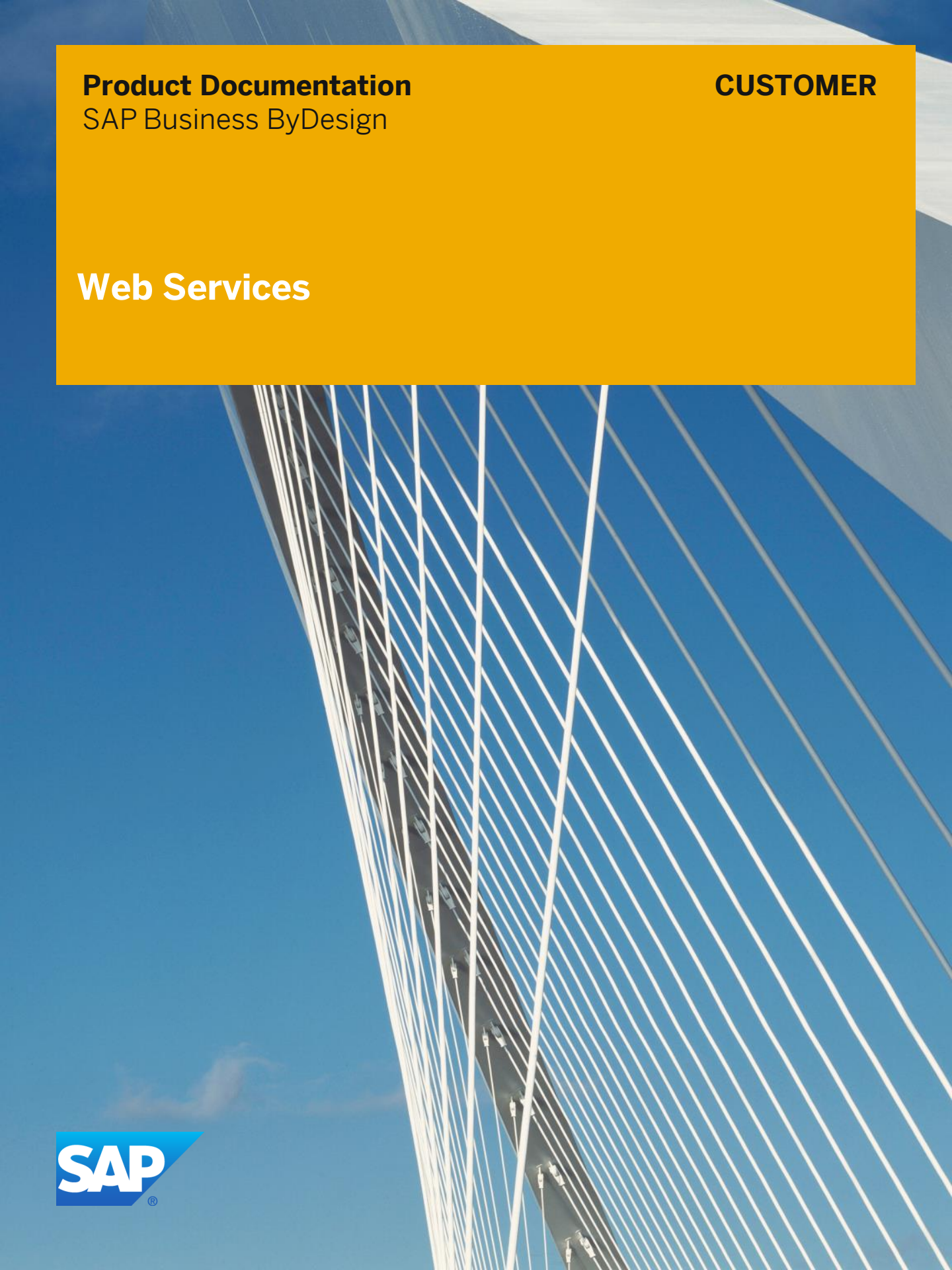
Product Documentation
SAP Business ByDesign

CUSTOMER

Web Services



Web Services

This document provides sample technical information about SAP-delivered Web services that supports Web service consumption.

Information about the business and functionality of the process itself can be found in the Help Center. The target audiences for this document are developers with experience in:

- SOAP, Web services, WSDL, WSIL, authentication
- XML, HTTP/HTTPS web requests

Here are some useful links:

- Help Center documentation – Use the Help Center to get any information about a functionality or a process related to the business.
- Global Data Type Documentation – SAP Global Data Types (GDTs) are SAP-wide unified data types representing business related content built according to UN/CEFACT CCTS. The WSDL for each Web service must be used for technical details of the interface. For more details about specific data types, you can refer to the GDT documentation.

