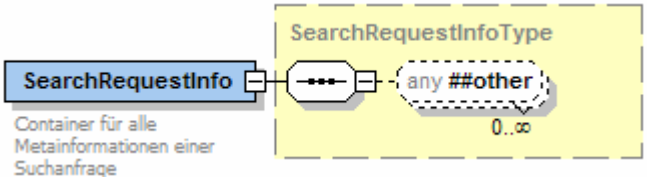
Notwendigerweise hat jede Suchanfrage eine `SearchRequestId`. Eine solche ID ermöglicht dem Client die Zuordnung der Antwort des Servers, die mit derselben ID zurückgesendet wird. Dadurch wird asynchrone Kommunikation möglich.



3.3.2. SearchRequestInfo

Beide Wurzelelemente für die Suchanfragetypen haben einen optionalen Kindelementtyp namens `SearchRequestInfo`, der Metainformationen zu dem Request kapselt.

Der Elementtyp `SearchRequestInfo` ist lediglich ein Container für alle Metainformationen die Suchanfrage betreffend.



Die vorliegende Version von XML-Search macht keine konkreten Vorgaben für Metadaten. Der Elementtyp `SearchRequestInfo` ist jedoch erweiterbar um implementierungsspezifische Elemente. Hier können technische Metadaten wie z.B. Workflow-Informationen zu Client und Server oder auch weitere fachliche Metadaten wie z.B. die anfragende Organisation untergebracht werden.

Der Erweiterungspunkt wird im XML-Schema durch die Verwendung einer so genannten *Wildcard* repräsentiert. Diese hat im vorliegenden Fall folgende Attribute:

namespace	#any
processContents	lax
minOccurs	0
maxOccurs	1

Das bedeutet, es können beliebig viele Erweiterungselemente eingefügt werden (maxOccurs="unbounded"), es muss aber keines hinzugefügt werden (minOccurs="0"). Die Elemente können zudem aus einem beliebigen Namensaum gewählt werden (namespace="#any"). Implementierungsspezifische Erweiterungen sollten allerdings aus einem anderen Namensraum als dem Zielnamensraum des hier definierten Schemas stammen. Es ist nicht notwendig, dass für die eingefügten Elemente ein Schema vorhanden ist, gemäß dem diese gültig sind. Es kann jedoch eines angegeben werden und damit eine Validierung erzwungen werden (processContents="lax"). Ein Schema kann z.B. direkt im Element selbst über das Attribut schemaLocation erfolgen:

```
<SearchRequestInfo>
  <base:ClientInfo
  xsi:schemaLocation=
  " http://bmi.gv.at/namespace/zmr-su/zmr/20040201#
../xsd/base.xsd">
    <base:Softwarehaus>ZMR HTML GUI</base:Softwarehaus>
    <base:ClientVersion>3.0</base:ClientVersion>
  </base:ClientInfo>
</SearchRequestInfo>
```

(3.4) Spezifische Kindelementtypen von SearchByExample

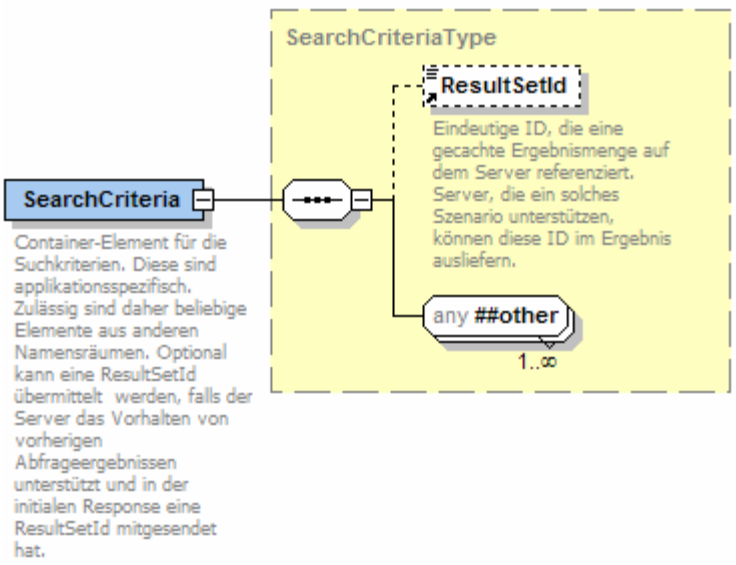
Neben den beiden Suchanfragetypen gemeinsamen Kindelementtypen SearchRequestId und SearchRequestInfo gehören die beiden Elementtypen ResultCriteria und SearchCriteria zum Wurzelelement SearchByExample. Sie werden im folgenden detailliert beschrieben.

3.4.1. SearchCriteria

Der notwendige Elementtyp SearchCriteria stellt einen Container für applikationsspezifische Suchkriterien zur Verfügung und bietet zusätzlich an, eine vom Server bereits erzeugte und vorgehaltene Ergebnismenge mittels einer ResulSetId zu referenzieren. Server, die dieses Verfahren unterstützen, können im Ergebnis eine ResultSetId mitliefern, die der Client dann hier referenzieren kann. Typischerweise wird ein Client diese ID nutzen, um z.B. die nächsten 50 Sätze einer vorherigen Abfrage zu holen.

Hinweis:

Aus Interoperabilitätsgründen hinsichtlich der Generierung von Programmcode (z.B. Java-Klassen) wurde auf die Nutzung einer xs:choice-Modellgruppe an dieser Stelle verzichtet, obwohl das die eigentlich intendierte Struktur korrekt darstellen würde. Es macht natürlich keinen Sinn, eine ResultSetId ***und*** *Suchkriterien mit zu senden, auch wenn diese nun theoretisch von der Struktur her zulässig wäre.*



Empfehlungen zur Struktur von Suchkriterien (SearchCriteria)

Die Struktur unterhalb des Elementtyps `SearchCriteria` sollte möglichst flach sein, d.h. so flach wie es dass Schema der Fachobjekte erlaubt. Im Idealfall ist dies eine einfache Liste von Elementen, die Suchkriterien repräsentieren. Diese Struktur ist leicht verarbeitbar und sollte in den meisten Fällen ausreichen. Ein Beispiel wäre:

```
<SearchCriteria>
  <pers:Vorname>Max</pers:Vorname>
  <pers:Familienname>Mustermann</pers:Familienname>
</SearchCriteria >
```

Beispiel 1: Abfrage mit flacher Struktur

Zulässig ist gemäß Schema aber auch eine verschachtelte Struktur für Fälle, in denen eine flache Liste nicht ausreicht oder laut Schema nicht möglich ist. Obige Liste benötigt eine weitere Verschachtelungsebene, um konform zum *PersonData*-Schema (s. **[persondata2]**) zu sein.

```
<SearchCriteria >
  <pers:PersonenName>
    <pers:Vorname>Max</pers:Vorname>
    <pers:Familienname>Mustermann</pers:Familienname>
  </pers:PersonenName>
</SearchCriteria >
```

Ein weiteres Beispiel wäre die jetzige Struktur im Entwurf für die Meldeauskunft. Hier dient eine teilweise befüllte Struktur des Elementtyps `NatuerlichePerson` aus dem *PersonData*-Schema optional gefolgt vom teilweise befülltem Elementtyp `PostAdresse` (ebenfalls aus dem *PersonData*-Schema) als Suchkriterium.

```
<SearchCriteria >
  <pers:NatuerlichePerson>
    <pers:PersonenName>
      <pers:Vorname>Max</pers:Vorname>
      <pers:Familienname>Mustermann</pers:Familienname>
    </pers:PersonenName>
    <pers:Geburtsdatum>2003-01-01</pers:Geburtsdatum>
```

```
  </pers:NatuerlichePerson>
  <pers:PostAdresse>
    <pers:Gemeindekennzahl>10001</pers:Gemeindekennzahl>
    <pers:Ortschaft>Testort</pers:Ortschaft>
    <pers:Zustelladresse>
      <pers:Strassenname>Teststrasse</pers:Strassenname>
    </pers:Zustelladresse>
  </pers:PostAdresse>
</SearchCriteria >
```

Beispiel 2: Meldeauskunft-Abfrage in XML-Search

Das *PersonData*-Schema ist ideal aufgebaut für die Formulierung von *Search by Examples* in XML, da alle Kindelemente der Elementtypen, die für die Formulierung einer Beispielanfrage relevant sind, optional sind. Somit sind solche Teilstrukturen als Suchbeispiel gleichzeitig valide gegen das *PersonData*-Schema.

Da zudem viele relevante Elemente des *PersonData*-Schemas global deklariert sind, können sie direkt unterhalb von `SearchCriteria` verwendet werden, und das Gesamtdokument bleibt valide. Dadurch können leicht zu verarbeitende möglichst flache Strukturen unterhalb von `SearchCriteria` gebildet werden.

