



Informatica® Cloud Application Integration
September 2018

4. Design

© Copyright Informatica LLC 1993, 2018

This software and documentation contain proprietary information of Informatica LLC and are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright law. Reverse engineering of the software is prohibited. No part of this document may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording or otherwise) without prior consent of Informatica LLC. This Software may be protected by U.S. and/or international Patents and other Patents Pending.

Use, duplication, or disclosure of the Software by the U.S. Government is subject to the restrictions set forth in the applicable software license agreement and as provided in DFARS 227.7202-1(a) and 227.7702-3(a) (1995), DFARS 252.227-7013(1)(ii) (OCT 1988), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 52.227-14 (ALT III), as applicable.

The information in this product or documentation is subject to change without notice. If you find any problems in this product or documentation, please report them to us in writing.

Informatica, Informatica Platform, Informatica Data Services, PowerCenter, PowerCenterRT, PowerCenter Connect, PowerCenter Data Analyzer, PowerExchange, PowerMart, Metadata Manager, Informatica Data Quality, Informatica Data Explorer, Informatica B2B Data Transformation, Informatica B2B Data Exchange Informatica On Demand, Informatica Identity Resolution, Informatica Application Information Lifecycle Management, Informatica Complex Event Processing, Ultra Messaging, Informatica Master Data Management, and Live Data Map are trademarks or registered trademarks of Informatica LLC in the United States and in jurisdictions throughout the world. All other company and product names may be trade names or trademarks of their respective owners.

Portions of this software and/or documentation are subject to copyright held by third parties, including without limitation: Copyright DataDirect Technologies. All rights reserved. Copyright © Sun Microsystems. All rights reserved. Copyright © RSA Security Inc. All Rights Reserved. Copyright © Ordinal Technology Corp. All rights reserved. Copyright © Aandacht c.v. All rights reserved. Copyright Genivia, Inc. All rights reserved. Copyright Isomorphic Software. All rights reserved. Copyright © Meta Integration Technology, Inc. All rights reserved. Copyright © Intalio. All rights reserved. Copyright © Oracle. All rights reserved. Copyright © Adobe Systems Incorporated. All rights reserved. Copyright © DataArt, Inc. All rights reserved. Copyright © ComponentSource. All rights reserved. Copyright © Microsoft Corporation. All rights reserved. Copyright © Rogue Wave Software, Inc. All rights reserved. Copyright © Teradata Corporation. All rights reserved. Copyright © Yahoo! Inc. All rights reserved. Copyright © Glyph & Cog, LLC. All rights reserved. Copyright © Thinkmap, Inc. All rights reserved. Copyright © Clearpace Software Limited. All rights reserved. Copyright © Information Builders, Inc. All rights reserved. Copyright © OSS Nokalva, Inc. All rights reserved. Copyright Edifecs, Inc. All rights reserved. Copyright Cleo Communications, Inc. All rights reserved. Copyright © International Organization for Standardization 1986. All rights reserved. Copyright © ej-technologies GmbH. All rights reserved. Copyright © Jaspersoft Corporation. All rights reserved. Copyright © International Business Machines Corporation. All rights reserved. Copyright © yWorks GmbH. All rights reserved. Copyright © Lucent Technologies. All rights reserved. Copyright © University of Toronto. All rights reserved. Copyright © Daniel Veillard. All rights reserved. Copyright © Unicode, Inc. Copyright IBM Corp. All rights reserved. Copyright © MicroQuill Software Publishing, Inc. All rights reserved. Copyright © PassMark Software Pty Ltd. All rights reserved. Copyright © LogiXML, Inc. All rights reserved. Copyright © 2003-2010 Lorenzi Davide, All rights reserved. Copyright © Red Hat, Inc. All rights reserved. Copyright © The Board of Trustees of the Leland Stanford Junior University. All rights reserved. Copyright © EMC Corporation. All rights reserved. Copyright © Flexera Software. All rights reserved. Copyright © Jinfonet Software. All rights reserved. Copyright © Apple Inc. All rights reserved. Copyright © Telerik Inc. All rights reserved. Copyright © BEA Systems. All rights reserved. Copyright © PDFlib GmbH. All rights reserved. Copyright © Orientation in Objects GmbH. All rights reserved. Copyright © Tanuki Software, Ltd. All rights reserved. Copyright © Ricebridge. All rights reserved. Copyright © Sencha, Inc. All rights reserved. Copyright © Scalable Systems, Inc. All rights reserved. Copyright © jQWidgets. All rights reserved. Copyright © Tableau Software, Inc. All rights reserved. Copyright © MaxMind, Inc. All Rights Reserved. Copyright © TMate Software s.r.o. All rights reserved. Copyright © MapR Technologies Inc. All rights reserved. Copyright © Amazon Corporate LLC. All rights reserved. Copyright © Highsoft. All rights reserved. Copyright © Python Software Foundation. All rights reserved. Copyright © BeOpen.com. All rights reserved. Copyright © CNRI. All rights reserved.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>), and/or other software which is licensed under various versions of the Apache License (the "License"). You may obtain a copy of these Licenses at <http://www.apache.org/licenses/>. Unless required by applicable law or agreed to in writing, software distributed under these Licenses is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the Licenses for the specific language governing permissions and limitations under the Licenses.

This product includes software which was developed by Mozilla (<http://www.mozilla.org/>), software copyright The JBoss Group, LLC, all rights reserved; software copyright © 1999-2006 by Bruno Lowagie and Paulo Soares and other software which is licensed under various versions of the GNU Lesser General Public License Agreement, which may be found at <http://www.gnu.org/licenses/lgpl.html>. The materials are provided free of charge by Informatica, "as-is", without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

The product includes ACE(TM) and TAO(TM) software copyrighted by Douglas C. Schmidt and his research group at Washington University, University of California, Irvine, and Vanderbilt University, Copyright (©) 1993-2006, all rights reserved.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (copyright The OpenSSL Project. All Rights Reserved) and redistribution of this software is subject to terms available at <http://www.openssl.org> and <http://www.openssl.org/source/license.html>.

This product includes Curl software which is Copyright 1996-2013, Daniel Stenberg, <daniel@haxx.se>. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at <http://curl.haxx.se/docs/copyright.html>. Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

The product includes software copyright 2001-2005 (©) MetaStuff, Ltd. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at <http://www.dom4j.org/license.html>.

The product includes software copyright © 2004-2007, The Dojo Foundation. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at <http://dojotoolkit.org/license>.

This product includes ICU software which is copyright International Business Machines Corporation and others. All rights reserved. Permissions and limitations regarding this software are subject to terms available at <http://source.icu-project.org/repos/icu/icu/trunk/license.html>.

This product includes software copyright © 1996-2006 Per Bothner. All rights reserved. Your right to use such materials is set forth in the license which may be found at <http://www.gnu.org/software/kawa/Software-License.html>.

This product includes OSSP UUID software which is Copyright © 2002 Ralf S. Engelschall, Copyright © 2002 The OSSP Project Copyright © 2002 Cable & Wireless Deutschland. Permissions and limitations regarding this software are subject to terms available at <http://www.opensource.org/licenses/mit-license.php>.

This product includes software developed by Boost (<http://www.boost.org/>) or under the Boost software license. Permissions and limitations regarding this software are subject to terms available at http://www.boost.org/LICENSE_1_0.txt.

This product includes software copyright © 1997-2007 University of Cambridge. Permissions and limitations regarding this software are subject to terms available at <http://www.pcre.org/license.txt>.

This product includes software copyright © 2007 The Eclipse Foundation. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at <http://www.eclipse.org/org/documents/epl-v10.php> and at <http://www.eclipse.org/org/documents/edl-v10.php>.

This product includes software licensed under the terms at <http://www.tcl.tk/software/tcltk/license.html>, <http://www.bosrup.com/web/overlib/?License>, <http://www.stlport.org/doc/license.html>, <http://asm.ow2.org/license.html>, <http://www.cryptix.org/LICENSE.TXT>, <http://hsqldb.org/web/hsqLicense.html>, <http://httpunit.sourceforge.net/doc/license.html>, <http://jung.sourceforge.net/license.txt>, http://www.gzip.org/zlib/zlib_license.html, <http://www.openldap.org/software/release/license.html>, <http://www.libssh2.org>, <http://slf4j.org/license.html>, <http://www.sente.ch/software/OpenSourceLicense.html>, <http://fusesource.com/downloads/license-agreements/fuse-message-broker-v-5-3-license-agreement>, <http://antlr.org/license.html>, <http://aopalliance.sourceforge.net/>, <http://www.bouncycastle.org/licence.html>, <http://www.jgraph.com/jgraphdownload.html>, <http://www.jcraft.com/jsch/LICENSE.txt>, http://jotm.objectweb.org/bsd_license.html, <http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231>, <http://www.slf4j.org/license.html>, <http://nanoxml.sourceforge.net/orig/copyright.html>, <http://www.json.org/license.html>, <http://forge.ow2.org/projects/javaservice/>, <http://www.postgresql.org/about/licence.html>, <http://www.sqlite.org/copyright.html>, <http://www.tcl.tk/software/tcltk/license.html>, <http://www.jaxen.org/faq.html>, <http://www.jdom.org/docs/faq.html>, <http://www.slf4j.org/license.html>, <http://www.iodbc.org/dataspace/iodbc/wiki/IODBC/License>, <http://www.keplerproject.org/md5/license.html>, <http://www.toedter.com/en/jcalendar/license.html>, <http://www.edankert.com/bounce/index.html>, <http://www.net-snmp.org/about/license.html>, <http://www.openmdx.org/#FAQ>, http://www.php.net/license/3_01.txt, <http://srp.stanford.edu/license.txt>, <http://www.schneier.com/blowfish.html>, <http://www.jmock.org/license.html>, <http://xsom.java.net>, <http://benalman.com/about/license/>, <https://github.com/CreateJS/EaselJS/blob/master/src/easeljs/display/Bitmap.js>, <http://www.h2database.com/html/license.html#summary>, <http://jsoncpp.sourceforge.net/LICENSE>, <http://jdbc.postgresql.org/license.html>, <http://protobuf.googlecode.com/svn/trunk/src/google/protobuf/descriptor.proto>, <https://github.com/rantav/hector/blob/master/LICENSE>, <http://web.mit.edu/Kerberos/krb5-current/doc/mitK5license.html>, <http://jibx.sourceforge.net/jibx-license.html>, <https://github.com/lyokato/libgeohash/blob/master/LICENSE>, <https://github.com/hjiang/jsonxx/blob/master/LICENSE>, <https://code.google.com/p/lz4/>, <https://github.com/jedisct1/libsodium/blob/master/LICENSE>, <http://one-jar.sourceforge.net/index.php?page=documents&file=license>, <https://github.com/EsotericSoftware/kryo/blob/master/license.txt>, <http://www.scala-lang.org/license.html>, <https://github.com/tinkerpop/blueprints/blob/master/LICENSE.txt>, <http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html>, <https://aws.amazon.com/ssl/>, <https://github.com/twbs/bootstrap/blob/master/LICENSE>, <https://sourceforge.net/p/xmlunit/code/HEAD/tree/trunk/LICENSE.txt>, <https://github.com/documentcloud/underscore-contrib/blob/master/LICENSE>, and <https://github.com/apache/hbase/blob/master/LICENSE.txt>.

This product includes software licensed under the Academic Free License (<http://www.opensource.org/licenses/afl-3.0.php>), the Common Development and Distribution License (<http://www.opensource.org/licenses/cddl1.php>), the Common Public License (<http://www.opensource.org/licenses/cpl1.0.php>), the Sun Binary Code License Agreement Supplemental License Terms, the BSD License (<http://www.opensource.org/licenses/bsd-license.php>), the new BSD License (<http://opensource.org/licenses/BSD-3-Clause>), the MIT License (<http://www.opensource.org/licenses/mit-license.php>), the Artistic License (<http://www.opensource.org/licenses/artistic-license-1.0>) and the Initial Developer's Public License Version 1.0 (<http://www.firebirdsql.org/en/initial-developer-s-public-license-version-1-0/>).

This product includes software copyright © 2003-2006 Joe Walnes, 2006-2007 XStream Committers. All rights reserved. Permissions and limitations regarding this software are subject to terms available at <http://xstream.codehaus.org/license.html>. This product includes software developed by the Indiana University Extreme! Lab. For further information please visit <http://www.extreme.indiana.edu/>.

This product includes software Copyright (c) 2013 Frank Balluffi and Markus Moeller. All rights reserved. Permissions and limitations regarding this software are subject to terms of the MIT license.

See patents at <https://www.informatica.com/legal/patents.html>.

DISCLAIMER: Informatica LLC provides this documentation "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of noninfringement, merchantability, or use for a particular purpose. Informatica LLC does not warrant that this software or documentation is error free. The information provided in this software or documentation may include technical inaccuracies or typographical errors. The information in this software and documentation is subject to change at any time without notice.

NOTICES

This Informatica product (the "Software") includes certain drivers (the "DataDirect Drivers") from DataDirect Technologies, an operating company of Progress Software Corporation ("DataDirect") which are subject to the following terms and conditions:

1. THE DATADIRECT DRIVERS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
2. IN NO EVENT WILL DATADIRECT OR ITS THIRD PARTY SUPPLIERS BE LIABLE TO THE END-USER CUSTOMER FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR OTHER DAMAGES ARISING OUT OF THE USE OF THE ODBC DRIVERS, WHETHER OR NOT INFORMED OF THE POSSIBILITIES OF DAMAGES IN ADVANCE. THESE LIMITATIONS APPLY TO ALL CAUSES OF ACTION, INCLUDING, WITHOUT LIMITATION, BREACH OF CONTRACT, BREACH OF WARRANTY, NEGLIGENCE, STRICT LIABILITY, MISREPRESENTATION AND OTHER TORTS.

Publication Date: 2018-09-25

Table of Contents

Preface	8
Chapter 1: Understanding Data Types and Field Properties.....	9
Introduction to Data Types and Field Properties.	9
Types of Data and Field Properties.	10
Formatting Dates, Times, and Numbers.	17
Using the Field Properties Dialog.	17
Controls in All Many or Some Field Properties Dialog.	17
Show List: Advanced Query.	18
Show List: Custom List.	18
Show List: List Child Objects.	18
Show List: None.	19
Show List: Object Query.	19
Show List: Picklist for Field.	20
Show List: Users in Role.	20
Using the Expression Editor.	20
Using Functions.	26
Specifying Functions and Variables.	35
Digital Signature Functions Overview.	36
Attachments.	37
HTTP Headers.	40
Chapter 2: Designing Processes.....	44
Creating a Process.	45
Setting Process Properties.	45
General Properties.	46
Start Properties.	47
Input Field Properties.	48
Output Field Properties.	50
Temporary Field Properties.	50
Messages Properties.	50
Advanced Properties.	52
Notes.	54
Adding Process Steps.	54
Assignment Step.	54
Service Step.	55
Subprocess Step.	56
Create Step.	57
Wait Step.	58
Receive Step.	59

Milestone Step.	59
End Step.	60
Throw Step.	60
Decision Step.	61
Parallel Path Step.	62
Jump Step.	62
System Service Actions.	62
Delete Object.	62
JMS Enqueue Service.	63
Run a Shell Command.	64
Run Cloud Task.	65
Fault Handling.	66
Fault Handling and Boundary Events.	66
Methods to Return a Fault from a Process.	68
Chapter 3: Using and Displaying Data.	70
Selecting and Displaying Objects.	70
Inserting Fields Using Picklists.	71
Inserting Fields in Tables.	72
Entering Fields as Text.	72
Displaying Columns.	73
Setting Data in Steps.	74
Using Fields in Steps.	74
Setting Source Values.	75
Setting Source Values: Content.	76
Setting Source Values: Field.	76
Setting Source Values: Formula.	76
Setting Source Values: Query.	77
Setting Source Values: Screen.	78
Chapter 4: Designing Guides.	79
Creating a Guide.	80
Setting Guide Properties.	81
General.	82
Start.	82
Screens.	83
Fields.	88
Outcomes.	88
Advanced.	90
Notes.	91
Adding Guide Steps.	91
Assignment Step.	92
Create Step.	93

Screen Step.	93
Decision Step.	95
Jump Step.	96
Process Call Step.	96
Embedded Guide step.	97
Service Step.	98
End/Milestone Step.	99
Setting the Appearance of a Guide.	102
Guide Appearance.	102
Guide screen Editor.	103
Guide Simulation.	109

Chapter 5: Creating Process Objects..... 111

Chapter 6: Designing Service Connectors..... 112

Option1: Creating a New Service Connector.	113
Defining Properties.	114
Specifying Functions and Variables.	115
Defining Actions.	116
Creating Service Connector Process Objects.	126
Testing the Service Connector.	127
Option 2: Generating a Service Connector by Importing a File.	129
Creating a Service Connector from a WSDL File.	129
Creating a Service Connector from a Swagger JSON File.	134

Chapter 7: Using Connectors..... 141

Creating a Connection.	141
Defining Connection Properties.	142
Specifying Event Sources.	144
Specifying Event Targets.	144
Viewing Published Metadata.	145

Chapter 8: System Services, Listeners and Connectors..... 147

Using the OData Provider.	147
Supported OData V4 and OData V2 URI Conventions.	148
Custom Composite Keys.	149
Using Salesforce System Services Listeners and Connectors.	152
Salesforce Outbound Messages.	152
Using a Salesforce Connection.	155
Using the Email Service.	157
Email Connection Properties.	157
Sending an Email from a Process.	159
Using Java Message Service (JMS)	159

Using the Shell Command Invoke Service Connector.	160
---	-----

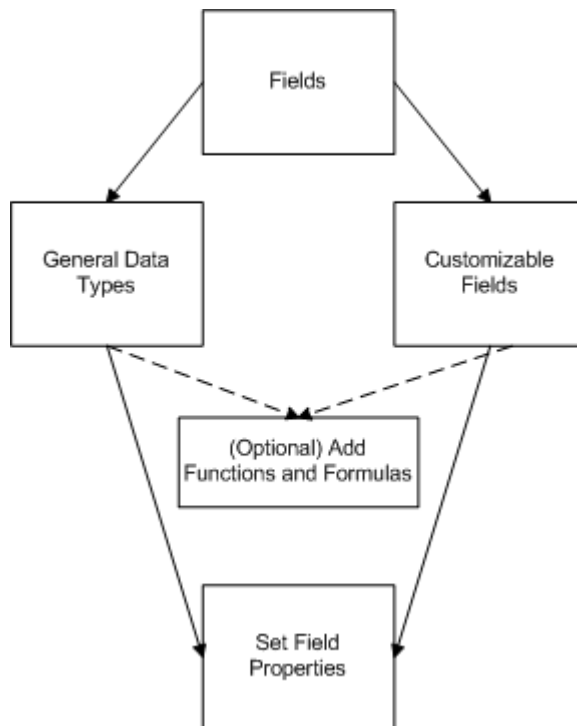
Preface

This module contains information about creating assets and working with data in Application Integration.

CHAPTER 1

Understanding Data Types and Field Properties

The following diagram explains what you learn in this chapter:

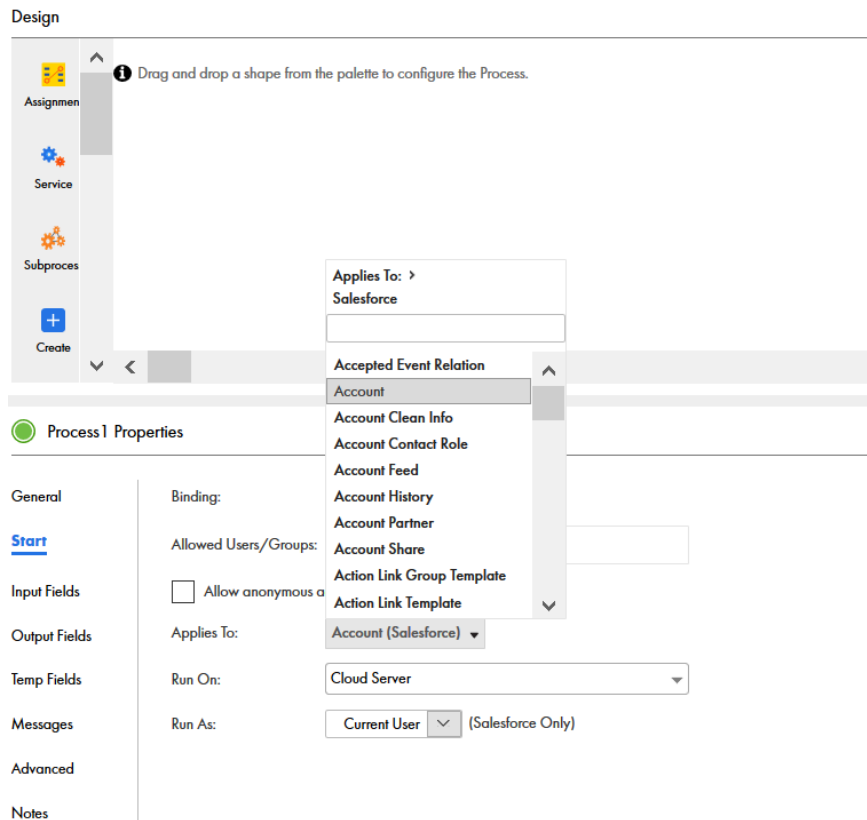


Introduction to Data Types and Field Properties

Process Designer uses information contained within your application database. Each database table contains object types that you can access through your connections to work with the fields needed in a process.

When you define process or step properties, you drill down to view the objects and select associated fields from a list.

For example, in following case, the Salesforce objects give you access to many object types that you can use in the process:



In most cases, a process determines what type of data is being stored based on the application. For other fields, you may need to define them as Simple types and select the appropriate data type. Apart from the data type, you might want to control other field properties. For example, you could specify a default value or hover text to help the user.

To update field properties, simply click in the field to edit.

Read the topics in this section to learn more about the field properties and data types.

Types of Data and Field Properties

You can specify Field Properties, derived from API data types and options, in the Informatica Process Designer. Field Properties may be either of the following:

- General data type properties. For example, specified on the **Fields** tab of the Process Properties.
- A field that you can tailor for the context. For example, a pick list on a Screen step.

The following base options are available for many data types in the Field Properties:

- **Default Value** or **Initial Value**: The value for an item when it is initially displayed.
- **Field**: The name of the field from which the **Field Properties** dialog is invoked.
- **Hover text**: Text that a user sees when they place their cursor over the field.
- **Required**: If checked, a value for this field must exist.
- **Show List**: Values to display are determined at runtime based on what is configured here.

The following table explains built in data types:

Note: Depending on where you access the Field Properties, the available data type options may vary.

Data Type	Description
Any	Use the Any data type if you want to declare fields that are process objects without setting the Reference To option. This means that the data type is an object, but you do not need to specify which object it is. If you use the Any object, you cannot drill further into the object. Some services return large amounts of data, but you may need only a small percentage of it. Use the Any data type to model all the data returned in to just a few fields. For example, when you use a REST proxy, it passes a received payload as input and then responds to this payload. The REST proxy would do some processing on the request or response, and it would not have to understand the whole message. A second example would be supporting heterogeneous lists. While simple REST APIs tend not to include heterogeneous lists, SOAP-based APIs do. For example, a SOAP API might have a "query" operation that returns data that depends on the query itself. Here, the static type of the output for the "query" operation would be an objectlist of the any type. A third example is calling a service that returns records that match a query. Each record can have a unique set of fields, so those could not be known at design time. At runtime, each field would just be turned into a child element of the record element, using a tagname that corresponds to the field name. The converted JSON returned as output might be: <pre> { "record": [{"ROWID_OBJECT": "2001", "CREATOR": "admin", ...}, {"ROWID_OBJECT": "2002", "CREATOR": "admin", ...}, ...] }</pre>
Attachment, Attachments	Use Attachment and Attachments to a process to pass through an attachment and extract data like file size and file name using functions on the response. The default Maximum File Size is 5,242,880 bytes and the default Maximum Number of Files is 10. You can change these field properties. Note: If you run a process on the cloud server, do not use an attachment whose size is more than the default 5,242,880 bytes. The cloud server cannot process attachments that are greater than 5,242,880 bytes.
Checkbox	A checkbox allows the user to make a true/false decision. This decision can alternatively be displayed as Yes/No or No/Yes For example, you could use a checkbox to indicate if an address is a work phone number or a home phone number. As an alternative, a checkbox can be displayed as yes/no values. - Show as: a picklist that has the following items: Yes/No and No/Yes. Your choice sets the labels that the user will see.
Currency	A currency field contains a monetary value. In addition to numbers and perhaps a decimal point, you can also use commas where they are needed. Commas are always optional. - Length/Decimal places: Specify how many numbers can be entered (the left box). Use the right box to enter the maximum number of digits to the right of the decimal point. If you enter 0, there are no digits to the right of the decimal point. - Hide currency symbol: When checked, a currency symbol does not display.

Data Type	Description
Date	The output date is in UTC, and output date and time are in ISO 8601 format. For example, if you enter an input date of 2016-03-29, and use the currentTime XQuery function to get date and time values, you see the following output: - Date: 2016-03-29Z - dateTime: 2016-03-29T06:00:48.525Z - Time: 06:00:48.525Z The following are sample valid date values: - 2001-10-26 - 2001-10-26+02:00 - 2001-10-26Z - 2001-10-26+00:00 - -2001-10-26 - -2000-04-01 If you do not pass a time zone, Informatica Process designers assumes the time to be in UTC. If you need to format the way in which the date appears, see <i>Formatting Dates, Times, and Numbers</i>.
Date Time	When selecting a Date Time field in a process, the user can select the time from a list. - Default Value: Either enter a specific date, a field, or an interval, as shown in the figure. - 30 Minute Increments: If checked, a time picklist displays lists hours and half hours. If it isn't checked, the user can type in any time value (for example, 10:37). The output time is in UTC, and output date and time are in ISO 8601 format. For example, if you enter an input date of 2016-03-29, and use the currentTime XQuery function to get date and time values, you see the following output: - Date: 2016-03-29Z - dateTime: 2016-03-29T06:00:48.525Z - Time: 06:00:48.525Z The following are sample valid dateTime values: - 2001-10-26T21:32:52 - 2001-10-26T21:32:52+02:00 - 2001-10-26T19:32:52Z - 2001-10-26T19:32:52+00:00 - -2001-10-26T21:32:52 - 2001-10-26T21:32:52.12679 If you do not pass a time zone, Informatica Process designers assumes the time to be in UTC. If you need to format the way in which the date and time appears, see <i>Formatting Dates, Times, and Numbers</i>.
Email	The content of this field is an email address. - Display # chars: The number of characters that are displayed within the process for this address. - Max # chars: The largest number of characters that can be entered for this address.
Formatted Text	The format is a representation of the kind of character a user can type within the text being entered. - Format: A pattern for text that the user will type. The characters you can when defining a format are: - A: An upper- or lowercase letter; that is A through Z. - 9: A number. - *: Any letter, number, or symbol Note: the "." is not listed. If you can type this character, it is displayed when the user sees the step.

Data Type	Description
Image	A field that will contain an image. This is most often used when an automated action retrieves an image. - Show preview: Displays a preview of the image. - Max width: The maximum width of the area in which the image is displayed. - Max height: The maximum height of the area in which the image is displayed. - Display # chars: The number of characters that are displayed within the process for this address.
Integer	A positive or negative whole number. Use the Digits field to enter the maximum amount of numbers the user can type. Numeric values display using the format of the user's locale. If you need to format the way in which the number appears, see <i>Formatting Dates, Times, and Numbers</i>.
Multi-Select Picklist	A picklist field from which the user can select one or more rows. To select multiple items, the user clicks each one while pressing the CTRL key. - Height (rows): The number of rows, each containing one list item that displays in the process. - Comma Separated List: The list of items within the list. Separate each item in the list with a comma. Each item specified is available in the list at runtime. Usage Notes - If you declare the data type of an input field as a Multi-Select Picklist or a Picklist and you do not enter values for the settings, you must use a semi-colon ";" as a separator instead of a comma. Be sure to use a semi-colon for an Equals condition and a comma for a Contains condition. - If the available values are not specified in the Properties for a Multi-Select Picklist or a Picklist and you receive a list of values from a search service, you must: 1. Specify the search service; and 2. Populate the list from a Screen step or Input options in other steps. Note that the list of available values exists only in the current step. To use the same list in other steps, specify the search service again. - You can also assign values to an undefined Multi-Select Picklist or Picklist in the Assignment step. In that case, use Formula as the Source and the value(s) of the Picklist or Multi-Select Picklist as the Content. When you enter a formula, be sure to enclose the values in quotes, for example: "value" for a Picklist and "value1;value2" for a Multi-Select Picklist.
Number	A positive or negative decimal number. Use the Length/Decimal places field to specify how many numbers can be entered (the left box). Use the right box to enter the maximum number of digits to the right of the decimal point. If you enter 0, there are no digits to the right of the decimal point. Numeric values display using the format of the user's locale. If you need to format the way in which the number appears, see <i>Formatting Dates, Times, and Numbers</i>.
Object ID	The object ID is the ID of the object instance. For example, it identifies one of your leads. (This value is assigned to the object by your application.) The unique parts of this dialog are discussed after this table. . Fields that are used here and in Field Properties dialogs are discussed at the top of this topic.

Data Type	Description
Object List	An object list is a set of object IDs. An object ID is the ID of an object instance. For example, it identifies one of your leads. - Reference To: The type of object that can be contained in the list. You must enter the object type's official name, not its label. If the list can contain more than one object types, you can enter a comma-separated list of object types. However, you should avoid doing this as there are query and table capabilities that can't be provided for fields unless they only have one Reference To type. - Display Fields: The columns (they must be manually entered) from the data associated with the object ID. - Height (rows): The number of rows, each containing one list item, that displays in the process. - Width (%): The percent of the area in the used to display information. A read-only object list has a limit of 100 rows, even if the query associated with uses a LIMIT clause, asking for more rows to be returned. Other fields are discussed at the top of this topic.
Percent	A number expressed as a per cent. - Precision: The maximum number of digits in the number. - Scale: The maximum number of digits to the right of the decimal point. If you enter 0, there are no digits to the right of the decimal point. Numeric values display using the format of the user's locale. Here are some examples--each of these examples shows a number greater than 100%: - Precision 5, scale 2: 123.45 or 123.4 or 123. or 123 - Precision 5, scale 0: 12345 or 12 or 12,345 If you need to format the way in which the percent appears, see <i>Formatting Dates, Times, and Numbers</i>.
Phone	A field into which you can type a phone number. - Format: Enter the character that represents a number, which is 9 and any other display characters. For example, 99-999 will display as "_ _ - _ _" in the process.
Picklist	A picklist field from which the user can select one row. - Comma Separated List: The list of items within the list. Separate each item in the list with a comma. Within the process, each item displays in its own row. Usage Notes - If you want to display text to the user and assign a different value to the field, separate the label from the value with an equals ("=") sign. For example, if you type <i>Red, White, Blue</i> as the values, the user sees these three strings and the selected string is assigned to the field. However, if you enter <i>Red=1, White=2, Blue=3</i>, the user sees the same three strings but the number value for the selected item is assigned to the field. - See also the Usage Notes for Multi-Select Picklists for information on receiving a list of values from a search service.
Rich Text Area	A field that will contain HTML commands as well as text. Enter this text using the HTML editor. - Height (rows): The number of rows within the area the process displays into which the user can type text. - Width (%): The width of the text area relative to the process width. You cannot enter JavaScript into a Rich Text Area.
Text	A control that changes the size and number of characters that the user can type text. - Max # chars: The maximum number of characters that can be typed when entering text. - Display # chars: The number of characters that are displayed within the process. As the process size is limited, use this field to show how much of what the user types is displayed.

Data Type	Description
Text Area	A control into which the user can enter text. This can display as one or more rows. - Width (%): The width of the text area relative to the process width. - Height (rows): The number of rows within the area the process displays into which the user can type text.
Time	Select the Time field to select time from a picklist. The field properties dialogs that are displayed are: The output time is in UTC, and output data and time are in ISO 8601 format. For example, if you enter an input date of 2016-03-29, and use the currentTime XQuery function to get date and time values, you see the following output: - Date: 2016-03-29Z - dateTime: 2016-03-29T06:00:48.525Z - Time: 06:00:48.525Z The following are sample valid Time values: - 21:32:52 - 21:32:52+02:00 - 19:32:52Z - 19:32:52+00:00 - 21:32:52.12679 If you do not pass a time zone, Informatica Process designers assumes the time to be in UTC. If you check <i>30 Minute Increments</i>, you see a time picklist that displays hours and half hours. If it is not checked, you can enter any time value. For example, you can enter 10:37. If you want to format the way in which the time appears, see <i>Formatting Dates, Times, and Numbers</i>.
URL	A field that will contain a URL. - For Read-only Display: Your choices are <i>Link</i>, <i>Button</i>, or <i>IFrame</i>. - For Read-only Label: Text that identifies the contents of the field. - Height: The height of the area in which the URL is displayed. - Width: The width of the area in which the URL is displayed. - Display # chars: The number of characters that are displayed within the process. As the process size is limited, use this field to show how much of what the user types is displayed. A URL can be an input, output or temporary process field or even a field from. The field's data is assumed to be a valid URL. To create a button in a screen step: 1. Create a field by using the <i>Fields</i> tab in the process properties. 2. Use an Assignment step to assign it data to the URL being linked to. This data, which will be a content, can contain references to other fields such as ID fields by using <code>{!FieldName}</code> syntax. Process Designer automatically inserts the data using the button to the right of the text entry area. This is very useful when linking to other pages or other data dependent hyperlinks. 3. On any screen following the Assignment step, place the field within a step by selecting the file from the Read-Only fields picklist. 4. Select the field's ... button to specify its properties. Change For Read-only Display to Button and enter the text you would like to appear in the button in the For Read-only Label.

Data Type	Description
XML	You can use XML data type with only with an Informatica Process Designer process. To use the XML data type in a process, you can do either of the following: - You can use the XML data type in the assignment step of a process. - You can use the XML data type with JMS or AMQP. Here, the payload can be XML data and you can design a process that sends or message or receives a message. If you use XML data with JMS or AMQP, wrap the payload within a root element. To map simple XML data elements to other fields, use formula and XQuery functions. For example, consider the following XML: <pre><o:order xmlns:o="urn:purchasing:system"> <o:number>123</o:number> <o:amount>9213.32</o:amount> </o:order></pre> You can use this XQuery function to search for the order number: <pre>\$input.myInboundOrder/*:number/text()</pre> When you query XML data with namespaces, you must use the asterisk (*) character.

Object ID

The object ID is the ID of the object instance. For example, it identifies one of your leads. (This value is assigned to the object by your application.)

- Reference To:** The type of object that the field refers to. You must enter the object's official name, not its label. If the object can refer to more than one type, you can enter a comma-separated list of object types. However, you should avoid doing this as there are query and table capabilities that can't be provided for fields unless they only have one **Reference To** type.

A polymorphic relationship is a relationship where the referenced objects can be one of several different object types. If the object ID can refer to more than one object, check the Polymorphic field. A picklist is now available for choosing these objects. For example, while the Account object was selected, objects that could also be referenced are "Accepted Event Relation", "Account Contact", and so on. In this list
- Display Fields:** The columns (they must be manually entered) from the data associated with the object ID.

When you are using this field in other steps, the information displayed differs. Here's a Process Designer example. When inserting the field, you have two choices.

If you insert the first (PolymorphicID), and click on the "..." in inserted field, the **Field Properties** dialog that displays shows all of the items selected in the Properties dialog:

However, when you select the item from within the Polymorphic section of the Insert Field for Update picklist, what you see are the individual items that were named. You will need to click again to get to the item that you actually want.

Object ID Notes and Comments

When an Object ID is placed in a canvas as a read-only field or put into a column in a table, it displays as a link to the object. If you want to show the Object ID as a number, insert it into a text field as "{!Id}". However, when an ID field is included in a result from a **Lookup** dialog, the ID's value displays.

When an object ID is displayed in a column or picklist, Process Designer displays with a meaningful name. For example, b6f0f0b2-9f38-4771-8365-58eb4f7b7d41 might display as "Acme". However, when an object ID field is included in the results from a Lookup Dialog, it displays using the value of the ID; for example, it might display as b6f0f0b2-9f38-4771-8365-58eb4f7b7d41.

If you use a content field to enter an object ID and are using a sandbox, you will need to adjust the ID when you move the process to your production environment. Rather than using a content, you could use a query to load the object ID by name for the object.

Formatting Dates, Times, and Numbers

You can format values within the **Field Properties** dialog (see [“Types of Data and Field Properties” on page 10](#)) or within a function invoked from a field where you set the Source to Formula.

The output of a format is a display value and therefore a string.

The `infa.format()` function has four arguments that enable you to specify the format. For example, if you set the Source to Formula, you can use an expression similar to the following:

```
infa.format($output.Created Lead.AnnualRevenue * .75, "#####.")
```

Process Designer evaluates the text of the format string and places a value in that format. For example, the format `"#,##0.00"` tells Process Designer that at least one number to the left of the decimal point must display as well as two to the right. If they are not present, use a zero. Also, depending upon the length, it can insert a comma.

Refer to [“Using Functions” on page 26](#) for more information.

The patterns you can use for formatting values are the same patterns used in Java classes. If you need information on:

- Dates, see <http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html> (the `SimpleDateFormat` Java class).
- Numbers, see <http://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html> (the `DecimalFormat` class).

Using the Field Properties Dialog

When you define steps within a process, you insert fields to handle input, output, or other variables that you might need to receive from a service or pass to another step in the process. Each of the fields you add has a set of field properties that you can configure to determine the field source, the formatting, display properties, and list handling.

For some fields, you can also use the Expression Editor to define complex formulas that determine the field values and other attributes.

The topics in this section describe the options in the Field Properties dialog and the kinds of queries that you can write for use with field properties.

Controls in All Many or Some Field Properties Dialog

Most **Field Properties** dialogs have the following controls:

- **Default Value:** Sets a default value for the information being displayed. Your choices are *Content* (object type), *Field* (has the object type), *Formula*, and *Screen* (user specified object).
- **Display Fields:** Names the fields to be displayed in columns and the order in which they display.