Abbreviations used:

- Service Interface — SI
- Service Operation — SO

Topics

General Information

WSDL

Many external applications consuming Web services have special requirements and restrictions regarding the format of WSDLs. Some external applications require service definition WSDLs describing the Web service signature. This is normally sufficient for the creation of static client-side proxies. Other external applications, normally those that do not create static client-side proxies, require binding WSDLs including the endpoint definition and authentication policy information.

In both cases, it may be the case that the external application imposes special restrictions on the structure or the size of WSDLs.

Microsoft InfoPath® requires binding WSDLs and considers elements with the attribute `minOccurs=0` as `mandatory`. However `minOccurs=0`means `optional` in SAP Web services. In order to circumvent this problem, the WSDL must be saved locally and an additional attribute `nillable=true`must be added to make a query parameter optional for Microsoft InfoPath®.

External applications have to take into account that Web service request and response message types can be enhanced with additional elements and attributes. Enhancements can be created by SAP, SAP partners, and key users. Enhancements of request message types are always optional elements or attributes. The SAP system does not require the external application to provide values in the request. Enhancements of response message types can contain mandatory elements or attributes. The external application must be able to process the extended response successfully.

XML element and attribute names are always stable. Technical definitions of data types can be enhanced in a compatible way. This may result in changed data type names. External applications can rely on XML element names and attribute names, but should not rely on data type names. Over various releases, the interface will be kept stable for use, that is, SAP, customers, and partners may add additional optional elements. Removal of elements or attributes will only be done if these are outdated and not used at all, so that there is no impact for consumers.

GDT Documentation

Global Data Type documentation is available here: http://scn.sap.com/docs/DOC-17516

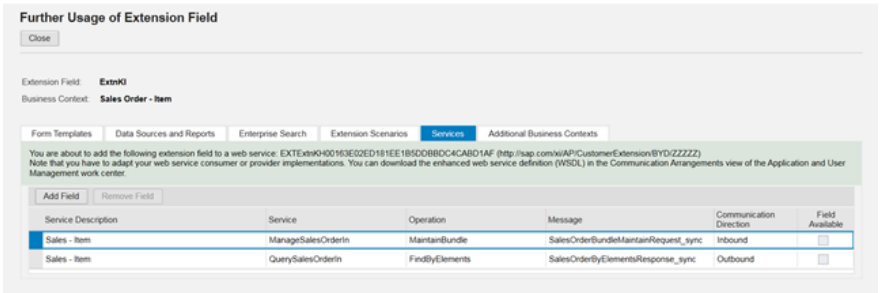
The WSDL provides all the information required to consume the Web service. Above is the link to the GDT documentation that provides additional information.

Extending Interfaces

It is now possible to use the extended fields in the service interfaces.

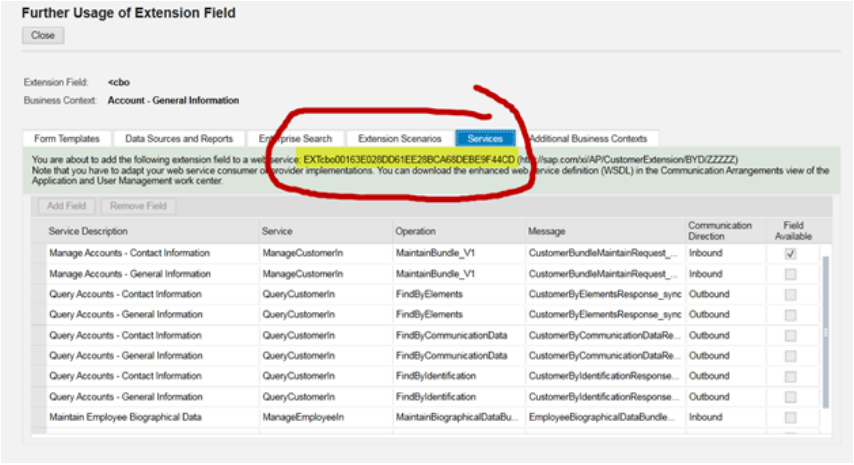
1. Navigate to the screen on which the extension field is available, and select *Enter Adaptation Mode* from the *Adapt*menu. In adaptation mode, select *Edit Screen* from the *Adapt* menu.
2. In the *Adaptation* panel, under *Extension Fields*, select the extension field from the list and under *Field Properties*, click *Further Usage*.
3. On the Services tab, you can add the extension field to the interface.
4. In the *Message* column, you can check if the extensibility is supported in the request message and/or message structure.

Downloaded WSDLs might be outdated by extension field changes. If required, download the WSDL again after the changes, and update your coding if required.



Sample: Further Usage of Extension Field

When a field is added, the name is not entirely taken over. Instead, a more technical name is used.