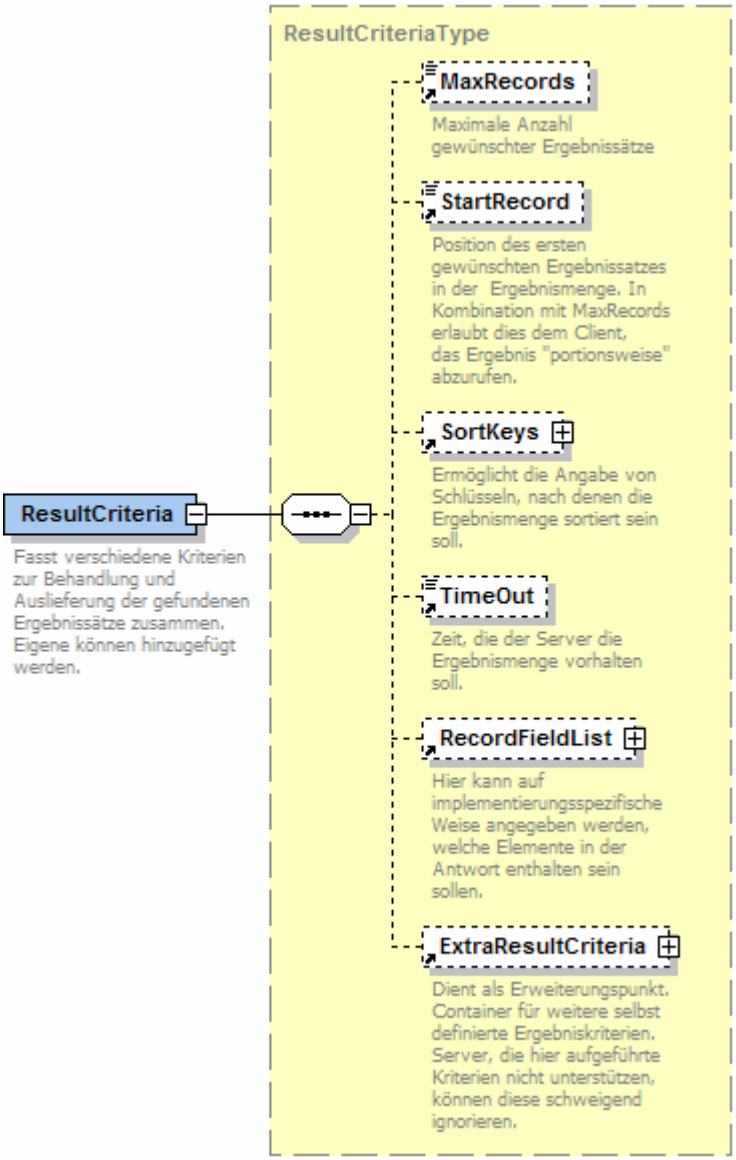
Schemata, die Objekte modellieren, welche Gegenstand von Suchabfragen sein können, sollten daher alle Elemente, die als Suchkriterien verwendbar sein müssen, global deklarieren, damit sie unterhalb des Elementtyps `SearchCriteria` dieses Schemas als Kinder einsetzbar sind.

Da evtl. nicht jedes Schema diese Bedingungen erfüllt, muss auch für die Wildcards, die die Suchkriterien repräsentieren, nicht zwingend ein Schema angegeben werden. Wird jedoch eines angegeben, so muss der Inhalt des Elementtyps `SearchCriteria` valide gegen das Schema sein.

Selbstverständlich muss mindestens ein Suchkriterium angegeben werden. Es können darüber hinaus beliebig viele angegeben werden.

3.4.2. ResultCriteria

Einem `SearchRequest` können im Falle einer *Search by Example* Kriterien mitgegeben werden, die dem Server Informationen liefert, wie der Client die Ergebnisse ausgeliefert haben möchte. Dazu dient der Elementtyp `ResultCriteria`.



Das Element `ResultCriteria` ist optional innerhalb eines `SearchRequest`, d.h. eine Suchanfrage muss keine Ergebniskriterien mitliefern. Dies hat seinen Grund darin, dass auf Einfachheit konzipierte Anwendungen vorstellbar sind, die keine Konfiguration zulassen oder deren Ergebnis immer in ein und derselben Weise zurückgeliefert wird und eine Konfiguration der Ergebnismenge daher nicht nötig ist. Letzteres ist z.B. bei einer SZR-Personenbindungsanfrage der Fall, die zwar eine *Search by Example* ist, aber immer nur ein eindeutiges Ergebnis zurückliefert. Ist das Ergebnis mehrdeutig, wird eine Fehlermeldung zurück gegeben.

Auch die Kindelementtypen des Container-Elementtyps `ResultCriteria` sind alle optional. Gibt ein Client jedoch keine Werte vor, liefert der Server die Ergebnisse mit seinen Standardvorgaben zurück, was evtl. nicht zum gewünschten Resultat führt.

Die Ergebniskriterien bilden die gängigen Parametereinstellungen für Ergebnisse von Suchanfragen ab und sind darüber hinaus um implementierungsspezifische Kriterien erweiterbar. Somit könnten weitere Elementtypen wie z.B. das im ZMR bei der Personensuche verwendete `InclusiveHistorie` eingebunden werden.

Im folgenden sind die bisher vordefinierten Elementtypen zusammengestellt und erläutert:

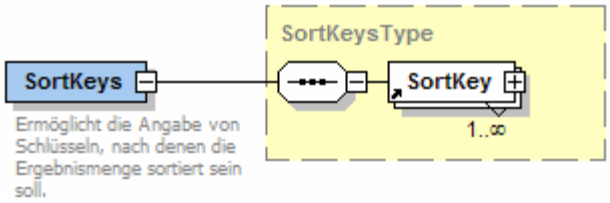
Elementtyp	Datentyp	Erläuterung
`MaxRecords`	`xs:nonNegativeInteger`	Maximale Anzahl gewünschter Ergebnissätze. Evtl. werden gar nicht so viele gefunden, in jedem Falle werden aber nicht mehr als diese Anzahl ausgeliefert. Ist der Wert `0`, sollte der Server keine Ergebnissätze, aber trotzdem die Anzahl der gefundenen Ergebnissätze zurückliefern (`FoundRecords`). Letzteres gilt nicht für Register die dies aus rechtlichen Dingen nicht dürfen.
`StartRecords`	`xs:nonNegativeInteger`	Position des ersten gewünschten Ergebnissatzes in der Ergebnismenge. Der erste Ergebnissatz hat die Position `0`. In Kombination mit `MaxRecords` erlaubt dies dem Client, das Ergebnis "portionsweise" abzurufen.
`SortKeys`	`complexType`	s. nächstes Unterkapitel
`TimeOut`	`xs:nonNegativeInteger`	Zeitspanne in Sekunden, die der Server die Ergebnismenge vorhalten soll. Der Wert `0` zeigt an, dass der Client nicht mehr auf eine vorgehaltene Ergebnismenge zugreifen möchte.
`RecordFieldList`	`complexType`	Hier kann auf implementierungsspezifische Weise angegeben werden, welche Elemente in der Antwort enthalten sein sollen. Auf diese Weise kann eine Art *Short List* definiert werden.
`ExtraResultCriteria`	`complexType`	Dient als Erweiterungspunkt der Ergebniskriterien. Innerhalb dieses

Elementtyp	Datentyp	Erläuterung
		Elementtyps können eigene selbst definierte Ergebniskriterien hinzugefügt werden. Kennt ein Server ein solches Kriterium nicht, sollte er es schweigend ignorieren.

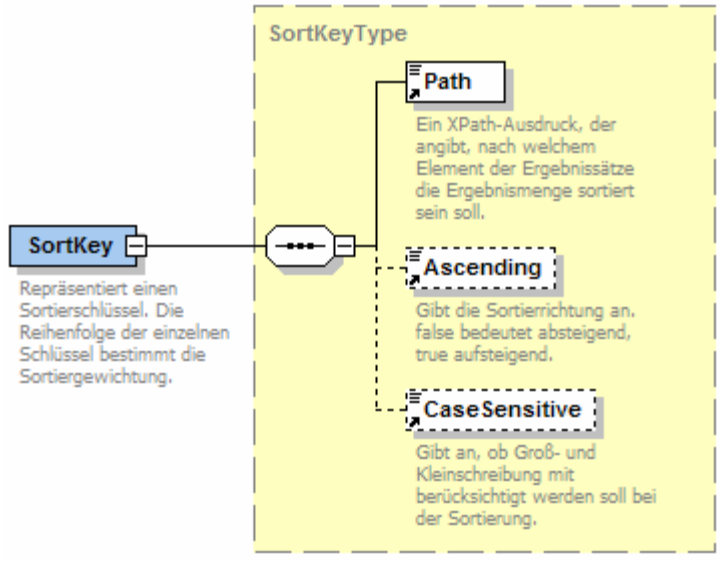
Nutzt ein Client einen der hier vorgegebenen Elementtypen in einer Anfrage und der Server unterstützt dieses Kriterium nicht, so sollte der Server im Response eine entsprechende Message zurücksenden, gefundene Ergebnisse aber gemäß seinem Standard ausliefern.

SortKeys

Der optionale Elementtyp `SortKeys` ermöglicht dem Client, Informationen über die gewünschte Sortierung an den Server zu senden. Gibt der Client keinen Sortierschlüssel an, wird die Sortierreihenfolge vom Server festgelegt.



Zulässig sind mehrere Schlüsselangaben mittels des Elementtyps `SortKey`. Die Reihenfolge der `SortKey`-Elemente gibt dabei die Gewichtung vor. Auf diese Weise kann z.B. eine Ergebnismenge zuerst nach Wohnort und bei gleichem Wohnort nach Familiennamen sortiert werden.



Der Elementtyp `SortKey` ist komplex und fasst alle Angaben zu einem Sortierschlüssel zusammen.