- **Field:** The name of the field from which the **Field Properties** dialog was invoked.
- **Height (rows):** Sets the number of rows that will be displayed.
- **Hover text:** Text that is displayed after a user places the cursor over the field.
- **Reference To:** The type of object to which the field applies. You cannot change this value—not in all dialog boxes.
- **Required:** Click this checkbox if this value must either be entered or cannot be deleted and not replaced.
- **Show List:** Defines the way in which objects are located.
- **Width (%):** Sets the width of the displayed information as a percentage of the width of the process.

Show List: Advanced Query

Selecting **Advanced Query** lets you finely tune the information retrieved for the list.

After the query executes, the default action is to return a list whose values are object IDs; the list also has the name field for display.

The source for each of the query's four parameters can be Content, Field, Formula, or Screen. In almost all cases, the source is Content. The items that you can set are follows:

- **Object Type:** The object type upon which the query executes.
- **Where Clause:** Enter an SQL WHERE clause. The text you enter here does not include the WHERE keyword. Here, Content means you will write the WHERE clause, Field means that the field contains the WHERE clause, etc.
- **Display Field:** Set a value here to return a field other than the current one. Process Designer will retrieve this data, and then display it within a picklist. If you do not specify a field, Process Designer uses the *namefield* (the first one, if there is more than one).
- **Value Field:** Returns a different value than what Process Designer would use by default. If you do not set a value field, the ID of the returned object is used.

Advanced Technique: You could use an advanced query to create a step that displays the query's results within a step. You might want to do this in a different "testing" guide.

Show List: Custom List

Selecting **Custom List** let you enter your own list of values. For example, you might individually type the names of the New England states. When the process displays, the user sees the items you enter in a picklist.

You must separate items from one another using a comma (",").

Notice that you can also set a *Default Value*.

Show List: List Child Objects

Use related (or child) objects in the query. While similar to using related objects within fields, using **List Child Objects** is simpler as you do not need to specify the query's condition.

After you select **List Child Objects**, you also need to specify the child's object type. Your choice for source is automatically set to "Content", which is the only allowable source.

Show List: None

Use this setting to display a search box that allows the user to specify any value for the field. If the field is a reference to an object, not using *Show List* lets the user locate objects of the type named in the *Reference To* area.

You control how to display information about objects using the *Display Columns* item.

Show List: Object Query

Use a query to retrieve object information from all of the objects that are selected by the *Where Clause*.

After you select **Object Query**, the Process Designer adds the Object Query controls to the **Field Properties** dialog. The *Source* for the *Where Clause* is always *Content*. While you can enter your condition directly into the text area, it is usually far easier to click the **Add Condition** button.

Note: The difference between a value and what is shown in the display field may not be clear. In many cases, they may be the same. The difference is that the display field is what is shown to the user while the value is what is stored. For example, if a currency value is stored with two places after the decimal, its display could omit these numbers. This distinction is almost always used with ID fields as you want to display something like a Name while using the ID as the actual value that is stored when the user selects it. Also, you may be using code tables where the code is something other than an Object Id. For example, imagine a *States* object where the code is the two letter abbreviation that is stored as the value in other objects, but you want to display the full name to users when they are selecting *State*.

After using this dialog, you can edit the information that it added.

Selecting the field from the picklist is suggested as it is sometimes not obvious what the field's internal name is. The text entered here is a standard SQL WHERE clause. Also, this text does not include the WHERE keyword.

If you are creating more than one condition, you can use the standard AND and OR operators. You can also use the NOT operator to invert the meaning of the condition.

Each condition has four parts:

- The name of a field in the object. In this example, the field is one of those contained within the *Account* object (this is the object named in the *Reference To* area).
- An operator that Process Designer uses when it compares the field value on the left with values on the right.
- The kind of data that will be compared. Your choices are *Content*, *Formula*, or *Field*. If you choose *Field*, you are comparing the contents within the field in the current object with the contents of the field in the type of object being queried.
- The data to which the field on the left is being compared. If the source is *Content* and you click within this area, a small icon appears to the right. After selecting it, Process Designer displays a picklist from which you can select the name of a field in another object that will be used when making the comparison.

To add an SQL LIMIT clause to a query, enter it in the **Where** clause text box. For example, enter "Limit 200" to limit the number of retrieved rows to 200. Add a space after the WHERE information and then enter the SQL LIMIT. Else, enter the SQL LIMIT information on a new line. ("Limit" can be in upper, lower, or mixed case.) By default, 100 rows are returned. If your process requires more than 100, you must add this clause. However, a read-only object list has a limit of 100 rows, even if you set a query associated with it to more than 100.

If you use a JDBC connection, you can use the ORDER BY unction to sort data.

To use the order by function, click **Order By** and select a field. For example, you can sort a list of names by a person's last name.

Note: You cannot use the order by function to make OData-enabled requests to a JDBC connector.

Show List: Picklist for Field

Record types are information that you associate with objects that let you select some of the objects. For example, suppose you have "On the Road" and "In the Office" records types for a lead. Your process can use these values to select what other information it should display. If the record type value is "On the Road", you could have a picklist with values A, B, and C. If the value was "In the Office", picklist values could be C, D, and E.

Use this option when using a "Create" Service step with a dependent picklist. While Object Type and Field Name are automatically filled in, you will need to enter a Controller Value. If the user will fill in the field containing the record type, set the Source to Field and also select the appropriate field that this is dependent on. If it is Content just enter its value.

For example, you have a Create Lead Service step that has a "Business Sector" picklist and an "Industry" picklist dependent on it. When the user selects "Technology" from the "Business Sector" picklist, Process Designer can display values set for that sector in the "Industry" picklist, values such as "Computer Hardware" and "Biotechnology." In order for this to operate correctly on the Create screen, you must edit the field properties for the "Industry" field and change the Controller Value to Field and select "Business Sector" as the field.

Your options when you select Picklist for Field are:

- *Object Type:* The object type containing the object field. This should already be filled in correctly, so you don't need to change it.
- *Field Name:* The name of the field that defines the picklist values. This should already be filled in correctly, so you don't need to change it.
- *Controller Value:* For dependent picklists to display correctly, fill in this controller value. Setting this value to be content is not a good idea since the ID for a record type would be different when used in sandbox and or in the product. What you should do is use a query to dynamically set the controller value. This query must return just one value.

Show List: Users in Role

Users in Role tells Process Designer to locate all users who have the role that you identify.

When you choose this option for Show List, you also select the Role name that Process Designer will use to locate users in that role. The source for each of these can be "Content", "Field", "Formula", or "Screen".

Using the Expression Editor

The Expression Editor allows you to edit field definitions when you define a process or service connector.

You can open the Expression Editor from Process Designer in two contexts:

- When you select Formula as the Source type for a field definition so you can get the field value from a formula.

- When you define the URL, parameters, or HTTP headers for a service connector.

Expression Editor allows you to:

- See syntax highlighting for XML and XQuery.
- View syntax validation feedback.
- Select code completion options that appear as you type.
- Insert common code fragments from a drop-down list.
- Use keyboard shortcuts.
- Choose fields from a list of available input, output, or temp fields defined for the process, the current step, and any upstream steps in the current process. When you choose a field, the appropriate value (for example, '\$input.Customer') is inserted into the editor.
- Choose a function from the list of available functions, grouped by category. The list includes some common XQuery functions and displays syntax help as a tool tip, when available.
- Insert the sample XML representing the selected item (for a process object xml element or for a child of a process object).
- Insert common code fragments from a drop-down list.

Opening the Expression Editor

Where available, click the **f(x)** icon to open the Expression Editor.

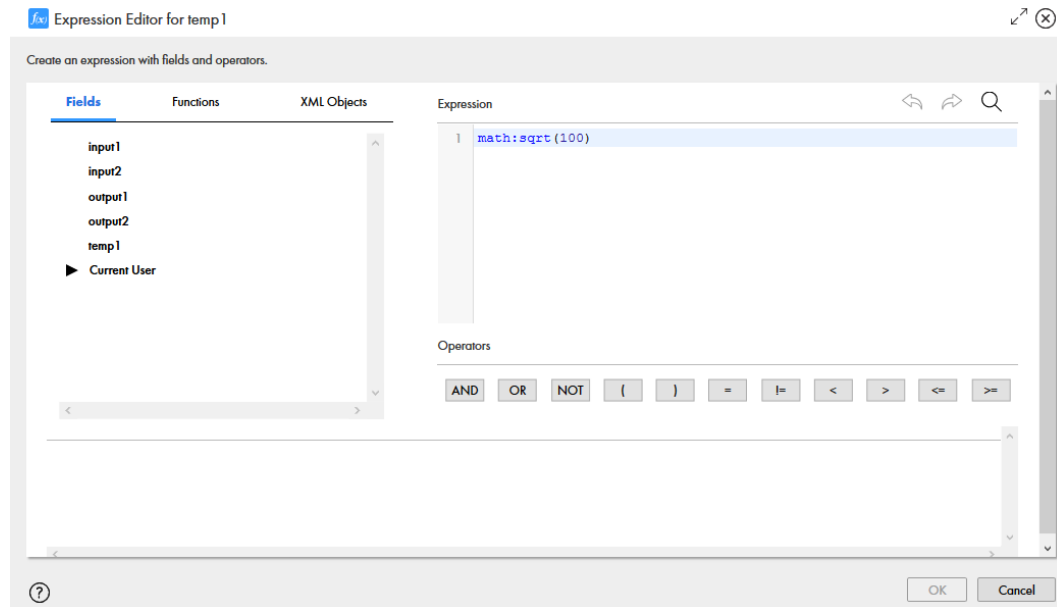
For example, in an Assignment step, you might add an input field with a Formula as the data source. You can select **Formula** from the list, as shown in the following image:

Assignment to temp1, input1 etc. Properties

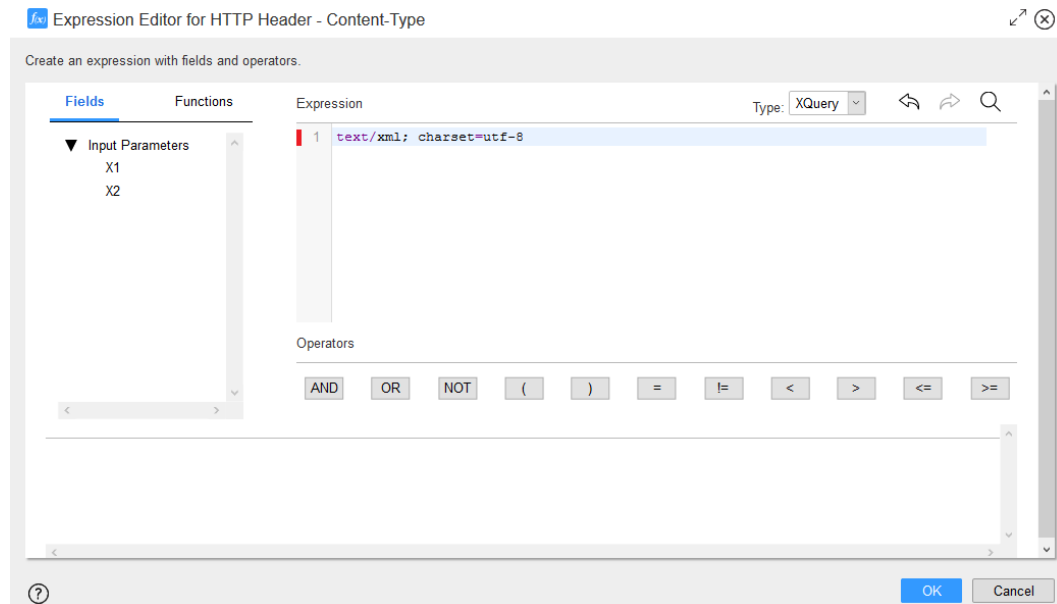
General	Target	Value
Assignments	temp1	Formula math:sqrt(100) f(x)
	input1	Field temp1
	output1	Field input1
	input2	Content 100 f(x)
	output2	Formula <input "="" type="text" value="util:base64Encode("/> f(x)

Add Field **>< Reorder**

After the Expression Editor opens, you can use the features described below. The field type and name appears in the window title, as shown in the following image (input.TestInput):



When opened from a service connector, you can choose a type (XML, XQuery, JSON, or Content) and see the fields available in the Connection Properties and Input Parameters, as shown in this image:



Editing Options

When using the Expression Editor, you can use the toolbar and/or the following keyboard shortcuts, which are available when the editor is active (the cursor is blinking):

Undo	Ctrl+Z
Redo	Ctrl+Y
Copy	Ctrl+C
Cut	Ctrl+X
Paste	Ctrl+V
Find	Ctrl+F
Indent four spaces	Tab
Show list of available variables	\$
Show list of available insertions (namespaces, functions, fields and common code fragments)	Ctrl+Space (You can also start typing in the editor to filter the list and select the function or variable you need.)

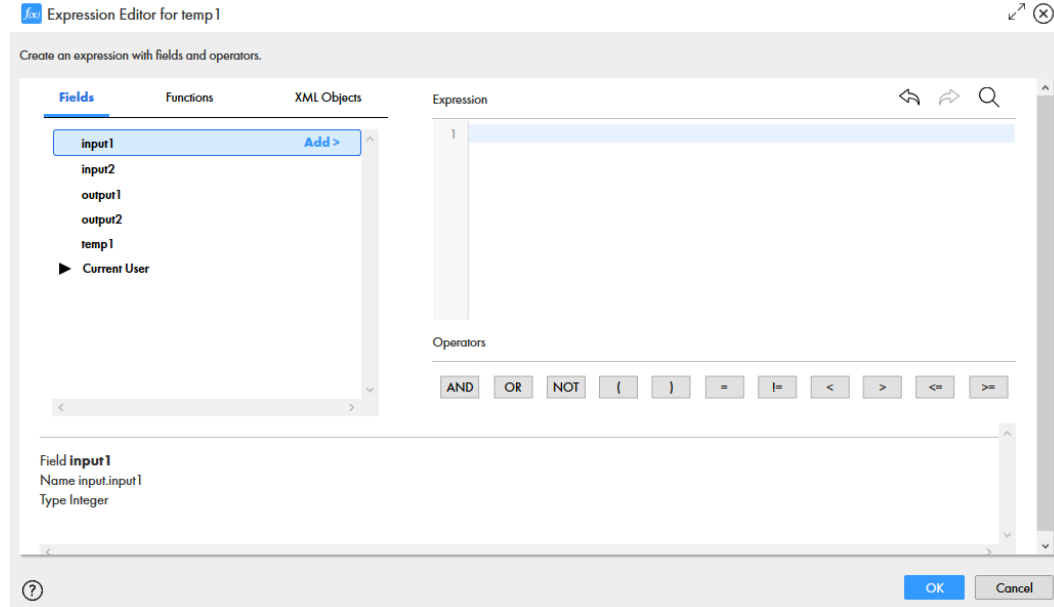
You can also expand the editor window to full screen using the toolbar icon. Click Esc to close the full screen editor and return to the canvas.

Building a Formula

To enter a formula that determines the value for the selected field, you can access these options from the toolbar:

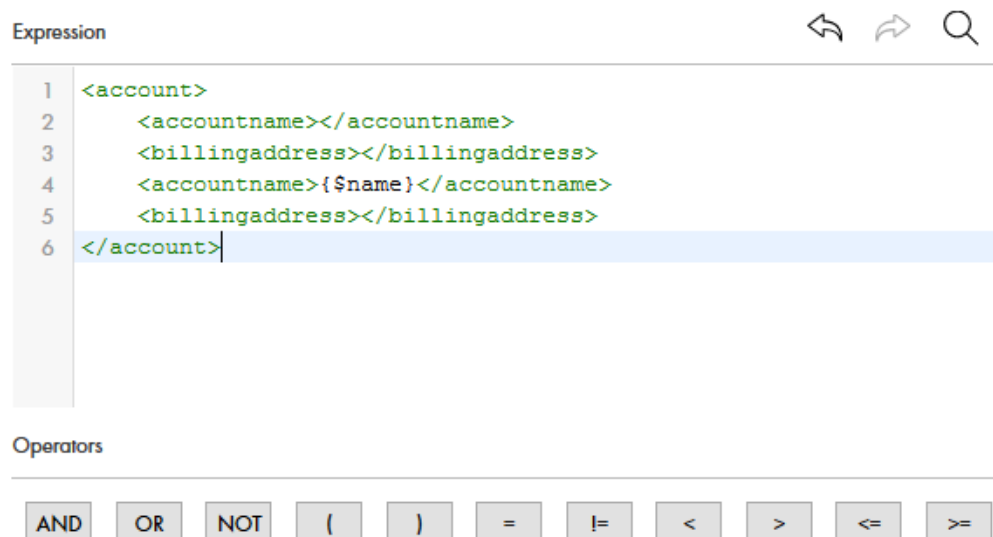
- Insert Field. Shows the list of available fields you can insert into the formula.
- Insert Function. Shows the list of available functions.
- XML Object. Shows a list of XML objects, if any, that you can add.

For example, in the following image, the **Insert** list shows the field names associated with the object:

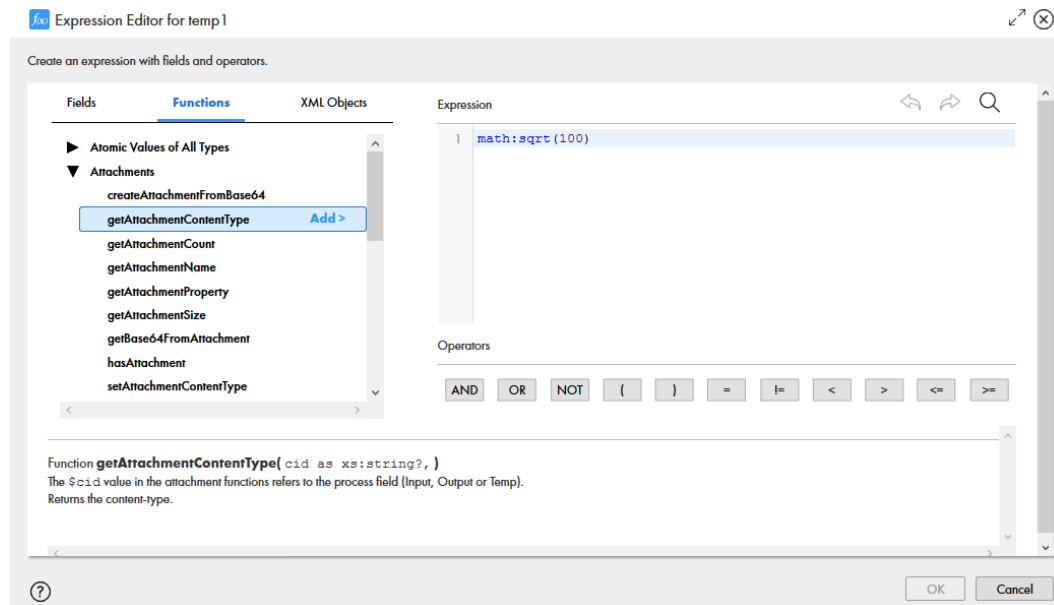


Choose **Functions** to show a list of those available, sorted by category.

Use the **XML Objects** list to add an XML code snippet for the field when you are defining a process object. For example, you might create an XML snippet to represent a process object and then use Insert Field to add the input field as shown in this image:



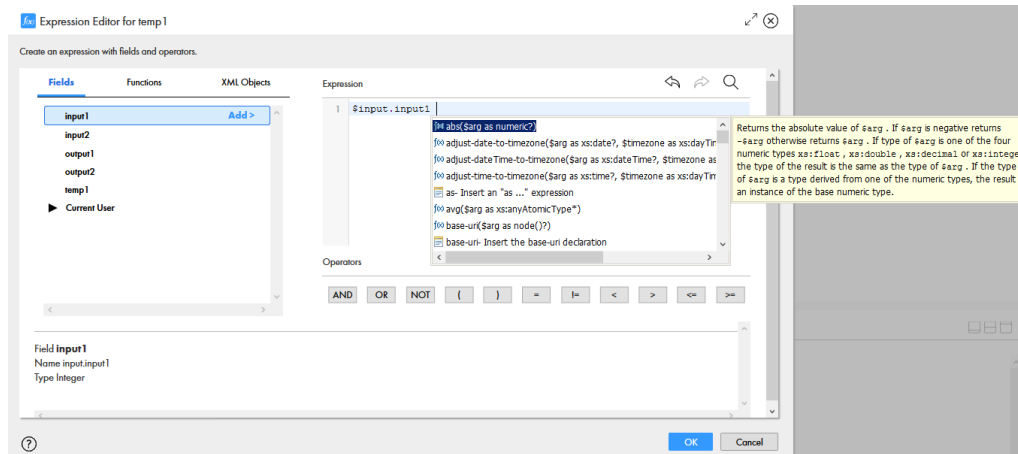
To get more information about a function, hover over the function name in the list to display a description. For example:



Note the following tips when you work with the editor:

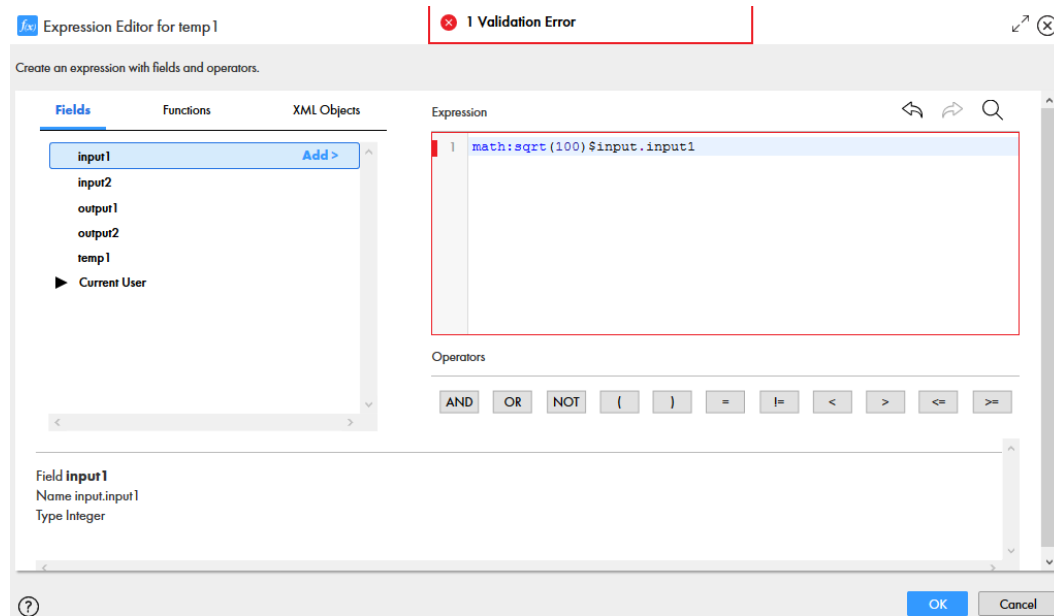
- Press **Ctrl+Space** to display a list of available insertions (namespaces, functions, fields and common code fragments).
- Press **\$** to display a list available functions, process fields and any locally declared XQuery variables.
- If the expression is used in XML, be sure to add braces around it, for example:

```
<Value>{2*fn:abs($input.In)}</Value>
```
- Enter a few characters to filter your list.



Syntax Validation

As you build a formula, the syntax is validated. If an error is detected, a red X appears next to the line that contains the error. Hover over the error indicator to display a description of the error:



As you type, the syntax is validated and the Expression Editor displays a message if it encounters an error condition.

XQuery error messages are defined by the XQuery engine. You can learn more about XQuery validation and see a list of XQuery error conditions (in *Appendix F*) here: <http://www.w3.org/TR/xquery-3/>.

Using Functions

You can use the following functions when working with XQuery in Process Designer or in any fields that use a formula to set a value.

For information on XQuery functions that are available in the Formula Editor but are not described below, see <http://www.xqueryfunctions.com/xq/alpha.html>.

For information on using the Formula Editor to work with these functions, see [“Using the Expression Editor” on page 20](#)

Atomic Values of All Types

The following functions are supported in processes and service connectors:

- boolean
- empty
- exists
- false
- nilled

- not
- number
- string
- true

Attachments

The functions described in the following table are available for handling attachments, only in processes.

The *\$cid* value in the attachment functions refers to the process field (Input, Output or Temp). For example, if the input parameter is called *customerPhoto* (of type attachment), to get the attachment size, use:

```
sff:GetAttachmentSize($input.customerPhoto)
```

Note: Be sure that your field names do not have spaces so they can be easily referenced in XQuery.

For more examples of using the attachment functions, see [“Attachments” on page 37](#).

Function	Syntax	Description
hasAttachment	sff:hasAttachment(\$cid as xs:boolean)	Checks if an attachment exists.
base64EncodeAttachment	sff:base64EncodeAttachment(\$cid as xs:string?)	Indicates base64-encoded attachment contents.
getAttachmentContentType	sff:getAttachmentContentType(\$cid as xs:string?)	Returns the content-type.
getAttachmentCount	sff:getAttachmentCount(\$cid as xs:long)	Returns the number of attachments.
getAttachmentName	sff:getAttachmentName(\$cid as xs:string?)	Returns the attachment (file) name if available.
getAttachmentProperty	sff:getAttachmentProperty(\$cid as xs:string?, \$attribute as xs:string)	Returns the attachment attribute, given the mime header name such as 'content-type'.
getAttachmentSize	sff:getAttachmentSize(\$cid as xs:long)	Returns the attachment size.
setAttachmentContentType	sff:setAttachmentContentType(\$cid as xs:string, \$val as xs:string)	Sets the attachment content-type.
setAttachmentName	sff:setAttachmentName(\$cid as xs:string, \$val as xs:string)	Sets the attachment name.
setAttachmentProperty	sff:setAttachmentProperty(\$cid as xs:string, \$attribute as xs:string, \$val as xs:string)	Sets the attachment mime header attribute value.

Function	Syntax	Description
createAttachmentFromBase64	sff:createAttachmentFromBase64 (\$contentName as xs:string, \$encodedContent as xs:string, \$mimeType as xs:string)	Creates an attachment from the base64-encoded content.
getBase64FromAttachment	sff:getBase64FromAttachment (\$cid as xs:string)	Returns base64-encoded attachment content from the variable, which has the attachment type.

Dates and Times

Process Designer supports most built-in XQuery date and time functions described at <http://www.xqueryfunctions.com/xq/alpha.html>, in addition to the following:

Function Name	Syntax	Description
now	date:now()	Returns the current time in milliseconds.
millisToDate	date:millisToDate(\$millis)	Converts the current time from milliseconds.
dateToMillis	date:dateToMillis(\$date)	Converts the time to milliseconds.
getLocale	date:getLocale()	Return a string representing the current locale the process is running in.
getTimeZone	date:getTimeZone()	Returns a string that is the timezone ID of where the process is executing or where the user is executing it.

Digital Signatures

The functions described in the following sections are available for use in digital signatures.

HMAC Functions

The following functions enable you to generate a digital signature by calculating a keyed-hash message authentication code (HMAC):

Function	Syntax	Description
hmacSHA1signature	<code>dsig:hmacSHA1signature(\$data as xs:string, \$key as xs:string, \$encoding as xs:string) as xs:string</code>	Calculates an HMAC using the SHA1 algorithm and the optional encoding. <code>\$encoding</code> is one of the following: - Base64 (default) - Base64Url - Hex - Hex64 <code>\$data</code> consist of four parts, including a JSON string in base64 encoding. To ensure that you get hex binary and not base64 when you use the <code>hash:hash</code> function on the payload, use the following expression: <pre>let \$md5hex := hash:hash(\$jsonPayload,"MD5") return xs:base64Binary(xs:hexBinary(\$md5hex))</pre>
hmacSHA256signature	<code>dsig:hmacSHA256signature(\$data as xs:string, \$key as xs:string, \$encoding as xs:string) as xs:string</code>	Calculates an HMAC using the SHA256 algorithm and the optional encoding where <code>\$encoding</code> is one of the following: - Base64 (default) - Base64Url - Hex - Hex64

Key Signing Functions

The following functions enable you to generate digital signatures based on private keys:

Function	Syntax	Description
signWithKeyFile	dsig:signWithKeyFile(\$messageToSign as xs:string, \$pathToKey as xs:string, \$encryptionAlgorithm as xs:string, \$digestAlgorithm as xs:string, \$encoding as xs:string) as xs:string	Generates a signature using an asymmetric algorithm and a private key, specified in a PKCS8 file. Arguments: - \$digestAlgorithm: SHA1 or SHA256 - \$encryptionAlgorithm: RSA (common) or DSA (if using SHA1 for \$digestAlgorithm). - \$pathToKey: The PKCS8 certificate as a Base64-encoded string (-----BEGIN PRIVATE KEY----- n-----END PRIVATE KEY-----\n) or a binary private key file. - \$encoding (optional), which may be: - Base64 (default) - Hex64 - Base64Url
signWithKeyString	dsig:signWithKeyString(\$messageToSign as xs:string, \$key as xs:string, \$encryptionAlgorithm as xs:string, \$digestAlgorithm as xs:string, \$encoding as xs:string) as xs:string	Generates a signature using an asymmetric algorithm and a private key, specified in a PKCS8 certificate encoded string. Arguments: - \$digestAlgorithm: SHA1 or SHA256 - \$encryptionAlgorithm: RSA (common) or DSA (if using SHA1 for \$digestAlgorithm). - \$key: The PKCS8 certificate as a Base64-encoded string (-----BEGIN PRIVATE KEY----- n-----END PRIVATE KEY-----\n) or a binary private key. - \$encoding (optional), which may be: - Base64 (default) - Hex64 - Base64Url
signWithCertificate	dsig:signWithCertificate(\$messageToSign as xs:string, \$pathToCertificate as xs:string, \$keyRecoveryPassword as xs:string, \$encryptionAlgorithm as xs:string, \$digestAlgorithm as xs:string, \$encoding as xs:string, \$keyStorePassword as xs:string, \$aliasName as xs:string, \$keyStoreType as xs:string) as xs:string	Generates a signature using a PKCS12 certificate. Arguments: - \$pathToCertificate: File location on the agent that contains either a PKCS12 or JKS (Java keystore) certificate. - \$keyRecoveryPassword: Password to access the key in the certificate. - \$keyStorePassword: Password to open the key store. If empty, assumes the keystore is not password-protected. - \$aliasName: Optional. Alias of entry in the keystore on the Secure Agent which contains the key. If the alias is not supplied (or empty), the first entry is used. - \$keyStoreType: Type of keystore, which may be: - PKCS12 (default) - JKS (Java keystore)

Hashing Functions

The following functions enable you to generate a message hash string:

Function	Syntax	Description
hash	hash:hash(\$string, \$alg)	Generates a hash string for a message using the specified algorithm (optional), where \$alg is one of the following: - MD5 (default) - SHA1 - SHA256

List Functions

The following functions allow you to work with lists:

Function	Syntax	Description
append	list:append(\$objectlist, \$newItem)	Appends a new item to a list.
count	list:count(\$objectlist)	Counts the items in a list.
head	list:head(\$objectlist)	Returns the first item in a list.
list	list:list(\$sequence)	Converts a sequence of IDs into a semicolon-separated list of IDs for an object list.
remove	list:remove(\$objectlist, \$position)	Removes the item at the specified position from the list.
replace	list:replace(\$objectlist, \$position, \$newItem)	Replaces an existing item in a list with a new value.
sequence	list:sequence(\$objectlistFieldName)	Converts a semicolon-separated list to a sequence. When an object list field is inserted into formula and content, the field is wrapped in the list:sequence(object_field_name) XQuery function. For object lists in hosted objects, this converts the semicolon-separated list of IDs in the object list into a sequence. For object lists for process objects, the value is already a sequence and the function returns the list unchanged.
tail	list:tail(\$objectlist)	Return all items from the list with the exception of the first item.

List Function Examples

Perhaps the most common example of using the list functions is to get an object from a list each time a step is invoked, possibly from within a repeated Process or Service step. For example:

```
LET mylist.Current:=list:head(mylist.List)
LET mylist.List:=list:tail(mylist.List)
```

The following example converts an object list in a semicolon-separated list, then iterates over each item:

```
for $objectid in list:sequence($objectlist)
...
```

The following example converts a sequence of IDs into a semicolon-separated list of IDs for an object list for hosted objects. For an object list for process objects, it returns a sequence of values:

```
let $mergedObjectLists :=
  ( as:sequence($objectlist1), as:sequence($objectlist2) )
return list:list($mergedObjectLists)
```

Math

In processes and service connectors, you can use the math functions described at <http://www.w3.org/2005/xpath-functions/math>.

Miscellaneous

The following miscellaneous functions are available:

Function	Syntax	Description
base64Decode	util:base64Decode()	Returns the base64-decoded version of the input string, typically used for attachments.
base64Encode	util:base64Encode(data, charSet)	Returns a base64-encoded version of the string provided. Default for charSet argument: UTF-8.
base64EncodeURL	util:base64EncodeUrl(str, charSet)	Returns a base64-encoded version of the string provided, safe to use in a URL. Any "+" and "/" characters are replaced with "-" and "_". Any "=" characters are removed. Default for charSet argument: UTF-8.
error	fn:error	Raises a custom error.
exactly-one	fn:exactly-one	Returns a sequence if it contains exactly one item, otherwise returns errors.
format	util:format(value, pattern, timeZoneId, locale)	Formats string content. Arguments: - value: The string being formatted. - pattern: A pattern describing how the value should be formatted. See "Formatting Dates, Times, and Numbers" on page 17. - timeZoneId: Optional. The time zone ID. If you do not use a stored value (typically returned by a call to date:getTimeZone()), you can omit this argument as Process Designer will call date:getTimeZone(). - locale: Optional. The locale. If you do not use a stored value (typically returned by a call to date:getLocale()), you can omit this argument as Process Designer will call date:getLocale(). If you use a locale argument but do not use a timeZoneId argument, you must add the comma that would follow timeZoneId. For example: util:format("789", "##00.00", "", date:getLocale())
generate-random-string	util:generate-random-string(length as xs:integer)	Generates a random string of the specified length.
generateUUID	util:generateUUID()	Generates a universally unique identifier.

Function	Syntax	Description
getCatalogResource	util:getCatalogResource()	Returns the resource based on a catalog location URL within the organization. Takes any catalog resource in the Informatica Cloud organization as its parameter and returns an element node. To display the element data, add a wrapper function. For example: <pre>serialize(util:getCatalogResource("project:/spi.ipd/services.xml"))</pre>
getProcessId	util:getProcessId()	Returns the process Id of the currently executing or completed process.
getUserName	util:getUserName()	Returns a string that is the login name or ID of the authenticated User running the process .
getUserSystem	util:getUserSystem()	Returns a string that is the name of the system that authenticated the user running the process.
one-or-more	fn:one-or-more	Returns a sequence if it contains one or more items, otherwise returns errors.
parseJSON	util:parseJSON(jsonStr)	Parses the provided JSON string and converts it to XML elements.
parseXML	util:parseXML(xmlStr)	Parses the provided XML string and converts it to an XML element.
random	util:random()	Returns a random number from 0 to 1.
resolveURN	util:resolveURN()	Retrieves the URN mapping for an organization.
safeNumber	util:safeNumber()	Holds a number value safely so it cannot be changed.
setProcessTitle	ipd:setProcessTitle()	Sets the title of the process. Note: Use this function only in Informatica Process Designer.
toJSON	util:toJSON(elements)	Converts the provided list of XML elements to a JSON string.
toXML	util:toXML(element)	Converts the provided XML element to an XML string.
zero-or-one	fn:zero-or-one	Returns a sequence if it contains zero or one items, otherwise returns errors.

Numbers

The following number functions are available in the Formula Editor:

- abs
- avg
- ceiling
- floor
- max

- min
- round
- round-half-to-even
- sum

Sequences

The following sequences functions are available in the Formula Editor:

- count
- distinct-values
- index-of
- insert-before
- last
- position
- remove
- reverse
- subsequence
- unordered

String

The following string functions are available in the Formula Editor:

- codepoint-equal
- codepoints-to-string
- compare
- concat
- contains
- default-collation
- ends-with
- lang
- lower-case
- matches
- normalize-space
- normalize-unicode
- replace
- starts-with
- string-join
- string-to-codepoints
- substring
- substring-after
- substring-before
- tokenize

- translate
- upper-case

XML

You can add common XML functions as you build expressions. From the Formula Editor, a list of common XML functions displays in these categories:

- XML Documents, URIs, and IDs (processes only)
- XML Namespaces and Names
- XML Nodes

Specifying Functions and Variables

To configure service connectors, you may need to specify variables and functions to define bindings, output fields, and other properties.

Built-in Variables

You can reference these variables in a service connector using an XQuery expression:

Variable	Variable Type	Description
\$VariableName	All connection properties	Properties, input and Other Parameters can be specified using this format.
\$ResponseStatusCode	Output field mapping	HTTP response code.
\$ResponseHeaders	Output field mapping	Contains the HTTP response header in an element list where each item is: <header name="Content-Type">text/plain</header> For example: \$ResponseHeaders[@name = "Content-Type"]/text()
\$RESTResponse	Output field mapping	Contains the RESTResponse XML data, including the headers and code. Visible in the Test Results for the Service Connector.

Output Field Mapping Functions

When you define bindings in the service connector, you can use many functions in the Expression Editor.

For information on all the available functions, see [“Using Functions” on page 26](#) and [“Using the Expression Editor” on page 20](#).

The following table describes functions available for service connector output field mappings:

Function	Syntax	Description
responseHeaderExists	<code>svc:responseHeaderExists(\$ResponseHeaders, headerName) : boolean</code>	Returns a Boolean value that indicates if a header parameter exists within the response.
getResponseHeader	<code>svc:getResponseHeader(\$ResponseHeaders, headerName, defaultValue) : string</code> <code>svc:getResponseHeader(\$ResponseHeaders, headerName) : string</code>	Returns a response header value. If the header parameter is not defined, this function returns the optional default value.
getResponseHeaderNames	<code>svc:getResponseHeaderNames(\$ResponseHeaders) : list of strings</code>	Returns a list that contains the names of all the parameters within a response header.