Sample: Further Usage of Extension Field

SOAP Faults

If there are issues regarding the authorization of the Web service call, the system returns an error message similar to the following one:

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
      <soap-env:Header/>
      <soap-env:Body>
            <soap-env:Fault>
                  <faultcode>soap-env:Server</faultcode>
                  <faultstring xml:lang="en">Authorization role missing for service "ServiceInterface
                  http://sap.com/xi/A1S/Global ManageContactIn <default> <default>", operation "Operation
                  http://sap.com/xi/A1S/Global MaintainBundle" (UTC timestamp 20130320080942; Transaction ID
                  00163E0290481EE2A4A6B0992E5E8C2D)</faultstring>
                  <detail/>
            </soap-env:Fault>
      </soap-env:Body>
</soap-env:Envelope>
```

In case there are technical errors during message processing, for example, an invalid XML structure, the system returns an error message similar to the following one:

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
      <soap-env:Header/>
      <soap-env:Body>
            <soap-env:Fault>
                  <faultcode>soap-env:Server</faultcode>
                  <faultstring xml:lang="en">Web service processing error; more details in the Web service error log on
                  provider side (UTC timestamp 20130320080819; Transaction ID 00163E0290481EE2A4A6AA6271768C2A)</faultstring>
                  <detail/>
            </soap-env:Fault>
      </soap-env:Body>
</soap-env:Envelope>
```

You can view the content of the error in the *Application and User Management* work center, *Business Communication Monitoring* view. In the *Show* drop down, select *Rejected Web Service calls*. You can also provide the name of the service interface and search for errors.

Query Services

Communication Timeout

Query operations are mass-enabled stateless synchronous Web service operations. Transferring or requesting amounts of data that are too large causes communication timeouts. The Web service consumer (external application) is responsible for ensuring reasonable sizes for mass operations.

Processing Conditions

Processing conditions are used to set the maximum number of returned hits per query or to retrieve the next chunk of data (paging).

Maximum Number of Hits

The maximum number of rows returned by a query is determined by the elements `QueryHitsUnlimitedIndicator`and `QueryHitsMaximumNumberValue`of the element `ProcessingConditions`.

If the `QueryHitsUnlimitedIndicator`is true, all data is returned. Otherwise, only up to `QueryHitsMaximumNumberValue` items are returned (if not specified, default 100 is applied).

```
<ProcessingConditions>
      <QueryHitsMaximumNumberValue></QueryHitsMaximumNumberValue>
      <QueryHitsUnlimitedIndicator></QueryHitsUnlimitedIndicator>
      <LastReturnedObjectID></LastReturnedObjectID>
</ProcessingConditions>
```

Paging

If more records exist than returned, the previous query sets `MoreHitsAvailableIndicator` to `true`and provides the `LastReturnedObjectID` that has to be passed as the `LastReturnedOjbectID` in the next query to retrieve the next bunch of records. Query parameters should not be changed, otherwise the behavior is undefined.

Processing conditions can be used for the paging of query results. However, it is crucial not to change the query selection parameters from request to request, as otherwise the query result might change. The last returned object ID from the response can be used to query the next set of data by using the same request with the last returned object ID as obtained from response.

Selection Parameter

Query selection parameters are mainly exposed as select options or ranges with inclusion/exclusion code*, interval boundary type code*, lower boundary value and upper boundary value.

" * " indicates a mandatory field. If the consumer does not pass the mandatory fields, the consumer will get an application error. These fields accept limited possible input entries as listed below. If the value passed is different from the provided list, an application error is expected.

The inclusion/exclusion code (element name `InclusionExclusionCode`) specifies the inclusion of a set into a result set or the exclusion of it.

Inclusion Exclusion Code	Description
E	Excluding
I	Including

The interval boundary type code (element name `IntervalBoundaryTypeCode`) is used to describe intervals by their boundaries. Depending on the operator, the lower boundary value or both boundary values are relevant.

Interval Boundary Type Code	Name	Description	Value
1	Equal to	Single value	=X
3	Between	Interval with closed lower and open upper boundary	[X, Y]
6	Less than	Interval with unlimited lower and open upper boundary	< X
7	Less than or equal to	Interval with unlimited lower and closed upper boundary	<= X
8	Greater than	Interval with open lower and unlimited upper boundary	> X
9	Greater than or equal to	Interval with closed lower and unlimited upper boundary	>= X

The interval boundary type codes 2, 4, and 5 are not supported and therefore not included in this list.

The lower boundary value (element name `LowerBoundaryInternalID`) and upper boundary value (element name `UpperBoundaryInternalID`) can be passed based on the selection for the interval boundary type code.