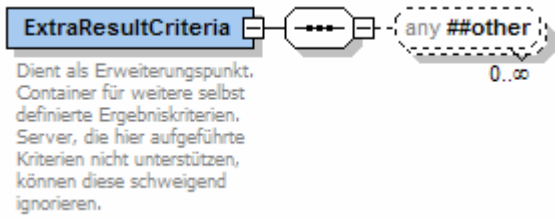
Über den notwendigen Elementtyp `Path` kann mittels eines XPath-Ausdrucks das Element im Ergebnissatz ausgewählt werden, nach dem sortiert werden soll. Soll

z.B. im Falle einer Telefonbuchanfrage nach dem Familiennamen sortiert werden, so müsste (gesetzt die Ergebnissätze gehorchen dem *PersonData*-Schema) der Ausdruck lauten: `/NatuerlichePerson/PersonenName/Familienname`.

Zusätzlich kann mittels des Elementtyps `Ascending` angegeben werden, ob die Sortierung auf- oder absteigend erfolgen soll (Datentyp: `xs:boolean`) sowie festgelegt werden, ob die Sortierung die Groß- und Kleinschreibung berücksichtigen soll (Elementtyp `CaseSensitive`, Datentyp ist `xs:boolean`). Letzteres gilt natürlich nur für Textwerte.

ExtraResultCriteria

Der Elementtyp `ExtraResultCriteria` dient als Erweiterungspunkt für eigene, selbst definierte Ergebniskriterien. Zulässig sind beliebige Elemente aus einem anderen Namensraum als dem Zielnamensraum von XML-Search. Falls ein Schema für diese Elemente angegeben wird, sollte der Inhalt gegen dieses Schema validiert werden.



(3.5) Spezifische Kindelementtypen von SearchById

Der für den Anwendungsfall *Search by Id* zur Verfügung gestellten Elementtyp `SearchById` hat nur einen (notwendigen) spezifischen Kindelementtyp namens `RecordId`, der im folgenden detailliert beschrieben wird.

3.5.1. RecordId

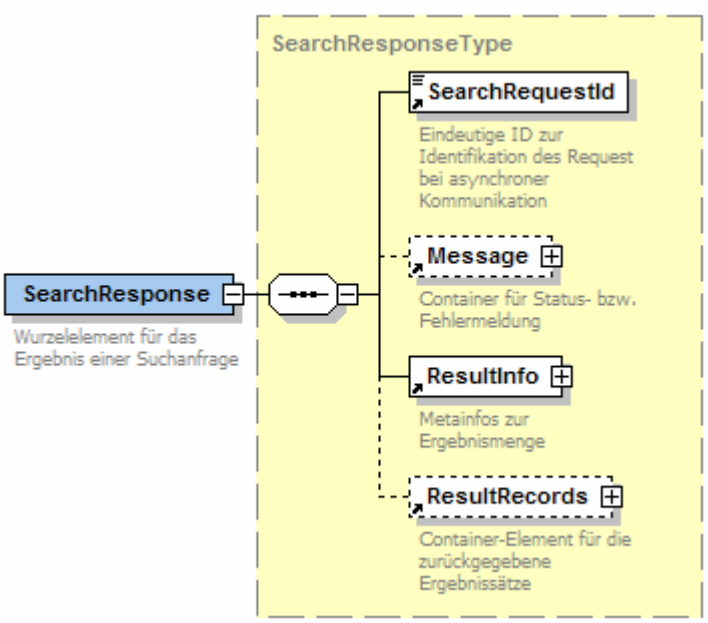
Zulässig sind Werte vom Typ `xs:nonNegativeInteger`. Der Wert des Elements muss übereinstimmen mit dem Wert des `id`-Attributs eines `ResultRecord`-Elements, das in einer vorhergehenden *Search by Example* vom Server zurückgeliefert wurde.

Ergebniskriterien sind bei einer *Search by Id* nicht vorgesehen, da die Ergebnismenge immer eine Detailsicht eines eindeutig durch die ID bestimmten Objekts sein sollte bzw. eine Fehlermeldung im Falle der Nichteindeutigkeit.

(3.6) SearchResponse

Der Elementtyp `SearchResponse` repräsentiert die Antwort des Servers auf eine Suchanfrage. In jedem Falle wird die vom Client beim Request mit gesendete `SearchRequestId` zurückgegeben. Dadurch wird eine Zuordnung der Response bei asynchroner Kommunikation sicher gestellt.

Als Container für die vom Server zurück gelieferten Ergebnisse dient der Elementtyp `ResultRecords` (s. Kapitel 3.6.3). Evtl. notwendige Status- bzw. Fehlermeldungen werden durch den Elementtyp `Message` repräsentiert (s. Kapitel 3.6.1).

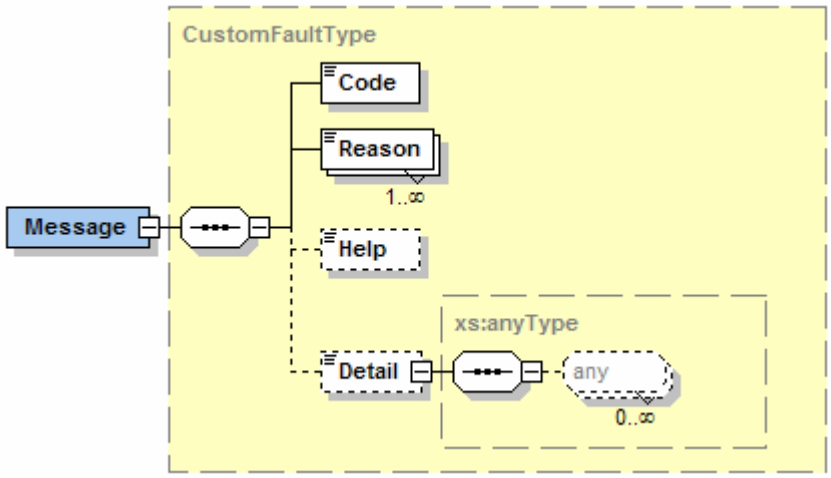


Den Ergebnissätzen muss ein Element `ResultInfo` zur Seite gestellt werden (vgl. Kapitel 3.6.2), das Metadaten zur Ergebnismenge enthält.

3.6.1. Message

Ein `Message`-Element beinhaltet eine Status- oder Fehlermeldung des Servers. **[soap-faults]**Es dient dazu eine Statusmeldung, Warnung oder Fehlermeldung zurückzuliefern zusammen mit gefundenen Ergebnissätzen. Typischerweise entsprechen diese Datensätze dann nicht vollständig den Erwartungen der Clientanwendung, sind z.B. nicht sortiert, weil der Server Sortierung nicht unterstützt. Sind keine vernünftigen Ergebnisse auslieferbar, sollte stattdessen ein SOAP-Fault generiert werden. Dieses Vorgehen ist konform zu **[soap-faults]**.

Das XML-Search-Schema inkludiert das von **[soap-faults]** zur Verfügung gestellte Standard-Schema für solche Fehlermeldungen und referenziert den darin deklarierten Elementtyp `Message`.

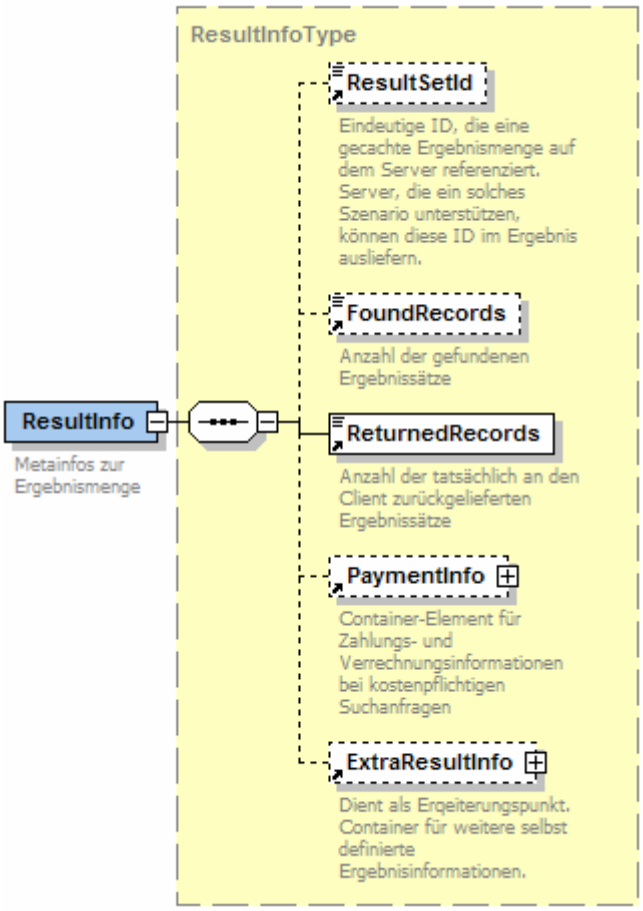


Diese Spezifikation stellt einige vordefinierte Message-Codes zur Verfügung. Es ist jedoch den spezifischen Implementierungen möglich, eigene Message-Codes zu definieren.

Richtlinien für Message-Codes und die von dieser Spezifikation vordefinierten Message-Codes mit Texten finden sich in Kapitel (5) dieser Spezifikation. Für einige Message-Codes ist zusätzlich der empfohlener Inhalt des `Detail`-Elementtyps angegeben.