Digital Signature Functions Overview

Using the digital signature functions available for Process Designer, you can use XQuery to create processes and service connectors that handle signing of content. The supported functions are either:

- Symmetric, signed using Hash-based Message Authentication Code (HMAC), which is based on a shared password and key, and allows you to use multiple key formats.
- Asymmetric, signed using private keys (for example, PKCS8, PKCS12, Java KeyStore).

One common use for signed digital content is to allow for API authentication in a service connector, which allows you to use hashing and related functions on string-based content. For example, you might need to use signed headers as part of the payload

For more information on each of the functions, see [“Using Functions” on page 26](#).

Note: There are currently no supported functions to verify signed content, or to sign binary content.

HMAC

The HMAC signing method is used with many services, including:

- Amazon Web Services (AWS)
- Twitter OAuth

Process Designer supports HMAC SHA1 and SHA256, with Base64, Hex, Hex64 or Base64Url encoding. For example, you might use one of these functions to sign content for AWS:

- `dsig:hmacSHA1signature($data as xs:string, $key as xs:string) as xs:string`
- `dsig:hmacSHA256signature($data as xs:string, $key as xs:string) as xs:string`

AWS Authentication

The AWS REST API requires the authentication header in this format:

Authorization: AWS AWSAccessKeyId:Signature

where:

Signature = `dsig:hmacSHA1signature($strToSign, "AwsSecretKeyId")`

The `AWSAccessKey` and `AwsSecretKeyId` are generated by AWS and accessible through the AWS Developer Console.

Note: `$strToSign` is combination of HTTP header values and other parameters.

For more information on REST authentication for AWS, see:

<http://docs.aws.amazon.com/AmazonS3/latest/dev/RESTAuthentication.html>

Asymmetric Private Key-Based Signing

The following private key methods are supported:

- PKCS8
- PKCS12
- Java KeyStore

Note: If using a private key to sign content, the keystore file must reside on a Secure Agent and the process or service connector must run on the Secure Agent in order to access the key. The file path to the key must be specified in the object properties (for example, "C:/certs/mykey.p12").

You can also:

- Deploy the PKCS8/12 artifact on an agent contribution and specify its location using the 'project://' scheme.
- Supply the key using encoded key content instead of a file location, which may be useful during development of a service connector. For example, you can paste the encoded content into a text area for test purposes.

PKCS8 Key File

To sign with the PKCS8 key file, use this function:

```
dsig:signWithKeyFile( $messageToSign as xs:string, $pathToKey as xs:string, $encryptionAlgorithm as xs:string, $digestAlgorithm as xs:string ) as xs:string
```

The key file can be binary or in Base64-encoded format.

P12 and Keystore Based Files

To sign with a P12 or KeyStore-based file, use this function:

```
dsig:signWithCertificate( $messageToSign as xs:string, $pathToCertificate as xs:string, $keyRecoveryPassword as xs:string, $encryptionAlgorithm as xs:string, $digestAlgorithm as xs:string ) as xs:string
```

For example, you can use this method for Google JWT (JSON Web Token) Authentication for service accounts. After you obtain a Google private key for access and save it on your Secure Agent, you create a JWT request and sign it with your private key.

For more information on JWT authentication, see: <https://console.developers.google.com> and <https://developers.google.com/identity/protocols/OAuth2ServiceAccount>.

Attachments

Process Designer exposes an endpoint so you can access attachments in a process using an event handler, internal API calls, custom service connectors or built-in connectors like the Amazon S3 Connector. This enables you to pass through an attachment and extract data about the attachment like the file size and file name using functions on the response (where supported).

When you design a process, you can pass attachments into a subprocess, service connector, or Service steps or return attachments (where the output field is defined using the Attachment type).

In a process, you can allow the user to add one or more attachments as inputs where the attachment is coming directly from the screen (not carried across steps using process fields). For example, you could use a process to capture real property data and allow the user to attach photos of the associated property. However, you cannot access the attachment data within a process.

Note: If you run a process on the cloud server, do not use an attachment whose size is more than the default 5,242,880 bytes. The cloud server cannot process attachments that are greater than 5,242,880 bytes.

Encoding

Attachments can be encoded as:

- multipart/form-data (for browser-based interactions)
- multipart/mixed

Data Types

You can specify named input parameters as attachments using two Simple data types:

- **Attachment** supports single attachments and allows you to define the maximum file size as a field property.
- **Attachments** supports multiple attachments and allows you to define the maximum file size and maximum number of files as field properties.

From within the process, you can access metadata about the attachment using an XQuery formula and built-in functions. For example, the following returns the size of the 'photo' input:

```
sff:getAttachmentSize($input.photo)
```

You can assign input attachment fields to outputs or temp fields using an Assignment step. In that case, be sure that the target temp or output field is also of type Attachment.

In this example, the second parameter (\$encodedContent) is a base64-encoded string:

```
sff:createAttachmentFromBase64("base64", "UOpGUIRA", "ascii")
```

Note: The attachment functions are available in processes but not in service connectors. See below for more information on handling attachments in service connectors.

Refer to [“Using Functions” on page 26](#) for more information on the functions available for use with attachments.

Attachment Payload

When using simple attachments, be sure that:

- Each named part has the same name (name and Content-Disposition) as the Input parameter in the process.
- The Content-Type of the part must also match the content type of the Input parameter. For all data types except attachments, Content-Type= text/plain (for example, text, checkbox, date).
- If the part represents a process object, its contents should be valid, serialized JSON content (with a Content-Type of application/json).
- You define process output field (s) of type Attachment (or Attachments). These fields are returned as attachments to the caller.

For example, if the process has input fields "first" (Text), "last" (Text), and "inputFile" (Attachment), the payload would be similar to the following:

```
POST /active-bpel/public/rt/Attachments_Test HTTP/1.1
Content-Type: multipart/form-data; boundary=----TheBoundary1234
-----TheBoundary1234
Content-Disposition: form-data; name="first"
John
```

```

-----TheBoundary1234
Content-Disposition: form-data; name="last"
Smith
-----TheBoundary1234
Content-Disposition: form-data; name="inputFile"; filename="filename.png"
Content-Type: image/png
... binary image data ...
-----TheBoundary1234--

```

The first part of the response should be a JSON document describing the response output fields. The JSON content will have the output field values (standard response).

If the output field is of type Attachment, the value will be a string containing the content-id (cid) of the part containing the attachment. For example, if the output fields are the "first" (Text), "last" (Text), and "inputFile" (Attachment), the response should look similar to:

```

Content-Type: multipart/mixed; boundary=-----TheBoundary1234

-----TheBoundary1234
Content-Type: application/json
{
  "first" : "John",
  "last" : "Smith",
  "inputFile" : "cid:fc1c6030-4b90-4981-b9d8-ab1d0ba0e84e"
}

-----TheBoundary1234
Content-Type: image/png
Content-Name: photo
Attachment-Created-At:1435182142837
Content-Id: fc1c6030-4b90-4981-b9d8-ab1d0ba0e84e
Content-Length: 1073
... binary image data ...

-----TheBoundary1234--

```

- If you do not enable **Output field is whole payload** in the process properties and you have a single output field of type attachment, the response contains both the attachment data and the metadata, as shown here:

```

--Boundary
content-type: ...

cid:outputFile

--Boundary
content-type: image/jpeg

[binary data]

```

If you enable **Output field is whole payload** and you have exactly one output field of type Attachment, you can receive a single attachment (of type text or binary) without the metadata. If a third-party wants to access only the attachment, it is easier to consume. For example:

```

HTTP/1.1 200 OK
Content-Type: image/jpeg
Content-Length: length

[... JPEG data ...]

```

Using Attachments in Service Connectors

You can define a service connector in order to submit an attachment from a process to a service connector's input or obtain an attachment from a service connector and submit it to a process as an attachment. However, you cannot send an attachment to the input of a process from a service connector.

Attachments are sent directly to the service input (including Salesforce-based services) from processes and service connectors. However, the attachments are not accessible from subsequent steps.

Note: You upload or download content from services like Amazon S3 and propagate the attachment back to the main process under the name defined in the output mapping. With the Amazon S3 Connector, you can read or write the contents of an S3 object both as a Base64-encoded string and as an attachment. For more information, refer to the *Amazon S3 Connector Guide*.

Using GET, POST, and PUT

You can define a service connector to use GET, POST, or PUT operations on a resource that does not support JSON or XML.

Note that:

- The input parameters (Attachment type) must be uploaded to the destination from the binding URL (POST/PUT). Use a payload format based on the binding type and number of attachments.
- If there are multiple attachment parameters and the binding is FORM, send all the parameters (including non-file parameters) as multipart/form-data.
- If there are one or more attachment parameters and the binding is JSON, send files as multipart/form-data and combine non-file parameters into a single JSON element.
- If there is exactly one file parameter, the binding is CUSTOM, and the binding payload is empty (no XML or JSON in a custom textarea), POST or PUT the raw file data as is.
- If there are multiple attachment parameters and the binding is CUSTOM and the payload is XQuery, the HTTP outbound payload should be multipart/related. The first part is the CUSTOM (XQuery) content, followed by parts for each of the file/attachment input parameters.
- You can assign input attachment fields to outputs or temp fields using an Assignment step. In that case, be sure that the target temp or output field is also of type Attachment.
- Downloading multiple files (using multipart/mime attachment) is not supported.

Note: The "sff" functions are available only in processes.

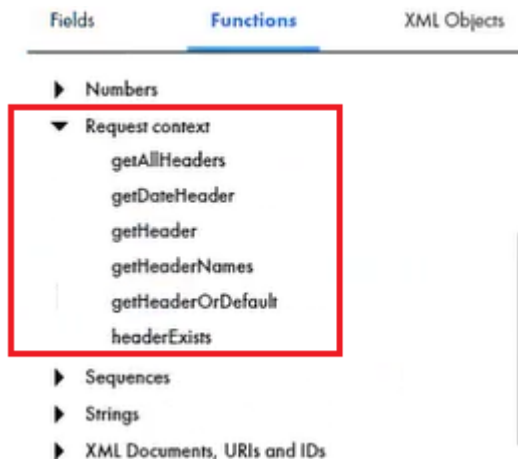
HTTP Headers

Use header functions to get details about headers that you pass when you invoke a process. You can get details about process headers and message event headers.

Use header functions to get information that is only available in the header that you pass when you invoke a process. Examples include dates, JSON Web Tokens, and security IDs.

You can assign the header value to a field and, optionally, perform XQuery operations on the value.

The following image shows the header functions available under the **Request context** section of the Expression Editor:



To understand how each function works, consider a process invoked with the following headers and values:

Header	Value
accept-language	en-US;q=0.9
client-ip	192.0.2.1
accept	text/xml
postman-token	0023-a5dfd-8vdg3-n2b3
CustomHeader	This is a custom header
ExecutionDate	Wed, July 25 2018 06:25:24 GMT

You can use the following header functions on their own, or along with other functions:

Get All Header Values

Use the **getAllHeaders** function to get a list of all headers and their values.

For example, if you use the following expression, you get a specific output:

Expression:

```
fn:getAllHeaders()
```

Output:

```
<headers>
  <header name="accept-language">en-US;q=0.9</header>
  <header name="client-ip">192.0.2.1</header>
  <header name="accept">text/xml</header>
  <header name="postman-token">0023-a5dfd-8vdg3-n2b3</header>
  <header name="CustomHeader">This is a custom header</header>
  <header name="ExecutionDate">Wed, July 25 2018 06:25:24 GMT</header>
</headers>
```

Get Date Headers

Use the `getDateHeader` function to get the value of a Date header. The function returns data in the `dateTime` type.

You can use other Date and Time functions to parse the value.

For example, if you use the following expression, you get a specific output:

Expression:

```
fn:year-from-dateTime(request:getDateHeader("ExecutionDate"))
```

Output:

```
2018
```

Get a Specific Header Value

Use the `getHeader` function to get the value of a specific header.

For example, if you use the following expression, you get a specific output:

Expression:

```
request:getHeader("accept")
```

Output:

```
text/xml
```

Get a list of Headers Names

Use the `getHeaderNames` function to get a sequence of header names without values.

For example, if you use the following expression, you get a specific output:

Expression:

```
request:getHeaderNames()
```

Output:

```
accept-language, client-ip, accept, postman-token, CustomHeader, ExecutionDate
```

Get a Header value or 'Default'

Use the `getHeaderOrDefault` to get a header value if the header exists, or some default value if the header does not exist.

To use this function, you must give the header name and a default value as expression parameters.

Let the default value be `Not Available`.

For example, if you use the following expression, you get a valid output:

Expression:

```
concat ("The header I want is",  
        request:getHeaderOrDefault("postman-token", "Not Available" )
```

Output:

```
The header I want is 0023-a5dfd-8vdg3-n2b3
```

If you use the following expression, you get a `Not Available` output:

Expression:

```
concat ("The header I want is",  
        request:getHeaderOrDefault("special-token", "Not Available" )
```

Output:

```
The header I want is Not Available
```

This is because the request does not contain a header with the name `special-token`.

Check if a Specific Header Exists

Use the `headerExists` function to check if a header with a specific name exists.

You get either `True` or `False` in the output.

For example, if you use the following expression, you get a specific output:

Expression:

```
request:headerExists("CustomHeader")
```

Output:

```
True
```

Headers from Message Events

If a process has message events, you can use any header functions to get headers from the message events. To do this, you must add the case-sensitive message event name as an expression parameter.

For example, if you pass a header with the name `CustomHeader` to the message event receive step, use the following expression to get the value of `CustomHeader` from the message event `ME1`.

```
request:getHeader("CustomHeader", "ME1")
```

CHAPTER 2

Designing Processes

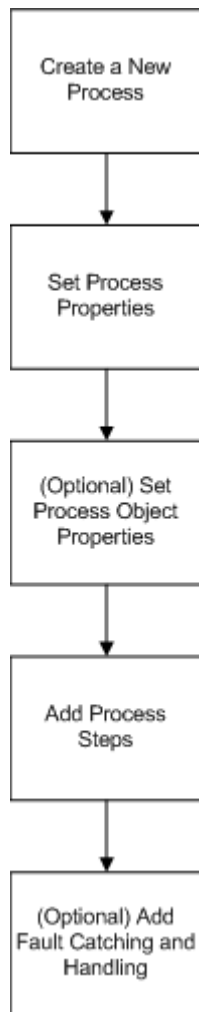
When you use Process Designer to integrate applications, you can:

- Build a process from a variety of step types, including data decision, subprocess, parallel path, receive, and wait.
- Automatically link steps in a process as you select data and define actions, bindings, and other process properties.
- Test your process and necessary connections during process design.
- Publish the process so it is automatically deployed and available to invoke as a REST/XML or JSON service.

For example, you might define a process to streamline Salesforce order processing as follows:

- Invoke services to obtain shipping information.
- Lookup current pricing from an inventory database that resides on-premises.
- Submit order details to SAP.

The following image shows how to design a process:



Creating a Process

1. In Application Integration, select **New**.
2. In the **New Asset** dialog box, select **Processes > Process**, and then click **Create**.

Setting Process Properties

1. In Application Integration, click **New** to open the **New Asset** window.
2. Select **Processes > Process**, and then click **Create**.

The Process Designer opens.

The following image shows the **Process Properties** panel:


Process1 Properties

General

Start

Input Fields

Output Fields

Temp Fields

Messages

Advanced

Notes

Step Type: Start

Name: *

Location: Default Select

Description:

- Enter the required properties.
- Optional. To view or edit process properties, click the **Start** step.
- Click **Save**.

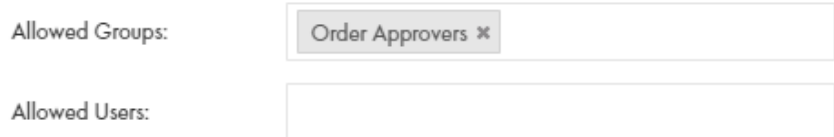
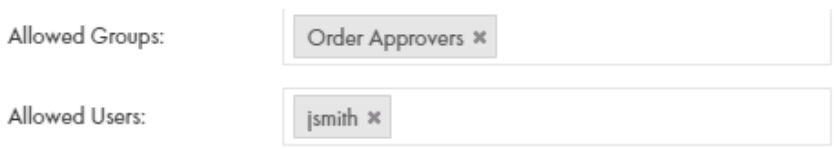
General Properties

You can specify the following General properties for processes:

Property	Description
Name	A descriptive name to identify the process in Process Designer and the name that appears when it is available for use in other objects such as subprocess steps. Note: - Process names cannot begin with a number. If you enter a number as the first character, Process Designer adds an underscore to the name. For example, "123" becomes "_123". - To change the name of a published process, you must first unpublish the process. Then, change the name and republish the process.
Location	The location of the project or folder you want the process to reside in. To select a location for the process, browse to the appropriate project or folder, or use the default location.
Description	A description of the process.

Start Properties

You can define the following Start properties for a process:

Property	Description
Binding	Select HTTP/SOAP or Event, depending on how this process is called. For example, if you are enabling a JMS or AMQP process, choose Event. Default: HTTP/SOAP.
Allowed Groups	The groups that have access to the process service URL at runtime. If the users in the group also have the Run privilege, they can invoke the process service URL. Use the Allowed Groups option when you want a group of users to have access to a service URL. For example, you have a group called 'Order Approvers'. If you enter Order Approvers in the Allowed Groups field, all users within the group have access to the service URL. The following image shows the Order Approvers group added to the Allowed Groups field:  Allowed Groups: Order Approvers x Allowed Users: You can enter more than one group in the field. You can enter data in the Allowed Groups and the Allowed Users fields. If a user falls into either the Allowed Groups or Allowed Users category, they will have access to the service URL. If you enter a role in the Allowed Groups or Allowed Roles field, you cannot select Allow anonymous access and vice-versa. If you do not enter a role in the Allowed Roles field nor do you not check Allow anonymous access, no service URL is available for this process. You can, however, call the process as a subprocess. If you want the process to appear in the API Manager service, you must either check Allow anonymous access or use the Allowed Groups or Allowed Roles field. For more information about managing APIs, see the <i>API Manager</i> documentation.
Allowed Users	The users that have access to the process service URL at runtime. If the users also have the Run privilege, they can invoke the process service URL. Use the Allowed Roles field when you want a specific user to have access to the service URL. The following image shows the a the user jsmith in the Allowed Users field and the group Order Approvers in the Allowed Groups:  Allowed Groups: Order Approvers x Allowed Users: jsmith x In this process, users in the Order Approvers group and the user jsmith have access to the service URL. You can enter more than one user in the field. If you enter a role in the Allowed Groups or Allowed Roles field, you cannot select Allow anonymous access and vice-versa. If you do not enter a role in the Allowed Roles field nor do you not check Allow anonymous access, no service URL is available for this process. You can, however, call the process as a subprocess. If you want the process to appear in the API Manager service, you must either check Allow anonymous access or use the Allowed Groups or Allowed Roles field. For more information about managing APIs, see the <i>API Manager</i> documentation.

Property	Description
Allow anonymous access	When checked, Process Designer lets anyone use this process. This means that access to this process is not tied to a specific role or to any credentials. If you select Allow anonymous access, you cannot enter a role in the Allowed Groups or Allowed Users fields, and vice-versa. If you do not enter a role in the Allowed Groups or Allowed Users field, nor do you not check Allow anonymous access, no service URL is available for this process. You can, however, call the process as a subprocess. If you want the process to appear in the API Manager service, you must either check Allow anonymous access or use the Allowed Groups or Allowed Users fields. For more information about managing APIs, see the <i>API Manager</i> documentation.
Applies To	The type of object associated with the process. The objects that display in this list are the process objects you defined separately, or generated in a defined connection. Drill-down in the list to see process objects and connections, and then select one of these items. Select a process object only if the process will be embedded in another process. Note: To change the Applies To property of a published process, you must first unpublish the process. Then, change the Applies To property and republish the process There are two special values: - Any: Choose Any if you do not want to associate the process with a specific object. When you select this option, the process does not automatically have access to an object's fields. Use this object type when a process creates a new object and does not access information within existing objects. - Home (Salesforce): Select Home if you want a process to be visible from Home or any initial location where users launch all processes, regardless of the object type. Note: If you directly invoke a process, the Applies To setting must be either Any or a specific object in a service, but not a process object.
Run On	Use this list to specify where this process will run. For a process, choose Cloud Server or a specific agent. If your process uses a connector that is event based, make sure that you run the process on the same agent that the connector runs on.
Run As (Salesforce only)	For Salesforce only, you can choose whether to run the process as the Current User or System. Choose Current User if the process should have the same privileges as the user who is running it. Choose System if the process should have system-level permissions to enable actions that the process user may not otherwise be able to access. For example, if a user does not have access to account objects, if the process is running as System, the process can still access the account objects to the extent needed to run the process.

Input Field Properties

The input fields available for a process are those contained in the object(s) to which the process applies. This is specified in the **Applies To** property (on the **Start** tab).

The Input field values are available in all steps of the process. You must define a name and type for the input field category for use in a process:

Property	Description
Input Format	Determines whether the input field can represent one or more fields, or the entire contents of the request. When you select Whole Payload, the input field represents the entire content of the request. For example, if you have a customer process object, its contents might be: <pre>"customer": {"name" : "Joe Smith", "street": "3 Main St", "city": "Shelton", "state": "CT"}}</pre> If you check this option, the following would be posted: <pre>{"name" : "Joe Smith", "street": "3 Main St", "city": "Shelton", "state": "CT"}</pre>
Name	The name of the input field.
Type	The data type.
Description	Description for the input field.
Required	If the field is required for a process to execute, check Required .

For more information on fields, see [“ Introduction to Data Types and Field Properties” on page 9.](#)

XML Payload

In JSON syntax, a root container is not required. If the payload is XML, however, the root element is required. When it sends XML, Process Designer always adds a root node. If the payload is XML, Process Designer always adds a root node when it sends the request.

For example, if the XML is wrapped:

```
<root>
  <oAddressPO>
    <Street>3 Enterprise Drive</Street>
    <State>CT</State>
    <Zipcode>6484.0</Zipcode>
    <City>Shelton</City>
  </oAddressPO>
  <oAddressPO>
    <Street>31 Enterprise Drive</Street>
    <State>CT1</State>
    <Zipcode>106484.0</Zipcode>
    <City>Shelton1</City>
  </oAddressPO>
</root>
```

If the XML is unwrapped:

```
<root>
  <_1>
    <Street>3 Enterprise Drive</Street>
    <State>CT</State>
    <Zipcode>6484.0</Zipcode>
    <City>Shelton</City>
  </_1>
  <_2>
    <Street>31 Enterprise Drive</Street>
    <State>CT1</State>
    <Zipcode>106484.0</Zipcode>
    <City>Shelton1</City>
  </_2>
</root>
```

Output Field Properties

The output fields available for a process are those contained in the object(s) to which the process applies. This is specified in the Applies To property (on the **Start** tab).

The Output field values are set when the process executes and are then available in the steps that follow. Define output fields only for use in an embedded process. These output fields do not appear in lists until the values are returned from an embedded process. You define a name and type for the output field category for use in a process.

Property	Description
Output Format	Determines whether the output field can represent one or more fields of the response, or the entire contents of the response. When you select Whole Payload , the output field represents the entire content of the response.
Name	The name of the output field.
Type	The data type.
Description	Description for the output field.
Initial Value	The initial value for the output field.

For more information on fields, see [“Introduction to Data Types and Field Properties” on page 9](#).

Temporary Field Properties

The temporary fields available for a process are those contained in the object(s) to which the process applies. This is specified in the Applies To property (on the **Start** tab).

The temporary field values will be used in all of the current process's steps. However, a temporary field's value is not available to an embedded process. You define a name and type for the temporary field category for use in a process:

Property	Description
Name	The name of the temp field.
Type	The data type.
Description	Description for the temp field.
Initial Value	The initial value for the temp field.

For more information on fields, see [“Introduction to Data Types and Field Properties” on page 9](#).

Messages Properties

If needed, you can define one or more message events for this process.

For each message, you can define the following properties (similar to Start events):

Property	Description
Input Fields	Specify the field name and type for each input field. If the field is required for the message to be valid, check Required . If this field is used for a correlated message receive event, check Use for Correlation .
Output Fields	Specify the field name and type for each output field.
Binding	Select REST/SOAP or Event, depending on how this message is called.
Allowed Users/Groups	Enter the users and groups who can access this message.
Allow Anonymous Access	When checked, anyone can use this message without any authentication. Note: If you enable this option, Process Designer ignores any limits you set with Allowed Users/Groups.
Input format is Whole payload	The input field can represent the entire contents of the request contained in this message.
Output format is Whole payload	The output field can represent the entire contents of the response. The way in which this changes a payload is the same as for an input field.

Correlated Message Events

When you design a process, you can use message events in connection with the Receive step to interact with a process after the process has started execution. For example, you might want to:

- Query the status of a process
- Update data or provide control to the process

Based on one or more input parameters that uniquely identify the process, the process engine matches the correlated message events with their intended business process instances.

For example, a purchase ordering process might have an OrderId property. Messages can store the value of OrderId in different parts of the order process using different names and you associate a specific process instance with a unique OrderId. When a message arrives with the correlated OrderId, it is dispatched to the correct process instance. You can create as many correlation sets as you need.

Correlation matches incoming messages with their intended business process instances, so business processes can support asynchronous request/response patterns. (By comparison, synchronous calls need not rely on correlation because the message context is maintained on the stack or across a TCP connection.)

You define message events at the process level. A message event can be either interrupting or non-interrupting.

Before You Use Correlation

Take these steps before you define a set of correlated message receive events:

- Identify the Process Designer steps that need to be correlated. They should share one or more pieces of common data.
- Define a property that identifies the piece of common data.
- Define a correlation set of data so you can implement this in Process Designer.

Message Event Design Guidelines

When you define correlated message events, note that:

- For each message event you define in the process properties, at least one input field should be designated as **Use for Correlation**. If this Input field is a Reference type (a process object), also specify a path to a simple-typed field (with a choice of "field" or "formula").
- You can use event inputs and outputs as fields in the process but they are available only downstream of the Receive step (similar to outputs of Service steps).
- If you use multiple message event definitions and two input parameters in different message events have the same name, they should have the same type and same correlation path, if the input parameters are used for correlation. This ensures that you can have a single correlation value for a step.

You also specify whether it is interrupting or non-interrupting.

With an *interrupting* message event:

- The event terminates the step.
- The event path can merge with main path.
- The event path must have a milestone, which serves as the reply to the message event receive step.

With a *non-interrupting* message event:

- The process does not wait for the message event.
- The event path must end with an end step, which serves as the reply to the message event receive step.

Client Interaction to Invoke a Message Event

You can invoke a message event similar to a process, but using a different endpoint. For example, the endpoint to start a process might be:

```
.../active-bpel/public/rt/00000I/OrderProcessor
```

To invoke a specific message event, you would use an endpoint similar to:

```
.../active-bpel/public/rt/00000I/OrderProcessor/event/CheckOrderStatus
```

To learn more about concepts related to correlated message events, also see the *Process Developer Guide*.

Advanced Properties

By default, Process Server terminates a process when an uncaught fault condition occurs. You can use Process Designer to suspend individual processes on uncaught faults.

Note: These process-specific properties work in conjunction with other exception management settings specified in the Application Integration Console.

You can select these Advanced properties to determine the type of logging that occurs on an individual process at runtime.

Property	Description
Suspend on Fault	If you select Suspend on Fault Process Server suspends the process when an uncaught fault occurs. If you do not select Suspend on Fault, Process Server terminates the process if a fault occurs and the process does not explicitly handle the fault. When the process is in the suspended state, review and identify error conditions. Then, you can then retry or complete the activity. For example, you can catch error conditions during employee or customer onboarding. After you resolve all errors, the process moves out of the suspended state and continues from that step. If you enable this option, full persistence is used. To learn how to manage processes at runtime and correct a problem with a faulting process, refer to the <i>Monitor</i> module.
Tracing Level	Select a Tracing Level from this list to determine the corresponding persistence and logging level settings: None, Terse, Normal, Verbose. See the discussion below for more information. The Tracing Level of a process implemented using Process Designer couples the Persistence and Logging levels that are handled separately by Process Developer.

Tracing Level Considerations

The Tracing Level you select determines the persistence and logging level as shown in this table:

Tracing Level	Persistence Level	Logging Level
None	Brief	None
Terse	Brief	Fault
Normal	Brief	Execution with Service Data
Verbose	Final	Execution with Data

Persistence Level

Regardless of the Tracing Level set here, Process Server uses full persistence in the following situations:

- When you select **Suspend on Uncaught Fault**.
- When a process includes a timer (Wait step), a message event, or a receive event.
- When one process invokes another to provide subprocess coordination.

For information about different persistence levels, see the *Setting the Process Version Persistence Type Persistence* topic in the *Monitor* section.

Logging Level

The Process Server logging level may be lower than the logging level you set for an individual process. Regardless of what you set at the process level, the logging will not exceed the Max Process Logging Level specified for Process Server. For example, the Max Process Logging Level might be set to Execution with Service Data. In that case, even if you select the Verbose tracing level, Process Server applies the Execution logging level for this process.

For information about different logging levels, see the *Logging Properties* topic in the *Administer* module.

Notes

Use the **Notes** tab to add information that you or other developers might need. The Notes you enter display only at design time. The guide or process users or consumers do not see the notes. For example, you might add a reminder about something that needs to be done before you deploy a process or guide.

Adding Process Steps

Add steps to access data, services, and perform related orchestration activities.

When you add a step, you also define properties for that step.

Perform one of the following tasks to add a step to a process:

- Drag a step from the palette on the left onto the canvas.
- Double click on the canvas and select a step from the **Step Type** list.

Assignment Step

Use the Assignment step to set a value to a field. To create an Assignment step, click a process, and then select **Assignment** from the Design canvas. Then, from the Assignments properties panel, click **Add Field** to assign Name and Source values.

The following image shows an Assignment step:

 Assignment to temp1, input1 etc. Properties

General	Target	Value
Assignments	temp1	Formula math:sqrt(100)  
	input1	Field temp1 
	output1	Field input1 
	input2	Content 100  
	output2	Formula util:base64Encode[ 

 Add Field  Reorder

To add a field, click **Add Field** and then add the following information for each field:

Name

This is the fully-qualified field name. You can select a field name from the list of fields you defined under .

Source

This is the source from which the field will take values from. The fields you see depend on the data type you defined under .

For example, if you define the data type as Date, you see the following Source options:

- Specific date
- Days from today
- Days before/after
- Field
- Formula

If you define the data type as Number, you see the following Source options:

- Content
- Field
- Formula

For more information about data types, see [“Using the Field Properties Dialog ” on page 17](#).

To change the order of fields in the Assignment step, perform the following steps:

1. Click **Reorder**.
2. In the **Reorder Expression** dialog box, select a field and use the arrow buttons to move the field up or down the order.
3. Click **Update**.

Service Step

When you add a Service step, you set some properties.

The following sections describes the Service step properties:

General

Property	Description
Step Type	The Service step.
Name	A descriptive name for the service.

Service

Property	Description
Service Type	The connection, process, or system service you add to the process. Select from a list of existing tasks. Note: You must have an existing item to add to a process. You cannot create a task in when you create a process.

When you add a service to a Service step, corresponding Input Fields are created.

Input Fields

Input Fields section shows the names of input fields that are most often used when using this kind of step. For Service steps that create objects, the input fields shown are the fields that are most frequently needed when the object is created. (If you do not need a field and it is optional, you can delete it.) You can choose additional (optional) input fields from the list.

Use the delete icon to remove a field. This removes input fields that you do not want to pass to the Service step. Some fields are required and cannot be deleted.

Fault Handling

Fault handling in Process Designer is based on the Business Process Model and Notation (BPMN) 2.0 specification, which defines the concept of a boundary event. The boundary events catch faults associated with specific steps, rather than the overall process scope. This means you can handle faults at the step level, not the process level. (Developers can also use Process Developer to define a fault handler on the process scope.)

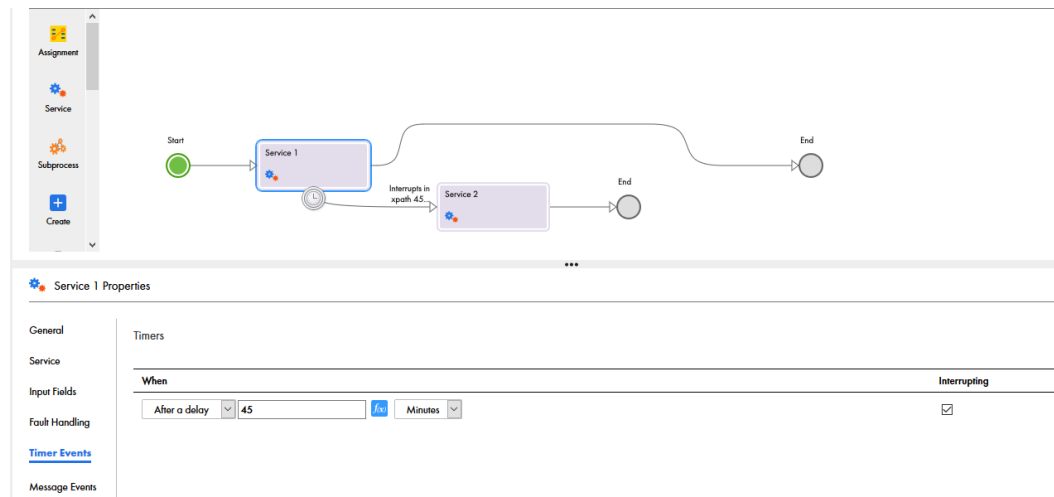
Visibility of Faults in Application Integration Console

When fault handling is enabled in a process, you can view the error marker on the step and some basic fault information in the Application Integration Console Processes list.

Timer Events

Use Timer Events to perform an action based on a schedule. You can specify whether you want the event to run **At specific time** or **After a delay**.

The following image shows a configured timer event:



Select **Interrupting** if you want the timer to interrupt the Service step. When you set an interrupting timer, the Service Step task is interrupted and the process only runs the task on the timer set

Message Events

Specify messages for use with message providers and correlation

Subprocess Step

A Subprocess step embeds one process within another process. When the Subprocess step executes, the embedded process executes. Use of these embedded processes is one of the more powerful and versatile features within Process Designer.

You can define the Input and Output associated fields for the Subprocess step, if any, based on the embedded process.

The following image shows the JMS Dequeue subprocess properties:

JMSDequeue Properties

General

Process:

Description

No description

Iteration Rule

Input Fields

Name	Required	Type	Description
in	☐	Object ID (Process Objects:TestPO)	

Output Fields

Name	Type	Description
out	Object ID (Process Objects:TestPO)	

Fault Handling

Timer Events

Message Events

Create Step

Use the Create step to add new object instances. For example, if you work with an Account object, you can use the Create step to add a new account.

The following image shows a Create step for the Account object:

Design

+ Account Properties

General

Connection:

Object:

Description

Create a step type Account.

Input Fields

Name	Required	Type
Account Name	☒	Text
Account Type	☐	Picklist
Parent Account ID	☐	Object ID (Salesforce:Account)
Billing Street	☐	Text Area
Billing City	☐	Text
Billing State/Province	☐	Text
Billing Zip/Postal Code	☐	Text
Billing Country	☐	Text

Fault Handling

To create new objects:

1. Choose the **Input Fields** tab and select all the fields that you will be adding to the object. Some are required; others are optional.
2. For each field name, set its source to one of the following values: **Content**, **Field**, or **Formula**.

Using Reference Fields

Note that:

- If you do not include a reference field on the Input Fields tab for the object you are creating, Process Designer uses the value of the current object at runtime.
- Reference fields for the *Applies To* object are set if they are not set explicitly in the Input tab.

- When the content of a reference field is set to empty, it is not included in the create statement. (This prevents problems that might occur when explicitly setting a field to empty or null could violate data constraints.)

Wait Step

When you add a Wait step, you set some properties.

The following table describes the Wait step properties:

Property	Description
Name	The name of the Wait step.
Wait	Properties that determine when and for how long the process pauses. You can choose from the following options: - At a specific time - After a wait period

Pause at a specific time

Select this option to pause the process at a particular time. Enter the **Time** you want the process to pause at, and optionally, a **Delay**. The **Delay** value can be an integer or a field that you define.

The following image shows the **At a Specific Time** option:

The screenshot shows the 'Wait 1 Properties' dialog box with the 'Wait' tab selected. The 'Continues:' dropdown is set to 'At a specific time'. The 'Time' field is set to '01:00 AM'. The 'Delay' field is set to '01:00 AM' with a unit dropdown set to 'Days'.

For example, set the process to pause at 1:00 am after three days. 1:00 am is the **Time** and three days is the **Delay**.

After a wait period

Select this option to pause the process after a period. The period begins when the process reaches the Wait step. Enter the **Wait Period** that you want the process to pause for. The **Wait Period** value can be an integer or a field that you define.

For example, set the process to pause for one hour from the time that the process reaches the Wait step.

The following image shows the **After a wait period** option:

The screenshot shows the 'Wait 1 Properties' dialog box with the 'Wait' tab selected. The 'Continues:' dropdown is set to 'After a wait period'. The 'Wait Period' field is set to '1' with a unit dropdown set to 'Hours'.

Receive Step

A Receive step can be defined in processes that make a Service and wait for some event data before the process can continue. For example, if the order process includes a step to request more credit information when the order is above a certain threshold, then a Receive step allows you to handle the blocking receive call.

The screenshot displays a BPMN editor interface. On the left, a vertical toolbar contains icons for Subprocess, Create, Receive, Wait, Milestone, and End. The main canvas shows a process flow starting with a green circle labeled 'Start', followed by a 'Receive 1' step (represented by a circle with an envelope icon), and ending with a grey circle labeled 'End'. Below the canvas, the 'Receive 1 Properties' dialog is open. It has two tabs: 'General' and 'Receive'. The 'Receive' tab is selected. Under the 'Message' section, a dropdown menu shows 'MessageEvent1'. Under the 'Correlation' section, there is a label 'InFieldMsgEvent', a dropdown menu set to 'Literal', and a text input field containing '<Give values>'. A small blue icon with '(x)' is next to the input field.