- If the interval boundary type code is 1, 6, 7, 8 and 9, only the lower boundary value should be passed. Neglecting this can result into an error.
- If the interval boundary type code is 3 both the lower boundary value and the upper boundary values are required. Neglecting this can result into an error.

Requested Elements

Requested elements (element name `RequestedElements`) enable Web service consumers to reduce the size of query response messages, thereby reducing transmission and response times, and avoiding timeouts.

Requested elements contain transmission request codes for controlling the transmission of corresponding query response node elements.

The transmission request codes are modeled as XML attributes following the syntax: `<element name>TransmissionRequestCode`.

The prefix `<element name>` is equivalent to the name of the node element in the response message to which the transmission request code applies.

Transmission Request Code	Description
1	Complete structure: The node element and all its elements, its child node elements and all their elements will be returned.
2	Complete node: The node element and all its elements will be returned
3	Node with key elements: The node element and its key elements will be returned. This is not supported by all Web service interfaces.
4	Excluded node: The node element and its child node elements will not be returned.

The transmission request code is optional. If no transmission request code is modeled or provided by the service consumer, the following rules apply:

- If no transmission request code is provided at all, the response message contains all the data.
- If the transmission request code of the parent node element is 1, its sub-node elements for which no transmission request code is specified are defaulted by 1.
- If the transmission request code of the parent node element is 2 or 3, its sub-node elements for which no transmission request code is specified are defaulted by 4.
- If the transmission request code of the parent node element is 4, its sub-node elements are not returned.

Response Message

The structure of the query response message consists of three parts:

- A business document-specific part containing the returned business documents
- Processing conditions
- Log items that contain system messages including errors, warnings, and information messages raised by the system while processing the Web service request

Retrieving Data in Different Languages

The default log on language is the language defined in the user master data. The default can be overruled by explicitly selecting a language via URL parameter, for example, `sap-language=DE`. The language code is the two letter ISO code. Since Web service calls are stateless, the parameter value can be changed for every call.

Maintain Services

General Rules for Using this Web Service

- Check Maintain Bundle vs. Maintain Bundle

  Maintain Bundle operations enable external applications to create and change business document data. Check Maintain Bundle operations enable external applications to simulate maintain bundle requests without changing business document data. In particular, Check Maintain Bundle operations have the following functions:
  - Return system messages similar to corresponding Maintain Bundle operations
  - Provide the same message type as the corresponding operation Maintain Bundle
  - Do not assign internal numbers from a productive number range interval (number range statuses are not increased)
  - Do not change business documents

Action Code

Action codes represent an instruction to the recipient of the Web service request to process transmitted message node elements.

Action Code	Description
01	Create; the system returns an error message if the node element already exists.
02	Update; the system returns an error message if the node element does not exist.
03	Delete; the system returns an error message if the node element does not exist.
04	Save; the system creates or changes the node element data.
05	Remove; the system deletes the node element. If the node element does not exist, the system does not send an error message.
06	No Action; the system does not change the node element.

Default action code: 04 (Save).

Action code 04 (Save) creates business documents if the system could not identify a matching target business document. This applies in particular if no business document ID or UUID is provided by the Web service consumer. The Web service consumer (external application) is responsible for providing correct business document IDs or UUIDs and to avoid accidental creation of duplicate business documents.

List Processing

The processing of node elements with cardinality > 1 (for example, a list of descriptions in different languages or a list of telephone numbers) can be controlled using List Complete Transmission Indicators (LCTI). The LCTI indicates whether a list of node elements is transmitted completely. The LCTI of a node element with cardinality > 1 is modeled as an attribute of its parent node element (attribute name: `<name of child element>ListCompleteTransmissionIndicator`).

LCTI	Description
false	The list of node elements is not transmitted completely and hence all node elements that are not transmitted remain unchanged. If the transmitted node elements in the list can be identified uniquely, the system processes the node elements according to the action code. If transmitted node elements in the list cannot be identified uniquely, the system appends the node element to the corresponding list of node elements in the target business document.
true	The list of elements is transmitted completely and hence all node elements that are not transmitted are removed. If no node element is transmitted, the complete list is removed.

Default list complete transmission indicator: false.

The LCTI refers to the completeness of the list of node elements and does not indicate the completeness of sub-elements.

Example

A new description with language code DE (German) is created while an existing description with language code EN (English) is updated. Moreover, an existing description with language code FR (French) remains unchanged and any other description [with language code ES (Spanish), for example] is deleted. An error is raised if there is already a German description or if the English or French description does not exist.

```
<Root actionCode="04" descriptionListCompleteTransmissionIndicator="true" >
      <Description actionCode="01">
            <Description languageCode="DE">neuer deutscher Text
      </Description>
      <Description actionCode="02">
            <Description languageCode="EN">changed english text
      </Description>
      <Description actionCode="03">
            <Description languageCode="FR">
      </Description>
</Root>
```

Empty and Missing Elements

Optional leaf elements in request messages that are not transmitted within a Web service request are not changed in corresponding business documents.