3.6.2. ResultInfo

Der Elementtyp `ResultInfo` dient der Angabe von zusätzlichen Informationen zur Ergebnismenge, die die Verarbeitung der Ergebnissätze durch den Client vereinfachen, sowie Verrechnungsinformationen für kostenpflichtige Anfragen. Der Elementtyp ist notwendig.



Die Anzahl der gefundenen Datensätze wird über den Elementtyp `FoundRecords`, die Anzahl der tatsächlich an den Client ausgelieferten Ergebnissätze wird im Elementtyp `ReturnedRecords` angegeben. Bei einer *Search by Id* sollten beide den Wert `1` besitzen. War die Suchanfrage ergebnislos sind beide Werte `0`.

Hinweis:

Der Elementtyp `FoundRecords` ist optional, da es Register gibt, die z.B. aus rechtlichen Gründen keine genaue Anzahl von gefundenen Datensätzen zurückliefern (dürfen). Im Regelfall sollte aber die Anzahl gefundener Datensätze immer mitgeliefert werden.

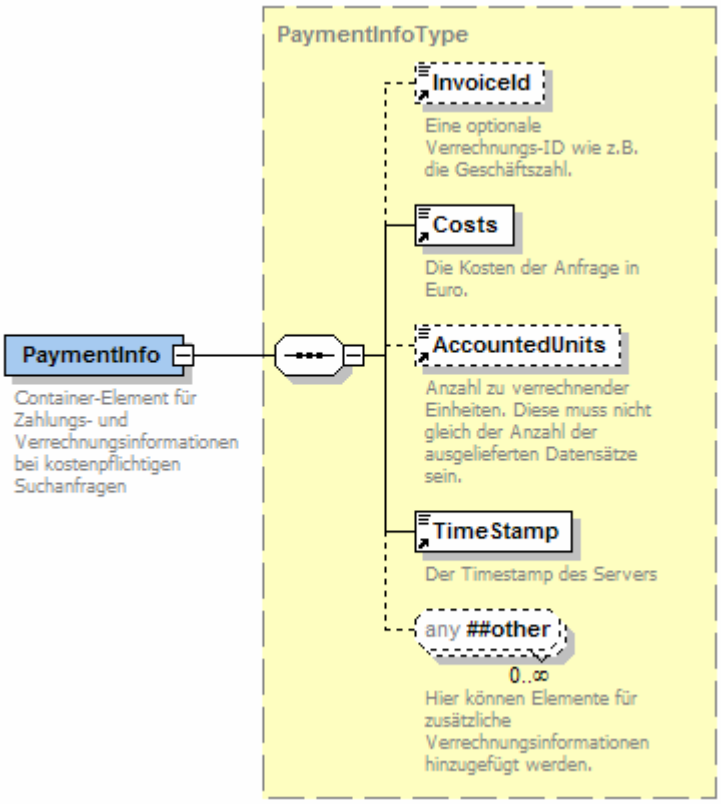
Auf den Elementtyp `PaymentInfo` wird in einem eigene Abschnitt weiter unten eingegangen.

Der Elementtyp `ExtraResultInfo` ist das Pendant zu `ExtraResultCriteria` im Request. Er dient als Erweiterungspunkt für eigene, selbst definierte Ergebnisinformationen und kapselt eine Wildcard.

Zusätzlich kann der Ergebnismenge eine ID vom Server mitgegeben werden. Falls der Server das Cachen von Ergebnismengen unterstützt, kann er eine solche ID ausliefern, auf die der Client dann in einer Folgeanfrage Bezug nehmen kann.

PaymentInfo

Der Elementtyp `PaymentInfo` dient der Aufnahme von Verrechnungsinformationen für Register, die kostenpflichtige Anfragen verarbeiten.



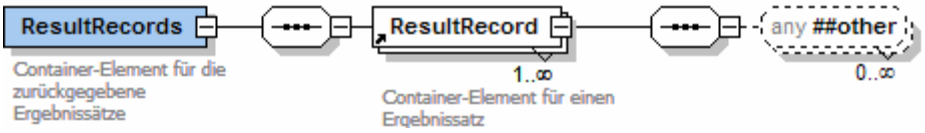
Die folgende Tabelle beschreibt die Kindelelementtypen von `PaymentInfo`:

Elementtyp	Datentyp	Erläuterung
`InvoiceId`	`xs:string`	Dient der optionalen Angabe einer Verrechnungs-ID, wie z.B. der Geschäftszahl.
`Costs`	`xs:float`	Kosten in Euro
`AccountedUnits`	`xs:nonNegativeInteger`	Anzahl der verrechneten Einheiten. Diese Zahl muss nicht gleich der Anzahl ausgegebener Datensätze sein.
`TimeStamp`	`xs:dateTime`	Ein Timestamp des Servers

Eine Wildcard, die beliebig viele Elemente aus anderen Namensräumen als dem Zielnamensraum von XML-Search zuläßt, dient der Erweiterung des Elementtyps `PaymentInfo` um anwendungsspezifische Elementtypen.

3.6.3. ResultRecords

Wenn Ergebnissätze gefunden wurden und der Client mindestens einen zurückgeliefert haben will (`MaxRecords > 0`), dann werden die gefundenen Ergebnissätze in einem Container-Element vom Typ `ResultRecords` an den Client ausgeliefert. Jeder Ergebnissatz ist in einem Element vom Typ `ResultRecord` zu kapseln.



Innerhalb von einem `ResultRecord`-Element sind beliebige Elementtypen aus anderen Namensräumen als dem Zielnamensraum des vorliegenden Schemas zulässig. Falls ein Schema für die Ergebnissätze verfügbar ist, sollte der XML-Prozessor der Anwendung dagegen validieren.

Der Elementtyp `ResultRecord` hat ein optionales Attribut `id` vom Typ `xs:nonNegativeInteger`. Die Liste der Ergebnisse beginnt mit der ID 0, was die Umsetzung in Programmiersprachen erleichtert, deren Listen ja größtenteils o-basiert sind. Über dieses Attribut kann der Server den ausgelieferten Ergebnissätzen eine eindeutige ID zuweisen. Diese kann vom Client dann in einer nach gelagerten *Search by Id* verwendet werden, um z.B. die Details eines bestimmten Datensatzes anzufordern.

Empfehlungen für die Struktur von Ergebnissätzen (ResultRecords)

Die Ergebnissätze und deren Anzahl können in Abhängigkeit von der in der Suchanfrage mitgegebenen Ergebniskriterien unterschiedlich aussehen. Wird z.B. über den Elementtyp `RecordFieldList` eine *Short List* für die Ergebnissätze definiert,, sollte nur ein für eine weitere Einschränkung der Ergebnismenge relevanter Ausschnitt der Detaildatenstruktur pro Ergebnissatz zurückgegeben werden. Wird jedoch keine Einschränkung gemacht, werden alle Datensätze in vollständiger Datenstruktur zurückgeliefert. Die Spezifizierung, welche Elemente eine *Short List* enthalten soll, ist in dieser Version implementierungsabhängig.

Schemata, die Objekte modellieren, welche als Ergebnissätze in einer *Short List* auftauchen können, sollten folgende Voraussetzungen erfüllen:

- Die entsprechenden Objekte müssen global deklariert sein.
- Damit ein Objekt als Ergebnissatz in einer *Short List* verwendet werden kann, muss es alle Elemente seiner Detailstruktur, die in dieser *Short List* nicht ausgeliefert werden, als optional deklarieren (`minOccurs="0"`).

Folgendes Beispiel zeigt eine *Short List*, die `NatuerlichePerson`-Elemente des *PersonData*-Schemas mit Namen und Vornamen als Ergebnis enthält.

```
<ResultRecords>
  <ResultRecord id="0">
    <pers:NatuerlichePerson>
      <pers:PersonenName>
        <pers:Vorname>Max</pers:Vorname>
```

```
        <pers:Familienname>Rilke</pers:Familienname>
      </pers:PersonenName>
    </pers:NatuerlichePerson>
  </ResultRecord>
  <ResultRecord id="1">
    <pers:NatuerlichePerson>
      <pers:PersonenName>
        <pers:Vorname>Max</pers:Vorname>
        <pers:Familienname>Ernst</pers:Familienname>
      </pers:PersonenName>
    </pers:NatuerlichePerson>
  </ResultRecord>
  <ResultRecord id="2">
    <pers:NatuerlichePerson>
      <pers:PersonenName>
        <pers:Vorname>Max</pers:Vorname>
        <pers:Familienname>Hertal</pers:Familienname>
      </pers:PersonenName>
    </pers:NatuerlichePerson>
  </ResultRecord>
  <ResultRecord id="3">
    <pers:NatuerlichePerson>
      <pers:PersonenName>
        <pers:Vorname>Max</pers:Vorname>
        <pers:Familienname>Mustermann</pers:Familienname>
      </pers:PersonenName>
    </pers:NatuerlichePerson>
  </ResultRecord>
  <ResultRecord id="4">
    <pers:NatuerlichePerson>
      <pers:PersonenName>
        <pers:Vorname>Max</pers:Vorname>
        <pers:Familienname>Reichenberg</pers:Familienname>
      </pers:PersonenName>
    </pers:NatuerlichePerson>
  </ResultRecord>
  <ResultRecord id="5">
    <pers:NatuerlichePerson>
      <pers:PersonenName>
        <pers:Vorname>Max</pers:Vorname>
        <pers:Familienname>Schneider</pers:Familienname>
      </pers:PersonenName>
    </pers:NatuerlichePerson>
  </ResultRecord>
</ResultRecords>
```