The output of a Receive step is available to be used later in the process, similar to output fields in Service steps.

For the **Correlation Assignments**, select the fields you are correlating with the incoming message event. You can use these assignments to cancel further execution of a request (interrupting) or to provide status (non-interrupting).

For more information, refer to: [“Correlated Message Events” on page 51](#)

Milestone Step

The Milestone step lets you specify one of two actions when the process ends.

Milestone 1 Properties

General

Milestone

Ending Type: ☒ Milestone: Send Synchronous Response ☐ End of Process

HTTP Status: 200 OK ▼

You choices are:

Milestone: Send Synchronous Response

Sends a synchronous reply to original request. This reply has all of the output fields defined in the **Process Properties** dialog. If a synchronous reply was already sent, no action occurs. This is not an error as it allows background and batch work to occur. You can if you like have additional steps after a milestone. This is used to send a synchronous message, and then continue executing.

Note: If you insert an Milestone step into a Parallel Paths step, the Ending Type must be set to Milestone.

End of Process

No further action occurs when the process ends.

End Step

An End step indicates the end of the process. When execution reaches this step, the process completes.

The following table describes the properties in an End step:

Property	Description
Title	The name of the step. You can edit this value.
Ending Type	The default value is <code>End of Process</code> . You cannot edit this value.
HTTP Status.	The HTTP response status code. The default value is <code>200 OK</code> . You can edit this value.

Throw Step

You can use a Throw step to construct an error payload.

The following table shows the properties in a Throw step:

Property	Description
Title	Required. A descriptive name for the Throw step.
Code	Required. A string value that contains the name of the faultInfo payload.
Detail	Optional. A string value that contains the faultInfo detail.
Reason	Optional. The faultInfo reason detail, which can be type \$any.

Decision Step

A Decision step allows a process to take different paths depending on the value of the field.

The following table describes the properties in a Decision step:

Property	Description
Title	The name of the Decision step.
Decision	The process takes a decision based on the fields and paths you define here. Select a field name from the list of fields you define under Start > Fields. Enter conditions and values that you want the Decision step to base a decision on. The conditions available depend on the field that you select. For example, if you select a field of type Simple > Text, the following conditions are available: - Equals - Starts With - Ends With - Starts with any of - Contains You can enter text values against the conditions you select. You can add multiple conditions to a Decision step. Each condition is a potential data path. For each path that you add, a corresponding branch appears on the UI. Drag branches to rearrange the order in which the branches appear on the UI. Most Decision steps have an Otherwise path. This path handles execution if no data meets the conditions in your tests.

Evaluating Paths

A process evaluates conditions based on the criteria you specify. Ensure that you construct paths with non-intersecting conditions.

For example, you create a Data Decision step with the following paths:

- Path 1: Field less than or equal to 100.
- Path 2: Field less than or equal to 75.
- Path 3: Field less than or equal to 25.
- Path 4: Otherwise

If the integer field for which the Data Decision step was created has a value of 25, the Data Decision step takes path 1. This is because 25 is less than 100 and path 1 is the first option.

To ensure that the Data Decision step follows the "Field less than or equal to 25" path, re-create the paths with the following criteria:

- Path 1: Integer between 0 and 25
- Path 2: Integer between 26 and 75.
- Path 3: Integer between 76 and 100.
- Path 4: Otherwise

Important: The process evaluates conditions in a top-down manner. Ensure that the Otherwise branch is the last path.

A Decision step can lead to another Decision step. For example, a branch could run if an annual income exceeds \$100,000. The next decision test along the same path could test if the city is Boston, or otherwise. Using this technique, you use Boolean AND logic because you base the test for the second condition on the

true branch of the first condition. In this example, you use the Decision step to set the condition "Annual Revenue exceeds \$100,000 AND city is Boston".

Similarly, to support Boolean OR logic, you can add a test for the second condition on any branch.

Parallel Path Step

When you add a Parallel Paths step, you set some properties.

The following table describes the properties in a Parallel Paths step:

Property	Description
Name	The name of the Parallel Paths step.
Parallel Paths	The paths that you want the process to run in parallel. Click Add to add a new branch. You can add multiple steps to each branch. To add steps to a branch, drag and drop a step from the palette on the left. Keep the following restrictions in mind when you use the Parallel Paths step in conjunction with the Jump step, : - You cannot jump to any branch of a Parallel Path step from outside of the Parallel Paths step. - You cannot jump to outside a Parallel Path step from within the Parallel Path step. You can jump from one branch of a Parallel Path step to another branch.

Jump Step

When you add a Jump step, you set some properties.

The following table describes the properties in a Jump step:

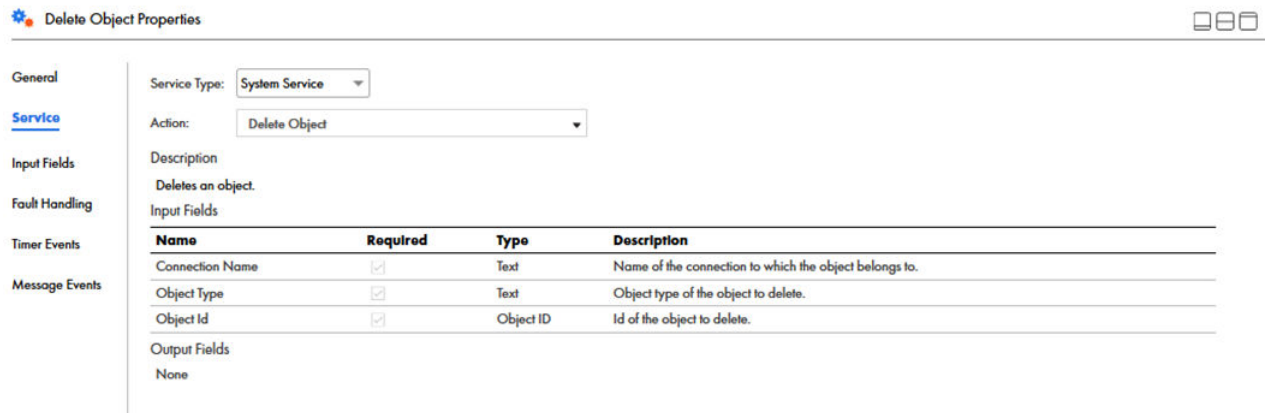
Property	Description
To	The target of the jump. Select from a list of available steps. More than one step can jump to the same target step. To see how many Jump steps have a particular step as their target, place the cursor over the arrow next to the target step. The Jump step has the following restrictions when used in conjunction with the Parallel Paths step: - You cannot jump to any branch of a Parallel Path step from outside of the Parallel Paths step. - You cannot jump to outside a Parallel Path step from within the Parallel Path step. You can jump from one branch of a Parallel Path step to another branch.

System Service Actions

When you configure a Service task in a process and select system service as the service type, you can choose from a list of supported actions.

Delete Object

This Delete Object action lets you delete a record. Depending upon your application, what is being deleted can be called an object or an object instance.



Delete Object Properties

General

Service Type: System Service

Action: Delete Object

Description: Deletes an object.

Input Fields

Name	Required	Type	Description
Connection Name	☒	Text	Name of the connection to which the object belongs to.
Object Type	☒	Text	Object type of the object to delete.
Object Id	☒	Object ID	Id of the object to delete.

Output Fields: None

As the figure shows, you will need to enter information for three fields.

JMS Enqueue Service

Use the JMS Enqueue Service to send a message to a JMS service, where it will be queued. When the receiving process is ready, the service sends the message to it. The message is sent using the Informatica Secure Agent to communicate with an on-premise service.

The data in the message is:

Messaging Manager Name

The name of the messaging manager that contains the destination.

Destination Name

The name of the message's destination.

Format

The format to be serialized to the destination.

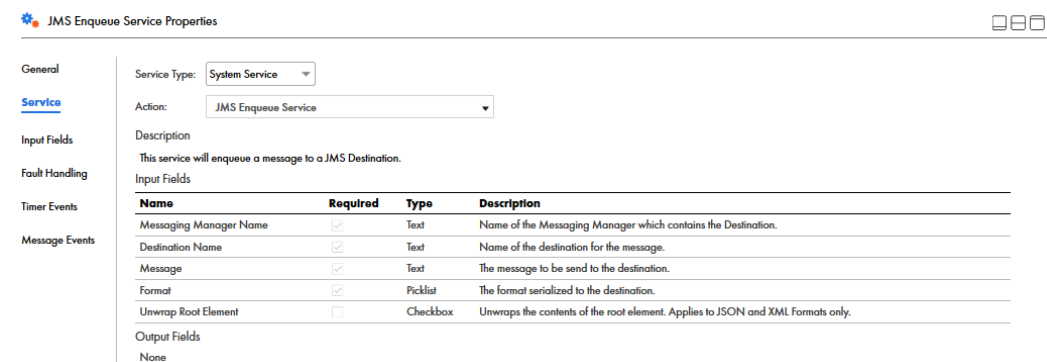
Message

The message to be sent to the destination.

Unwrap Root Element

This describes how the process object is serialized when it is sent to the destination.

The following image shows the JMS Enqueue Service properties:



JMS Enqueue Service Properties

General

Service Type: System Service

Action: JMS Enqueue Service

Description: This service will enqueue a message to a JMS Destination.

Input Fields

Name	Required	Type	Description
Messaging Manager Name	☒	Text	Name of the Messaging Manager which contains the Destination.
Destination Name	☒	Text	Name of the destination for the message.
Message	☒	Text	The message to be send to the destination.
Format	☒	Picklist	The format serialized to the destination.
Unwrap Root Element	☐	Checkbox	Unwraps the contents of the root element. Applies to JSON and XML Formats only.

Output Fields: None

Run a Shell Command

You can use the Run a Shell Command action in a Service step to automate the call to a shell command as part of your application integration. For example, you might add this action in a subprocess to move files on the agent or to read the file contents of a particular directory by running `cp` or `cat` shell commands.

Note: When you use this action in a process, be sure to select a Secure Agent where you want to run the shell command in the **Server Configuration > System Services** list in Application Integration Console. However, you can access a shell command from a process running on the Cloud Server by calling the shell command in a subprocess that contains this Service step. Running system commands with this step is not recommended.

Enabling the Shell Command Service

Before you can execute a process that includes this action, enable the shell service in the Application Integration Console with the following steps:

1. In Application Integration Console, choose **Server Configuration**.
2. Choose the agent where you want to run the shell service from the **Server Configuration** list.
3. On the **System Services** tab, choose **Shell Service**.
4. Click **Save**.

Input and Output Fields

When you include a Service step for Run a Shell Command, you can use the following options:

	Name	Type	Description
Input Fields	Command	Text	Required. The command name to be executed, which is specific to the operating system (Windows or Linux).
	Character Set	Text	The character set of the command to be run. Default, if not provided: UTF-8.
	Merge Output	Checkbox	When checked, the contents of the output are merged with the error string, if any. By default, this is enabled.
	Input	Text	File name with path to use for the input, if any.
	Working Directory	Text	Required. Path of the directory where the command executes

	Name	Type	Description
Output Fields	Error String	Text	String of error returned, if any.
	Response Code	Text	0 (success) or 1 (failure).
	Output String	Text	String of output returned, if any

The following image shows the properties required to run a Shell command:

Run Cloud Task

You can use a Service step to interact with Data Integration applications.

First, ensure that you have set up the Salesforce integration. Then, synchronize the Data Integration service with Salesforce.

Perform the following steps to create a Run Cloud Task Service step:

1. Insert a Service step at the place in the process where you want to interact with Application Integration.
2. In the Service properties, select the action **Run Cloud Task**.
3. Configure the required properties:
 - **Task Name:** The name of the cloud task that you want Process Designer to start.
 - **Wait for Task to Complete:** If you select this, Process Designer waits for the task to complete before resuming execution.
 - **Max Wait:** The maximum number of seconds that Process Designer waits for a task to complete. The default value is 300 seconds, or 5 minutes.
Important: You must enter a Max Wait value of less than 604800 seconds, that is, seven days or fewer. Process Designer waits for a maximum of seven days for a task to complete.
 - **Fail on Cloud Task Errors:** Faults when the synchronization or mapping fails.

The following image shows the four fields that you can configure for a Run Cloud Task Service step:

 Run Cloud Task Properties

General
Service
Input Fields
Fault Handling
Timer Events
Message Events

Service Type: System Service

Action: Run Cloud Task

Description

Start the specified Informatica Cloud Task.

Input Fields

Name	Required	Type	Description
Task Name	☒	Cloud Task	The name of the task to start.
Wait for Task to Complete	☐	Checkbox	Wait for the task to complete before returning.
Max Wait	☐	Integer	The maximum time in seconds to wait for a task to complete.
Fail on Cloud Task Errors	☐	Checkbox	Throw fault if ICS task completed with errors.

Output Fields

Name	Type
Run Id	Integer
Task Status	Text
Success Source Rows	Integer
Failed Source Rows	Integer
Success Target Rows	Integer
Failed Target Rows	Integer
Start Time	Date Time
End Time	Date Time
Error Message	Text

- Log in to Data Integration in a separate browser tab.
- Copy the Task Name of the task to run from the **Explore** tab.
- Go back to the Run Cloud Task Service page, and then paste the Task Name in the **Task Name** field on the **Input Field** tab.

After you publish and run the process that contains this step, you can see the task execution history in Application Integration Console.

Fault Handling

When you integrate applications with Process Designer, you can handle error conditions by:

- Catching a fault in a process using boundary events.
- Returning faults from a process either with output fields or by adding a Throw step to the process to throw a fault.

Fault Handling and Boundary Events

A boundary event is an event that catches an error that occurs on the boundary of a particular step, that is, within the scope of the step where it is defined. When you enable fault handling for a step in your process, you are defining a boundary event. The fault handler listens for a fault on that step when the process executes. When the fault is caught, the step is interrupted and the process then follows the path you define for handling the fault. In the fault path, for example, you might send an email notification and terminate the process, retry after waiting for a specified interval, suspend the process, or complete the process using an alternative path.

For example, you can catch a fault that occurs when a REST service returns an error code to the process because the host system is not accessible or because the process provided invalid data.

Note: A fault handler on a step is an interrupting boundary event. This means that when a fault occurs, the process follows the fault path and the main path from the step does not execute.

Design Guidelines for Fault Handling

When you define fault handlers, you must note the following points:

- Determine whether the boundary event occurs at a single step or in a set of steps. If the boundary event occurs on a set of steps you have two options:
 1. Add fault handling for each step individually.
 2. Define the steps inside a wrapping process.
The latter option allows you to then add the fault handler on a Subprocess step and invoke the wrapped process.
- If you have a Parallel Path step for fault handling, be sure to provide unique field names for each field that contains fault information. Because you define the fault information output field at the process level, there is a chance that fault information from one path can overwrite the fault information of another path. To avoid this, be sure to set a unique field name for fault information in each path of a Parallel Path step.
- To handle faults, you have several options:
 1. If there is an uncaught fault, suspend the process and debug it using the Application Integration Console. In this case, you must select Suspend on Uncaught Fault in the Advanced process properties. If you do not catch the fault, it will be suspended and visible in the Application Integration Console. You can also rethrow the fault after you catch it.
 2. Log the fault (for example, by sending an email notification) and continue gracefully.
 3. Add a Wait step to wait for some interval and then retry the step by adding a Jump step. However, you cannot use a Jump step from the fault path. This means you must merge back to the main process path and then add the Jump step.
- Because you get the fault detail (in the faultInfo process object) as an any type value, you can use XQuery to access the detail, provided you know the structure of the fault information returned from the service.

Faults can be triggered by:

- Service step (using a service connector or an automated step created in the process).
- Create step (a special kind of Service step available in Process Designer).
- Host system interaction.
- Throw step defined in a process.
- Other faults.

Faults Triggered by Service Connectors

If a fault is triggered by a service connector, the faultInfo process object includes a description of the error and returns this to the process.

When you enable fault handling in Process Designer, you also catch any explicit runtimeError faults that might be generated by an SOA connector. For example:

```
<wsdl:operation name="read">
  <wsdl:input message="tns:readRequest"></wsdl:input>
  <wsdl:output message="tns:readResponse"></wsdl:output>
  <wsdl:fault name="runtimeError" message="tns:faultResponse"/>
</wsdl:operation>
```

Step Types that Support Fault Handling

You can enable fault handling on the following step types:

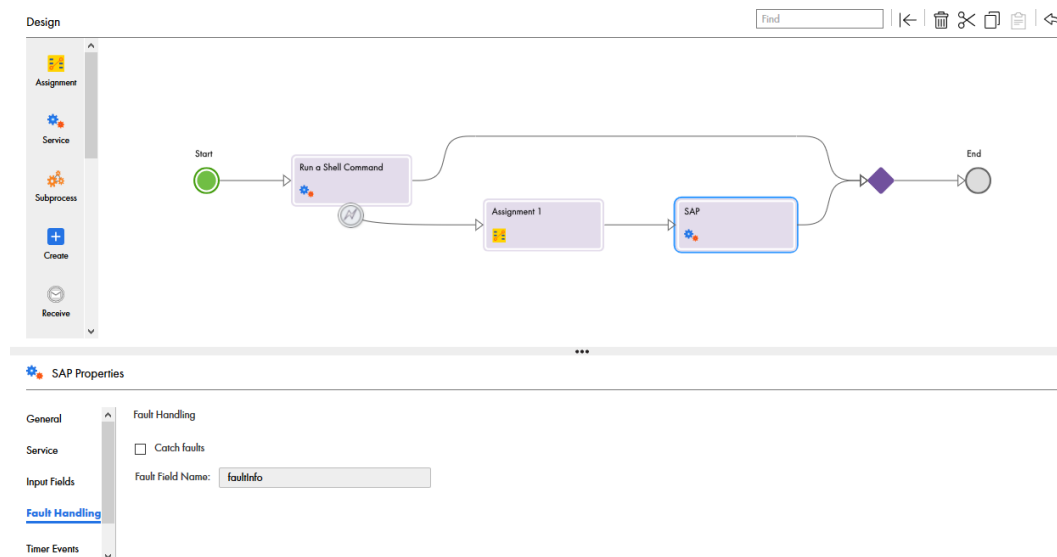
- Create
- Service
- Subprocess

If enabled, the fault handler catches all faults on the individual step. Because it is interrupting, it creates a separate execution path in the process. You specify a fault name field so you can get all the fault details in the process object.

On the **Fault Handling** tab for each of these Step types, you can set these properties:

Property	Description
Catch Faults	Required. Check to enable fault handling on this step. By default, this option is disabled.
Fault Field Name	Required if Catch Faults is enabled. The fault name, which defaults to faultInfo.

The following image shows a service step with fault handling enabled:



You can also configure a specific fault type, reason, and description in a Throw step. You can then call the Throw step from other processes when it is caught by a pattern or process, rather than using an output field to pass fault information.

Methods to Return a Fault from a Process

1. **Use an output.** This is suitable when you foresee that the error might occur and you want to use a data decision step to branch the process based on the normal process flow.
2. **Throw a fault.** This is suitable when the only appropriate response to the error is to pass the error up to the caller or log the error and suspend the process. For example, if you attempt to access a web service that responds with the HTTP status code, 403 Forbidden, it indicates that the service understood the request but refuses to take any action, because access is denied. Throwing a fault is normally the best option to handle this error because you are unlikely to resolve it in the normal process flow.

Throw Step Usage

When you add a Throw step to a process, the error information is propagated by the Throw step. This allows you to catch the fault when you call it, for example, through a Subprocess step.

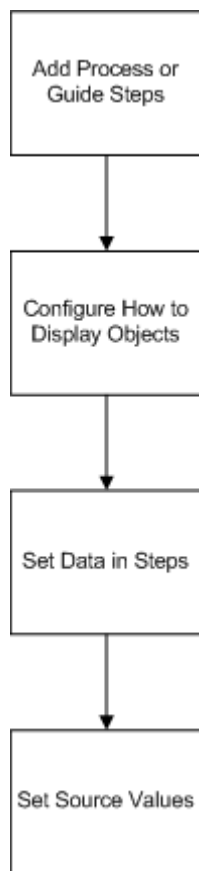
Note that:

1. The name must be a string value.
2. Throw is a terminating step so it does not support any outbound links.
3. A Throw step can be used to rethrow information caught using a fault handler. In that case, the payload for the step is a `faultInfo` element that contains the parameters returned by the service, service connector, or SOA connector.
4. You can also use a Throw step to construct specific fault information that you want to use as part of your process.

CHAPTER 3

Using and Displaying Data

The following diagram explains what you learn in this chapter:



Selecting and Displaying Objects

Within a step, you may want your users to select one or more object. You can display this information in a variety of ways:

- A pop-up search dialog for searching for objects by name
- A picklist from which they can select one object
- A multi-select picklist for selecting more than one

- A table of objects, and you can select which of the object's fields it should display in separate column. If only one object can be selected, Guide Designer adds a column containing radio buttons; otherwise, it adds checkboxes.

Note: If you are just displaying one object, you can display the object's fields as a one-row, multi-column table.

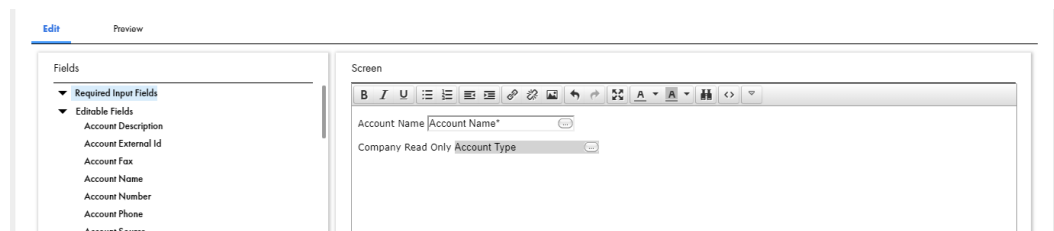
By default, Guide Designer displays a search dialog box when users are searching for information for fields that can be updateable. You can override the default and tell Guide Designer to display object information using either a picklist or table. Picklists and tables are populated using a query that filters the possible objects down to just those that should be presented. For example, Guide Designer could show only the contacts that are associated with an account.

You can customize the fields being shown for each object. If only one field from these objects is displayed, Guide Designer shows this information as a list. However, if you are displaying more than one (for example, a lead's first and last name), Guide Designer shows each within its own column in a table.

Note: Object queries are not available when the field you are updating can reference more than one type of object (that is, the **Reference To** control within the **Field Properties** dialog includes multiple object types). If this is your situation (that is, you cannot use a regular query), you must select and use an advanced query.

Inserting Fields Using Picklists

There are many places where you add or use field names.



When a user runs a guide, Process Designer replaces the field with the value within the object. For example, if an Account Type field was inserted and the Account Type is "Grade 1", users will see "Grade 1" instead of the field name when they are running the guide.

Three kinds of lists exist in steps:

- **Read-Only Field:** A read-only field is a field whose value cannot be changed. You can tell that it is read-only field because it displays in gray.

The text in front of the field was added as descriptive text in front of the field. If you didn't add a description, the user might not know what information was being displayed.

Note: When an object ID is placed in a canvas as a read-only field or put into a column in a table, it displays as a link to the object. If you want to show the object ID's value, insert it into a text field as "{! Id}".

- **Editable Field:** An editable field lets the user see the current value of the field as well as modify its value.
-

Click the "..." button to see a field properties dialog. For more information, see ["Introduction to Data Types and Field Properties" on page 9](#).

Autogenerated Fields

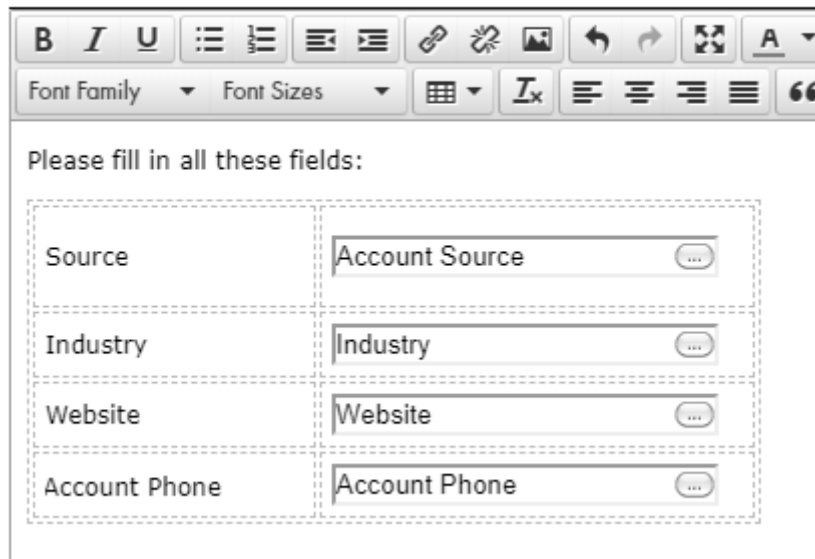
A field can be virtual; that is, the object has a field that is actually a composite key or a composite field. When Guide Designer displays the name of this virtual and autogenerated field, it displays it as "Object_name" + "_IID".

Inserting Fields in Tables

When you insert a field into a step, clicking on the field's name in a picklist inserts a label and a text field into which the field information will display. If you are inserting more than one field, the step can be hard to read and does not look good. Placing the elements within a table can solve both of these problems.

Here's an example:

Screen



The screenshot shows a software interface with a rich text editor toolbar at the top. Below the toolbar, the text "Please fill in all these fields:" is displayed. Underneath this text is a table with four rows. Each row contains a label on the left and a text input field on the right. The labels are "Source", "Industry", "Website", and "Account Phone". The corresponding text input fields contain the values "Account Source", "Industry", "Website", and "Account Phone". Each text input field has a small icon to its right, likely for clearing the field or showing a picklist.

Source	Account Source
Industry	Industry
Website	Website
Account Phone	Account Phone

When handling fields in a table:

- If the cursor is not in the right-most cell, Guide Designer inserts the label at the current cursor position. The field is inserted into the cell to the right of the cursor. If the cell to the right has something in it, Guide Designer appends the field to this information.
- If the cursor is in the right-most cell and the cell isn't empty, Guide Designer places the label and the field into the first two columns of the row that follows. If that row has text in it, Guide Designer inserts a row immediately beneath the current row, and then inserts the label and field. If that row is empty, just write place the information into the cell.
- If the cursor is in an empty right-most cell or if the table has only one column, Guide Designer places the field (not the label) into this cell.

Entering Fields as Text

You can enter text and the value of a field as the information you type in text boxes such as the one in the following.

To specify a field value, choose Content as the Source and then click the small icon to the right of the text box. From the list of fields that displays, select the field you want to enter. If you know the name of the field, you can also type it in the text box as `{ $name_of_field }`.

This example shows the values of four fields from the current object as the Content for the Description field.

Event Properties

General

Create

Input Fields

Input Fields (2)

Edit Screen

Preview Screen

Add Field

Name	Required	Value
Start Date Time	☐	Specific date 12:00 PM
Description	☐	Content { \$Account.Name } { \$Who.Lead.Id } { \$What.Campaign.Id }

You can add other text to display with the field values. For example, you could specify:

The name is: { \$name }

Displaying Columns

Use the **Display Fields** field to tell Guide Designer which of the object's fields it should display in a table. If the field can be updated, each row also has either a radio button if users can only select one object or a checkbox if they can select more than one.

Use the **Add Field Column** button to add columns. You can use the arrow buttons to the right of the name to change the order. If you do not want to use the default column heading, enter the text you prefer.

Here's a four column table:

Field Properties

Field: Account_ID (input)

Reference To: Account

Required: ☐

Default Value: Content --None Selected--

Field	Label				Properties
Account Name	Account Name	☐	☐	☐	...
Billing City	Billing City	☐	☐	☐	...
Billing State/Province	Billing State/Province	☐	☐	☐	...
Account Phone	Account Phone	☐	☐	☐	...

Add Field

☒ Paginate Results ☐ Allow Filtering ☒ Allow Column Sorting

Hover text:

Height (rows):

Show List: None

OK Cancel

The three check boxes are as follows:

- *Paginate Result*: When selected, the output is paginated.
- *Allow Filtering*: When selected, the user can enter text in a text box to select some of the displayed results.

- *Allow User Column Sorting*: When selected, the user can click on a column heading to sort the results by that column's contents.

Note: When an ID field is included in a result from a Lookup dialog, the ID's value displays.

Setting Data in Steps

Some steps that you will create ask users to enter information before the step's actions occur. Examples include Service and Create. In contrast, other steps already have the information they need when they begin executing. If the step has the information it needs, the action occurs automatically and the user doesn't see it. Instead, the user sees the step that follows. Although this topic uses a Create step as an example, there is no difference setting data in other kinds of steps

In the step properties section, you can see information about the fields associated with the action or object. If an entity or object is not selected, no data appears in the Information tab.

Opportunity Properties

General

Connection: Salesforce

Object: Opportunity

Description

Create a step type Opportunity.

Input Fields

Name	Required	Type	Description
Account ID	☐	Object ID (Salesforce:Account)	
Private	☐	Checkbox	
Name	☒	Text	
Description	☐	Text Area	
Stage	☒	Picklist	
Amount	☐	Currency	
Probability (%)	☐	Percent	
Quantity	☐	Number	
Close Date	☒	Date	
Opportunity Type	☐	Picklist	
Next Step	☐	Text	
Lead Source	☐	Sublist	

This section lists all of the fields that can supply data when this step creates an opportunity object. The fields that you will actually be using are named and set within the Input tab.

Using Fields in Steps

The data types that you name in the property sheet may be numbers, text, time, and so on. These data types determine how the data is handled in a process and displayed in a guide.

The following image shows the Account Type field:

Field Properties

Field:	Account Type
Required:	☒
Default Value:	Content ▼ Partner ▼
Hover text:	

Show List: None ▼

The Default Value and Show List fields appear only when you define the field in a step:

- **Default Value:** Select Content, Field, or Formula.
 - Content displays a control where you can enter a specific value.
 - Field displays a list from which you can select the default value.
 - Formula allows you to set the value using a formula you enter here.

Note: If you select a default value for a field that is on a screen "for update", the user does not see the default value unless the field's current value was not set before the step displays. If it was set, the user sees this current value. Service Call steps differ as the default value is always shown since there is no existing value.

- **Show List:** For information, see ["Selecting and Displaying Objects" on page 70](#).

Setting Source Values

You can tell Guide Designer about the source of the values that will be assigned to a field. The following figure shows part of a Create step where fields have different sources:

Input Fields (2)

Name	Required	Value
Account Name	☒	Content
Parent Account ID	☐	Field Account ID

- **Add:** If the field being entered is of type `ObjectList`, selecting this item will let you append an item to that list.
- **Content:** The values used by the step is what you define here and does not change. For example, the Rating field, which was defined as having one of three values, will be set to one of these three as you are designing this guide. The value you set here is what will be used when this object is created. However, if this were text, the text may also have embedded fields. This example has three different kinds of content fields. The differ because Process Designer knows what kind of data is being set. For example, Converted was defined as a checkbox, Rating as a picklist with predefined values, and Zip/Postal Code is defined as text.
- **Field:** The value comes from a field in an object.
- **Formula:** You will use numeric operators such as:
 - +
 - -
 - *
 - div
 - mod

You can use built-in and XQuery functions to set a value.
- **Query:** (This is not shown.) You tell Guide Designer which information it should locate. For example, Guide Designer might look for a company whose name is "Acme".
- **Screen:** (Guides only) You must insert an input field within the Input Screen tab's editing area.

Setting Source Values: Content

A Content source is one in which the person designing the guide is deciding the value that is passed as input to a step. For example, you could use Content to set the Description field of a Create step creating an event to "Demonstrate the product" rather than requiring the guide's user to provide the description.

Sometimes, you want to create content that contains text and field values. For example, you could have a "mailing" field that contains a person's name and address. You would add this information by including field strings.

Click the (x) icon to see a list of fields.

After you select a field, Process Designer places a field code into the text box at the cursor's position. You can individually select as many fields as you need; however, you can only select one item at a time.

Here's an example:

Event Properties

General

Create

Input Fields

Input Fields (2)

Edit Screen

Preview Screen

Add Field

Name	Required	Value
Start Date Time	☐	Specific date 12:00 PM
Description	☐	Content

Field selection dropdown:

- {\$Account.Name}
- {\$Who.Lead.Id}
- {\$What.Campaign.Id}

You can also add text to this inserted field indicator. For example, in front of `{ $Account.name }`, you could type "Name: ".

When Process Designer executes this step, it replaces each field indicator with the text extracted from the field.

Setting Source Values: Field

If you select Field for the Source, Process Designer displays a collapsed list to the right. You can now select the field whose value Process Designer will write into the object when this step executes. As with other places where you select fields, the bottom of the picklist may show related objects whose fields can be used.

General

Create

Input Fields

Input Fields (3)

Edit Screen

Preview Screen

Add Field

Name	Required	Value
Account Name	☒	Field
Parent Account ID	☐	Field
Annual Revenue	☐	Field

Field selection dropdown:

- Click To Select Field
- Click To Select Field
- Parent Account > Annual Revenue

Setting Source Values: Formula

Fields that take numeric input fields can use a formula to set a value. For example, you can use this with the Assignment step to add a number to an existing field. Use numeric operators such as +, -, *, div, and mod to perform an operation upon fields.

You can also use any XQuery function within a formula. (These are defined at <http://www.w3.org/TR/xpath-functions-30/>.)

Evaluation Notes

Process Designer evaluates formulas as you might expect. Note that:

- `$Field1.Field2.Field3` is pre-processed to de-reference down to the contents of `Field3` and would bind the contents to its value.
- Special characters in field names are removed. Fields that start with numbers are given an "x" prefix.
- If `$Account` is a by-value object, you must use an XPath path expression to get at its contents; for example, `$Account/Owner/Name`.
- Object lists should be sequences containing either ID values (for by-reference object lists) or the XML representation of the records. For example, two object lists can be appended using the following expression: `($a,$b)`.
- Object references within a list are separated by semi-colons; for example, `(a; b; c)`.

For information on available functions, see ["Using Functions" on page 26](#).

Setting Source Values: Query

Use a query to retrieve object information from all of the objects that are selected by the Where Clause.

Query controls appear in different steps and in some dialogs so what you may see may differ slightly.

What you will be doing is creating an SQL WHERE clause. While you can enter your condition directly into the text area, it is usually far easier to click the **Add Condition** button and the Order By picklist. After Process Designer inserts this information, you can edit it if you need to. Letting Process Designer do the work is recommended as it is sometimes not obvious what the field's internal name is. The text entered here is a standard SQL WHERE clause. Also, this text does not include the WHERE keyword.

Each condition has four parts:

- The name of a field in the object. In this example, the field is one of those contained within the object.
- An operator that Process Designer uses when it compares the field value on the left with values on the right.
- The kind of data that will be compared. Your choices are Content, Field, or Formula. If you choose Field, you are comparing the contents within the first field in the current object (the one on the extreme left of the **Condition** dialog) with the contents of the field in the type of object being queried.
- The data to which the field on the left is being compared. If the source is Content and you click within this area, a small icon appears to the right. After selecting it, Process Designer displays a picklist from which you can select the name of a field in another object that will be used when making the comparison.

Pressing the *Order By* picklist button lets you select the fields that the query sorts upon when it displays information. For example, you might want to sort the displayed information by a person's last name. You can add additional keys by reselecting a field from this picklist. The *Order By* clause is placed within the text box. This lets you edit it after it is inserted. You can also directly type the Order By clause here as well.

Note: You can add an SQL LIMIT clause to your query by entering it in the Where clause text box. For example, you can limit the number of retrieved rows to 200 by typing "Limit 200". When adding this clause, be sure to add a space after the where information or place it on its own line after the where. ("Limit" can be in upper, lower, or mixed case.) By default, 100 rows are returned. If your guide requires more than 100, you must add this clause.

Specifying Conditions in a WHERE Clause

Your ability to define a condition within a WHERE clause depends upon what Produce Designer is interacting with.

Direct Interaction with a Relational Database

Most databases fully support the SQL WHERE operator. If you encounter problems, you should consult the documentation for that database.

Using an Informatica Connector

If you need more than one condition, only simple cognations (field operator values) are supported. You may join multiple conditions using either AND or OR. You can also use the NOT operator to invert the meaning of the condition.

Salesforce

The WHERE condition must conform to SOQL requirements. Consult the Salesforce SOQL documentation for more information.

Using a WHERE Clause When No Answer Is Expected

One common situation, and a situation that isn't at all unique to Process Designer, is that you are creating a WHERE clause that is based on a multiple criteria. However, not all of the criteria will have values. For example, an account and a territory, and the territory is optional. Setting up a WHERE clause such as `Account={$account} AND Territory={$territory}` wouldn't work if no territory value was used.

The solution for creating a WHERE clause with fields that are optional is to use the LIKE operator; for example, `Account LIKE '%{$account}%' AND Territory LIKE '%{$territory}%'`. If territory doesn't exist, this would be the same and entering `Account LIKE '%{$account}%' AND Territory LIKE '%'`. The LIKE following the AND will always evaluate to true.

Setting Source Values: Screen

If you select Screen for the Source, the end user provides data for this field. This means that you must add a field to the step's Input Screen tab so that the guide's user can type information. In the following example, the source on the Input tab for three fields was set to Screen:

The screenshot shows the 'Opportunity' form editor in 'Edit' mode. On the left, the 'Fields' panel lists 'Required Input Fields' (Close Date, Name, Stage) and 'Read-only Fields'. The 'Screen' panel on the right shows a text area with three fields: 'Name', 'Stage', and 'Close Date', each with a 'Screen' source value. The 'Field Detail' panel at the bottom left shows a table with 'Field' and 'Close Date'. The 'Footer' panel at the bottom right shows a 'Continue Button' with a 'Continue' label. The 'OK' and 'Cancel' buttons are at the bottom right.