While updating a postal address, the request updates the city name, street postal code, street name, and house ID. The country code remains unchanged, as the element `CountryCode`is not contained in the XML document (commented out in the code snippet below).

```
<PostalAddress actionCode="02">
      <!-- <CountryCode>DE</CountryCode> -->
      <CityName>Heidelberg</CityName>
      <StreetPostalCode>69117</StreetPostalCode>
      <StreetName>Hauptstrasse</StreetName>
      <HouseID></HouseID>
</PostalAddress>
```

Note: The request deletes the house ID or updates the house ID with its initial value.

Communication Timeout

Maintain Bundle and Check Maintain Bundle operations are mass-enabled stateless synchronous Web service operations. Transferring or requesting amounts of data that are too large causes communication timeouts. The Web service consumer is responsible for ensuring reasonable sizes of data for mass operations.

Message Header

Maintain Bundle and Check Maintain Bundle operations support transparent retries (idempotency) in case of lost server responses caused by bad network connections. Using idempotency is optional and normally not required if the network connection is OK. If the Web service consumer fills the UUID element of the `BasicMessageHeader` message element, the server returns the cached response if the same UUID is received within a short time period. This allows the consumer to perform multiple retries in case the original response did not reach the consumer in time. The maximum time span a consumer performs retries should not exceed some minutes. Responses for requests with a filled UUID are cached for 2-4 hours. Reusing the same UUID after this period is not allowed and results in a fault message response.

Change State ID

By using change state identifier (element name `ChangeStateID`), external applications can enforce that a modifying operation is not executed because the state of the business document has changed since the external application last read its data.

The change state ID is an uninterpretable string that is provided by query and read operations and can be utilized by all modifying operations. If the change state identifier is provided when calling a modifying operation, the system does not perform the operation if the state of the business document instance has changed since the change state ID was computed. If the change state ID is not provided by the Web service consumer, the system performs the Web service operation without checking the state of the business document.

The Web service consumer (external application) is responsible for preventing accidental changes to business documents.

Object Node Sender Technical Identifier

Request node elements with cardinality > 1 contain an object node sender technical identifier to relate response message elements and log items to corresponding node elements in the request message.

The object node sender technical identifiers are provided as `ObjectNodeSenderTechnicalID`s in the request message types and referred to as `ReferenceObjectNodeSenderTechnicalID`s in corresponding response message types.

If the object node sender technical ID is initial, the object node sender technical ID of the parent node element in the request is returned as the reference object node sender technical ID. If the object node sender technical IDs of all parent node elements are initial, the reference object node sender technical ID is returned as initial as well.

- The values specified in the `ObjectNodeSenderTechnicalID` are transient values that establish the correspondence between elements only for a single call. The Web service consumer is not required to specify them or to use the same values for different calls. Also, the service provider does not interpret these values at all, but returns them to the Web service consumer instead in the `ReferenceObjectNodeSenderTechnicalID` elements.
- The `ObjectNodeSenderTechnicalID` is also used to identify failed business document modifications in a mass operation.

Example

Request:

```
<Child>
      <ObjectNodeSenderTechnicalID>999_A<ObjectNodeSenderTechnicalID>
      <Content>Child A: Some correct content</Content>
</Child>
<Child>
      <ObjectNodeSenderTechnicalID>999_B<ObjectNodeSenderTechnicalID>
      <Content>Child B: Some erroneous content</Content>
</Child>
```

Response:

```
<Log>
      <Item>
            <ReferenceObjectNodeSenderTechnicalID>999_B</ReferenceObjectNodeSenderTechnicalID>
            <Note>Error message for Child B</Note>
      </Item>
</Log>
```

Response Message Structure

The structure of the response message consists of two parts:

- A business document-specific part containing information about IDs and UUIDs of the created and changed business documents
- Log items containing system messages including errors, warnings, and information messages raised by the system during processing of the Web service request

Message Header in Asynchronous Services

While for synchronous messages no addressing information is needed in the message payload, all asynchronous web service messages contain a `BusinessDocumentMessageHeader` element (BDMH) as common payload section, which is used for exchanging addressing information between communication partners on business level. Its structure can be seen in the WSDL file downloaded from the communication arrangement UI.