Beispiel 3: Short List mit komplexen Ergebnisfachobjekten

Im Unterschied zu den Suchkriterien im Request sind die Objektstrukturen der Ergebnissätze im Response Fachobjekte im Sinne von **[xml-g]**, die in Ihrer verschachtelten Struktur ausgegeben werden. Bei einer *Short List* mit Teilen Ihrer Attribute, bei einer *Long List* oder *Search by Id* mit allen befüllten Attributen. Folgendes Beispiel zeigt die Objektstrukturen einer Suchanfrage in Gegenüberstellung zu einem Detailergebnis:

```
<Suchkriterien>
  <pers:PersonenName>
    <pers:Vorname>Max</pers:Vorname>
    <pers:Familienname>Mustermann</pers:Familienname>
  </pers:PersonenName>
</Suchkriterien>
```

Beispiel 4: Flache Struktur der Suchkriterien

```
<ResultRecords>
  <ResultRecord id="0">
    <pers:NatuerlichePerson>
      <pers:PersonenName>
        <pers:Vorname>Rainer</pers:Vorname>
        <pers:MittlererName>Maria</pers:MittlererName>
        <pers:Familienname>Rilke</pers:Familienname>
      </pers:PersonenName>
      <pers:Familienstand>ledig</pers:Familienstand>
      <pers:Geschlecht>männlich</pers:Geschlecht>
      <pers:Geburtsdatum>1875-12-04</pers:Geburtsdatum>
      <pers:Geburtsort>Prag</pers:Geburtsort>
      <pers:Geburtsstaat>Deutschland</pers:Geburtsstaat>
      <pers:Sterbedatum>1926-12-29</pers:Sterbedatum>
      <pers:Staatsangehoerigkeit>
        <pers:ISOCode3>DEU</pers:ISOCode3>
       </pers:Staatsangehoerigkeit>
      <pers:Bekenntnis>katholisch</pers:Bekenntnis>
      <pers:Beruf>Lyriker</pers:Beruf>
    </pers:NatuerlichePerson>
  </ResultRecord>
</ResultRecords>
```

Beispiel 5: Baumartige Fachobjekt-Struktur eines Ergebnissatzes

(4) Beispielablauf einer Suchanfrage (Telefonbuch)

Zur Erläuterung der Schemaspezifikation sei hier an einer klassischen Telefonbuchanfrage die Anwendung des Schemas demonstriert. Aus Gründen der Übersichtlichkeit wird in den folgenden Listings auf Namensraum-Deklarationen und Referenzierung der Schemata verzichtet.

(4.1) Search by Example

Eine einfache Telefonbuchabfrage könnte z.B. über einen Client, der eine Suchmaske zur Eingabe von Familiennamen und Vornamen als Suchkriterium zur Verfügung stellt, ablaufen.

Der Client-Applikation könnte folgenden zum vorliegenden allgemeinen und *PersonData*-Schema konformen Request absenden. Dieser stößt eine Suche nach allen Personen an, deren Vorname "Max" ist. Sortiert werden soll nach dem Familiennamen der gefundenen Personen und zwar aufsteigend.

```
<?xml version="1.0" encoding="UTF-8"?>
<SearchByExample>
  <SearchRequestId>21372132</SearchRequestId>
  <SearchRequestInfo>
    <base:Organisation>
      <base:BehoerdenNr>000000</base:BehoerdenNr>
    </base:Organisation>
  </SearchRequestInfo>
  <ResultCriteria>
    <MaxRecords>6</MaxRecords>
    <StartRecord>1</StartRecord>
    <SortKeys>
      <SortKey>
        <Path>
          /NatuerlichePerson/PersonenName/Familienname
        </Path>
        <Ascending>true</Ascending>
      </SortKey>
    </SortKeys>
    <TimeOut>1800</TimeOut>
    <RecordFieldList>
      <tel:Field>
        /NatuerlichePerson/PersonenName/Familienname
      </tel:Field>
      <tel:Field>
        /NatuerlichePerson/PersonenName/Vorname
      </tel:Field>
    </RecordFieldList>
  </ResultCriteria>
  <SearchCriteria>
    <pers:PersonenName>
      <pers:Vorname>Max</pers:Vorname>
    </pers:PersonenName>
  </SearchCriteria>
</SearchByExample>
```

Eine Suche nur nach dem Vornamen "Max" findet potentiell mehrere Personen. Ausgeliefert werden jedoch maximal 6 (<MaxRecords>6</MaxRecords>). Im Element RecordFieldList ist zudem angegeben, welche Elemente für die Personenobjekte der Ergebnisliste ausgeliefert werden sollen, also im Grunde eine Art Short List definiert.

Ein gültiges Ergebnis könnte folgendermaßen aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<SearchResponse>
  <SearchRequestId>21372132</SearchRequestId>
  <ResultInfo>
    <FoundRecords>28</FoundRecords>
    <ReturnedRecords>6</ReturnedRecords>
  </ResultInfo>
  <ResultRecords>
    <ResultRecord id="0">
      <pers:NatuerlichePerson>
        <pers:PersonenName>
          <pers:Vorname>Max</pers:Vorname>
          <pers:Familienname>Rilke</pers:Familienname>
        </pers:PersonenName>
      </pers:NatuerlichePerson>
    </ResultRecord>
    <ResultRecord id="1">
      <pers:NatuerlichePerson>
        <pers:PersonenName>
          <pers:Vorname>Max</pers:Vorname>
          <pers:Familienname>Ernst</pers:Familienname>
        </pers:PersonenName>
      </pers:NatuerlichePerson>
    </ResultRecord>
    <ResultRecord id="2">
      <pers:NatuerlichePerson>
        <pers:PersonenName>
          <pers:Vorname>Max</pers:Vorname>
          <pers:Familienname>Hertal</pers:Familienname>
        </pers:PersonenName>
      </pers:NatuerlichePerson>
    </ResultRecord>
    <ResultRecord id="3">
      <pers:NatuerlichePerson>
        <pers:PersonenName>
          <pers:Vorname>Max</pers:Vorname>
          <pers:Familienname>Mustermann</pers:Familienname>
        </pers:PersonenName>
      </pers:NatuerlichePerson>
    </ResultRecord>
    <ResultRecord id="4">
      <pers:NatuerlichePerson>
        <pers:PersonenName>
          <pers:Vorname>Max</pers:Vorname>
          <pers:Familienname>Reichenberg</pers:Familienname>
        </pers:PersonenName>
      </pers:NatuerlichePerson>
```

```
    </ResultRecord>
    <ResultRecord id="5">
      <pers:NatuerlichePerson>
        <pers:PersonenName>
          <pers:Vorname>Max</pers:Vorname>
          <pers:Familienname>Schneider</pers:Familienname>
        </pers:PersonenName>
      </pers:NatuerlichePerson>
    </ResultRecord>
  </ResultRecords>
</SearchResponse>
```

Die Ergebnisliste beinhaltet im Beispiel 28 Sätze, von den 6 zurückgeliefert werden. Die zurückgegebenen Objekte sind mit Vorname und Familienname befüllte `NatuerlichePerson`-Fachobjekte, die im Unterschied zu den Suchkriterien baumartig verschachtelt sind. Diese ausgelieferte XML-Struktur könnte im Client als Liste von Vor- und Nachnamen abgebildet werden.

Hinweis:

Das Beispiel zeigt zudem einen möglichen Weg, die Felder für eine Short-List zu definieren. Ähnlich wie bei der Angabe der Felder, nach denen sortiert werden soll (Elementtyp `SortKey`), könnten hier XPath-Ausdrücke zum Einsatz kommen.

(4.2) Search by Id

Der Benutzer könnte nun den passenden Ergebnissatz selektieren und der Client daraufhin für diesen Ergebnissatz Detaildaten anfordern. Request und Response einer solchen *Search by Id* könnten dann wie folgt aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<SearchById>
  <SearchRequestId>21372133</SearchRequestId>
  <SearchRequestInfo>
    <base:Organisation>
      <base:BehoerdenNr>000000</base:BehoerdenNr>
    </base:Organisation>
  </SearchRequestInfo>
  <RecordId>1</RecordId>
</SearchById>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<SearchResponse>
  <SearchRequestId>21372133</SearchRequestId>
  <ResultInfo>
    <FoundRecords>1</FoundRecords>
    <ReturnedRecords>1</ReturnedRecords>
  </ResultInfo>
  <ResultRecords>
    <ResultRecord id="0">
      <pers:NatuerlichePerson>
        <pers:PersonenName>
          <pers:Vorname>Rainer</pers:Vorname>
          <pers:MittlererName>Maria</pers:MittlererName>
          <pers:Familienname>Rilke</pers:Familienname>
```

```
        </pers:PersonenName>
        <pers:Familienstand>ledig</pers:Familienstand>
        <pers:Geschlecht>männlich</pers:Geschlecht>
        <pers:Geburtsdatum>1875-12-04</pers:Geburtsdatum>
        <pers:Geburtsort>Prag</pers:Geburtsort>
        <pers:Geburtsstaat>Deutschland</pers:Geburtsstaat>
        <pers:Sterbedatum>1926-12-29</pers:Sterbedatum>
        <pers:Staatsangehoerigkeit>
          <pers:ISOCode3>DEU</pers:ISOCode3>
        </pers:Staatsangehoerigkeit>
        <pers:Bekenntnis>katholisch</pers:Bekenntnis>
        <pers:Beruf>Lyriker</pers:Beruf>
      </pers:NatuerlichePerson>
    </ResultRecord>
  </ResultRecords>
</SearchResponse>
```

(5) Message-Codes

Die Fehlerbehandlung von XML-Search entspricht der in **[soap-faults]** vorgeschlagenen. Für allgemeine Server-Fehler sind die dort definierten Message-Codes zu verwenden. Für XML-Search-spezifische Fehler sind die hier vordefinierten Fehlercodes zu verwenden.