Note: The text you enter will look better if placed in tables. For more information, see [“Inserting Fields in Tables” on page 72](#).

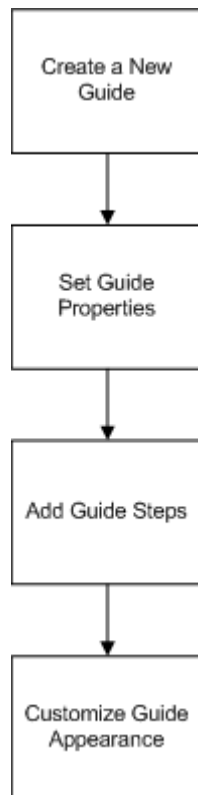
Like all fields, you can click within it to display a properties dialog. See [“Introduction to Data Types and Field Properties” on page 9](#) for more information.

CHAPTER 4

Designing Guides

A guide is a set of screens that prompts users to review, enter, or confirm data. For example, a step might display account details or prompt the user to confirm the status of a sales call. Behind the scenes, steps interact with your application by extracting and storing data. Guides run within mobile apps or on traditional platforms such as a PC or a Mac.

The following image shows the guide design process:

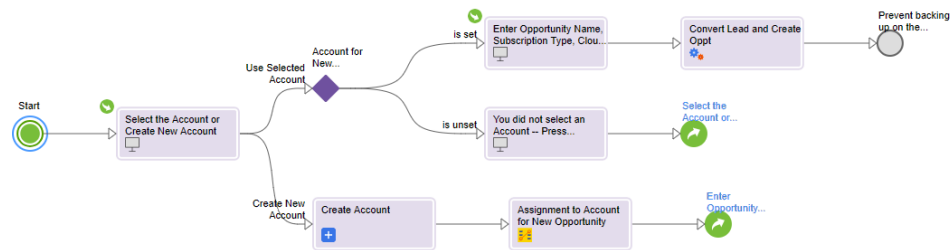


You can create guides without technical expertise or formal training.

You want to automate the process a sales manager follows to create a new account or update an existing account in Salesforce.

To create the guide, you perform the following general steps:

1. Create the screens that contain questions and possible answers. As the options are defined, the sequence of steps from the script appear as a flowchart in the canvas. The following image depicts the guide:

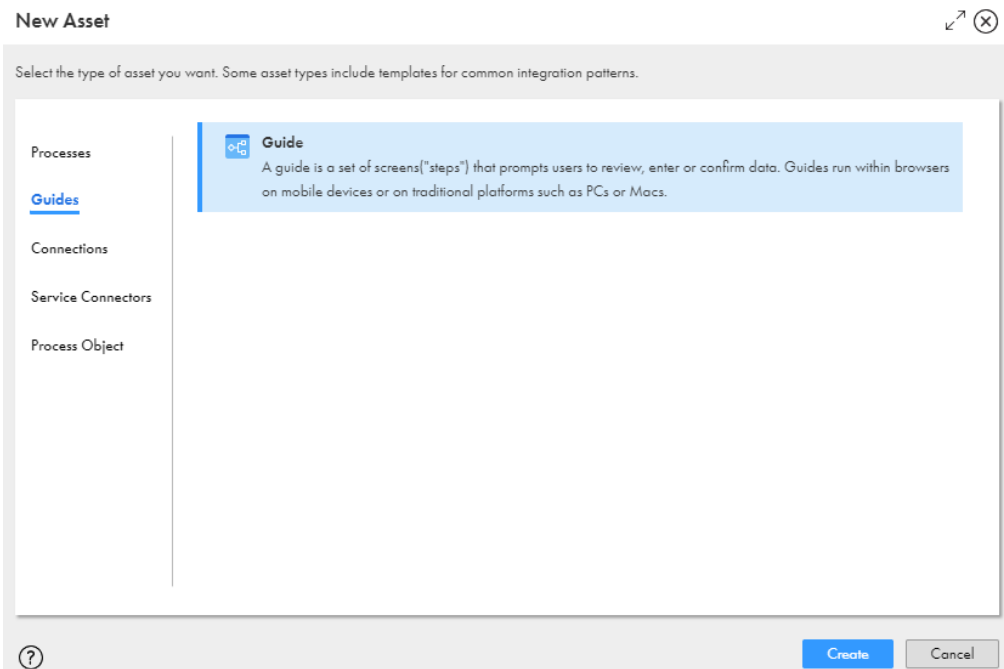


2. Insert optional automated actions. For example, call a process that converts a lead and creates a new opportunity. When you run the guide, if the guide takes the appropriate path, the process is called.
3. Publish the guide so that users can access it for sales activities.

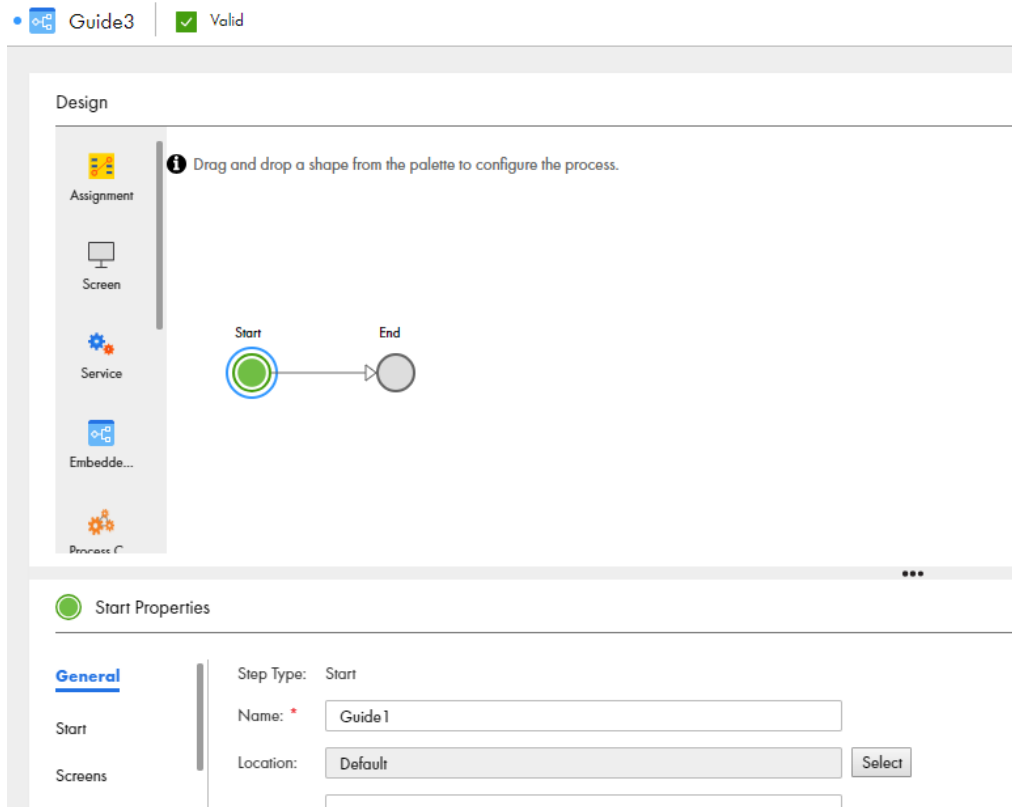
Creating a Guide

Perform the following steps to create a guide:

1. In Application Integration, select **New**.



2. In the **New Asset** dialog box, select **Guides > Guide**, and then click **Create**.
A guide template opens.



3. Enter guide properties and add steps.

Setting Guide Properties

1. Select the **Start** step.
2. Enter properties on the following tabs:
 - General
 - Start
 - Screens
 - Input Fields
 - Output Fields
 - Temp Fields
 - Outcomes
 - Advanced
 - Notes

General

You can specify the following General properties for guides:

Property	Description
Name	A descriptive name to identify the guide in Guide Designer and the name that appears when it is available for use in other objects such as subprocess steps. Note: - Guide names cannot begin with a number. If you enter a number as the first character, Guide Designer adds an underscore to the name. For example, "123" becomes "_123". - To change the name of a published process, you must first unpublish the guide. Then, change the name and republish the guide.
Location	The location of the project or folder you want the process to reside in. To select a location for the guide, browse to the appropriate project or folder, or use the default location.
Description	A description of the guide.

Start

You can define the following Start properties for a guide:

Property	Description
Applies To	The type of object associated with the guide. The objects that display in this list are the process objects you defined separately, or generated in a defined connection. Drill-down in the list to see process objects and connections, and then select one of these items. Select a process object only if the guide will be embedded in another guide. Note: To change the Applies To property of a published guide, you must first unpublish the guide. Then, change the Applies To property and republish the guide Choose Any if you do not want to associate the guide with a specific object. When you select this option, the guide does not automatically have access to an object's fields. Use this object type when a guide creates a new object and does not access information within existing objects. Note: If you run a guide, the Applies To setting must be either Any or a specific object in a service, but not a process object.
Run As (Salesforce only)	For Salesforce only, you can choose whether to run the guide as the Current User or System. Choose Current User if the guide should have the same privileges as the user who is running it. Choose System if the guide should have system-level permissions to enable actions that the guide user may not otherwise be able to access. For example, if a user does not have access to account objects and the guide is running as System, the guide can still access the account objects to the extent needed to run the guide.

Screens

Use the Screens tab to configure where you want the guide to display and the languages you want the guide to support.

You can define the following properties for the Screen step:

Property	Description
Intended Display	The environment in which the guide will run. Select Browser if you want the guide to run on a browser of the following devices: - Desktop - Laptop - Tablet Select Mobile App if you want the guide to run on the following mobile applications: - Informatica - Salesforce S1
Support multiple languages	The list of languages supported by the guide for localization. Select a language from the Add Language list.

Guide Translations

Use the **Start > Screens > Support Multiple Languages** option to select the languages that you want the guide to appear in.

This section discusses the following topics:

- [Creating content for a Translated Screenon page 84](#)
- [What You Can Change in a Translated Screenon page 86](#)
- [Simulating a Translated Screenon page 87](#)
- [What Do End Users Do to Select a Translated Guideon page 87](#)

Select **Add a language** to see the list of languages.

The following image shows a sample **Languages** list:

The screenshot shows the 'Start Properties' dialog box with the 'Screens' tab selected. Under 'Support multiple languages', there is a table listing languages. The table has three columns: 'Name', 'Guide Name', and 'Guide Description'. The 'Default' language is highlighted. Other languages listed are Catalan, French, and Spanish, all with the same 'Guide Name' and 'Guide Description' as the default.

Name	Guide Name	Guide Description
Default	Close Lead	Close a lead in salesforce
Catalan	Close Lead	Close a lead in salesforce
French	Close Lead	Close a lead in salesforce
Spanish	Close Lead	Close a lead in salesforce

By default, all languages use the name and description of the Default language. If you want translated names and descriptions to appear, enter the **Guide Name** and **Guide Description** against each language.

To remove a language, click **Delete**.

You cannot delete the **Default** item.

The following image shows the **Properties Detail** dialog box for a guide that has added languages:

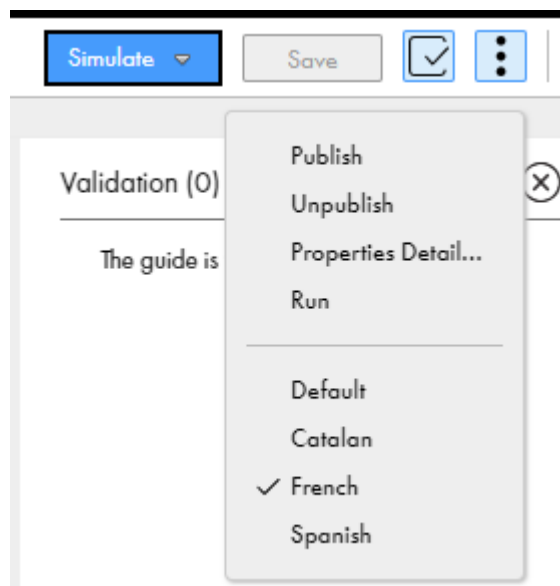
Properties Detail for Close Lead

Basic

Unique Name:	Close_Lead-1
Publication Status:	☒ Published
Published On:	2018-07-03 15:38
Published By:	sabraham@informatica.com
Applies To:	* Any *
Run As (Salesforce Only):	\$current
Languages:	Default: Catalan(ca), Default: French(fr), Default: Spanish(es)

Creating Content for a Translated Screen

Select a language from the **Actions** option, as shown in the following image:



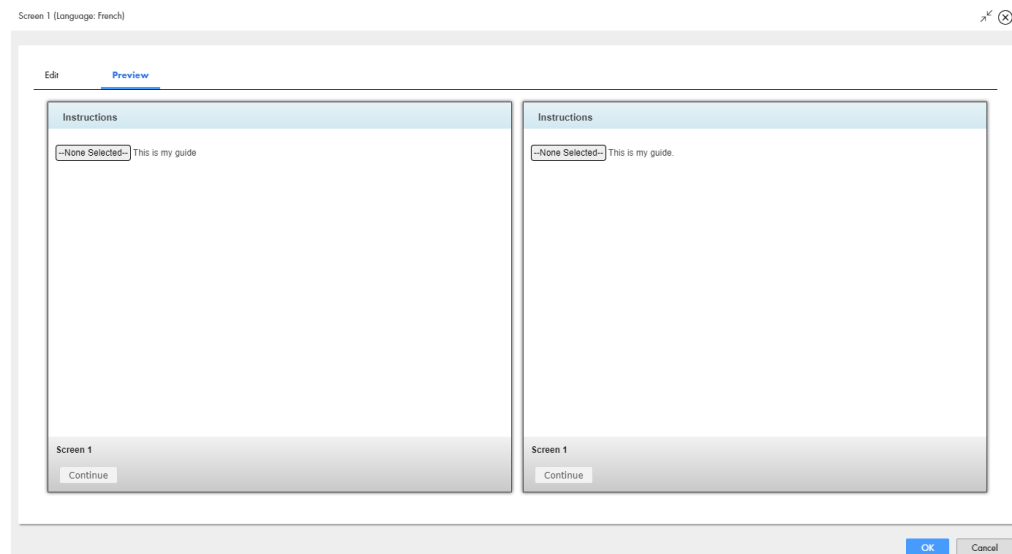
To edit the content of a translated screen, perform the following steps:

1. Go to **Actions** and select the language .
2. Select the step that you want to edit.
3. In the step properties section, click **Preview Screen**.

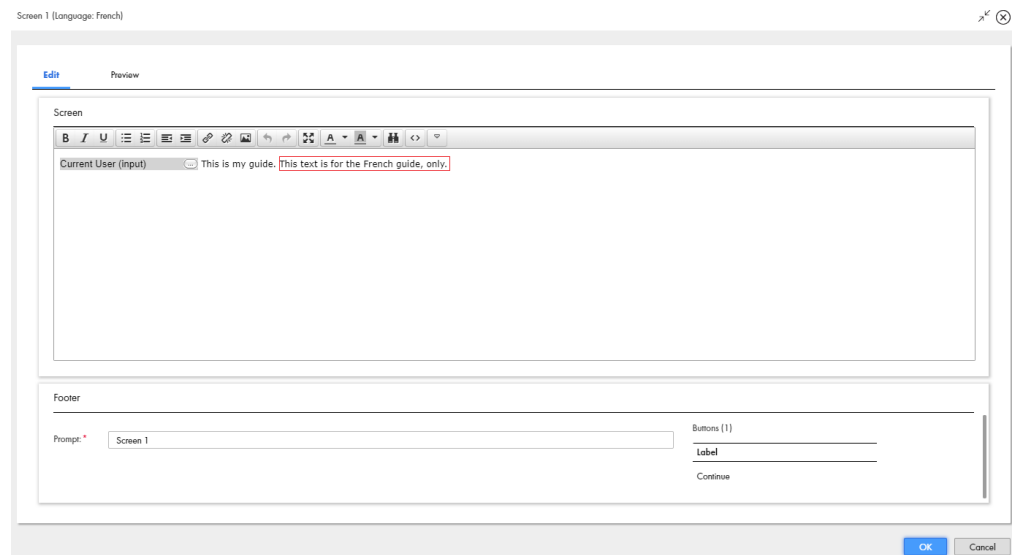
You see a window with sections for the Default language and for the translated language.

Note: The **Edit Screen** and **Preview Screen** options are available for all steps except for the Decision and Jump steps.

The following image shows the default screen on the left and the translated screen on the right:

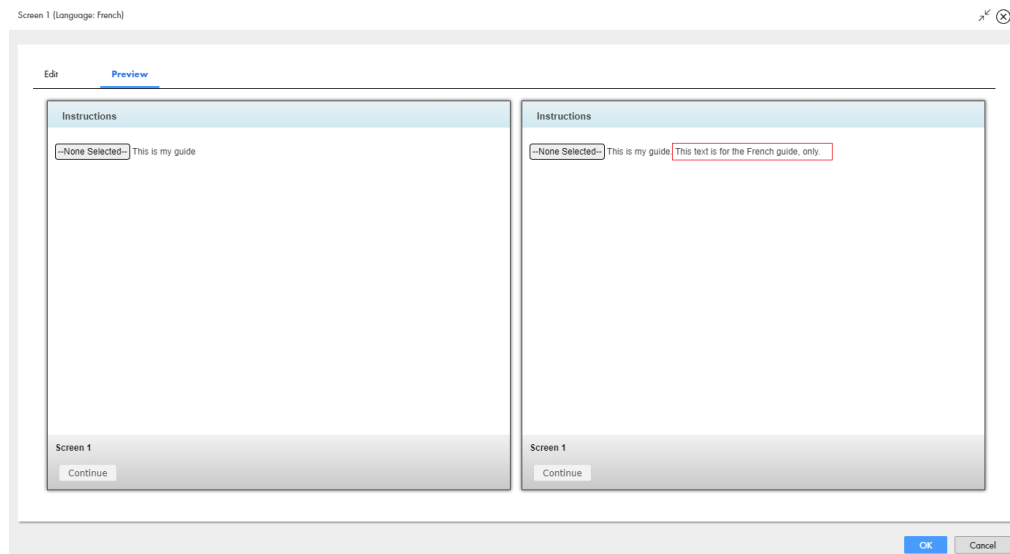


4. Select the **Edit** tab and enter the content that you want the translated guide to contain. The following image shows the **Edit** tab with content entered:



5. Select the **Preview** tab.

The following image shows that the content that you entered appears only in the French screen:



To make changes that apply to all guide languages, select **Default** from the Actions option, select a step, and then click **Edit Screen**.

The following table lists what happens to a translated screen when you make a change to the default screen:

Change Made in the Default Screen	Action in Translated Screen
Add a field	The field is added to the bottom.
Delete a field	Text is added to the translated screen telling you that a field was deleted. You will need to delete this inserted text from the translated screen.
Add an answer	The answer is added. All translations use the default answer until you translate the answer for each language.
Delete an answer	The answer in the same position is deleted.
Reorder answers	Answers are reordered identically.
Add text	Text added to the Default is added to languages that are not translated.
Delete text	Text is deleted from languages that are not translated.

What You Can Change in a Translated Screen

You can only change text within the Instruction, Prompt, Title, and Answers areas. You can also change the layout of information within the translation screen. For example, you can place two bulleted items in default screen within a table in the translation screen.

You cannot change text that is part of the guide. Examples are "Instructions", "Restart", "History", and so on.

Behind the scenes, Application Integration keeps the default and all translated screens synchronized, ensuring that the basic structure between the default and all translated screens is maintained. It prevents you from changing the step type, adding a field, or reordering answers by not allowing you to use them. You are

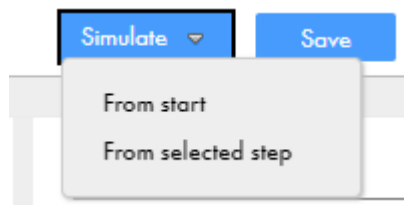
prevented from some actions by controls not being selectable. However, you can still accidentally delete a field or an answer. If you do delete a field or answer, Guide Designer displays an error box and reinserts what was deleted and tells you it did this. If there is text associated with something you accidentally deleted, the information from the Default is copied. For example, if you had removed an answer, Guide Designer inserts the text of the answer that was in the default into the translated screen.

Simulating a Translated Screen

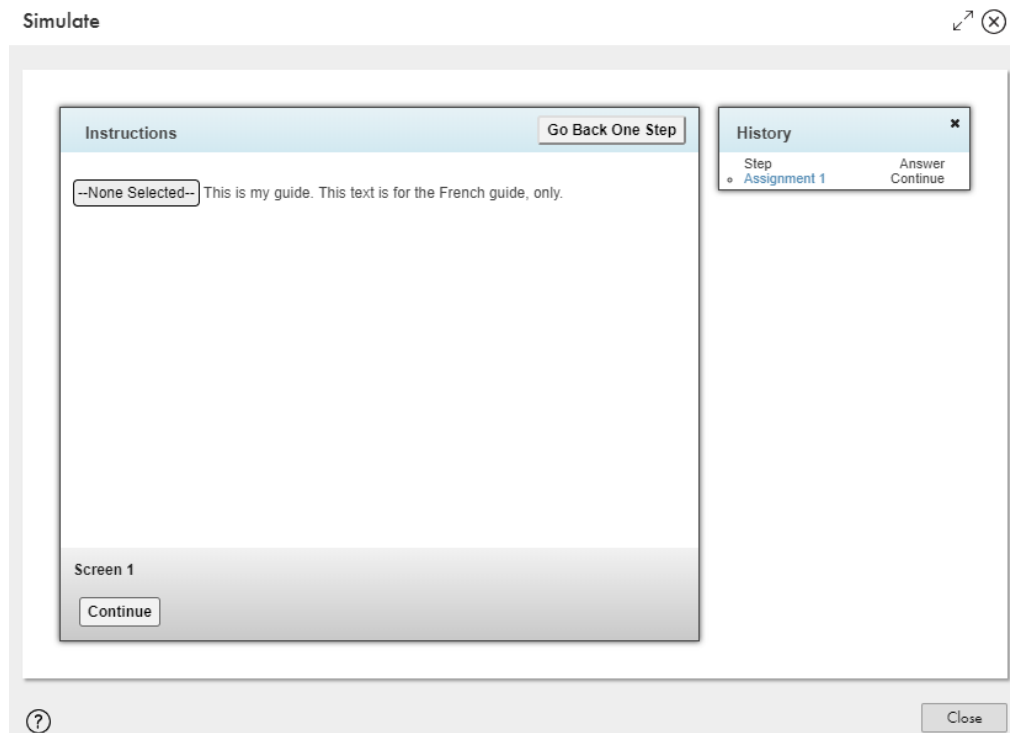
To view how a user will see a translated screen, perform the following steps:

1. Go to **Actions > Select Language**.
2. On the canvas, select the step you want to view.
3. From the **Simulate** list at the top of the page, select **From selected step**.

The following image shows the Simulate list:



The following image shows the simulated step:



What Do End Users Do to Select a Translated Guide?

End users do not need to select a translated guide. Application Integration uses the browser locale to select a language to display.

Salesforce only: Salesforce users are always associated with a language. Depending upon which version of Salesforce you are using, you can see this setting within the Personal Information section of "My Personal Information" or within the Languages & Time Zone section of "My Settings | Personal". Guide Designer checks this value and if a translated guide exists for that language, it displays that guide.

Fields

The fields available for a guide or process are those contained in the object(s) to which the guide or process applies. This is specified in the Applies To property (on the General tab).

You define a name and type for each of the following field categories, for use in a guide or process:

Field Categories	Description
Input	The Input field values are available in all steps of the process or guide. An embedded guide may need to access information from the embedding process or guide. Creating input fields is also a way to define this information so that it can be passed to the embedding guide. If the field is required for the guide or process to execute, check Required .
Output	The Output field values are set when the process executes and are then available in the steps that follow. Define output fields only for use in an embedded process or guide. These output fields do not appear in lists until the values are returned from an embedded process or guide.
Temporary	The Temporary field values be used in all of the current guide's steps. However, a temporary field's value is not available to an embedded guide or process.

For information on how to use input and output fields in embedded guides, see [Embedded Guide stepon page 97](#).

For more information on fields, see [“Introduction to Data Types and Field Properties” on page 9](#).

Outcomes

When a user finishes the last step in the guide, you may want to note how it ended. These endings are called 'outcomes'.

Use the **Start > Outcomes** to define possible outcomes.

In the following image, six outcomes are defined for a guide:

Start Properties

General

Start

Screens

Input Fields

Output Fields

Temp Fields

Outcomes

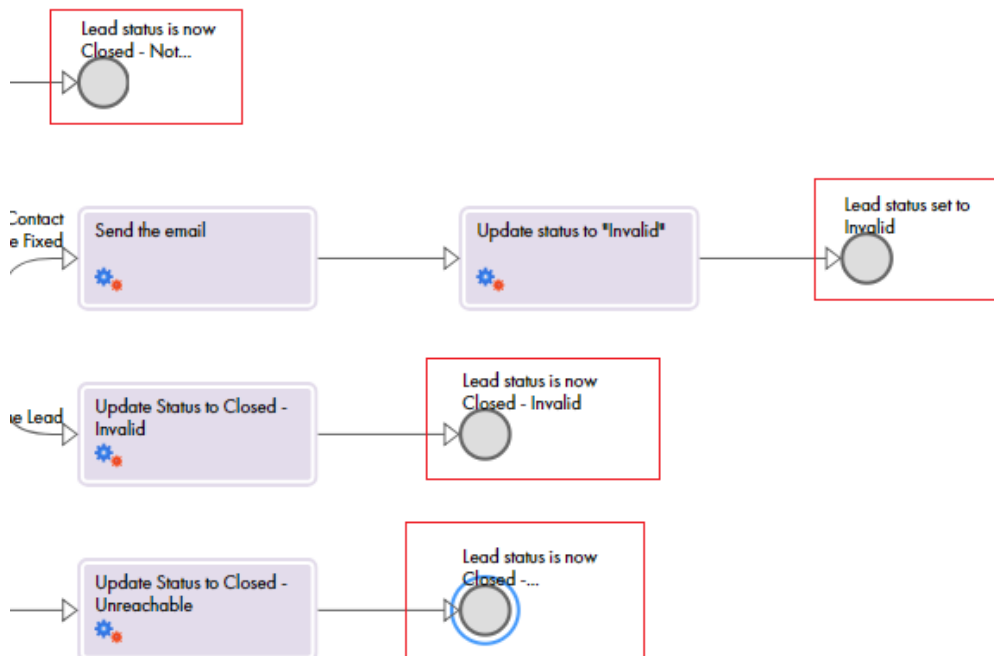
Advanced

Notes

Outcomes (6)

Outcome
Closed - Not Converted
Closed - Invalid
Closed - Unreachable
Invalid
Closed - Not Converted (voice message)
Closed - Not Converted (email message)

The following image shows the canvas view of a guide with part of a guide with four outcomes:



You can also add an outcome while editing an [End/Milestone Step on page 99](#) by selecting **Add new outcomes** in the list of available outcomes.

Advanced

Use the Advanced tab to set the following properties:

Allow Restart

Determines whether users can restart this guide from the first step. If you enable this option, when the guide restarts, it undoes previously executed steps.

Execution Logging

If checked, Application Integration logs details about the guide steps and execution, so it is available for reports.

Rollback Screens

If checked, your application can undo changes made to the database when a user clicks **Go Back One Step** or clicks a step in the History panel. If it is disabled, changes are not undone.

If an embedded guide enables roll back, then the undo is applicable only to the scope of the embedded guide, unless the parent guide also enables roll back.

Always Start New

If checked, opening a guide always creates a new instance of the guide.

Feedback email list

If you want users to send feedback, add one or more email addresses (separated by commas) to receive feedback. Users can then click the feedback option on the guide to display the following dialog:

The dialog shows the guide's name and the step being executed.

Show Guide Only When (Salesforce only)

You can specify criteria that determine when this guide displays in the "Guides" area on a user's Salesforce page.

The following image shows the **Advanced** tab:

The screenshot shows the 'Start Properties' dialog box with the 'Advanced' tab selected. The left sidebar lists various tabs: General, Start, Screens, Input Fields, Output Fields, Temp Fields, Outcomes, **Advanced** (highlighted in blue), and Notes. The main content area contains the following settings:

- ☒ Allow Restart
- ☒ Execution Logging
- ☒ Rollback Screens
- ☐ Always Start New
- Feedback Email List:
- Show Guide Only When: ▼
-

Show Guide Only When

Note: This feature only applies to guides that execute within Salesforce.

You can determine which guides display within the "Guides" area on a user's Salesforce page by entering criteria in the guide properties. Enter the criteria in the **Show Guide Only When** field on the Guide Properties **Advanced** tab.

The criteria must be entered as a Boolean expression. You can use the functions and operators described in the Salesforce documentation. See the examples below.

Use the **Insert Field** list to select insert objects and fields into the expression.

This expression can be as complex as you need. For example, here is an expression that tells Guide Designer that it should only display a guide when an account's owner is one of three people:

```
CONTAINS($Account.Owner.Name, "John Smith")
|| CONTAINS($Account.Owner.Name, "Bob Smith")
|| CONTAINS($Account.Owner.Name, "Jim Smith")
```

Note: Do not use Salesforce field syntax (enclosing field names in curly braces with the "!" prefix) when you enter field names. The expression is passed directly to Salesforce as an Apex expression. If you pass an invalid Apex expression, Salesforce returns an error message.

Example 1

If the field is a list, you can use an equals sign to select one of the items from the list. For example:

```
Account.Type = 'Analyst'
```

Example 2

To define multiple criteria, use "&&" (logical AND) and "||" (logical OR) in your expression. For example:

```
(Account.Type = 'Analyst' && Account.testfield__c= true) || (Account.Phone = '1234')
```

In this case, the expression means that:

1. The account type must be Analyst AND the testfield must be true, OR
2. The account phone number must be 1234.

In this case, the parentheses group the first two tests so they are evaluated as a single unit.

In the following example, the parentheses cause this expression to be evaluated differently:

```
(Account.Type = 'Analyst') && (Account.testfield__c= true || Account.Phone = '1234')
```

Here, the same criteria means that:

1. The account type must be Analyst AND
2. Either the testfield must be true OR the account phone must be 1234.

Notes

Use the **Notes** tab to add information that you or other developers might need. The Notes you enter display only at design time. The guide or process users or consumers do not see the notes. For example, you might add a reminder about something that needs to be done before you deploy a process or guide.

Adding Guide Steps

Add steps to access services and data, and to perform multiple related activities.

When you add a step, you also define properties for that step.

Perform one of the following tasks to add a step to a guide:

- Drag a step from the palette on the left onto the canvas.
- Double click on the canvas and select a step from the **Step Type** list.

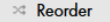
Assignment Step

Use the Assignment step to set a value to a field. To create an Assignment step, click a process, and then select **Assignment** from the Design canvas. Then, from the Assignments properties panel, click **Add Field** to assign Name and Source values.

The following image shows an Assignment step:

 Assignment to temp1, input1 etc. Properties

General	Target	Value
Assignments	temp1	Formula math:sqrt(100)  
	input1	Field temp1 
	output1	Field input1 
	input2	Content 100  
	output2	Formula util:base64Encode[ 

To add a field, click **Add Field** and then add the following information for each field:

Name

This is the fully-qualified field name. You can select a field name from the list of fields you defined under .

Source

This is the source from which the field will take values from. The fields you see depend on the data type you defined under .

For example, if you define the data type as Date, you see the following Source options:

- Specific date
- Days from today
- Days before/after
- Field
- Formula

If you define the data type as Number, you see the following Source options:

- Content
- Field
- Formula

For more information about data types, see [“Using the Field Properties Dialog ” on page 17](#).

To change the order of fields in the Assignment step, perform the following steps:

1. Click **Reorder**.
2. In the **Reorder Expression** dialog box, select a field and use the arrow buttons to move the field up or down the order.
3. Click **Update**.

Create Step

Use the Create step to add new object instances. For example, if you work with an Account object, you can use the Create step to add a new account.

The following image shows a Create step:

Name	Required	Type
Account Name	☒	Text
Account Type	☐	Picklist
Record Type ID	☐	Object ID (
Parent Account ID	☐	Object ID (

Perform the following tasks to create new objects:

1. Choose the **Input Fields** tab and select the fields that you want to add to the object. Some are required, some are optional.
2. Set the source to one of the following values: **Content**, **Field**, **Formula**, or **Screen**.

Use the **Edit Screen** option to place the fields and labels or prompts for the user, if any.

Using Reference Fields

Note the following information about reference fields:

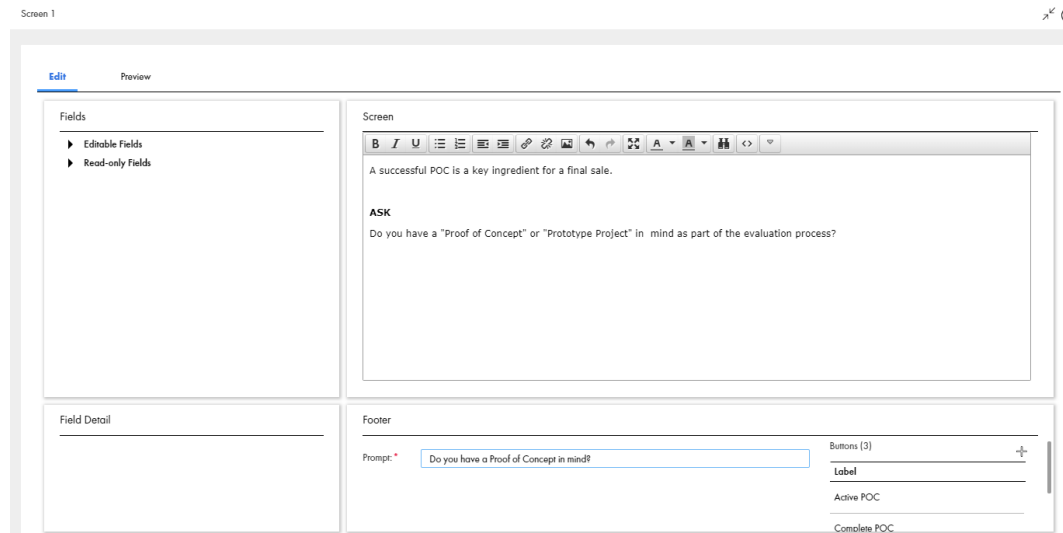
- If you do not include a reference field on the **Input Fields** tab for the object you are creating, Guide Designer uses the value of the current object at runtime.
- Reference fields for the *Applies To* object are set if they are not set explicitly in the Input tab.
- When the content of a reference field is set to empty, it is not included in the create statement. This prevents problems that might occur when explicitly setting a field to empty or null could violate data constraints.

Screen Step

Use the a Screen step to perform the following tasks:

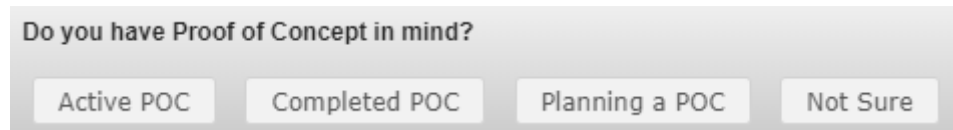
- Show instructions about what is being displayed, what to enter, and so on.
- Update an object's data based on user input.

The following image shows a Screen step that contains what a sales rep would say to a prospect:



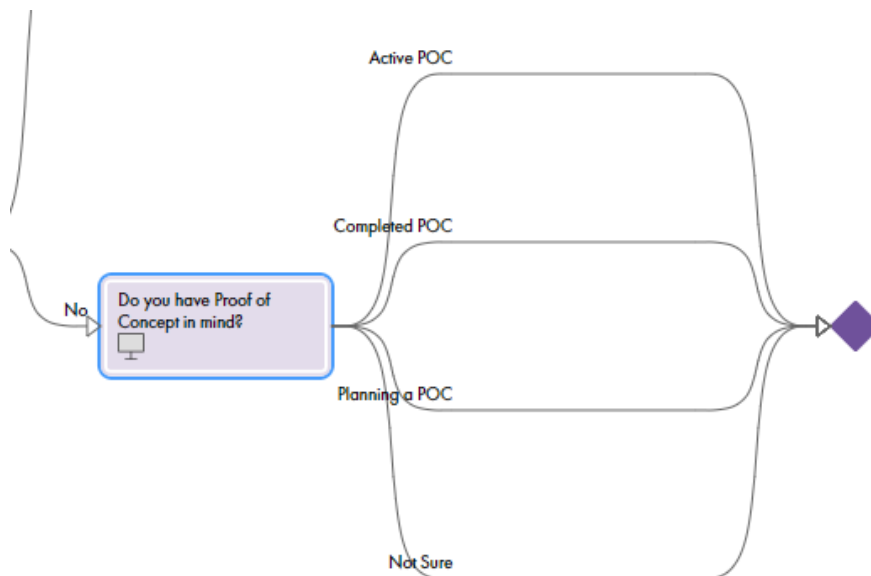
When the sales representative runs the guide, they see the question that they need to ask as well as buttons for each possible answer.

The following image shows a part of a simulated Screen step:



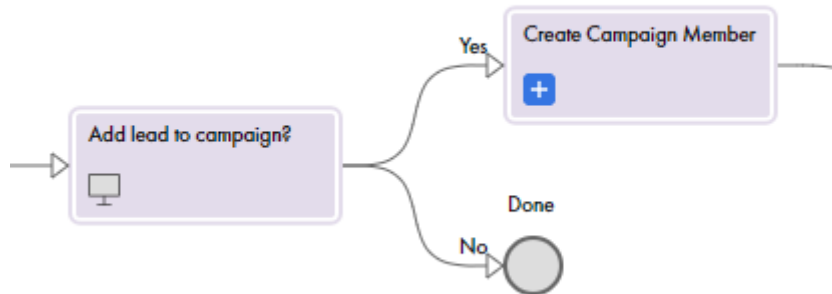
After you add a Screen step with different choices, Guide Designer creates a branch for each answer.