The BDMH is based on the following convention: "UN/CEFACT Standard BusinessDocumentMessage Header Technical Specification - Working Draft - Revision 2.2.5". Comparing `BusinessDocumentMessageHeader` to the header information from the usual message transfer protocols "Reliable Messaging", "OASIS ebXML MSG", "OASIS ebXML CPP/CPA", "Rosetta Net RNIF 2.0", and so on, demonstrates that the BDMH contains redundant information compared to these technical transfer protocols. However, the BDMH is only used at business application level and not at such a technical level.

The ID (`BusinessDocumentMessageID`) of the BDMH is clearly distinguishable from the technical Message Id:

- The ID (`BusinessDocumentMessageID`) remains unchanged even when the message is sent via multiple, successive middleware systems ("messaging systems").
- In contrast, the technical MessageID is issued at the level of the technical middleware system and generally changes as follows:
  - Each time the BusinessDocument is resent or forwarded by a middleware system
  - When the BusinessDocument is split into multiple technical messages by a middleware system

The `BusinessDocumentMessageHeader` is used for the following:

- For error handling, messages are referenced by their unique identification
- For tracing and monitoring of a BusinessDocument and its processing status at business application level
- For managing and monitoring business processes

The BDMH consists of following elements:

- **ID (and UUID)**

  Mandatory to identify the business document unequivocally, exactly one of the fields shall be filled, not both. Some middleware tools just display the ID element, therefore SAP cloud solutions, as well as most SAP Suite applications, will only fill ID, with a globally unique value.

  The xml attributes within the elements ID/UUID are not used by SAP cloud solutions and might be removed in the future.

- **ReferenceID (and ReferenceUUID)**

  Used in confirmation messages to identify the original message (see ID and UUID) confirmed by the confirmation message. The usage is mandatory in confirmation messages. If the ID element was filled in the request, it is to be moved into ReferenceID of the confirmation message. If UUID was filled, ReferenceUUID is filled instead. SAP Cloud solutions are using element ReferenceID only.

  The xml attributes within the elements ReferenceID/ReferenceUUID are not used by SAP cloud solutions and might be removed in the future.

- **CreationDateTime**

  This timestamp is filled by the sender with the time of message creation. The timestamp has to be a UTC timestamp (e.g. 2013-09-15T10:40:50Z).

  Filling is mandatory. It is used as a major sort/filter criterion for monitoring and error handling.

- **TestDataIndicator**

  This Indicator marks a message to be a test or simulation message. This avoids that data will be stored in the SAP cloud application.

  Usage: Test messages can be used to observe, for example, the routing behavior of middleware components configuration or test data and mapping quality during development or integration project phase.

- **(ReconciliationIndicator)**

  When set, the sender indicates it is sending its current state ("gross data") with the intention to let the receiver adjust to the most recent sender state as part of error handling.

  SAP Cloud solutions will not fill or evaluate this indicator in B2B communication.

- **SenderParty (and SenderBusinessSystemId), RecipientParty (and RecipientBusinessSystemId)**

  These elements are used to specify sending and recipient business systems in application-to-application (A2A) integration or collaboration business partners in business-to-business (B2B) integration.

  Each SAP cloud solution tenant has its own alphanumeric unique tenantID, which is displayed in the *Communication Arrangements* work center. When sending asynchronous A2A messages, this will be filled automatically by the system in the message header:

  ```
  <SenderParty><InternalID schemeID="CommunicationSystemID" schemeAgencyID="310">KRIG1S</InternalID></SenderParty>
  ```

  - **Specifics to SAP Business Suite integration scenarios:**

    SAP Suite Systems as well as SAP NetWeaver PI systems will identify business systems in message-based communication by a BusinessSystemID (schemeID=" BusinessSystemID") as maintained in the System Landscape Directory (SLD) or by an ALE LogicalSystemID (schemeID=" LocalSystemID") used in IDOCSs. For compatibility reasons, SAP cloud solutions allow to maintain these as alternative IDs in the Communication System UI.

  - **Use of Standard IDs in B2B scenarios for collaboration partner identification:**

    In B2B communications usually no system information or system local identifiers are used, therefore the sender and receiver business parties are identified via common standard IDs such as:

    - schemeID="DUNS" schemeAgencyID="16"
    - schemeID="GLN" schemeAgencyID="9"
    - schemeID="SCAC" schemeAgencyID="182"

      The standard IDs are maintained in the SAP cloud solution in the application UI of a Business Partner (for example, Supplier or Account).

  Use: In inbound processing of a SAP cloud solution, the sender/recipient ID information is used to find a communication arrangement assigned to a corresponding business partner (B2B) or communication system (A2A).

The child element `ContactPerson` of the `SenderParty` or `RecipientParty` is not supported by SAP and might be removed in future releases.

- **BusinessScope (optional)**

  When filled, this element is used to transfer meta information about the business process during which the message used. Unless explicitly documented for a specific service interface, this may be ignored.