Message-Codes sind in Klassen eingeteilt und angelehnt an die Status-Code-Definitionen des HTTP-1.0-Protokolls (s. **[http-1.0]**), allerdings vierstellig (gemäß **[soap-faults]**). Alle Message-Codes sind mit Standard-Texten verknüpft. Einige Messages werden in dieser Spezifikation vordefiniert, innerhalb der Klassen kann jedoch jede Implementierung eigene Message-Codes und korrespondierende Texte definieren. Dies sollte ausschließlich in der Message-Klasse 6 geschehen.

(5.1) Message-Klassen

Die Klasse eines vierstelligen Message-Codes wird durch die erste Ziffer bestimmt. Folgende Message-Klassen sind in Anlehnung an die Status-Code-Definitionen des HTTP-1.0-Protokolls vordefiniert:

Message-Klasse	Bedeutung
2	Aktion erfolgreich ausgeführt
3	Zur erfolgreichen Ausführung sind weitere Aktionen notwendig
4	Client-Fehler
5	Server-Fehler
6	Von Implementierungen definierte Fehler

(5.2) Message-Codes und Message-Texte

Innerhalb der vordefinierten Message-Klassen stellt diese Spezifikation die in der folgenden Tabelle vordefinierten Message-Codes zur Verfügung. In der letzten Spalte bedeutet ein "F", das die Message als SOAP-Fault zu werfen ist, während ein "X" Messages bezeichnet, bei denen noch Ergebnisse mit ausgeliefert werden können und daher der `Message`-Elementtyp zur Rückgabe der Fehlermeldung verwendet werden kann. In der Spalte *Detail* ist angegeben, welchen Inhalt das `Detail`-Element der XML-Fehlermeldung haben sollte. Der Detail-Text kann nicht direkt als Inhalt eines SOAP-Fault-`detail`-Elements auftauchen, da dort nur Elementinhalt zulässig ist. Aus diesem Grunde sollte der Text in das für diese Aufgabe global deklarierte Element `FaultHint` von XML-Search eingebettet werden.

Nr.	Text	Bedeutung	Detail	
2040	No records found	Die Suchanfrage wurde erfolgreich ausgeführt, Ergebnisse wurden aber keine gefunden.		X
4010	Required search criteria missing	Es fehlt ein für eine vollständige Suchanfrage benötigtes Suchkriterium.	Name des Suchkriteriums	S
4020	Start record position out of range	Die angeforderte Position für den ersten Ergebnissatz einer Suchanfrage ist größer als die Position des letzten Ergebnissatzes der Ergebnismenge.		S
4021	Specified number of MaxRecords too large	Die vom Client angeforderte Anzahl maximaler Ergebnisse ist zu hoch. Die Ergebnisse können bis zur möglichen maximalen Anzahl ausgeliefert werden.	Maximale Anzahl auslieferbarer Ergebnissätze.	X
4030	Result too large to send	Die Ergebnismenge übersteigt die mögliche bzw. festgelegte Übertragungsgröße. Die Ergebnisse können bis zur möglichen maximalen Anzahl ausgeliefert werden.	Maximale Ergebnisgröße	X
4040	Sorting is not supported	Der Server unterstützt keine Sortierung. Die Ergebnisse können unsortiert ausgeliefert werden.		X
4041	Too many records to sort.	Die Ergebnismenge übersteigt die mögliche bzw. festgelegte Größe für eine Sortierung. Die Ergebnisse können unsortiert ausgeliefert werden.	Maximale Anzahl Ergebnissätze, die sortiert werden kann.	X
4042	The provided sort key is not supported	Das gewünschte Sortierkriterium wird nicht unterstützt. Die Ergebnisse können unsortiert ausgeliefert werden.	XPath-Ausdruck des nicht unterstützten Suchkriteriums	X
4050	Unsupported search criteria	Das eingesetzte Suchkriterium wird vom Server nicht unterstützt.	Name des Suchkriteriums	S
4052	Required search criteria not supplied	Ein notwendiges Suchkriterium wurde nicht angegeben.	Name des fehlenden Suchkriteriums	S
4060	Caching of result sets not supported	Das Vorhalten von Ergebnismengen auf dem Server wird nicht unterstützt.		S

Nr.	Text	Bedeutung	Detail	
4061	ResultSetId doesn't exist	Die vom Client gesendete ResultSetId existiert nicht (mehr).	ResultSetId	S
4062	Time out too long	Der vom Client angeforderte TimeOut für die auf dem Server vorgehaltene Ergebnismenge ist zu lang. Ergebnisse werden trotzdem ausgeliefert.	Maximale Time-Out-Zeit.	X

Zukünftige Versionen von XML-Search können vordefinierte Message-Codes hinzufügen.

Hinweis:

Damit die obigen Fehlercodes in einem SOAP-Fault als Inhalt des `faultcode`-Elements verwendet werden können, müssen Sie mit einem Buchstaben beginnen, da sie sonst nicht dem SOAP-Schema genügen, das als Inhalt des `faultcode`-Elements einen Qualified Name fordert. Ein solcher darf jedoch nicht mit einer Ziffer beginnen. Daher sollte ein Fehlercode mit dem Präfix `F` versehen werden, wenn er in einem SOAP-Fault verwendet wird.

Hinweis:

*Laut **[soap-faults]** sind die Fehlercodes, wenn Sie in einem SOAP-Fault als Inhalt des `faultcode`-Elements verwendet werden, mit dem Namenraumpräfix für das Schema der Anwendung (hier also XML-Search) zu versehen. Wurde z.B. das Präfix `sw` mit dem XML-Search-Namensraum verknüpft, so sähe der Inhalt des `faultcode`-Elements für den Fehlercode 4010 wie folgt aus: `sw:F4010`.*

(5.3) Beispiele für Messages

Im folgenden wird jeweils ein Beispiel für eine Fehlermeldung als SOAP-Fault und für eine XML-Fehlermeldung gegeben.

5.3.1. SOAP-Fault

Folgendes Listing zeigt eine vollständige SOAP-Fault-Response für die Message mit dem Fehlercode 4010. Der SOAP-Fault wird geworfen, da ein notwendiges Suchkriterium fehlte und somit keine sinnvollen Ergebnisse ausgeliefert werden können.

```
<env:Envelope
  xmlns:sw="http://reference.e-government.gv.at/namespace/xml-sw/1#"
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  ../xsd/specific/envelope.xsd">
  <env:Body>
    <env:Fault>
      <faultcode>sw:F4010</faultcode>
      <faultstring>
        Required search criteria
        missing
      </faultstring>
      <detail>
        <sw:FaultHint>Familienname</sw:FaultHint>
      </detail>
    </env:Fault>
```

```
  </env:Body>
</env:Envelope>
```

5.3.2. XML-Fehlermeldung mit Message-Elementtyp

Folgendes Beispiel zeigt die XML-Fehlermeldung für den Message-Code 4040. Die Message besagt, dass der Request, der eine Sortierung der Ergebnismenge angefordert hatte, zwar bearbeitet werden konnte, jedoch das Feature Sortierung nicht unterstützt wird vom Server.

```
<Message>
  <Code>4040</Code>
  <Reason>Sorting is not supported</Reason>
</Message>
```

(6) Konventionen für Wildcarding

Mindestens sollten Implementierungen die folgenden beiden Wildcards unterstützen:

*****	Maskiert 0 oder mehr Zeichen
?	Maskiert genau ein Zeichen.

Für Nutzung weitergehender Wildcarding-Konventionen der Implementierung wird die Orientierung an einer Untermenge der in Perl üblichen regulären Ausdrücke, wie sie in der W3C-XML-Schema-Empfehlung definiert ist, empfohlen (s. **[wxs-2]**, Anhang F).

(7) Die Webservice-Schnittstelle

Diese Spezifikation stellt über das beschriebene Schema hinaus ein WSDL-Dokument zur Verfügung, das den logischen Teil der Webservice-Schnittstelle ausdefiniert.

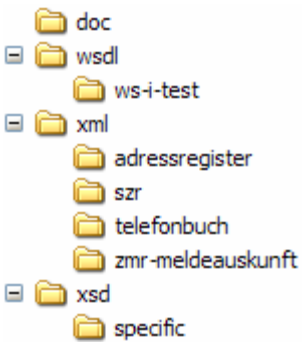
Das WSDL-Dokument ist streng nach den Richtlinien aus **[wsi-1]** gestaltet. Es wurde gegen die Test Suite der WS-I-Organisation getestet. Der Testbericht liegt wie auch das WSDL-Dokument dem Gesamptpaket dieser Spezifikation bei.