The following image shows a Screen step with four possible answers:



You can add steps to each branch of the Screen step.

The following image shows a Screen step with two answers:



You can insert the following field types when you edit a Screen step:

- Editable Fields: Fields that the user can edit. You can proceed to the next step even if you do not enter values in these fields.
- Read-only Fields: Fields that the end user cannot edit.

To create a translated version of a Screen step, see [Guide Translation on page 83](#).

Decision Step

The following table describes the properties in a Decision step:

Property	Description
Title	The name of the Decision step.
Decision	The guide takes a decision based on the fields and paths you define here. Select a field name from the list of fields you define under Start > Fields. Enter conditions and values that you want the Decision step to base a decision on. The conditions available depend on the field that you select. For example, if you select a field of type Simple > Text, the following conditions are available: - Equals - Starts With - Ends With - Starts with any of - Contains You can enter text values against the conditions you select. You can add multiple conditions to a Decision step. Each condition is a potential data path. For each path that you add, a corresponding branch appears on the UI. Drag branches to rearrange the order in which the branches appear on the UI. Most Decision steps have an Otherwise path. This path handles execution if no data meets the conditions in your tests.

Evaluating Paths

A guide evaluates conditions based on the criteria you specify. Ensure that you construct paths with non-intersecting conditions.

For example, you create a Decision step with the following paths:

- Path 1: Field less than or equal to 100.
- Path 2: Field less than or equal to 75.
- Path 3: Field less than or equal to 25.
- Path 4: Otherwise

If the integer field for which the Decision step was created has a value of 25, the Decision step takes path 1. This is because 25 is less than 100 and path 1 is the first option.

To ensure that the Data Decision step follows the "Field less than or equal to 25" path, re-create the paths with the following criteria:

- Path 1: Integer between 0 and 25
- Path 2: Integer between 26 and 75.
- Path 3: Integer between 76 and 100.
- Path 4: Otherwise

Important: The guide evaluates conditions in a top-down manner. Ensure that the Otherwise branch is the last path.

A Decision step can lead to another Decision step. For example, a branch could run if an annual income exceeds \$100,000. The next decision test along the same path could test if the city is Boston, or otherwise. Using this technique, you use Boolean AND logic because you base the test for the second condition on the true branch of the first condition. In this example, you use the Decision step to set the condition "Annual Revenue exceeds \$100,000 AND city is Boston".

Similarly, to support Boolean OR logic, you can add a test for the second condition on any branch.

Jump Step

When you create a guide, one set of choices might require some activities and a branch would occur. After these activities are completed, the flow of steps should rejoin the actions of another branch.

For example, a guide that leads a sales rep through a conversation with a prospect could take many different branches as the rep talks about features and overcomes objectives. However, at some point in some of these branches, the rep should schedule a meeting. The scheduling step and the steps that follow may be same. Rather than add a scheduling step to each branch that needs one, you could instead jump to a scheduling step that has the scheduling steps.

Set the following property when you create a Jump step:

Property	Description
To	The target of the jump. Select from a list of available steps. More than one step can jump to the same target step. To see how many Jump steps have a particular step as their target, place the cursor over the arrow next to the target step. The Jump step has the following restrictions when used in conjunction with the Parallel Paths step: - You cannot jump to any branch of a Parallel Path step from outside of the Parallel Paths step. - You cannot jump to outside a Parallel Path step from within the Parallel Path step. You can jump from one branch of a Parallel Path step to another branch.

Process Call Step

Use a Process Call step to invoke a process from within a guide.

Guides need not be self-contained. Most guides embed externally created sets of actions, as follows:

- **Process Calls.** The actions defined in a process are placed within the current guide.
- **Service Calls.** The actions defined in a service are placed within the current guide.
- **Embedded Guides.** Another separate guide is placed within the current guide.

The way in which you use a Process Call step is the same as for using an embedded guide step. See [Embedded Guide step on page 97](#) for more information.

Embedded Guide step

Use an Embedded Guide step to better orchestrate and manage your business processes. Design guides that embed other guides, similar to how you would create a subprocess that is called from another process.

Using an embedded guide has the following advantages:

- **Avoids Repetition:** If you place commonly used patterns in a separate guide, you can save time and create other efficiencies if you place the discrete set of steps into a guide that can be embedded and reused in other guides. When you change the embedded guide, you can easily make those changes available to all guides that use it.
- **Clarifies Design:** Embedding can make your design look simpler. Removing steps and placing them into an embedded guide means that the embedding object only shows one step rather than all of the steps. This can simplify the diagram on the canvas. Also, if a problem occurs in the ways steps occur, embedding can make it easier to locate a problem.
- **Clarifies Logic:** If there are many steps that need to operate on other objects, this allows you to create logical chunks of work that each apply to different objects (such as opportunities and leads). This decreases the complexities of working with multiple object types in a single guide.

You handle the data received from a guide and the data that can get sent back to it using the Input and Output fields defined in the guide properties.

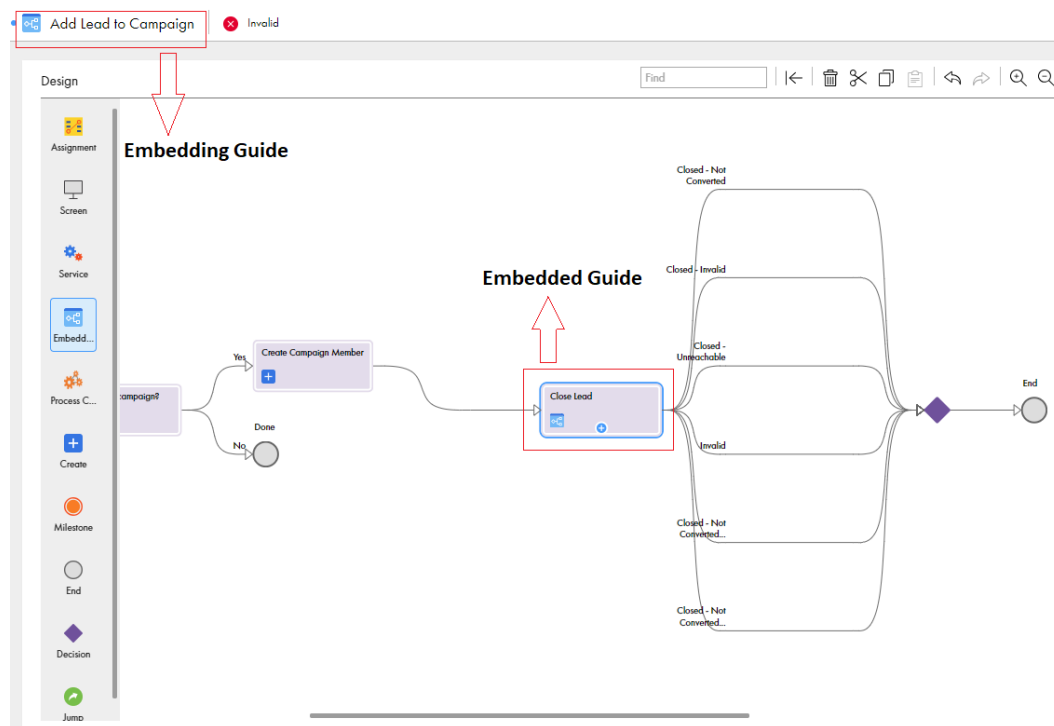
For example, suppose there are a set of steps associated with entering client information. If there are different ways that your users interact with clients and many need to enter client information, you could group these client information steps into their own guide. Now, whenever a guide needs to enter client information, it just "embeds" the guide. This also means that all embedding guides use the same steps

When you create an Embedded Guide step, you enter the following properties:

Property	Description
General	The name of the Embedded Guide step.
Guide	Select the guide that you want to embed within the current guide. The current guide is called the embedding guide. The guide that the embedding guide embeds is called the embedded guide.

Property	Description
Iteration Rule	Configure whether you want the embedded guide to run on a single object or on all objects. If you choose Run on a single object, the embedded guide runs on the specific object it receives as input. The embedded guide passes control to the step that follows in the embedding guide when execution completes. If you choose Run on each object in list until, you also choose an event that controls guide execution: You can also choose from the following objects to process at runtime: - Field: A single object or a list of objects is passed to the embedded guide (based on the selected Run choice). - Query: Available only with Run on a single object. Allows you to define a WHERE condition that selects the object passed to the embedded guide. - List: Available only with Run on each object in list until outcome. Allows you to define a WHERE condition that selects the objects passed to the embedded guide.
Input Fields	Displays the input fields available for the selected object. If the embedded guide receives one of these fields as input, the value of that field may be updated while the embedded guide executes. If so, the value is updated in the parent guide after this embedded guide completes execution.

The following image shows an embedding guide that contains an embedded guide:



Service Step

Use a Service step to connect to an external service, a system service, or a process.

When you add a Service step, you need to configure the Service Type, Connection, and Action.

The following table describes the Service step properties:

Property	Description
Service Type	The connection, process, or system service you add to the process. Select from a list of existing tasks. Note: You must have an existing item to add to the guide. You cannot create an item when you use the Service step.
Connection	The connection to the service connector.
Action	The action contained within the connection.

The following image shows a Service step with the Salesforce connection selected:

Submit For Approval Properties

General

Service Type: Connection

Connection: Default > Salesforce

Action: Submit For Approval

Description

Submits an object for approval based on the approval process associated with the object.

Input Fields

Name	Required	Type	Description
Object to approve	☐	Object ID	The object to submit for approval, by default it is the object the guide is running on (applies to object). You can optionally specify an object other than the current object for example an Account, Contact, or custom object.
Approver	☐	Object ID (Salesforce User)	If the approval process requires manual selection of the approver then this should be the ID of the user for the request.
Comments	☐	Text	Optional comments to add to the submission.

Output Fields

Input Fields

The **Input Fields** section shows the names of input fields that are most often used when using this kind of step. For Service steps that create objects, the input fields shown are the fields that are most frequently needed when the object is created. (If you do not need a field and it is optional, you can delete it.) You can choose additional (optional) input fields from the list.

Use the delete icon to remove a field. This removes input fields that you do not want to pass to the Service step. Some fields are required and cannot be deleted.

End/Milestone Step

The End/Milestone step marks the end of a guide's execution or a milestone.

When you add an End step to a guide, you can configure the step to be a **Milestone** type step or an **End of Guide** type step. A Milestone ending indicates the end of one phase in a guide's activities. .

Note: You can make the text that you enter more readable if you insert text and fields into a table. For more information, see *Inserting Fields in Tables*.

End of Guide Step

An End of Guide step marks the place where a guide finishes executing. Every branch in a guide concludes with an End step, unless it is a [Jump Step on page 96](#). The An End of Guide step specifies an outcome.

The following image shows a sample **End of Guide** step:

End Properties

General

End

Ending Type: ☐ Milestone ☒ End of Guide

☒ Show Screen: [Edit Screen](#) [Preview Screen](#)

Outcome:

Preprocessing:

Master Record > Owner > Group ▼

Set the following properties when you select the **End of Guide** option:

- **Ending Type:** Milestone
- **Show Screen:** Select this option for the user sees the milestone step. If you do not select it, the user does not see the milestone step. Instead, the action indicated in the On Done Button area immediately occurs after this step executes.
- **Outcome:** A list containing the outcomes you defined in the Start step. You can also add a new outcome.
- **Preprocessing:** Select an option from this list to configure the action occurs after the Milestone step completes.
 - **Refresh Current Object:** The page from which you launched the guide is refreshed so that the action performed by the guide is visible on the object's page.
 - **Go to Other Object:** When selected, you see a **Click To Select Field** list to the right. Select one of the items in it. Many are IDs associated with the current object. This is especially valuable for a guide whose primary purpose is to create a new object (for example, a lead). You might then have the End step specify that the user should be taken to the object after the guide finishes executing.
 - **Go to URL:** When selected, you see a **Click To Select Field** list to the right. Select a field that is of type URL.

Milestone Step

Milestones are especially useful when a user hands off work to someone else. (If the guide is handed off, the user who will be handing the guide off should let the next user know that something needs to be done. This may be built into your application's workflow rules.) When that user starts the guide, it starts at the step following the milestone. Milestones should also be used if the user normally stops executing the guide at a step to do something else.

In some cases, you may simply be using a milestone stop to record an event that occurred and the guide will automatically continue to the next step.

The following image shows a sample **Milestone** step:

Milestone Properties

General

Milestone

Ending Type:

☒ Milestone
☐ End of Guide

☒ Show Screen:

Edit ScreenPreview Screen

☐ Allow user to go back:

Preprocessing

Refresh Current Object ▼
Refresh Current Object
Go to Other Object
Go to URL
Refresh Current Object and Continue
Continue

After a guide hits a milestone, further activities do not need to occur. However, the guide will be terminated if no further actions occur within 14 days. Guides without a Milestone guides are terminated in seven days. Once action begins after a milestone is reached, the guide will be terminated if it does not either end or reach another milestone in seven days.

Set the following properties when you select the **Milestone** option:

- **Ending Type:** Milestone
- **Show Screen:** Select this option for the user sees the milestone step. If you do not select it, the user does not see the milestone step. Instead, the action indicated in the On Done Button area immediately occurs after this step executes.
- **Allow user to go back:** Select this option for the user to be able to go back to steps executed before the milestone. In either case, all steps that occurred within the guide display in the guide's history. However, if the user cannot go back, the user cannot select one of these previously executed steps.
- **Preprocessing:** Select an option from this list to configure the action occurs after the Milestone step completes.
 - **Refresh Current Object:** The page from which you launched the guide is refreshed so that the action performed by the guide is visible on the object's page.
 - **Go to Other Object:** When selected, you see a **Click To Select Field** list to the right. Select one of the items in it. Many are IDs associated with the current object. This is especially valuable for a guide whose primary purpose is to create a new object (for example, a lead). You might then have the End step specify that the user should be taken to the object after the guide finishes executing.
 - **Go to URL:** When selected, you see a **Click To Select Field** list to the right. Select a field that is of type URL.
 - **Refresh Current Object and Continue:** After the object upon which this guide is executing is refreshed, execution moves to the step that follows.
 - **Continue:** Notes that the milestone was reached and execution continues with the next step.

Setting the Appearance of a Guide

You can customize the appearance of a guide. You can select from built-in themes to determine what a guide will look like when a user runs the guide on a desktop browser. You can also customize the instructions that display when the guide runs.

Guide Appearance

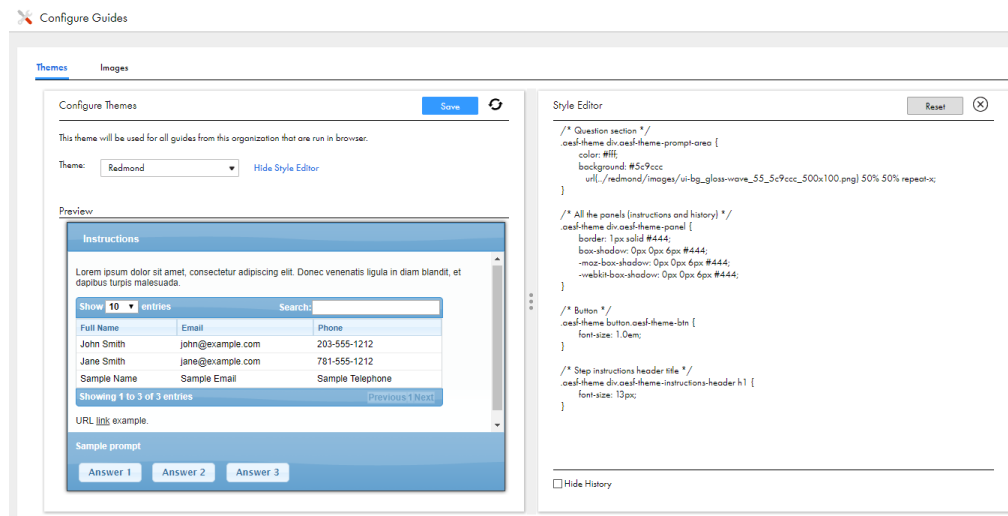
Use the **Configure Guides** page to specify guide themes and manage guide images at an organizational level.

Use the tabs on the **Configure Guides** page to configure guide settings.

Themes

Select from a list of built in themes to configure how you want your guide.

The following image shows the **Configure Guides** page with the **Theme** tab selected:



Click **Show Style Editor** to view and edit the CSS for a guide step, paragraph, or character in the theme. You can use any CSS property normally available for the element type.

For example, to change the color of the steps header text to blue, set the following CSS:

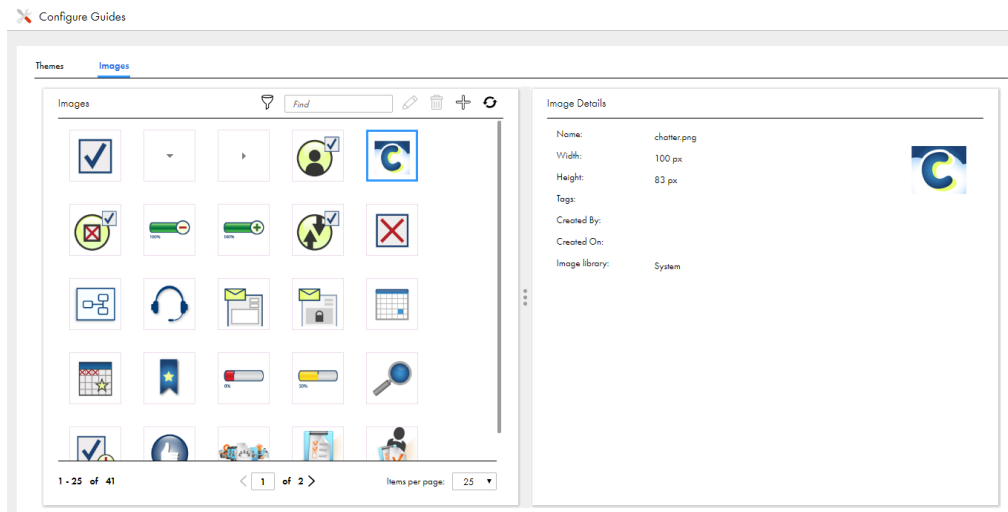
```
aesf-theme div.aesf-theme-instructions-header h1 {
    color:blue;
}
```

When you select, configure and save a theme, it applies to all guides in your organization.

Images

Use the images tab to view and edit icons that you use in guides.

The following image shows the **Configure Guides** page with the **Images** tab selected:

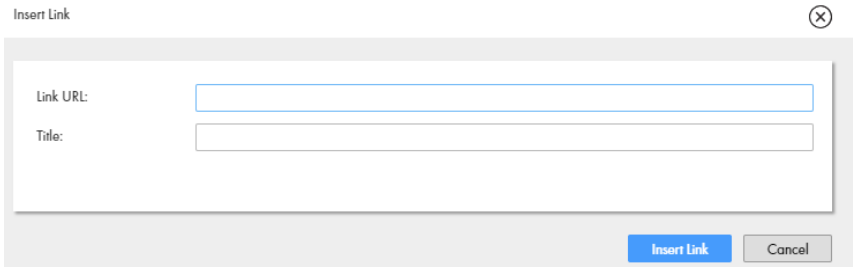


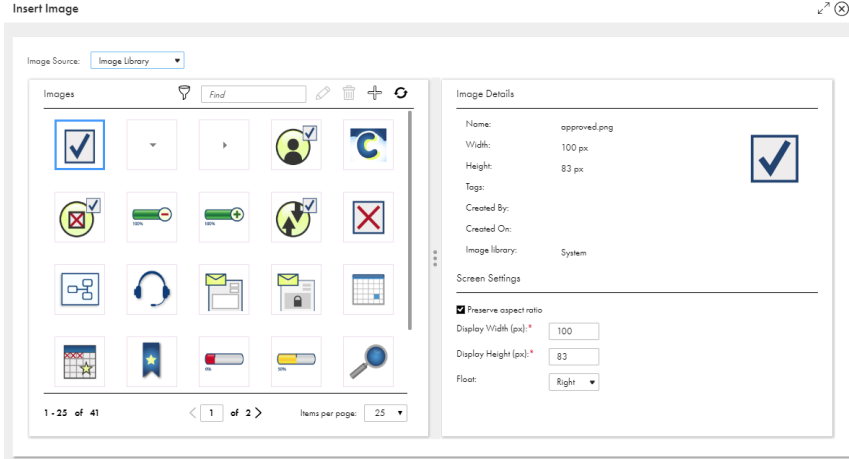
When you click **Add Image** at the top of the Images tab, the Add Image dialog box opens. You can upload .gif, .jpg, .jpeg, or .png images from your system to the Application Integration image library.

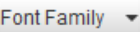
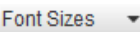
Guide screen Editor

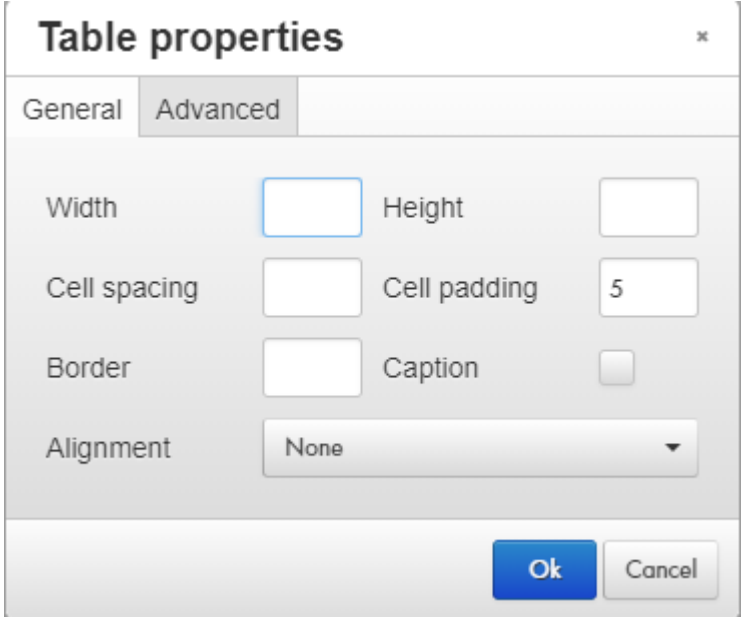
Use the screen editor to insert and edit instructions and fields into a screen that displays in a guide. You can use the following tools in the screen editor:

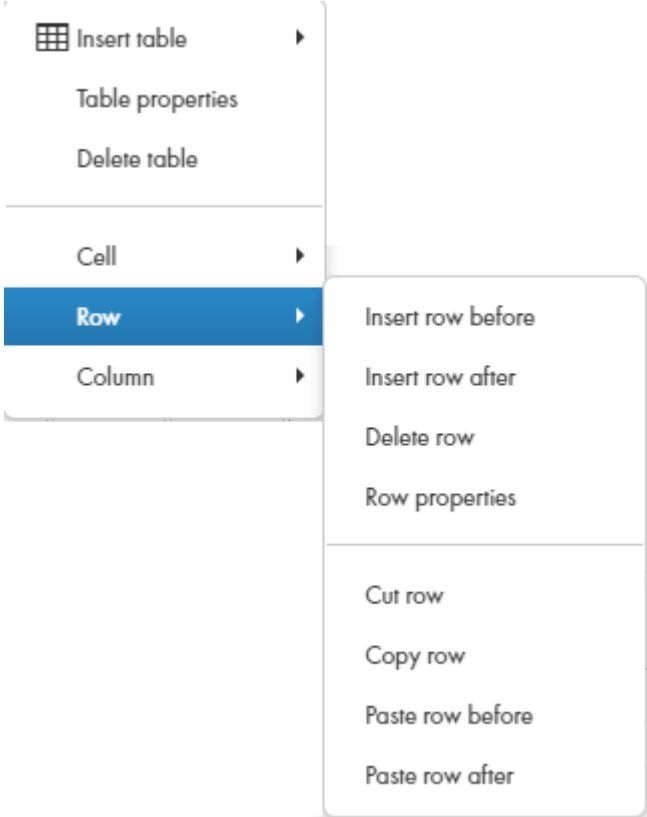
Button	Description
 (Bold)	Sets selected text to bold or indicates that the text you are about to type will be bold. If you press this button when bold text is selected, the bolding is removed. (Keyboard: Ctrl+B)
 (Italic)	Sets selected text to <i>italic</i> or indicates that the text you are about to type will be italic. If you press this button when italic text is selected, the italic is removed. (Keyboard: Ctrl+I)
 (Underline)	<u>Underlines</u> selected text or indicates that the text you are about to type will be underlined. If you press this button when underlined text is selected, the underlining is removed. (Keyboard: Ctrl+U)
 (Bullet list)	If the current paragraph does not have a bullet, adds a bullet and changes the paragraph's indent. If the paragraph has a bullet, this command removes the bullet and removes a level of indenting. When you press Enter within or at the end of a paragraph, the next paragraph also has a bullet. To end the bullet list, either type two returns or reselect this icon.
 (Numbered list)	If the current paragraph does not have a number, adds a number and changes the paragraph's indent. If the paragraph has a number, this command removes the number and removes a level of indenting. If the preceding paragraph is part of a numbered list, selecting this command adds it to that list. When you press Enter within or at the end of a paragraph, the next paragraph also has a number. To end the numbered list, either type two returns or reselect this icon.
 (Decrease indent)	If the current paragraph or selected paragraphs are indented, it removes a level of indenting.

Button	Description
 (Increase indent)	Adds a level of indenting to the current paragraph or to selected paragraphs.
 (Insert/edit link)	Changes the selected text into a link. After you select this button, Guide Designer displays the following dialog:  The files in this dialog are: - Link URL: Where you type a Web address. - Title: Text that appears when a user moves the cursor over the link. (This text may not appear in all browsers.) If you press this button while your cursor is anywhere in a link, you can edit the link's information. If you forget to select text, the link will not display. However, it will be created. Correct this by pressing the <> button to locate the link. If you know HTML, you can edit the link directly. If you do not, just delete it. (An empty link's text is similar to "<a>".)
 (Remove link)	Removes a previously created link. Your cursor can be anywhere within the link. The text in the "Text to display" field remains after you select this button.

Button	Description
 (Insert/edit image)	If an image is not selected, press this button to insert a new image. To change an existing image or its properties, select the image before pressing this button. After pressing this button, Guide Designer displays the following dialog:  The images you see in the top area were previously imported into Guide Designer using the Settings button located on the Design Home page. You can also upload an image for use in the current screen step. An image uploaded here is not available for use in other steps within this guide or in any other guide. You can also insert any image available on the web into the step. Do this by dragging it from the web site directly into the screen editor at the position at which you want to inset it. The fields within this dialog are: - Search area: Use the left text box to search for an image using text that you typed either in the <i>Title</i> or <i>Description</i> fields. Use the right text box to type a tag that was used to label an image. As you type a tag's text, you will see tags that match what you are typing. - URL: If the is accessible using a URL, enter it here. - Title: A title for the image. - Description: Text that describes the image. Notice the text that was added to this dialog. - Width x Height: Use these fields to change the width and height of an image. You can either enter the size in pixels or enter a percent of the original size. The values you type can be less than or more than what the image's original dimensions were. If you only type a width and you entered pixels, the size is changed proportionally. For example, if you change the display width to 235 in the previous example, the display height will be 93 or 94 depending upon your browser. If you entered a percent value, that percent is also applied to the height. - Original W x H: The images width and height as it is stored. - Float: Use the options within this picklist to indicate the image's position in relation to where you inserted it. Your choices are None (it is displayed in line), Left (the image floats to the left), and Right (the image floats to the right). - Browse: Press this button to display a dialog that lets you select an image using your computer's file explorer. After you select and confirm your choice, the image will appear here.
 (Undo)	Eliminates the previously executed action. (Keyboard: Ctrl+Z)
 (Redo)	If you have previously selected Undo, pressing this restores the change. (Keyboard: Ctrl+Y)
 (Fullscreen)	Enlarges the size of the step so that it completely fills the area beneath the Guide Designer tabs. If you have already enlarged it, pressing this button restores the step to its original size. Note: You cannot insert fields into the step when the editor is in full screen mode.

Button	Description
 (Text color)	Pressing this button displays a palette of colors that can be used when displaying text. If no text is selected, pressing this button and selecting a color changes the color of the text you type. If you move the cursor away from this text, the color is what exists at the cursor. Pressing this button and selecting a color while text is selected changes the color of the selected text.
 (Background color)	Pressing this button displays a palette of colors that can be used for coloring the background behind text. If no text is selected, pressing this button changes the background color of the text you type. If you move the cursor away from this text, the background color is what exists at the cursor. Pressing this button and selecting a color while text is selected changes the background color of the selected text.
 (Find and replace)	Pressing this button tells Guide Designer to display a dialog into which you can type the text you wish to locate in the current screen. If you enter text in the Replace with field and press the Replace button, the selected text is replaced. If you select Replace all, all the text that can be found is replaced.
 (Source code)	If text is being displayed, pressing this button tells Guide Designer to display the HTML that is being used to format the text in a different browser window, which is the HTML source editor. This button is not a toggle. After you finish making changes in the source editor, press its OK button.
 (Show/hide more tools)	Adds a second row of screen editor buttons. If this second row is showing, pressing this button removes it.
 (Font family)	Pressing this button displays a picklist of the fonts that can be used to display text. If no text is selected, pressing this button and selecting a font family changes the font of the text you type. If you move the cursor away from this text, the font will be what exists at the cursor. Pressing this button and selecting a font while text is selected changes the font of the selected text.
 (Font size)	Pressing this button displays a picklist of the font sizes that can be used to display text. If no text is selected, pressing this button and selecting a size changes the size of the text you type. If you move the cursor away from this text, the font size is what exists at the cursor. Pressing this button and selecting a size while text is selected changes the size of the selected text.
 (Table)	Pressing this button tells Guide Designer to display a picklist of table commands. Select Insert Table to display a grid of boxes. Use your mouse to select the number of rows and columns that will appear within the table. If the cursor is within a table, you can select other commands in this picklist.

Button	Description
<i>table properties</i>	Selecting Table Properties within the picklist displays a dialog box with two tabs.  Use this dialog to display the height and width of the table. (You should probably set the width as a percent -- use the percent symbol.) The height is seldom set. Cell spacing sets the space between table cell. Cell padding is the amount of space between a cell's contents and the cells edge. If you want a border, enter a number to indicate its width. If you want a caption to a table (this is not usual given the small size of the step's canvas), select this button. A small area above the table will appear. Finally, use the Alignment picklist to select None, Left, Center, or Right. None and Center are the most commonly used. The Advanced tab lets you set the table's style, and border and background colors. Select the Delete Table command to remove the table from the canvas.
<i>cell properties</i>	If you select Cell from the Table picklist, a second picklist displays. Use this picklist to define cell properties, merge two cells together, or split cells that were merged. When defining cell properties, you can specify the cells width and height (normally, you wouldn't select a height), its type (Cell or Header), the Scope (None, Row, Column, Row group, or Column group) of what is changed, and the horizontal and vertical alignment of a cell's contents.

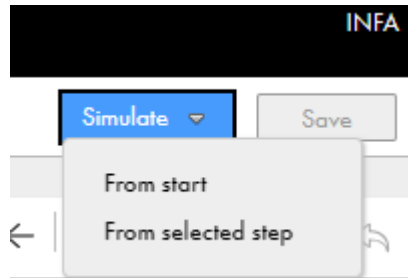
Button	Description
<i>row properties</i>	If you select Row from the Table picklist, a second picklist displays.  Row properties are similar to cell properties. Other commands in this picklist are self-explanatory
 (Clear formatting)	If you've changed the way in which text displays, pressing this button tells the screen editor to remove your changes.
 (Align left)	Aligns currently selected paragraphs or the paragraph containing the cursor to the left margin.
 (Align center)	Centers currently selected paragraphs or the paragraph containing the cursor within the left and right margins.
 (Align right)	Aligns current selected paragraphs or the paragraph containing the cursor to the right margin.
 (Justify)	Adds spaces between words to lines in the currently selected paragraphs or the paragraph containing the cursor so that the text is justified; that is, all text is aligned to the left and right margins.
 (Blockquote)	Changes the indent of selected paragraphs or the paragraph containing the cursor. If the paragraphs are already within a blockquote, this restores the original margin.

Guide Simulation

Use the **Simulate** option in Guide Designer to see how a guide appears when you run a guide.

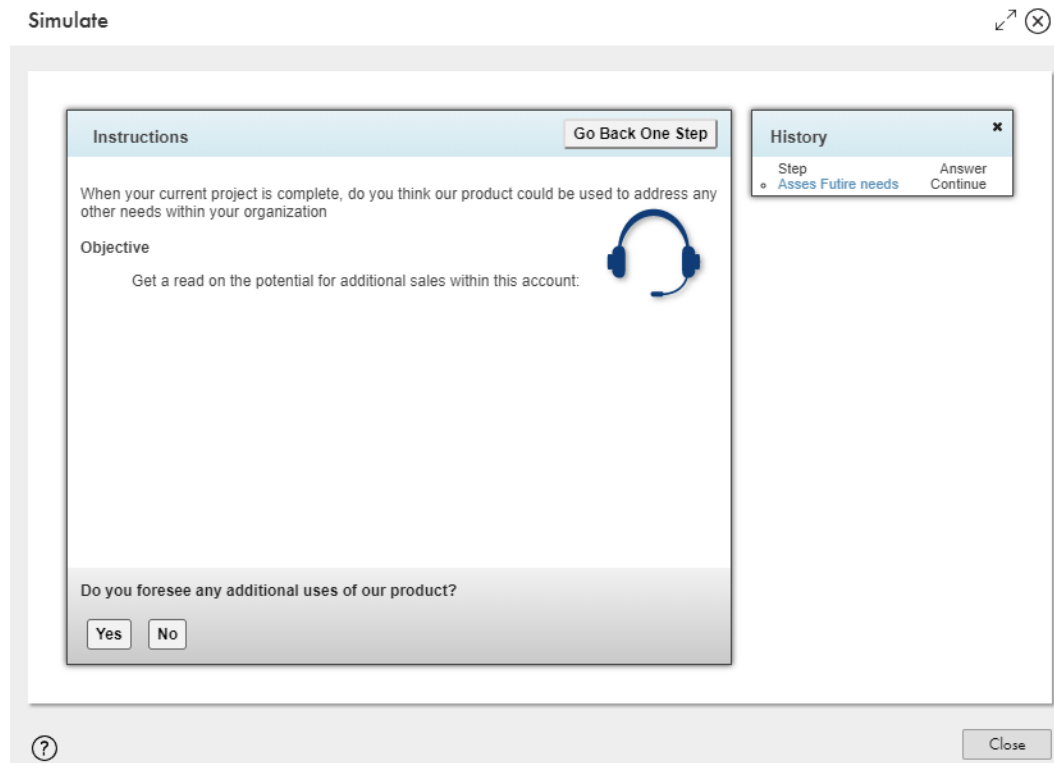
You can simulate a guide in Guide Designer from the beginning or from any step.

The following image shows the **Simulate** option:



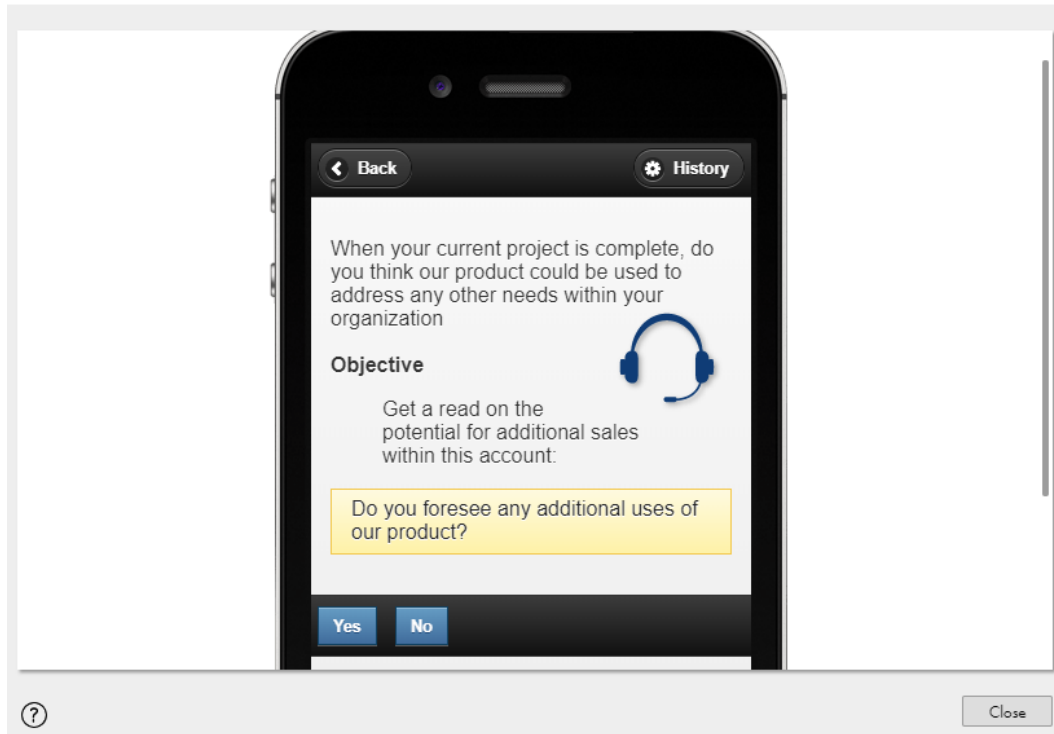
The type of simulation you see depends on whether you configured Browser or Mobile App display type in **Start > Screen > Intended Display**.

The following image shows guide simulation with the display configured as Browser:



The following image shows guide simulation with the display configured as Mobile App:

Simulate



CHAPTER 5

Creating Process Objects

To design a business process, you often need to retrieve sets of structured data. For example, instead of separate items for each kind of demographic data, you could have a group that contains name, address, and phone number. These groups are often called objects. Process objects group and structure this data for use in processes. The named process objects also help you to access returned information more efficiently.

You can define process objects in two ways:

1. As a standalone process object for use in your applications.
2. Within a service connector to handle data returned from a service.