Sample BDMH in B2B communication

```
<BusinessDocumentMessageHeader>
      <ID>1234567890ABCDEF0123456789ABCDEF</ID>
      <CreationDateTime>2012-03-28T12:00:00.1234567Z</CreationDateTime>
      <SenderParty>
            <StandardID schemeAgencyID="9">1010000002400</StandardID>
      </SenderParty>
      <RecipientParty>
            <StandardID schemeAgencyID="9">1010000007000</StandardID>
      </RecipientParty>
      <BusinessScope>
            <TypeCode listID="25201" listAgencyID="310">2</TypeCode>
            <ID schemeID="10555" schemeAgencyID="310">92</ID>
      </BusinessScope>
      <BusinessScope>
            <TypeCode listID="25201" listAgencyID="310">3</TypeCode>
            <ID schemeID="10555" schemeAgencyID="310">97</ID>
      </BusinessScope>
</BusinessDocumentMessageHeader>
```

Where:

- TypeCode ListID = ‘25201’ as the ID of the BusinessScopeTypeCodelist at SAP
- ListAgencyID: ’310’ for SAP
- TypeCode: ‘2’ for receiving process
- TypeCode: ‘3’ for sending process
- SchemeID: 10555 for the ID-Schema of the BusinessScopeID at SAP
- SchemeAgencyID: ‘310’for SAP
- ID for the receiving or sending process

Sample Payload for Third-Party Integration (Third-Party System -> SAP Cloud solution)

The id of the receiving SAP cloud solution tenant is generated by SAP cloud operations. It can be displayed in the *Communication Arrangements* UI and needs to be communicated to the person setting up the connectivity from the sending system.

The id of the sender represents the sender business system, which needs to be aligned beforehand and to be maintained in the *Communication System* UI of the receiving SAP cloud solution.

```
<BusinessDocumentMessageHeader>
      <ID>1234567890ABCDEF0123456789ABCDEF</ID>
      <CreationDateTime>2012-03-28T12:00:00.1234567Z</CreationDateTime>
      <SenderParty>
            <InternalID schemeID="CommunicationSystemID" schemeAgencyID="310">SYS_001</InternalID>
            <!-- Identifier of remote business system communicated by sending party and maintained in Communication
            System UI -->
      </SenderParty>
      <RecipientParty>
            <InternalID schemeID="CommunicationSystemID" schemeAgencyID="310">KRIG1S</InternalID>
            <!-- SAP Tenant Identifier as seen in Communication Arrangement UI -->
      </RecipientParty>
```

```
</BusinessDocumentMessageHeader>
```

Sample Payload for Third-Party Integration (SAP Cloud solution -> Third-Party Sytem)

```
<BusinessDocumentMessageHeader>
      <ID>1234567890ABCDEF0123456789ABCDEFID</ID>
      <CreationDateTime>2012-03-28T12:00:00.1234567Z</CreationDateTime>
      <SenderParty>
            <InternalID schemeID="CommunicationSystemID" schemeAgencyID="310">KRIG1S</InternalID>
            <!-- SAP Tenant Identifier as seen in Communication Arrangement UI ->
      </SenderParty>
      <RecipientParty>
            <InternalID schemeID="CommunicationSystemID" schemeAgencyID="310">SYS_001</InternalID>
            <!-- Identifier of remote business system communicated by sending party and maintained in Communication
            System UI -->
      </RecipientParty>
</BusinessDocumentMessageHeader>
```

Sample Code to Consume a Service

Below is some sample code that allows us to consume the interface. There are two ways in which you can consume the Web service. Create the SOAP XML and make a Web service call or write client code that creates the XML from client proxies and makes a call to the Web service. The following examples in C# and Java creates such an XML snippet. To achieve this, client side proxies must be generated from the WSDL that can be downloaded from the corresponding communication arrangement.

The communication arrangement and the communication system must be created before you start using the Web service. Once the add-on is deployed and the scoping in business configuration is done, these can be created. Please refer the Help Center documentation for information on how to create a communication arrangement.