Zudem wurde die Generierung von Client-Klassen aus dem beigefügten WSDL mit verschiedenen Code-Generatoren aus der Java- und .NET-Welt erfolgreich getestet.

In Anhang C (Kapitel (9)) findet sich ein Listing des WSDL-Dokuments.

(8) Anhang A: Mit der Spezifikation ausgelieferte Dateien

Das Gesamtpaket dieser Spezifikation hat folgende Verzeichnisstruktur:



Das Verzeichnis *doc* enthält dieses Dokument (*xml-sw.doc*).

Das Verzeichnis *wsdl* enthält das WSDL-Dokument (*XmlSwService.wsdl*).

Das Verzeichnis *wsdl/ws-i-test* enthält folgende Dateien:

- Bericht des WSDL-Test auf Konformität zu **[wsi-1]** in XML (*report.xml*)
- Stylesheets zur Transformation des XML-Berichts in HTML (*report.xsl* und *common.xsl)*.

Der Report kann direkt im Internet Explorer oder Mozilla durch Öffnen der Datei *report.xml* angesehen werden.

Das Verzeichnis *xml* enthält verschiedene Beispiel-Instanzdokumente zur Demonstration des W3C XML Schemas für XML-Search. Im einzelnen werden folgende Beispiele zur Verfügung gestellt:

- Adressregister-Use-Cases (Verzeichnis *adressregister*)
- Personenbindungsanfrage Stammzahlenregister nach **[spec-szr]** (Verzeichnis *szr*)
- Fiktive Telefonbuchanfrage, die in Kapitel (4) dieses Dokuments ausführlich beschrieben wird (Verzeichnis *telefonbuch*)
- Meldeauskunft des Zentralen Melderegisters (Verzeichnis *zmr-meldeauskunft*)

Das Verzeichnis *xsd* enthält das W3C XML Schema für XML-Search (*xml-sw.xsd*). Das Verzeichnis *xsd/specific* enthält verschiedene Schemata, die von XML-Search importiert werden oder zur Validierung der Beispieldateien dienen.

(9) Anhang B: W3C XML-Schema

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

<!--

************************************************

File      xml-sw.xsd

Version   1.0 / 23.03.2004

Author    Franz-Josef Herpers / fjh consulting

Copyright (c) 2004, 2005 Bundeskanzleramt Österreich

************************************************

-->
<xs:schema
  targetNamespace="http://reference.e-
government.gv.at/namespace/xml-sw/1#"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://reference.e-government.gv.at/namespace/xml-
sw/1#"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="1.0.0">

  <xs:include  schemaLocation="specific/CustomFault.xsd"/>

  <!-- Element Declarations (complex) -->
  <xs:element name="SearchByExample"
    type="SearchByExampleType">
    <xs:annotation>
      <xs:documentation>Wurzelelement für Suchabfragen vom
        Typ "Search by Example"</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="SearchById" type="SearchByIdType">
    <xs:annotation>
      <xs:documentation>Wurzelelement für Suchabfragen vom
        Typ "Search by Id"</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="SearchRequestId" type="xs:string">
    <xs:annotation>
      <xs:documentation>Eindeutige ID zur Identifikation des
        Request bei asynchroner
      Kommunikation</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="SearchRequestInfo"
    type="SearchRequestInfoType">
    <xs:annotation>
```

```
      <xs:documentation>Container für alle Metainformationen
        einer Suchanfrage</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="SortKeys" type="SortKeysType">
    <xs:annotation>
      <xs:documentation>Ermöglicht die Angabe von
        Schlüsseln, nach denen die Ergebnismenge sortiert
        sein soll.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="SearchResponse"
    type="SearchResponseType">
    <xs:annotation>
      <xs:documentation>Wurzelelement für das Ergebnis einer
        Suchanfrage</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="ResultInfo" type="ResultInfoType">
    <xs:annotation>
      <xs:documentation>Metainfos zur
      Ergebnismenge</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="RecordId" type="xs:nonNegativeInteger">
    <xs:annotation>
      <xs:documentation>ID eines Datensatzes. Sie muss
        übereinstimmen mit der ID eines Datensatzes (Wert
        des id-Attributs eines ResultRecord-Elements), den
        der Server in einem vorhergehenden Request
        zurückgeliefert hat.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="ResultRecords">
    <xs:annotation>
      <xs:documentation>Container-Element für die
        zurückgegebene Ergebnissätze</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ResultRecord" maxOccurs="unbounded"
        />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="SearchCriteria"
    type="SearchCriteriaType">
    <xs:annotation>
      <xs:documentation>Container-Element für die
        Suchkriterien. Diese sind applikationsspezifisch.
        Zulässig sind daher beliebige Elemente aus anderen
        Namensräumen. Optional kann eine ResultSetId
        übermittelt werden, falls der Server das Vorhalten
        von vorherigen Abfrageergebnissen unterstützt und in
```

```
        der initialen Response eine ResultSetId mitgesendet
        hat.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="ResultCriteria"
    type="ResultCriteriaType">
    <xs:annotation>
      <xs:documentation>Fasst verschiedene Kriterien zur
        Behandlung und Auslieferung der gefundenen
        Ergebnissätze zusammen. Eigene können hinzugefügt
        werden.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="SortKey" type="SortKeyType">
    <xs:annotation>
      <xs:documentation>Repräsentiert einen
        Sortierschlüssel. Die Reihenfolge der einzelnen
        Schlüssel bestimmt die
      Sortiergewichtung.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="ExtraResultCriteria">
    <xs:annotation>
      <xs:documentation>Dient als Erweiterungspunkt.
        Container für weitere selbst definierte
        Ergebniskriterien. Server, die hier aufgeführte
        Kriterien nicht unterstützen, können diese
        schweigend ignorieren.</xs:documentation>
    </xs:annotation>
    <xs:complexType mixed="false">
      <xs:sequence>
        <xs:any namespace="##any" processContents="lax"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ExtraResultInfo">
    <xs:annotation>
      <xs:documentation>Dient als Erqeiterungspunkt.
        Container für weitere selbst definierte
        Ergebnisinformationen.</xs:documentation>
    </xs:annotation>
    <xs:complexType mixed="false">
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ResultRecord">
    <xs:annotation>
      <xs:documentation> Container-Element für einen
        Ergebnissatz</xs:documentation>
    </xs:annotation>
```

```
    <xs:complexType>
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:nonNegativeInteger"
        use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="PaymentInfo" type="PaymentInfoType">
    <xs:annotation>
      <xs:documentation>Container-Element für Zahlungs- und
        Verrechnungsinformationen bei kostenpflichtigen
        Suchanfragen</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="RecordFieldList">
    <xs:annotation>
      <xs:documentation>Hier kann auf
        implementierungsspezifische Weise angegeben werden,
        welche Elemente in der Antwort enthalten sein
        sollen.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- Element Declarations (simple) -->
  <xs:element name="TimeOut" type="xs:nonNegativeInteger">
    <xs:annotation>
      <xs:documentation>Zeit, die der Server die
        Ergebnismenge vorhalten soll</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="MaxRecords"
type="xs:nonNegativeInteger">
    <xs:annotation>
      <xs:documentation>Maximale Anzahl gewünschter
        Ergebnissätze </xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="StartRecord"
    type="xs:nonNegativeInteger">
    <xs:annotation>
      <xs:documentation>Position des ersten gewünschten
        Ergebnissatzes in der Ergebnismenge. In Kombination
        mit MaxRecords erlaubt dies dem Client, das Ergebnis
        "portionsweise" abzurufen.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="Path" type="xs:string">
```

```
    <xs:annotation>
      <xs:documentation>Ein XPath-Ausdruck, der angibt, nach
        welchem Element der Ergebnissätze die Ergebnismenge
        sortiert sein soll.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="Ascending" type="xs:boolean">
    <xs:annotation>
      <xs:documentation>Gibt die Sortierrichtung an. false
        bedeutet absteigend, true
      aufsteigend.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="CaseSensitive" type="xs:boolean">
    <xs:annotation>
      <xs:documentation>Gibt an, ob Groß- und
        Kleinschreibung mit berücksichtigt werden soll bei
        der Sortierung.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="ResultSetId" type="xs:string">
    <xs:annotation>
      <xs:documentation>Eindeutige ID, die eine gecachte
        Ergebnismenge auf dem Server referenziert. Server,
        die ein solches Szenario unterstützen, können diese
        ID im Ergebnis ausliefern.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="FoundRecords"
    type="xs:nonNegativeInteger">
    <xs:annotation>
      <xs:documentation>Anzahl der gefundenen Ergebnissätze
      </xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="ReturnedRecords"
    type="xs:nonNegativeInteger">
    <xs:annotation>
      <xs:documentation> Anzahl der tatsächlich an den
        Client zurückgelieferten Ergebnissätze
      </xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="Costs" type="xs:float">
    <xs:annotation>
      <xs:documentation>Die Kosten der Anfrage in
      Euro</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="InvoiceId" type="xs:string">
    <xs:annotation>
      <xs:documentation>Eine optionale Verrechnungs-ID wie
        z.B. die Geschäftszahl</xs:documentation>
    </xs:annotation>
```