Process objects generated in a service connector are available only within the service connector. Standalone process objects are available in any service connector. Both are used similarly in processes.

Using the Process Object editor, you can define standalone process objects. They are then useable in the same way as built-in data types such as integer, text, or phone. In the Service Connector editor, you can generate process objects based on the data available from the service and the actions, bindings, and other properties you specify there.

CHAPTER 6

Designing Service Connectors

Process Designer allows you to interact with programs and services that are in Data Integration, inside your firewall, or anywhere in the cloud, as long as they have an API. For example, Google provides many APIs to access the services they provide.

To use these processes and services:

1. Create a Service Connector, an object that defines the interaction between Process Designer and the service, so the connector can send and receive data.
2. Define a connection that a process can use to access the service.
3. Add a Service step to the process where you want to use the service.

For example, this step uses the Get Vehicle Detail action to obtain information from Edmunds:

Get Makes Properties

General

Service

Input Fields

Fault Handling

Timer Events

Message Events

Service Type: Connection

Connection: Edmunds2

Action: Get Makes

Description

No description

Input Fields

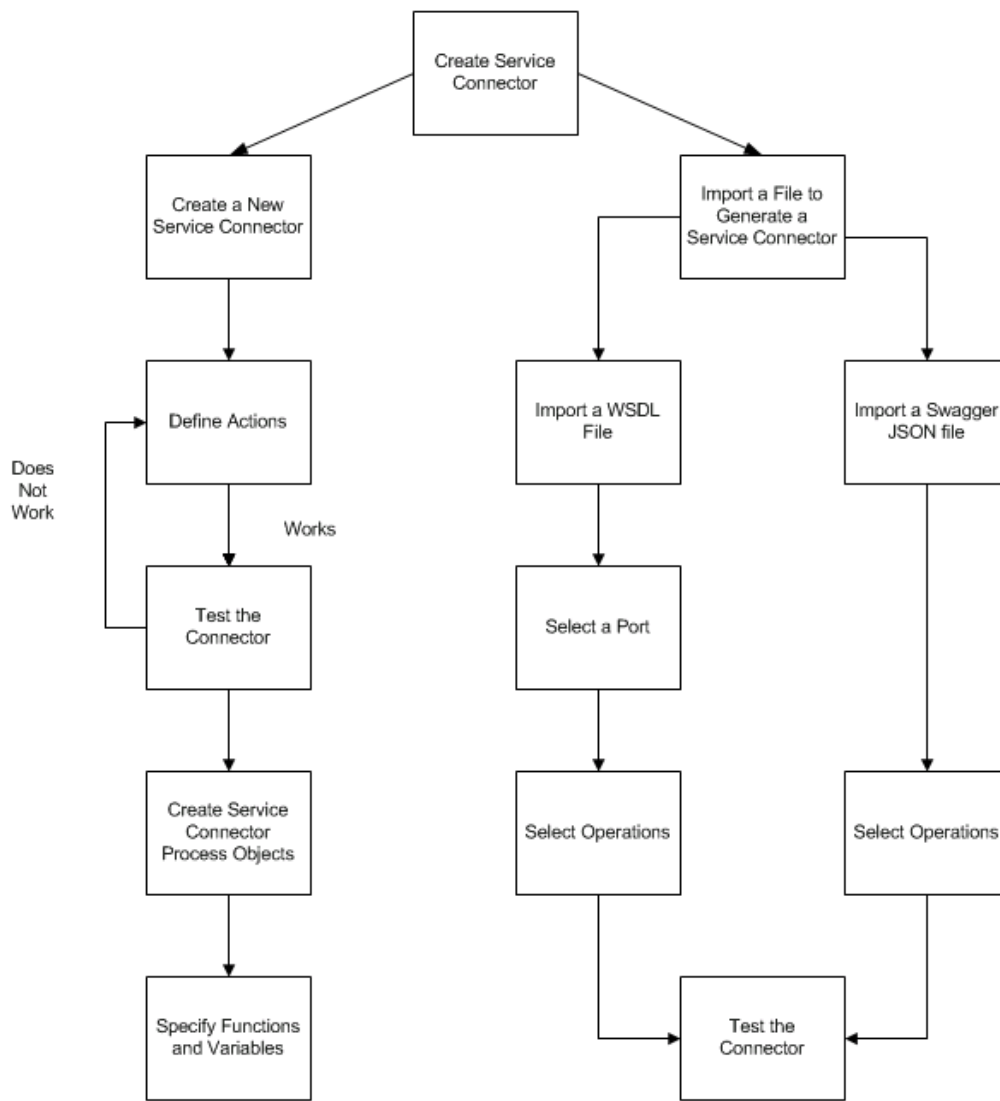
Name	Required	Type	Description
Year	☐	Text	1990 - present

Output Fields

Name	Type	Description
MakesResult	Object ID [Edmunds2.root]	

The services you enable using a service connector are accessible from processes.

The following image shows the process you go through to create a service connector:



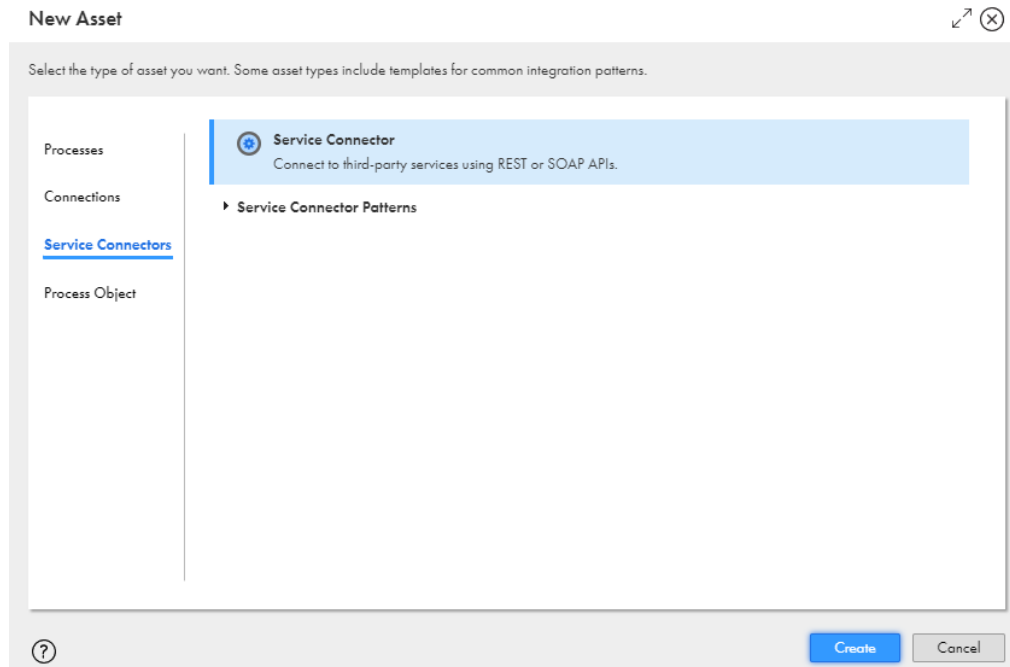
Option1: Creating a New Service Connector

You can create a service connector by defining properties and specifying functions and variables. You can also define the actions associated with a service connector.

To create a service connector, follow these steps:

1. In Application Integration, click **New > Service Connectors**.

2. In the **New Asset** dialog box, click **Service Connector** and then click **Create**.



Defining Properties

To create a service connector and define the properties, follow these steps:

1. In Application Integration, click **New > Service Connectors**.
2. In the **New Asset** dialog box, click **Service Connector** and then click **Create**.
3. Define the following basic properties for the service connector on the **Definition** tab:
 - **Name:** The name by which the service connector is made available for processes.
 - **Location:** Specify the project or folder to contain the service connector.
 - **Description:** Enter a description for the service connector.
 - **Agent Only:** Check if this service connector should only be available to run on the Secure Agent.

- **Connection Properties:** Enter the properties required to connect to this service.

Cloudinary2 Valid Save

Definition Actions Process Objects

Name: Cloudinary2

Location: Pramod Browse...

Description: asda

Agent Only: ☒

Connection Properties

Name	Description	Test With	Type	Required	Encrypt
CloudinaryApiKey	API Key	475257312787584	string	☐	☐
CloudinarySecret	Secret	dZGVrdCiw601mipRvQL_43PpiOc	string	☐	☐
CloudinaryName	Cloud Name	dli78hbgj	string	☐	☐
UploadPreset	Preset for unsigned requests	mqohy1xo	string	☐	☐

For more information about the Actions and Process Objects that define service connectors, see [“Defining Actions” on page 116](#) and [“Creating Service Connector Process Objects” on page 126](#).

Specifying Functions and Variables

To configure service connectors, you may need to specify variables and functions to define bindings, output fields, and other properties.

Built-in Variables

You can reference these variables in a service connector using an XQuery expression:

Variable	Variable Type	Description
\$VariableName	All connection properties	Properties, input and Other Parameters can be specified using this format.
\$ResponseStatusCode	Output field mapping	HTTP response code.
\$ResponseHeaders	Output field mapping	Contains the HTTP response header in an element list where each item is: <header name="Content-Type">text/plain</header> For example: \$ResponseHeaders[@name = "Content-Type"]/text()
\$RESTResponse	Output field mapping	Contains the RESTResponse XML data, including the headers and code. Visible in the Test Results for the Service Connector.

Output Field Mapping Functions

When you define bindings in the service connector, you can use many functions in the Expression Editor.

For information on all the available functions, see [“Using Functions” on page 26](#) and [“Using the Expression Editor” on page 20](#).

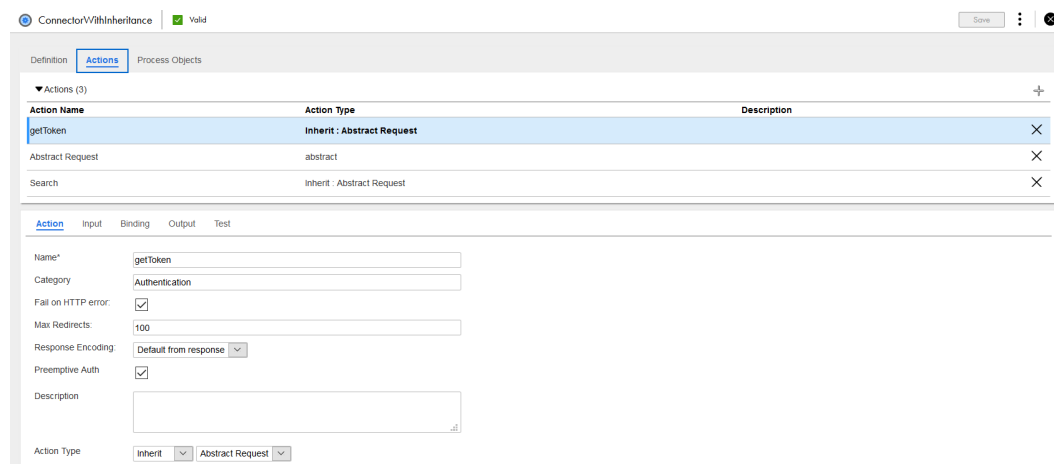
The following table describes functions available for service connector output field mappings:

Function	Syntax	Description
responseHeaderExists	svc:responseHeaderExists(\$ResponseHeaders, headerName) : boolean	Returns a Boolean value that indicates if a header parameter exists within the response.
getResponseHeader	svc.getResponseHeader(\$ResponseHeaders, headerName, defaultValue) : string svc.getResponseHeader(\$ResponseHeaders, headerName): string	Returns a response header value. If the header parameter is not defined, this function returns the optional default value.
getResponseHeaderNames	svc.getResponseHeaderNames(\$ResponseHeaders) : list of strings	Returns a list that contains the names of all the parameters within a response header.

Defining Actions

On the Actions tab, you can create and describe one or more actions associated with a service connector.

The following image shows the **Actions** tab:



Choose the row you want to edit and then enter the following information in the Action details tab:

Click **+** to add a new row.

- **Action Name:** (required) Enter the name that appears in lists when referencing this Action in service connectors and connections. Do not use spaces or special characters in this name.
- **Category:** If you have many service connectors, each with multiple actions, you can create categories to help users navigate within processes.

- **Fail on HTTP error:** Check this option if you want Process Designer to acknowledge that an error occurred when you send a request to a server using this action. For more information see [“Checking for HTTP Errors” on page 124](#).
- **Max Redirects:** Specifies the maximum number of redirects that an action can make. When an action accesses an endpoint, the endpoint might redirect the action to another endpoint. Enter a value to control the number of redirects.
The default **Max Redirects** value is 100.
Enter **0** to disable redirects.
For example, if you enter **3**, the service connector action makes a maximum of three redirects and receives three redirect responses from the endpoint. If the endpoint requests a fourth redirect response, you see an error message.
- **Preemptive Auth:** Sends authentication details, that is, user credentials, along with a request to an endpoint. Select **Preemptive Auth** when you know that an endpoint requires authentication. If you do not select **Preemptive Auth** and the endpoint requires authentication, the following events occur:
 1. The service connector sends a request to the endpoint.
 2. The endpoint requests authentication.
 3. The service connector sends user credentials to the endpoint.
 If you select **Preemptive Auth** and the endpoint requires authentication, you avoid an extra request.
- **Action Type:** You can select one of the following action types for each action. Refer to [“Shared Service Actions” on page 117](#) for more information.
 - **General** actions are published for use only with a particular service connector.
 - **Abstract** actions are not published but can be shared with other actions, that is, made available for reuse as a template for other actions.
 - **Inherited** actions inherit properties from an Abstract action.
- **Description:** Enter a description or notes about this action.

For each action, specify additional details on the **Input**, **Binding**, **Output**, and **Test Results** tabs.

Shared Service Actions

Using shared service actions, you can define common inputs, bindings, and outputs, then reuse these definitions in other actions that “inherit” from the parent “abstract” action.

For example, you might have multiple actions which all share the same:

- Input parameter session-id
- HTTP Header such as content-type
- Binding method (GET/POST) and URL
- Authentication Credentials
- Output fields

In that case, you can:

1. Define an Abstract type action that contains all of the common definitions:

The screenshot shows the 'Actions' tab in the Informatica Cloud Service Connector configuration. A table lists 16 actions, with 'ICSBaseGET' highlighted as an abstract action. Below the table, the configuration for 'ICSBaseGET' is shown in the 'Action' tab. The configuration includes fields for Name (ICSBaseGET), Category, Fail on HTTP error (checked), Max Redirects (100), Response Encoding (Default from response header), Preemptive Auth (checked), and Description (Login to ICS using the connection's username and password). The Action Type is set to 'Abstract'.

2. Create multiple Inherit type actions that use those common definitions and extend or add to certain parameters.
3. For example, an Inherit action can:
 - Define a new input parameter (for example, \$messageId).
 - Extend the binding URL and append the new parameter to the base with "{messageId}".

For example:

The screenshot shows the 'Actions' tab in the Informatica Cloud Service Connector configuration. A table lists several actions, with 'Get Mapping Task' highlighted as an inherit action. Below the table, the configuration for 'Get Mapping Task' is shown in the 'Action' tab. The configuration includes fields for Name (Get Mapping Task), Category (Task), Fail on HTTP error (checked), Max Redirects (100), Response Encoding (Default from response header), Preemptive Auth (checked), and Description (Get the details of a mapping configuration task). The Action Type is set to 'Inherit' and the parent action is 'ICSBaseGET'.

Notes on Usage

Shared service actions:

- Are not available at design time within IPDs and Guides.
- Inherit inputs, bindings, and outputs from a single parent (Abstract) action.
- Do not allow abstract actions to inherit from other abstract actions.
- Allow each Service Connector to have multiple abstract actions.

After you specify the Inherit Action Type, many fields are disabled.

However, for specific Binding and other options, you can choose to:

- **Inherit:** Use all input parameters from the Abstract action, as is. This is the default.
- **Exclude:** Exclude the parameter so it is not available to the action at design time or runtime.
- **Overwrite:** Specify parameters to overwrite what was inherited.

Input Tab

Use the **Input** tab to define input data items that are unique to the service to which you are sending data.

For example:

The screenshot shows the Service Connector Editor interface. At the top, there's a header bar with 'Connector: WithInheritance' and a 'Valid' status indicator. Below this, there are tabs for 'Definition', 'Actions', and 'Process Objects'. The 'Actions' tab is active, showing a list of actions. The first action, 'getToken', is selected and highlighted in blue. Below the actions list, there's a sub-tabbed interface for the selected action. The 'Input' sub-tab is active, showing a table of input fields. The table has columns for 'Name*', 'Label', 'Type', 'Required', 'Description', 'Parameter', and 'Test with'. One row is visible with 'attachment' as the name, an empty label, 'Attachment' as the type, and 'hosts.txt' as the test with value. There are icons for adding (+) and deleting (X) rows.

Action Name	Action Type	Description
getToken	Inherit : Select Action	
Abstract Request	abstract	
Search	Inherit : Abstract Request	

Name*	Label	Type	Required	Description	Parameter	Test with
attachment		Attachment	☐		☒	hosts.txt

Depending on the service, you may have no data items or many.

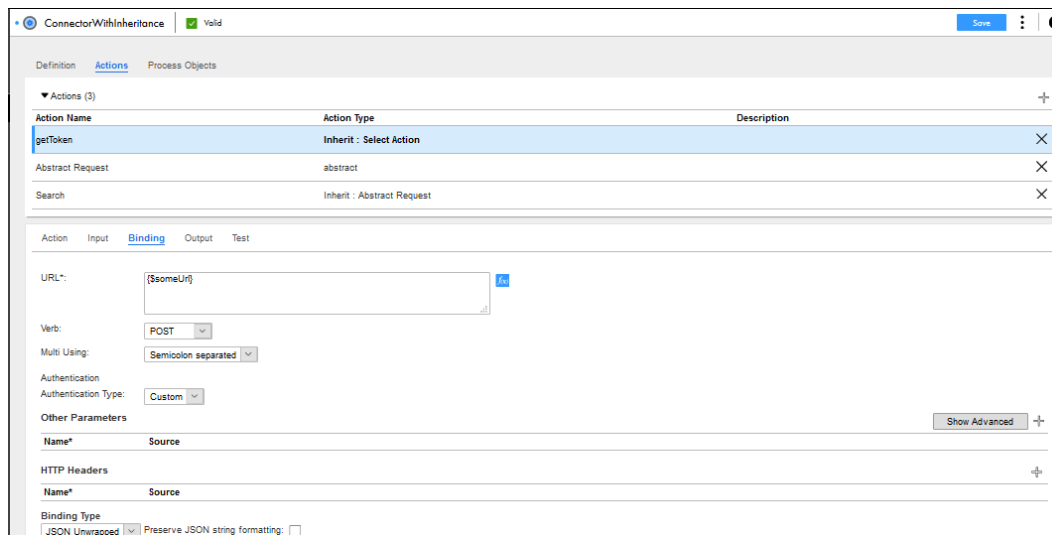
For each item, enter the following properties:

- **Name:** Required. The name of the input data item.
- **Label:** The name of the item to display within a process. If not specified, the Name is displayed.
- **Type:** Select the data type of the item from the list. If you select the Reference or Object List type, you can also choose a Process Object.
- **Required:** Check if a value must be set for this parameter.
- **Description:** Enter a description of the parameter.
- **Parameter:** When checked, the service connector passes this parameter to the service. If the parameter is only used when constructing another parameter and does not need to be passed to the service, uncheck this field. For example, the service might expect a date to be in RFC 1123 format but you want the process that calls it to pass in the Process Designer's normal format, an XSD date.
- **Test with:** Enter a value that Process Designer will use to test the action. If the input is an attachment, you can also upload a sample attachment file.

Use the **+** icon to add a new item. Press **X** to delete the current row.

Service Connector Bindings

For each action in a service connector, the Binding options enable you to specify the interface and parameters required to communicate with a service.



Binding Properties

Specify values for these options:

- **URL:** Enter the URL of the REST Service.
- **Verb:** Choose how to send data to the service:
 - **GET, DELETE, HEAD, OPTIONS, and TRACE:** Process Designer automatically generates the query parameters.
 - **POST and PUT:** Process Designer adds a Binding Type field from which you can then choose JSON, Form, URL, or Custom. If you choose Custom, a Body field appears where you can type the information that is sent to the service. Consult the API documentation to determine what you should enter.
- **Multi Using:** This option determines how query string parameters are generated when you need to specify multiple values for a parameter. Choose one of the following:
 - **Semicolon separated:** for a semicolon-separated multi-select list, Process Designer adds input parameter values as a single string. (For example, *?param=a;b;c*). This is the default.
 - **Comma separated:** for a comma-separated multi-select list, Process Designer adds input parameter values as a single string. For example, *?param=a,b,c*.
 - **Append brackets:** for a semicolon-separated multi-select list, Process Designer maps semicolon-separated values to multiple query parameters of the same name, and appends `[]` to the parameter name. For example, *colors[]=red&colors[]=blue&name=JW*.
 - **Append numbers:** for a semi-colon separated multi-select list, Process Designer maps semicolon-separated values to multiple query parameters of the same name and adds a number to the end of the field name, that is, 1..N. For example, *colors1=red&colors2=blue&name=JW*.
- **Authentication:** You have two options to define the authentication information that the service requires: Basic or Custom. If you choose:
 - **Basic,** enter the user name and password, if required.
Enter text or an XQuery expression in the user name and password fields. If you enter an XQuery expression, you can pass the password value through a connection instead of hard coding it in the service connector.

Select **Show password** to see the value of the password on screen. When you export the service connector with **Show password** selected, you also export the password.

- **Custom**, add the security parameters based on the service requirements. For example, you might need to send signed data using an HTTP header.

Other Parameters

Use this section to add input parameters required by the service. The value may be specified as a literal or expression. Refer to documentation from the service you are using.

To add a row and enter multiple parameters, click the **+** sign. To remove a row, click the **X**.

For each row, enter:

- **Name**: the name of the parameter as specified within the service being called.
- **Content**: the value to specify for the parameter, if any.

Click **Show Advanced** to select these optional parameters for the Binding:

- **Sign With**: The value that is computed will first be signed using SHA1, and the result of the signature will be this value.
- **Sign Using**: Choose to sign the field using SHA1 or SHA256.
- **Encode Using**: Choose to encode the field as Base64 and Hex64Upper.
- **Temp**: If checked, Process Designer will not generate a parameter when using GET or POST. For more information, see the Parameters described here: ["Input Tab" on page 119](#).
- **Attachment Base64**: The base64 encoding of the attachment associated with the <inputName>.
- **Attachment Name From**: The file name of the attachment associated with the <inputName>.

Click **Hide Advanced** to hide the fields.

HTTP Headers

If the service requires one or more HTTP headers, enter the Name and Content (value of the parameter) for each header in this section.

To define a SOAP-based service connection, you must add both the "SOAPAction" and "Content-Type" headers. The value of SOAPAction can be empty, dynamically set using GET, or defined in the WSDL.

Note: If the service does not set the content-type in the response header, the service connector attempts to infer the content type from the payload. For example, if the response starts and ends with either "{...}" or "[...]", the response is parsed as JSON. If the response payload starts with an angle bracket ("<"), it is treated as XML. If the response payload cannot be parsed due to malformed JSON or XML, it is processed as a regular string.

Note: As you type in the Name field, Process Designer displays commonly used values, including:

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language
- Content-Type
- SOAPAction

Some services request an MD5 signature. You can generate an MD5 signature in the **HTTP Headers** section of service connector actions.

To generate an MD5 signature, perform the following steps:

- Enter a name for the HTTP header. For example, enter **Signature**.

- Go to **source > Formula Editor**.
- Under **type**, select **XQuery**.
- Under **Insert Field**, select **PAYLOAD_DIGEST_MD5**.

When you test or run the service connector, the used variable contains an MD5 checksum for the generated request payload.

Content Type and Character Sets

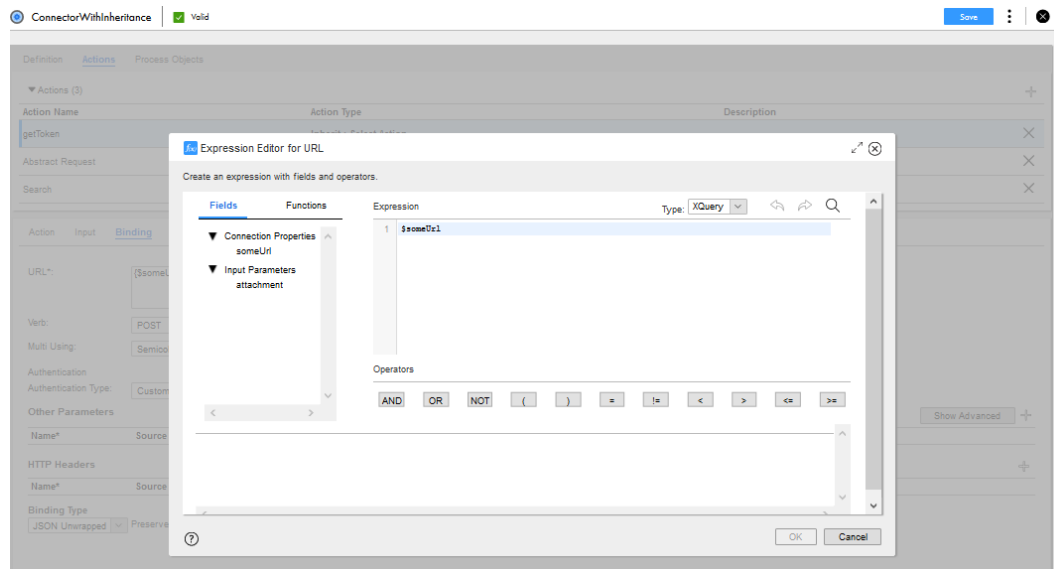
You can determine the charset used for a request payload as noted here:

- Specify a charset value in the Content-Type header. In this case, the header determines the charset of the request payload. In this case, the charset is based on the value specified in the header unless the request format is an attachment type, in which case each part of a multipart/form-data request will be treated separately. If you set the Content-Type and want Process Designer to append the default charset automatically, add a semi-colon after the specified Content-Type, as shown here:
Content-Type=application/json;
- Do not specify a charset value in the Content-Type header so this value can be set automatically based on the request format. To ensure that the content can be decoded correctly, Process Designer appends the default charset (UTF-8) to the Content-Type. For example, the value might be set as follows:
Content-Type=application/json;charset=UTF-8

Expression Editor for Service Connector Properties

To define the binding parameters, you can use the Expression Editor accessible for the URL, Other Parameters, and HTTP Headers fields.

To open the Expression Editor, click **f(x)** next to the field you want to edit. When it opens, the Expression Editor gives you access to a list of available functions and the fields defined for the service connector, as shown in this image:



Based on the type selected for the expression, the Expression Editor applies syntax validation. In this case, the `post_url` field is validated as an XQuery variable.

For more information, see [“Using the Expression Editor” on page 20](#).

Output Tab

Use the **Output** tab to define how the service connector will parse data returned from a service and place it into variables. Typically, the returned data is XML or JSON converted to XML. You can map this XML to output fields in two ways:

1. Directly map the data to a property based on XML tag names or JSON properties.
2. Use Expression to extract elements.

For example:

Connector: WithinInheritance | Valid

Definition | **Actions** | Process Objects

▼ Actions (3)

Action Name	Action Type	Description
getToken	Inherit: Abstract Request	
Abstract Request	abstract	
Search	Inherit: Abstract Request	

Action | Input | Binding | **Output** | Test

Output Fields

Name*	Type	Description	Get From
AllModelsMake	XML		Expression \$RESTResponse
CarMakesCount	Number		Property CarMakesCount
CarMakesList	Object List Process Object Any Type		Property CarMakesList

For each output data item, specify:

- **Name:** The name of a variable into which a returned value is placed.
- **Type:** The data type of the value being written to the variable. If the type is `objectList` or `reference`, Process Designer displays a list of Process Objects so you can choose one of the objects defined within the Process Objects tab.
- **Description:** Text describing the variable
- **Get From:** Select one of the following:
 - **Property:** To enter a named value (which is the name used within the XML returned by the service) to be placed within the variable.
 - **Expression:** To write an XSLT expression to parse the XML returned by the service. In the example figure, we use the XSLT `count` function to return one value for the `AllModelsCount` variable. The expression used for the `CarMakesList` returns a list. Click ... to open an XQuery Editor tab appears where you can type the query.
 - **HTTP Response Header:** To enter the part of the response header to assign to the field. See the details below.
 - **HTTP Response Status Code:** To check the HTTP response status code.
 - **Entire Response:** To assign the complete contents of the response payload to the field.
 - **Simplified XML:** To rearrange data so it can be used by process objects. For more information, see [“Simplified XML” on page 124](#).

For details on handling HTTP errors, see [“Checking for HTTP Errors” on page 124](#).

Simplified XML

Much of the complexity in standard XML, with a mixture of namespace, attributes, siblings, and children, is not needed within Process Designer in order to present data to people designing processes. While you sometimes need to use the XQuery option for Get From, this is not necessary for many applications.

Simplified XML rearranges data within XML so that it can be used by process objects. This rearrangement treats attributes as children, removes namespaces, makes siblings with the same name consecutive, and places text into CDATA sections.

If you select Simplified XML for the Get From option, Process Designer displays a text field. If you do not enter anything in the field, Process Designer processes all of the received XML. If you enter the name of an element, Process Designer begins simplifying at this element, only processing this element and elements nested within it.

If you type a period ("."), this is treated as an empty field. (This is the default.)

Checking for HTTP Errors

When you send a request to a server and it cannot complete the request, it returns an error status code. There are two ways in which a service connector can handle HTTP errors (4xx and 5xx status code in the HTTP response).

1. For each Action defined on the **Actions** tab, you can select Fail on HTTP error, as shown in the illustration below. By default, this option is enabled. If the HTTP request returns an HTTP error, the service connector fails.
2. Specify how to handle the HTTP errors for each Action on the **Output** tab. To implement this option, disable Fail on HTTP error.

Fail on HTTP Error

If you enable Fail on HTTP error and an HTTP error is returned:

- A process faults and you can view the process details in the Application Integration Console.
- A process displays a dialog to the end user with message text showing the HTTP error received.
- The service connector throws faults in the same format a process throws faults:

```
<sf:faultResponse>
  <sf:code>CODE</sf:code>
  <sf:reason>REASON</sf:reason>
  <sf:details>DETAILS</sf:details>
</sf:faultResponse>
```

HTTP error codes follow the format HTTP_NNN, for example, HTTP_404. The reason string contains the HTTP status message.

Note: Any internal or system errors return SERVICE_CONNECTOR_ERROR and one of the following reason strings:

- CATALOG_ERROR
- OTHER_PARAMETERS_ERROR
- AUTHENTICATION_ERROR
- CUSTOM_HEADERS_ERROR
- BINDING_URL_ERROR
- ALTER_REQUEST_AUTHENTICATION_ERROR
- PROCESS_RESPONSE_AUTHENTICATION_ERROR
- OUTPUT_PARAMETERS_ERROR

If you disable Fail on HTTP Error and an HTTP error is returned:

- A process does not fault and the service connector passes the 4xx or 5xx status codes to the process so that you can handle the error in the process.
- A process faults and displays a failure message.

If the target service uses error codes that users might encounter as a matter of course, enable this option to provide process users with meaningful information.

The following image shows the Fail on HTTP error option enabled:

The screenshot shows the 'ConnectorWithInheritance' configuration window. The 'Actions' tab is active, displaying a table of actions. The 'getToken' action is selected, and its configuration is shown in the 'Action' tab below. The 'Fail on HTTP error' checkbox is checked, and the 'Max Redirects' is set to 100. The 'Response Encoding' is set to 'Default from response headers'. The 'Preemptive Auth' checkbox is also checked. The 'Action Type' is set to 'General'.

Action Name	Action Type	Description
getToken	general	
Abstract Request	abstract	
Search	Inherit : Abstract Request	

Action Configuration:

Name*: getToken
Category: Authentication
Fail on HTTP error: ☒
Max Redirects: 100
Response Encoding: Default from response headers
Preemptive Auth: ☒
Description:
Action Type: General

Handle HTTP Errors

To handle HTTP errors for a process, first be sure that you have disabled **Fail on HTTP error**.

You then need to check the HTTP response status code within the process.

To do this, define an "HTTP Response Status Code" as a variable in the **Output** tab. You can also specify a variable using `$ResponseStatusCode` in Expression fields (see the available functions below). In this way, you can pass the status code as part of the service connector's output and handle the response in the process.

These variables, like all variables you create in the **Output** tab, display when you click **Test**.

If you check Fail on HTTP error and an HTTP error status code is returned, the output contains a message.

The screenshot shows the 'Test' tab of the configuration window. The 'Test Server' is set to 'Information Cloud Hosted Agent'. The 'Test' button is clicked, and the result is displayed. The result shows a 500 HTTP status code and a message: 'Failed. Could not initialize class com.activeее.rt.hub.cloud.service.client.rest.invoke.AeICSRestInvokeHandler.' The 'HTTP Status' is 500. The 'URL' is 'https://api.cloudinary.com/v1_1/dti78hbgil/image/upload?signature=29c2a00214c18811fbad2b5b75c1f72kd076c&api_key=475257312787584×tamp=1515352362464'. The 'Output Fields' section shows the 'Response Payload' with a value of 1. The 'REST Request' and 'REST Response' sections are also visible.

Test Results:

Test Server: Information Cloud Hosted Agent
Result: ❌ Failed. Could not initialize class com.activeее.rt.hub.cloud.service.client.rest.invoke.AeICSRestInvokeHandler.
HTTP Status: 500
HTTP responses with 4xx and 5xx status codes will cause fault of the action. If you want to handle such HTTP responses in your action please change the 'Fail on HTTP error' property of this action.
URL: https://api.cloudinary.com/v1_1/dti78hbgil/image/upload?signature=29c2a00214c18811fbad2b5b75c1f72kd076c&api_key=475257312787584×tamp=1515352362464
Output Fields:
Output:
Value:
Response Payload: 1
REST Request:
REST Response:

HTTP Response Header Information Using Expression

You can use one of the following to get details from the response headers:

`$ResponseHeaders[@name = "Content-Type"]/text();`

This constant contains the HTTP response header.

`fn:getResponseHeader( $ResponseHeaders, header_name[, default_value] );`

Returns a response header value. If the header parameter is not defined, this function returns the optional default value.

`fn:responseHeaderExists( $ResponseHeaders, header_name );`

Returns a Boolean value that indicates if a header parameter exists within the response.

`fn:getResponseHeaderNames( $ResponseHeaders );`

Returns a list that contains the names of all the parameters within a response header.

Creating Service Connector Process Objects

On the **Process Objects** tab, you can define one or more process objects for a service connector, to group data and create a structured object. When defined in the service connector, the process objects are available to processes that use the service connector. If, for example, a service is returning demographic information such as name, address, and phone number, you might create a single Demographic process object that contains this information.

You can associate each process object with one or more of the data elements returned by the service using the **Actions/Output** tab.

To create service connector process objects:

1. Create or open an existing service connector.
2. Click the **Process Objects** tab.
3. Click the **+** sign to add a new process object or select an existing process object from the list. Each process object appears as a line item in the tab.

The screenshot shows the 'Process Objects' tab in a service connector configuration interface. At the top, there are tabs for 'Definition', 'Actions', and 'Process Objects', with 'Process Objects' being the active tab. Below the tabs, there is a table with two columns: 'Process Object Name' and 'Description'. The table contains two entries: 'Organization' with the description 'Describes an ICS organization' and 'CustomFunctionConfig' with the description 'Defines attribute config for maplets used in the task.' Each entry has a delete icon (X) to its right. Below the table, there is a 'Properties' section with a 'Name' field containing 'CustomFunctionConfig' and a 'Description' field containing 'Defines attribute config for maplets used in the task.' There is also a 'Fields' section which is currently empty.

4. On the **Object** tab, specify the following for each process object:
 - **Name.** Required. Enter a process object name that identifies the process object. This name appears in the lists where the process object is available for selection.
 - **Description.** Enter an optional description.
5. On the **Fields** tab, specify the following for each process object:
 - **Field Name.** Enter a name for each field in the process object.
 - **Type.** Select the data type of the value being written to the variable using one of the built-in data types such as Text, Integer, or Date Time. If the type is Object List, you also select a Process Object.

Testing the Service Connector

When you use the Service Connector editor, click **Test** to send a request to the service and display this response data in the Test Results tab:

- **Generate Process Objects:** choose to see the process objects you have defined for the service connector (see below for more information).
- **Result** is the status of the test.
- **HTTP Response Status:** the HTTP status code.
- **URL:** the URL where test data was sent. Any query parameters are added to the URL shown here.
- **Output** and **Value:** show data returned from the service that is assigned to the variables defined in the Output tab. If you are using XSLT, it is used to extract the data from the payload.
- **Response Payload:** payload sent to Process Designer from the service (not the full response sent by the service).
- **REST Request:** the data sent to the service from Process Designer.
- **REST Response:** all of the data sent back to Process Designer from the service (whereas the response payload is only a portion of the returned data).

Here, you can see the differences between the payload, request, and response data (for illustration purposes only; the actual test results display only request or response data):

Cloudinary2 Valid Save

Definition Actions Process Objects

▼ Actions (2)

Action Name	Action Type	Description
uploadImage	general	
unsignedUpload	general	

Action Input Binding Output Test

Test Server: Informatica Cloud Hosted Agent Test

Result: ☒ Successful

HTTP Status: 400 HTTP responses with 4xx and 5xx status codes will cause fault of the action. If you want to handle such HTTP responses in your action please change the 'Fail on HTTP error' property of this action.