Sample C# Code to Consume a Service

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace WSDL_CL_T2
{
      class Program
            static void Main(string[] args)
            {
                  try
                  {
                        //Create an instance of the WCF proxy
                        ManageBusinessPartner_TemplateInClient client = new ManageBusinessPartner_TemplateInClient();
                        client.Endpoint.Behaviors.Add(new InspectorBehavior()); //Trace Request/Response XML to Console !
                              client.ClientCredentials.UserName.UserName = "_CP100120";
                              client.ClientCredentials.UserName.Password = "Welcome1";
                        //Read Financial Data
                        BusinessPartner_TemplateFinancialDataByIDQueryMessage_sync para1=new
                        BusinessPartner_TemplateFinancialDataByIDQueryMessage_sync ();
                        para1.BusinessPartner_Template = new BusinessPartner_TemplateFinancialDataByIDQuery();
                        para1.BusinessPartner_Template.InternalID = "S100800";
                        DateTime startTime = DateTime.Now;
                        BusinessPartner_TemplateFinancialDataByIDResponseMessage_sync ret =
                        client.ReadFinancialData(para1);
                        DateTime stopTime = DateTime.Now;
                        Console.WriteLine("Started: {0}", startTime);
                        Console.WriteLine("Ended: {0}", stopTime);
                        TimeSpan elapsedTime = stopTime - startTime;
                        Console.WriteLine("Elapsed time in ms:" + elapsedTime.TotalMilliseconds);
                        if(ret.BusinessPartner_Template==null)
                              Console.WriteLine("Business Partner not found!");
                        else
                        {
                              if(ret.BusinessPartner_Template.BankDetails!=null)
                                    Console.WriteLine("Bank Account Holder Name:
                                    {0}",ret.BusinessPartner_Template.BankDetails[0].BankAccountHolderName);
                              else
                                    Console.WriteLine("No Bank Account maintained !");
                        }
                        client.Close();
                  }
                  catch (Exception e)
                  {
                        Console.WriteLine("{0} Exception caught.", e);
                  }
            //Wait for any key to close the window
            ConsoleKeyInfo result = Console.ReadKey();
            }
      }
}
```

Sample Java Code to Consume a Service

```
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import javax.xml.ws.BindingProvider;
import com.sap.xi.a1s.global.ManageMaterialIn;
import com.sap.xi.a1s.global.MaterialMaintainConfirmationBundleMaterialSyncV1;
import com.sap.xi.ap.common.gdt.ProductInternalID;
import com.sap.xi.a1s.global.MaterialMaintainConfirmationBundleMessageSyncV1;
import com.sap.xi.a1s.global.MaterialMaintainRequestBundleDescription;
import com.sap.xi.a1s.global.MaterialMaintainRequestBundleMaterialSyncV1;
import com.sap.xi.a1s.global.MaterialMaintainRequestBundleMessageSyncV1;
```

```
import com.sap.xi.a1s.global.Service;
import com.sap.xi.a1s.global.StandardFaultMessage;
import com.sap.xi.ap.common.gdt.SHORTDescription;
public class Client {
	/**
	* @param args
	*/
	public static void main(String[] args) {
		// TO DO: Auto-generated method stub
		System.out.println("Hello World");
		ManageMaterialIn bnd = new Service().getBinding();
		Map<String,Object> requestContext = ((BindingProvider)bnd).getRequestContext();
		URL addressURL = null;
		try {
			addressURL = new URL("https://my300238.vlab.sapbydesign.com/sap/bc/srt/scs/sap/managematerialin1");
		} catch (MalformedURLException e) {
			// TO DO: Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			// TO DO: Auto-generated catch block
			e.printStackTrace();
		}
		// Create the Request
		requestContext.put(BindingProvider.USERNAME_PROPERTY, "ADMINISTRATION01");
		requestContext.put(BindingProvider.PASSWORD_PROPERTY, "Welcome1");
		requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, addressURL.toExternalForm());
		MaterialMaintainRequestBundleMessageSyncV1 requestMessage = new MaterialMaintainRequestBundleMessageSyncV1();
		MaterialMaintainRequestBundleMaterialSyncV1 material = new MaterialMaintainRequestBundleMaterialSyncV1();
		ProductInternalID internalID = new ProductInternalID();
		internalID.setValue("MCF-0001");
		material.setInternalID(internalID);
		MaterialMaintainRequestBundleDescription description = new MaterialMaintainRequestBundleDescription();
		SHORTDescription desc = new SHORTDescription();
		desc.setLanguageCode("EN");
		desc.setValue("Test Description");
		description.setDescription(desc);
		material.getDescription().add(description);
		requestMessage.getMaterial().add(material);
		// Create the Response
		MaterialMaintainConfirmationBundleMessageSyncV1 responseMessage = null;
		try {
			responseMessage = bnd.maintainBundleV1(requestMessage);
		} catch (StandardFaultMessage e) {
			// TO DO: Auto-generated catch block
			e.printStackTrace();
		}
		if (responseMessage.getMaterial() != null ){
			List<MaterialMaintainConfirmationBundleMaterialSyncV1> response = responseMessage.getMaterial();
			Iterator<MaterialMaintainConfirmationBundleMaterialSyncV1> itr = response.iterator();
			while (itr.hasNext()){
				System.out.println("Material ID Created is " + itr.next().getInternalID().getValue().toString());
			}
		}
	}
}
```

www.sap.com/contactsap