```
  </xs:element>
  <xs:element name="AccountedUnits"
    type="xs:nonNegativeInteger">
    <xs:annotation>
      <xs:documentation>Anzahl zu verrechnender Einheiten.
        Diese muss nicht gleich der Anzahl der
        ausgelieferten Datensätze sein.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="TimeStamp" type="xs:dateTime">
    <xs:annotation>
      <xs:documentation>Der Timestamp des Servers
      </xs:documentation>
    </xs:annotation>
  </xs:element>
  <!-- Complex Types -->
  <xs:complexType name="SearchByExampleType">
    <xs:sequence>
      <xs:element ref="SearchRequestId"/>
      <xs:element ref="SearchRequestInfo" minOccurs="0"/>
      <xs:element ref="ResultCriteria" minOccurs="0"/>
      <xs:element ref="SearchCriteria"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SearchByIdType">
    <xs:sequence>
      <xs:element ref="SearchRequestId"/>
      <xs:element ref="SearchRequestInfo" minOccurs="0"/>
      <xs:element ref="RecordId"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ResultInfoType">
    <xs:sequence>
      <xs:element ref="ResultSetId" minOccurs="0"/>
      <xs:element ref="FoundRecords" minOccurs="0"/>
      <xs:element ref="ReturnedRecords"/>
      <xs:element ref="PaymentInfo" minOccurs="0"/>
      <xs:element ref="ExtraResultInfo" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SearchResponseType">
    <xs:sequence>
      <xs:element ref="SearchRequestId"/>
      <xs:element ref="Message" minOccurs="0"/>
      <xs:element ref="ResultInfo"/>
      <xs:element ref="ResultRecords" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SearchRequestInfoType">
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

```
  <xs:complexType name="SortKeyType">
    <xs:sequence>
      <xs:element ref="Path"/>
      <xs:element ref="Ascending" minOccurs="0"/>
      <xs:element ref="CaseSensitive" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ResultCriteriaType">
    <xs:sequence>
      <xs:element ref="MaxRecords" minOccurs="0"/>
      <xs:element ref="StartRecord" minOccurs="0"/>
      <xs:element ref="SortKeys" minOccurs="0"/>
      <xs:element ref="TimeOut" minOccurs="0"/>
      <xs:element ref="RecordFieldList" minOccurs="0"/>
      <xs:element ref="ExtraResultCriteria" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SearchCriteriaType">
    <xs:sequence>
      <xs:element ref="ResultSetId" minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SortKeysType">
    <xs:sequence>
      <xs:element ref="SortKey" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="PaymentInfoType">
    <xs:sequence>
      <xs:element ref="InvoiceId" minOccurs="0"/>
      <xs:element ref="Costs"/>
      <xs:element ref="AccountedUnits" minOccurs="0"/>
      <xs:element ref="TimeStamp"/>
      <xs:any namespace="##other" processContents="lax"
        minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Hier können Elemente für
            zusätzliche Verrechnungsinformationen
            hinzugefügt werden.</xs:documentation>
        </xs:annotation>
      </xs:any>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

(10) Anhang C: WSDL

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

<!--

*****************************************************
File       XmlSwService.wsdl

Version   1.0 / 23.03.2004

Author    Franz-Josef Herpers / fjh consulting

Copyright (c) 2004, 2005 Bundeskanzleramt Österreich

*****************************************************

-->
<wsdl:definitions name="Search"
  targetNamespace="urn:gv:at:search"
  xmlns:tns="urn:gv:at:search"
  xmlns:sw="http://reference.e-
government.gv.at/namespace/xml-sw/1#"
  xmlns:soapbinding="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <wsdl:documentation> Dieses WSDL-Dokument beschreibt den
    XML-Search-Service. </wsdl:documentation>
  <wsdl:types>
    <xsd:schema>
      <xsd:import
        namespace="http://reference.e-
government.gv.at/namespace/xml-sw/1#"
        schemaLocation="../xsd/xml-sw.xsd"/>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="SearchByExample">
    <wsdl:documentation>
      Nachricht, die für eine Suchanfrage
      vom Typ Search by Example vom Client abgesendet wird
    </wsdl:documentation>
    <wsdl:part name="body" element="sw:SearchByExample"/>
  </wsdl:message>
  <wsdl:message name="SearchById">
    <wsdl:documentation> Nachricht, die für eine Suchanfrage
      vom Typ Search by Id vom Client abgesendet wird
    </wsdl:documentation>
    <wsdl:part name="body" element="sw:SearchById"/>
  </wsdl:message>
  <wsdl:message name="SearchResponse">
    <wsdl:documentation> Nachricht, die vom Server als
      Antwort auf eine der beiden Suchanfragen
```

```
      zurückgesendet wird
    </wsdl:documentation>
    <wsdl:part name="body" element="sw:SearchResponse"/>
  </wsdl:message>
  <wsdl:portType name="Search">
    <wsdl:operation name="searchByExample">
      <wsdl:documentation> Operation für den Abfragetyp
        Search by Example
      </wsdl:documentation>
      <wsdl:input message="tns:SearchByExample"/>
      <wsdl:output message="tns:SearchResponse"/>
    </wsdl:operation>
    <wsdl:operation name="searchById">
      <wsdl:documentation> Operation für den Abfragetyp
        Search by Id
      </wsdl:documentation>
      <wsdl:input message="tns:SearchById"/>
      <wsdl:output message="tns:SearchResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SearchServiceSoapBinding"
    type="tns:Search">
    <wsdl:documentation> Definition des Binding für den
      Search Service
    </wsdl:documentation>
    <soapbinding:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="searchByExample">
      <soapbinding:operation
        soapAction="http://localhost:8090/xml-
sw/SearchService/searchByExample"
        style="document"/>
      <wsdl:input>
        <soapbinding:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soapbinding:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="searchById">
      <soapbinding:operation
        soapAction="http://localhost:8090/xml-
sw/SearchService/searchById"
        style="document"/>
      <wsdl:input>
        <soapbinding:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soapbinding:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="SearchService">
    <wsdl:port binding="tns:SearchServiceSoapBinding"
```

```
      name="SearchService">
      <soapbinding:address
        location="http://localhost:8090/xml-
sw/SearchService"
      />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

(11) Referenzen

[fxpp] Goland, Yaron Y; Schlimmer, Jeffrey C.. 2000. "Flexible XML Processing Profile (FXPP)". UPnP Forum Technical Committee. Internet: *http://www.upnp.org/download/draft-goland-fxpp-01.txt*.

[http-1.0] Berners-Lee, T.; Fielding; R.; Frystyk, H. 1996. " Hypertext Transfer Protocol -- HTTP 1.0". RFC 1945. Internet: *http://www.ietf.org/rfc/rfc1945.txt*.

[http-1.1] Fielding R. [u.a.]. 1999. "Hypertext Transfer Protocol -- HTTP/1.1". RFC 2616. Internet: *http://www.ietf.org/rfc/rfc2616.txt*.

[nsp-1.0] Bray, Tim; Hollander, Dave; Layman, Andrew. 1999. "Namespaces in XML". W3C Recommendation. Internet: *http://www.w3.org/TR/1999/REC-xml-names-19990114/*.

[persondata2] Naber, Larissa. 2004. "PersonData Struktur: XML Spezifikation". AG Kommunikationsarchitektur. Internet: *http://reference.e-government.gv.at/*.

[rm] Hörbe, Rainer. 2004. Release-Management für Anwendungen: Best Practice. AG Kommunikationsarchitektur. Internet: *https://w4.wien.gv.at/Groups/externe/e-government/kommarch/ergebnisdokumente/release_management/index_html*.

[soap-1.1] Box, Don [u.a.]. 2000. "Simple Object Access Protocol (SOAP) 1.1". W3C Note. Internet: *http://www.w3.org/TR/2000/NOTE-SOAP-20000508/*.

[soap-faults] Liehmann, Michael. 2005. SOAP-Faults und deren Behandlung im Rahmen der Kommunikationsarchitektur des Bundes.

[spec-szr] Hörbe, Rainer. 2004. "Spezifikation Stammzahlen-Register (SZR-N)". BMI. Internet: *https://portal.bmi.gv.at/ref/szr/specszr.pdf*.

[uri] Berners-Lee, T. 1998. "Uniform Resource Identifiers (URI): Generic Syntax". RFC 2396. Internet: *http://www.ietf.org/rfc/rfc2396.txt*.

[url] Berners-Lee, T.; Masinter, L.; McCahill, M. 1994. "Uniform Resource Locators (URL)". RFC 1738. Internet: *http://www.ietf.org/rfc/rfc1738.txt*.

[wsdl-1.1] Christensen, Erik [u.a.]. 2001. "Web Services Description Language 1.1". W3C Note. Internet: *http://www.w3.org/TR/2001/NOTE-wsdl-20010315.*

[wsi-1] Ballinger, Keith [u.a.]. 2004. "Basic Profile Version 1.0". Web Services Interoperability Organization. Internet: *http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html*.

[wxs-1] Thompson, Henry S. [u.a.]. 2004. "XML Schema Part 1: Structures Second Edition". W3C Recommendation. Internet: *http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/*.

[wxs-2] Biron, Paul V.; Malhotra Ashok. 2001. "XML Schema Part 2: Datatypes". W3C Recommendation. Internet: *http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/*.

[xml-g] Luef, G.; Grandits, Franz. 2004. "E-Government: XML-Strukturen für Geschäftsobjekte". AG Kommunikationsarchitektur. Internet: *http://reference.e-government.gv.at/*".