URL: https://api.cloudinary.com/v1_1/dt78hbgj/image/upload?signature=f552ae51e945012f26208da9614c087489208db72&api_key=4752573127875848×tamp=1515400478551

Output Fields Generate Process Objects

Output	Value
uploadedUrl	

► Response Payload:

► REST Request:

▼ REST Response:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <rest:RESTResponse xmlns:rest="http://schemas.activebpel.org/REST/2007/12/01/aerest.xsd"
3   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
4   status="400">
5   <rest:headers>
6     <rest:header name="Status" value="400 Bad Request"/>
7     <rest:header name="X-UA-Compatible" value="IE=edge,chrome=1"/>
8     <rest:header name="Cache-Control" value="no-cache"/>
9     <rest:header name="Server" value="cloudinary"/>
10    <rest:header name="X-Request-Id" value="ae10e6c620db72"/>
11    <rest:header name="X-Cld-Err" value="Missing required parameter - file"/>
12    <rest:header name="Connection" value="keep-alive"/>
13    <rest:header name="X-XSS-Protection" value="1;mode=block"/>
14    <rest:header name="Content-Length" value="51"/>
15    <rest:header name="Date" value="Mon, 08 Jan 2018 08:34:39 GMT"/>
16    <rest:header name="Content-Type" value="application/json; charset=utf-8"/>
17  </rest:headers>
18  <rest:payload contentType="application/json">{"error":{"message":"Missing required parameter - file"}}</rest:payload>
19 </rest:RESTResponse>
```

Note: If the response includes an attachment, you also have an option to **Download Attachment**.

Generate Process Objects

A process object is usually tied to a specific set of elements. You may sometimes have a large number of identical objects, each of which applies to one set of elements. For example, when you define the data being returned, you create a process object whose sole purpose is to define a subset of returned data. These

objects are *non-reusable* and can only be used by a single process object field. The object names are also the name of a field.

You can also create objects that are *reusable*. For example, the refType element in NetSuite has two fields: a name and an internalID. Instead of creating many objects, each of which contains these two fields, you can create a single, reusable process object.

1. Click **Generate Process Objects** to show the Process Objects you have defined for the service connector.
2. Select the process objects you want to make reusable and then press **Next**.
Process Designer displays a list of generated process objects.
3. Click **Finish**.

Service Connector Timings

You can see the HTTP Response Parsing time, the HTTP Execution Time, and the Redirection Count in the tab of a service connector.

You can see the following HTTP headers in the **Rest Response** section of the tab:

HTTP Response Parsing Time

The time that a service connector takes, in milliseconds, to parse a response from the URL you entered in the **Binding** tab. The HTTP Response Parsing time is usually zero for small requests. For large requests, especially with requests with attachments, the HTTP Response Parsing time increases.

See the HTTP header `X-AE-HTTP-RESPONSE-PARSING-TIME-IN-MILLIS` for the HTTP Response Parsing Time.

Redirection Count

The number of redirects, if any, made by the URL that you entered in the **Binding** tab.

For example, if a service request redirects to a service that in turn redirects to another service, the Redirection Count is two. If a service request goes through with no redirects, the Redirection Count is zero.

See the HTTP header `X-AE-REDIRECTION-COUNT` for the Redirection Count.

Note: If a GET HTTP request redirects to another GET HTTP request, the Redirection Count remains zero. This is a limitation.

HTTP Execution Time

The response time, in milliseconds, of the URL that you entered in the **Binding** tab. The HTTP Execution Time does not include the response parsing time, or the time taken to by the service connector to perform other tasks.

See the HTTP header `X-AE-HTTP-EXECUTION-TIME-IN-MILLIS` for the HTTP Execution Time.

The following image shows the `X-AE-HTTP-RESPONSE-PARSING-TIME-IN-MILLIS`, `X-AE-REDIRECTION-COUNT`, and `X-AE-HTTP-EXECUTION-TIME-IN-MILLIS` HTTP headers:

Action Input Binding Output **Test**

Test Server: Informatica Cloud Hosted Agent Test

Result: Successful

HTTP Status: 400 HTTP responses with 4xx and 5xx status codes will cause fault of the action. If you want to handle such HTTP responses in your action please change the 'Fail on HTTP error' property of this action.

URL:

Output Fields Generate Process Objects

Output	Value
uploadedUrl	
▶ Response Payload:	
▶ REST Request:	
▼ REST Response:	<pre> 1 <rest:RESTResponse xmlns:rest="http://schemas.activelink.org/REST/2007/12/01/aeREST.xsd" 2 statusCode="200"> 3 <rest:headers> 4 <rest:header name="X-Frame-Options" value="DENY"/> 5 <rest:header name="X-AE-HTTP-RESPONSE-PARSING-TIME-IN-MILLIS" value="0"/> 6 <rest:header name="Server" value="Apache-Coyote/1.1"/> 7 <rest:header name="Cache-Control" 8 value="no-cache,no-store,max-age=0,must-revalidate"/> 9 <rest:header name="X-Content-Type-Options" value="nosniff"/> 10 <rest:header name="Pragma" value="no-cache"/> 11 <rest:header name="Expires" value="0"/> 12 <rest:header name="X-XSS-Protection" value="1;mode=block"/> 13 <rest:header name="Content-Length" value="0"/> 14 <rest:header name="X-AE-REDIRECTION-COUNT" value="2"/> 15 <rest:header name="Date" value="Tue, 21 Feb 2017 19:38:43 GMT"/> 16 <rest:header name="X-AE-HTTP-EXECUTION-TIME-IN-MILLIS" value="42"/> 17 </rest:headers> 18 </rest:RESTResponse> </pre>

Option 2: Generating a Service Connector by Importing a File

You can import a Web Service Definition Language (WSDL) or a Swagger JSON file to create a Cloud Application Integration service connector.

Creating a Service Connector from a WSDL File

To create service connector from a WSDL file, perform the following steps:

1. In Application Integration, click **New**.
2. In the **New Asset** dialog box, click **Service Connector Patterns**.
3. Select **Create from WSDL** and then click **Create**.
4. In the **WSDL Source** step, provide a name, location of the project or folder, and description.
5. Choose a WSDL file to import. You can use either of the following options in **WSDL File**:
 - To use a WSDL file on your local system, select **File** and select the file.
You can also select a .zip file that contains a self contained WSDL file, or a WSDL file along with an XML schema file.
 - To use a URL that contains a WSDL file, select **URL**.

The following image shows the WSDL Source tab:

The screenshot shows a wizard window titled "New Service Connector from WSDL". It has three tabs: "1 WSDL Source", "2 Service and Operations", and "3 Summary". The "WSDL Source" tab is active. Below the tabs, it says "Provide identifying information for service Connector and WSDL file." The form contains the following fields:

- Name:** A text box containing "ServiceConnector".
- Location:** A text box containing "Dimple" and a "Browse" button.
- Description:** A large empty text area.
- WSDL File:** A section header.
- WSDL source:** Radio buttons for "File" (selected) and "URL".
- Select File...:** A file selection box showing "HelloWorldPort.wsdl" and a "Choose File..." button.

At the bottom of the wizard are four buttons: "< Back", "Next >" (highlighted in blue), "Finish", and "Cancel".

6. Click **Next**.

The following image shows the **Services and Operations** tab:

The screenshot shows a dialog titled "New Service Connector from WSDL" with three tabs: "1 WSDL Source", "2 Service and Operations" (selected), and "3 Summary". The "Service and Operations" tab contains the following fields and sections:

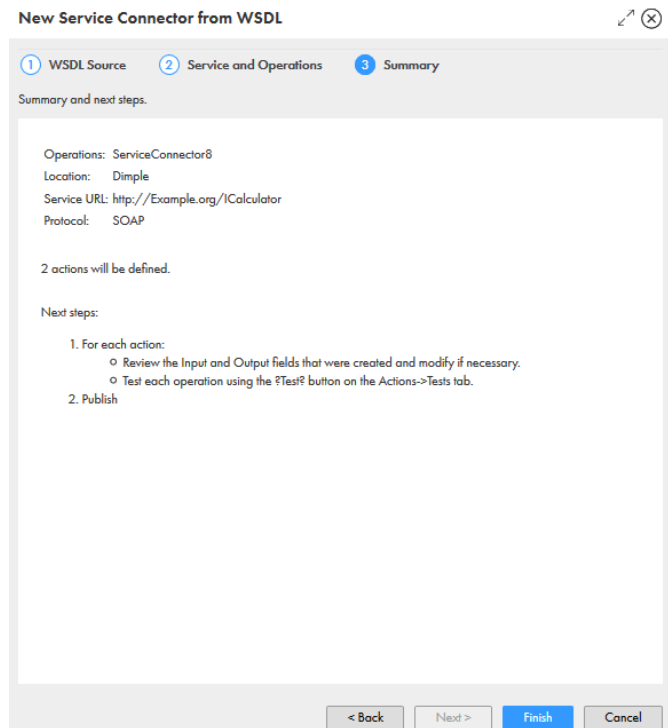
- Service Name:** A dropdown menu with "CalculatorService" selected.
- Service Port:** A dropdown menu with "ICalculator" selected.
- Service URL:** A text field containing "http://Example.org/ICalculator".
- Operations (2):** A table with two columns: a checkbox and a text field. The table contains three rows:

☒	Operation
☒	Add
☒	Subtract

At the bottom of the dialog are four buttons: "< Back", "Next >" (highlighted in blue), "Finish", and "Cancel".

7. In the **Service Name** field, select the service name of the WSDL file you imported from the list.
You see that the URL where you want the service connector to send requests to is present in the **Service URL** field. You do not need to edit this URL.
8. In the **Service Port** field, select the port that you want the service connector to use.
9. In the **Operations** section, select the operations that you want the service connector to provide.
The operations that you select will be available to you as actions when you create a process and use this service connector.
10. Click **Next**.
11. In the **Summary** step, review the service connector configurations.

The following image shows the Summary step:



12. Click **Finish**.

WSDL Files with Qualified Elements

If you have a WSDL file with user defined data types, you can add namespace prefixes and type names to create a WSDL file with qualified elements. You can use this WSDL file to create a service connector.

You can create a service connector from a WSDL file that contains user defined data types. Add namespace prefixes and type names to the user defined data to create a WSDL file with qualified elements. You can use this WSDL file to generate a service connector. The namespace prefix does not appear in the process object and you see a correct payload.

For example, consider a WSDL file that has the user defined elements RecordId and AddressComplete.

First, define a schema with the type name 'Address'. The following is a sample schema with a complexType name Address:

```
xs:complexType name="Address">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="1" name="RecordId" type="xs:string" />
    <xs:element minOccurs="0" maxOccurs="1" name="AddressComplete"
type="xs:string" />
  </xs:sequence>
</xs:complexType>
<s:element minOccurs="0" maxOccurs="1" name="Address" type="tns:Address" />
```

Next, in the WSDL file, add the namespace prefix 'tns', as in the following sample:

```
<wsdl:message name="InputRequest">
  <wsdl:part name="parameters" element="tns:Address" />
</wsdl:message>
<wsdl:portType name="AddressValidationSoap">
  <wsdl:operation name="Process">
    <wsdl:input message="tns:InputRequest" />
    ...
```

```

    </wsdl:operation>
  </wsdl:portType>

```

You can use this WSDL file to create a service connector. You will get an Address process object with two fields, RecordId and AddressComplete.

WSDL Files with Repeating Elements

You can create a service connector from a WSDL file with nested and repeating elements.

For example, consider the following WSDL and WSDL schema files with the nested, repeating elements 'shelf' and 'book':

```

1  <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
2  <wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3      xmlns:libSchema="http://www.library.com/library.xsd"
4      targetNamespace="http://www.library.com/library.wsdl"
5      xmlns:lib="http://www.library.com/library.wsdl"
6      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
7      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
8      name="library">
9
10     <wsdl:types>
11         <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
12             <xsd:import
13                 namespace="http://www.library.com/library.xsd"
14                 schemaLocation="./schema/library.xsd">
15             </xsd:import>
16         </xsd:schema>
17     </wsdl:types>
18
19     <wsdl:message name="request">
20         <wsdl:part name="request" element="libSchema:library" />
21     </wsdl:message>
22
23     <wsdl:message name="response">
24         <wsdl:part name="response" element="libSchema:bookCount" />
25     </wsdl:message>
26
27     <wsdl:portType name="Library">
28         <wsdl:operation name="countBooks">
29             <wsdl:input message="lib:request" />
30             <wsdl:output message="lib:response"/></wsdl:output>
31         </wsdl:operation>
32     </wsdl:portType>
33 </wsdl:definitions>
34

```

```

1  <?xml version="1.0"?>
2  <xs:schema elementFormDefault="qualified"
3    targetNamespace="http://www.library.com/library.xsd"
4    xmlns:xs="http://www.w3.org/2001/XMLSchema"
5    xmlns:ns="http://www.library.com/library.xsd">
6
7    <xs:element name="book">
8      <xs:complexType>
9        <xs:attribute name="name" type="xs:string" use="required" />
10     </xs:complexType>
11   </xs:element>
12   <xs:element name="shelf">
13     <xs:complexType>
14       <xs:sequence>
15         <xs:element ref="ns:book" minOccurs="0" maxOccurs="100" />
16       </xs:sequence>
17       <xs:attribute name="name" type="xs:string" use="required" />
18     </xs:complexType>
19   </xs:element>
20   <xs:element name="library">
21     <xs:complexType>
22       <xs:sequence>
23         <xs:element ref="ns:shelf" minOccurs="0" maxOccurs="20" />
24       </xs:sequence>
25       <xs:attribute name="name" type="xs:string" use="required" />
26     </xs:complexType>
27   </xs:element>
28
29   <xs:element name="bookCount" type="xs:integer" />
30
31 </xs:schema>

```

You can use this WSDL and WSDL schema file to create a service connector using the WSDL to Service Connector tool. When you import the service connector, you see three process objects: Library, Shelf, and Book. When you select a parent object, you also see the nested object in the right side of the bottom panel.

elementFormDefault

You can use WSDL files that contain the elementFormDefault attribute in the WSDL schema.

The WSDL file can contain elementFormDefault="qualified" or elementFormDefault="unqualified".

Choice Elements

You can use choice schema elements to create a payload in a WSDL file.

Use the If element to show what should appear in the request.

The WSDL to Service Connector considers top level choice elements. This means that if a WSDL schema file has nested choice elements, the WSDL to Service Connector takes the first choice element.

Creating a Service Connector from a Swagger JSON File

You can use the Service Connector Generator tool to create a service connector from a Swagger 2.0 JSON file.

1. In Application Integration, click **New**.
2. In the **New Asset** dialog box, click **Service Connector Patterns**.

3. Select **Create from Swagger** and then click **Create**.
4. In the **Swagger Source** step, provide a name, location of the project or folder, and description.
5. Choose a swagger file to import. You can use either of the following options of **Swagger Source**:
 - To use a Swagger JSON file on your local system, select **File**.
 - To use a URL that contains a Swagger JSON file, select **URL**.

The following image shows the Swagger Source tab:

The screenshot shows a dialog box titled "New Service Connector from Swagger" with a close button (X) in the top right corner. The dialog has three tabs: "1 Swagger Source" (active), "2 Service and Operations", and "3 Summary". Below the tabs, it says "Provide identifying information for service Connector and Swagger file." The form contains the following fields and controls:

- Name:** A text input field containing "ServiceConnector3".
- Location:** A dropdown menu set to "Default" and a "Browse" button.
- Description:** A large text area.
- Swagger File:** A section header.
- Swagger source:** Two radio buttons, "File" (selected) and "URL".
- Select File...:** A file selection input field showing "mathSwagger.json" and a "Choose File..." button.

At the bottom of the dialog, there are four buttons: "< Back", "Next >" (highlighted in blue), "Finish", and "Cancel".

6. Click **Next**.

The following image shows the **Services and Operations** tab:

New Service Connector from Swagger ✓ ? ✕

1 Swagger Source 2 Service and Operations 3 Summary

Service Name: Swagger ICalculator

Service URL: https://SERVICE_HOST/SERVICE_BASE

Operations (3)

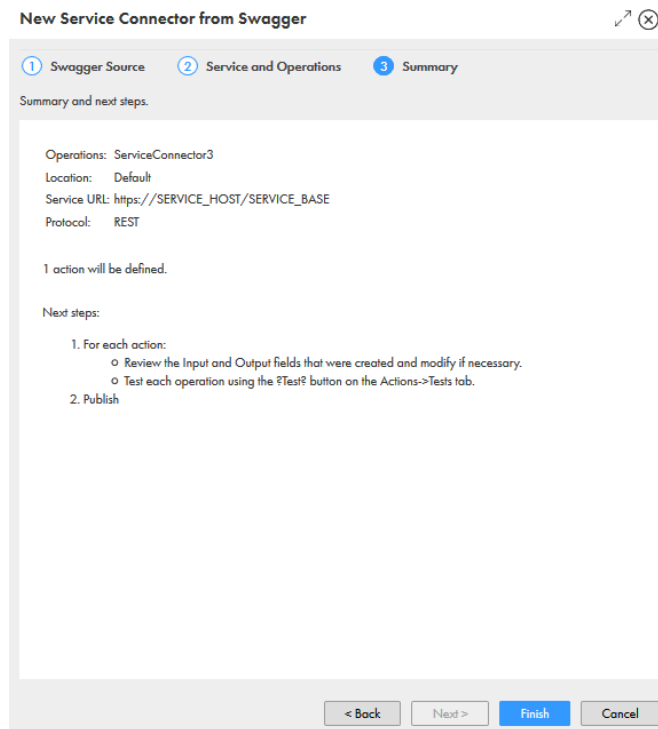
Operation	Description
☒ root	
☒ /	
☒ POST	Adds two numbers

< >

< Back Next > Finish Cancel

7. Optionally, to remove an operation from the service connector, unselect that operation.
The operations here are available to you as actions when you create a process and use this service connector.
8. Click **Next**.
9. In the **Summary** step, verify the configurations.

The following image shows the configurations in the **Summary** step:



10. Click **Finish**.

HTTP Operations

You can import a Swagger file that contains the DELETE, PUT, POST, and GET HTTP operations.

The service connector that you generate has actions corresponding to the HTTP operation in the Swagger file. If the Swagger file has multiple HTTP operations, the service connector will have multiple actions.

For example, if you use a swagger file with a POST HTTP operation, you get a service connector with a POST action.

The following image shows a single POST HTTP operation within a Swagger file:

```

- paths: {
  - /pet: {
    - post: {
      - tags: [
        "pet"
      ],
      summary: "Add a new pet to the store",
      description: "",
      operationId: "addPet",
      - consumes: [
        "application/json",
        "application/xml"
      ],
      - produces: [
        "application/xml",
        "application/json"
      ],
      - parameters: [
        - {
          in: "body",
          name: "body",
          description: "Pet object that needs to be added to the store",
          required: true,
          - schema: {
            $ref: "#/definitions/Pet"
          }
        }
      ],
      - responses: {
        - 405: {
          description: "Invalid input"
        }
      },
      - security: [
        - {
          - petstore_auth: [
            "write:pets",
            "read:pets"
          ]
        }
      ]
    }
  }
}

```

The following image shows the action in service connector that you get when you use the preceding Swagger file:

The screenshot shows the 'Actions' tab of a service connector interface. It contains a table of actions and a configuration panel for the selected 'addPet' action.

Action Name	Action Type	Description
addPet	general	Add a new pet to the store
updatePet	general	Update an existing pet
getUserByName	general	Get user by user name
deleteUser	general	Delete user. This can only be done by the logged in user.
updateUser	general	Updated user. This can only be done by the logged in user.

Below the table, the configuration for the 'addPet' action is shown. The 'Binding' tab is active, displaying the following settings:

- URL*: `http://($hostName)/v2/pet`
- Verb: `POST`
- Multi Using: `Semicolon separated`
- Authentication Type: `Custom`

There are expandable sections for 'Other Parameters' and 'HTTP Headers'.

You see that the Add Pet action corresponds to the verb POST.

JSON Payload

You can import a Swagger file with different types of JSON payloads. The Service Connector you generate reflects the JSON payload of the Swagger file.

The following table shows the JSON payload features that you can use, a sample Swagger excerpt, and the Input field of the corresponding service connector:

JSON Payload Feature	Sample Swagger Content	Service Connector Input Field
Lists of Objects	<pre>schema: { type: "array", - items: { \$ref: "#/definitions/User" } }</pre>	
References	<pre>schema: { \$ref: "#/definitions/Pet" }</pre>	
Single Select Enums	<pre>items: { type: "string", - enum: ["available", "pending", "sold"], default: "available" },</pre>	
Simple types	<pre>{ name: "orderId", in: "path", description: "ID of pet that needs to be fetched", required: true, type: "integer", maximum: 10, minimum: 1, format: "int64" }</pre>	
Attachments	<pre>- { name: "file", in: "formData", description: "file to upload", required: false, type: "file" }</pre>	
Lists of simple types	<pre>- parameters: [- { name: "tags", in: "query", description: "Tags to filter by", required: true, type: "array", - items: { type: "string" }, collectionFormat: "multi" }],</pre>	

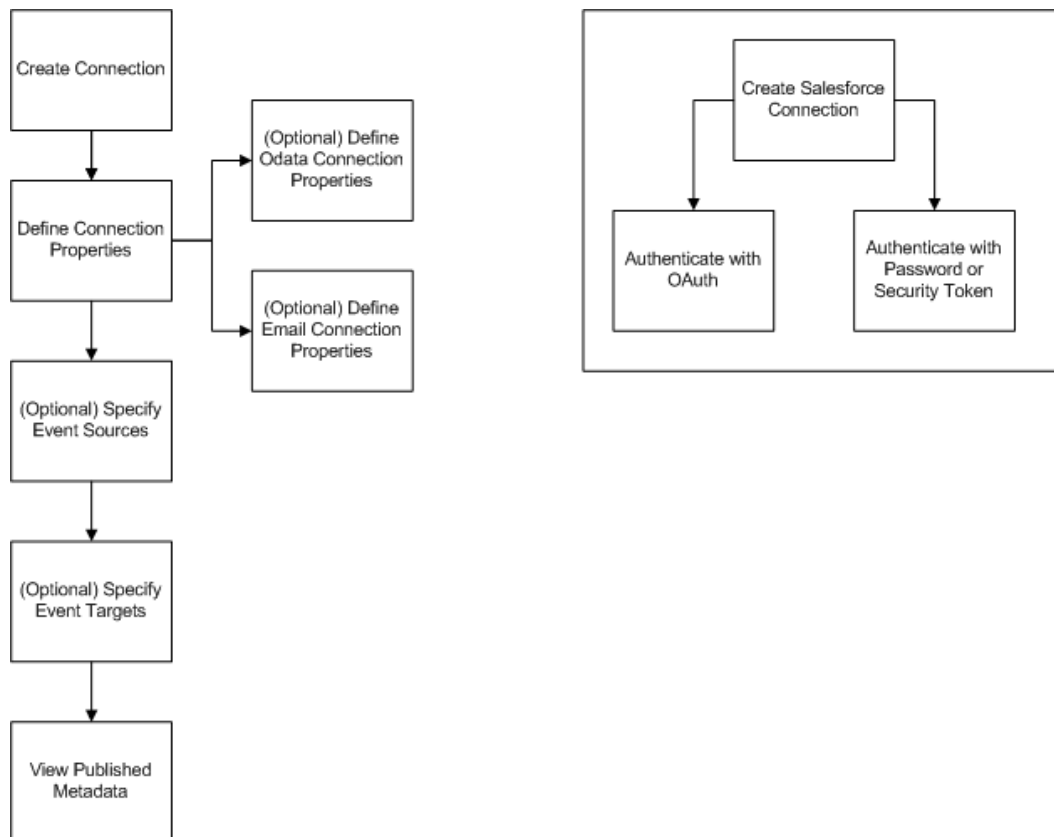
JSON Payload Feature	Sample Swagger Content	Service Connector Input Field						
Multi Select Enums	<pre>parameters: [- { name: "status", in: "query", description: "Status values that need to be considered for filter", required: true, type: "string", items: { type: "string", enum: ["available", "pending", "sold"], default: "available" }, collectionFormat: "multi" },]</pre>	<table><thead><tr><th>Name*</th><th>Label</th><th>Type</th></tr></thead><tbody><tr><td>status</td><td>status</td><td>Multi-Select Picklist ▼ Options: available, pending, sold</td></tr></tbody></table>	Name*	Label	Type	status	status	Multi-Select Picklist ▼ Options: available, pending, sold
Name*	Label	Type						
status	status	Multi-Select Picklist ▼ Options: available, pending, sold						
Objects with References to Other Object Lists	<pre>tags: { type: "array", - xml: { name: "tag", wrapped: true }, - items: { \$ref: "#/definitions/Tag" }, },</pre>	<table><thead><tr><th>Name*</th><th>Label</th><th>Type</th></tr></thead><tbody><tr><td>tags</td><td>tags</td><td>Object List ▼ Process Object string</td></tr></tbody></table>	Name*	Label	Type	tags	tags	Object List ▼ Process Object string
Name*	Label	Type						
tags	tags	Object List ▼ Process Object string						

CHAPTER 7

Using Connectors

After you create a service connector, create a connection to use in a process.

The following image shows the steps you follow to create a connection:

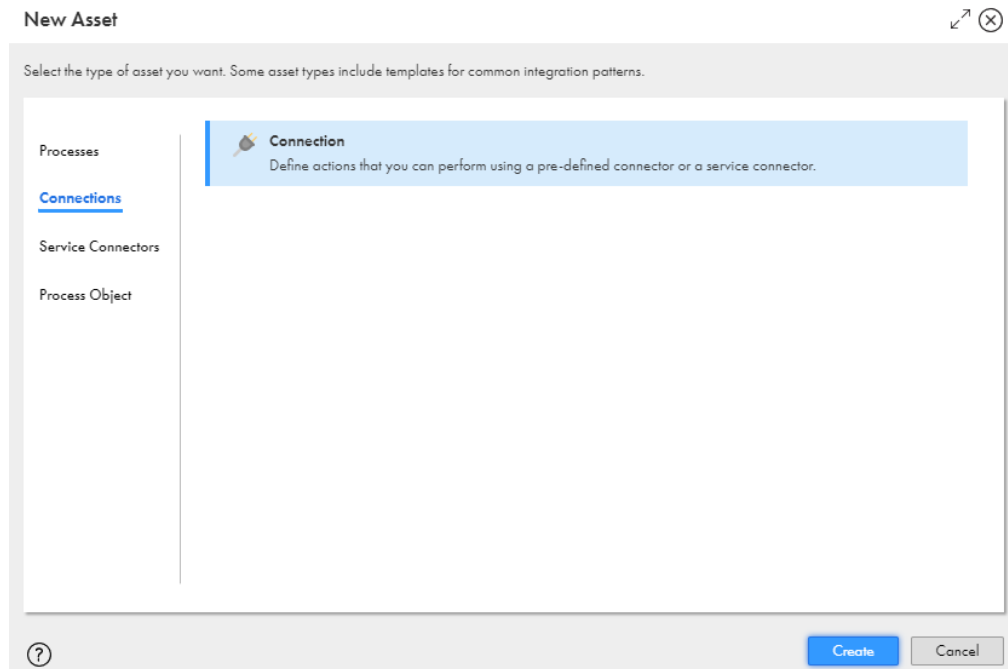


Creating a Connection

Perform the following steps to create a connection:

1. In Application Integration, click **New**.

2. In the **New Asset** dialog box, click **Connections > Connection**, and then click **Create**.



Defining Connection Properties

For each connection, you enter descriptive properties, identify the scope, and set authentication parameters, if required.

These options display for all Connections:

- **Connection Type:** From this list, select a published service connector, native toolkit connector (such as JDBC), or web service connector (such as Salesforce). The Connection Type you select determines which Connection Properties display.
Note: After you publish a connection, you cannot change its Connection Type.
- **Name:** Enter a name that identifies the connection within processes and in the Design. This name should contain only letters, numbers, and hyphens.
Important: If you are defining a connection to be used for publishing processes in Salesforce, the Connection Name must be *Salesforce*, to ensure that the processes is accessible to users in Salesforce. Only one Salesforce connection is permitted for each Application Integration organization.
- **Description:** Enter a description to identify the connection.
- **Tags:** You can enter one or more custom tag names to organize your objects into different groups. Separate multiple tags with a comma.
- **Run on:** Select the agent where you want to access a service behind your firewall or *Cloud Server or any agent*. This list displays the agents available for your organization.
- **Connection Test:** Displays the results of any connection test run on the Cloud Server or the agent named in the Run on field.
- **OData Enabled:** Choose **Yes** to utilize the OData protocol to access data services from the Cloud that exist on-premise such as those available via a JDBC or an SAP Table Reader connection.
Note: If you publish an OData enabled connection to a Secure Agent, the OData URL on the **Connection** page is an Informatica Cloud URL. You do not see a Secure Agent URL.

To construct the Secure Agent URL, replace all text `odata/v4/Oracle` in the Informatica Cloud URL with `https://<host>:<port>/process-engine/`

For example, if the Informatica Cloud URL is `https://na1.ai.dm-us.informaticacloud.com/active-bpel/odata/v4/Oracle`, the Secure Agent URL is `https://localhost:7443/process-engine/odata/v4/Oracle`

- **Allowed Roles for OData:**

The roles that have access to this connection at runtime. Users assigned these roles have access to an OData service URL. You can enter a custom role or a system-defined role. You can enter more than one role in this field.

IMPORTANT: If you enter a custom role in the **Allowed Roles for OData** field, a user invoking the OData service URL must have, or belong to groups that have, the custom role and the and Service Consumer role.

Example:

Your organization has a custom role, Testers, with the Run privilege enabled for Application Integration. You create a process and enter 'Testers' in the Allowed Roles field.

For a user to be able to use the OData service URL, the following conditions need to be met:

- Condition 1: The user has the Testers role, a role that this connection requires.
- Condition 2: The user has the Service Consumer role. Application Integration requires this role to invoke any service URL.

A user with the Testers role and the Service Consumer role will be able to invoke the OData service URL. This is because *both*, Condition 1 and Condition 2, are satisfied.

A user with *only* the Testers role will be unable to invoke the OData service URL. This is because Condition 1 is satisfied but Condition 2 is not satisfied.

A user with *only* the Service Consumer role will also be unable to invoke this OData service URL. This is because Condition 2 is satisfied but Condition 1 is not satisfied.

The remaining properties are determined by the connector (the Connection Type).

For example, the Salesforce Connection Type in the following figure requires a User Name, Password, Security Token, and Service URL:

The screenshot shows the 'Properties' tab of the 'SfoAuthConn' connection in Informatica Cloud. The 'Connection Details' section includes fields for Name, Location, Description, Type (Salesforce), Run On (Cloud Server or any Secure Agent), Connection Test, OData-Enabled (No), and Allowed Roles for OData. The 'Authentication' section shows Authentication Type set to OAuth. Below these are several rows for OAuth-specific properties:

Name	Value	Description
Authorization URL	<code>https://login.salesforce.com/services/oauth2/authorize</code>	Enter the Salesforce OAuth authorization URL. Default value is <code>https://login.salesforce.com/services/oauth2/authorize</code> . For Sandbox, use <code>https://test.salesforce.com/services/oauth2/authorize</code> .
Token Request URL	<code>https://login.salesforce.com/services/oauth2/token</code>	Enter the OAuth token request URL. For production, use <code>https://login.salesforce.com/services/oauth2/token</code> . For sandbox, use <code>https://test.salesforce.com/services/oauth2/token</code> .
Session Duration	60	Enter the number of minutes to wait before refreshing the session. Default: 60 minutes.
Authorization Status	Authorized The last update was by shwar on 2016-03-05 17:34.	Indicates the current status and the last time that authorization was completed.
Authorize Access	<button>Authorize</button>	Click to initiate the authorization workflow using OAuth.

Specifying Event Sources

Event Sources are available for some connections based on the service connector.

For the following AMQP connection, providing access to the AMQP listener service, you can add one or more event sources:

The screenshot shows the 'Event Sources' tab for 'Connection3'. It features a table for defining event sources. The first entry is an 'AMQP Source' with the following details:

Name	Value	Description
Destination Type:	queue	Type of destination, either 'queue' or 'topic'.
Destination:		The name of the queue or topic.
Payload Format:	Xml	Type of message payload expected on the destination, such as Xml, Json, Text or Binary.
Objectlist Field Name:	object	When the Payload Format is Json and the queue/topic accepts a JSON message which is an array of objects this value will be used as the field name of the object list for the AMQP Message body.
Dead Letter Queue Name:		Name of a queue that will be used to route message that encounter a failure during processing.
Other Attributes:		Advanced attributes. Please contact technical support.

First, click **Add Event Source** to add a new event source for this connection. Then enter the following information for each Event:

- **Name:** Enter a unique name for each Event Source. The name can contain letters, digits, a period (.), dash (-) or underscore (_).
- **Description:** Enter a description, if needed, for the event source.
- **Enabled:** Select No to disable this event source until you are ready to deploy this event source. This gives you the flexibility to configure multiple event sources and determine when to enable them.

The **Properties** vary based on the connection type. In this case, AMQP requires Destination Type, Destination, and Payload Format.

A message displays on the tab to notify you if the preview option is not enabled or supported for the connection type.

Specifying Event Targets

Similar to Event Sources, you can specify one or more Event Targets for each connection on a separate tab.

Each connector might define different types of event targets. For example, using the File connector, you can choose either File Writer or Delimited Content Writer File Writer for each event target you define. Each event target includes a set of properties unique to that event target type.

To create a new event target:

1. Click **Add Event Target** (and choose the target type, if applicable). This adds a new row.
2. Highlight the row and add an event **Name** and **Description**.
3. Edit the **Properties** for the specific connector.

Connection3 Save Test ⋮ ✕

Properties Event Sources **Event Targets** Metadata

Add Event Target

▼ **AMQP Target : AMQPTarget** ✕

Event:

Name:

Description:

Properties:

Name	Value	Description
Destination Type:	queue	Type of destination, either 'queue' or 'topic'.
Destination:	local-queue	The name of the queue or topic.
Payload Format:	Xml	Type of message payload expected on the destination, such as Xml, Json, Text or Binary.
Other Attributes:		Advanced attributes. Please contact technical support.

Viewing Published Metadata

After you publish the connection, you can see the objects associated with the connection.

In the following image, you see the published metadata of a service connector:

FileConn Valid Save Test ⋮ ✕

Properties Event Sources Event Targets **Metadata**

Connection published and synchronized on: 12/15/2017 10:56:51 PM. Preview data is not supported or enabled for this connection.

▼ **Actions**

Search:

Action Name	Description
FileWriter	This event target can be used to write some output to a file.

▼ **Objects**

Search:

Name	Label	Type	Description
▼ DelimitedContent	Delimited content		Delimited file parsing results.
fileInfo	File information	FileConn:FileInformation	
header	Record field headers	FileConn:Header[]	
totalRowCount	Total rows count	int	
record	Records	FileConn:Record[]	
▼ Field	Record Field		Record field.
value	Value	string	
▶ FileInformation	File Information		File information object.
▶ FileWriteTask	File write task		File write request object.

Note: For some connection types, published metadata is not available.

CHAPTER 8

System Services, Listeners and Connectors

This chapter includes the following topics:

- [Using the OData Provider, 147](#)
- [Using Salesforce System Services Listeners and Connectors, 152](#)
- [Using the Email Service, 157](#)
- [Using Java Message Service \(JMS\) , 159](#)
- [Using the Shell Command Invoke Service Connector, 160](#)

Using the OData Provider

OData is a REST-based and standardized protocol that provides access to data over the Web. You can use the OData protocol to access data services from the Cloud, including internal data sources like those available with a JDBC connection.

OASIS has standardized on [OData V4](#). Informatica recommends that you use version to use OData V4, and it is the default version.

Your organization can expose OData feeds on an endpoint such as:

`https://[host].rt.informaticacloud.com/active-bpel/odata/[version]/[connection name]/[data source name]`

For example, these two endpoints expose the *sampleparts* table:

- **OData Version 2:** `https://[host].rt.informaticacloud.com/active-bpel/odata/v2/Parts/sampleparts`

~or~

- **OData Version 4:** `https://[host].rt.informaticacloud.com/active-bpel/odata/v4/Parts/sampleparts`

Data is available in Atom XML format or JSON, including support for the XML content-type.

When the OData schema is generated, note the following:

- The IID fields are excluded.
- The key fields are added to the OData Key definition.
- The schema uses the same type as the native data type for each field.

You can enable OData in connections configured to run on Secure Agents. The Secure Agent, rather than opening a port, opens up an outbound connection to the Informatica Cloud servers through which all communication occurs. The Secure Agent then has access to any on-premises applications or data sources.

For the best practices on using the recommended version of OData, OData Version 4, along with a JDBC connector like Salesforce Connect, for example, see [here](#).

Note: If you publish an OData enabled connection to a Secure Agent, the OData URL on the **Connection** page is an Informatica Cloud URL.

You do not see a Secure Agent URL. To construct the Secure Agent URL, replace all text `odata/v4/Oracle` in the Informatica Cloud URL with `https://<host>:<port>/process-engine/`.

For example, if the Informatica Cloud URL is `https://ps1w2.rt.informaticacloud.com/active-bpel/odata/v4/Oracle`, the Secure Agent URL is `https://localhost:7443/process-engine/odata/v4/Oracle`.

Supported OData V4 and OData V2 URI Conventions

You can use many OData V4 and OData V2 URI conventions to access data.

Supported OData Version 4 Conventions

You can use the following OData V4 URI conventions to access data:

- Section 2. URL Components
- Section 3 Service Root URL
- Section 4.1 Addressing the Model for a Service
- Section 4.3 Addressing Entities
 - You can use only Canonical URLs
- Section 4.4 Addressing References between Entities
- Section 4.6 Addressing a Property
- Section 4.7 Addressing a Property Value
- Section 5 Query Options
- Section 5.1 System Query Options
- Section 5.1.1 System Query Option \$filter
 - Section 5.1.1.1 Logical Operators
 - Section 5.1.1.1.1 Equals
 - Section 5.1.1.1.2 Not Equals
 - Section 5.1.1.1.3 Greater Than
 - Section 5.1.1.1.4 Greater Than or Equal
 - Section 5.1.1.1.5 Less Than
 - Section 5.1.1.1.6 Less Than or Equal
 - Section 5.1.1.1.7 And
 - Section 5.1.1.1.8 Or
- Section 5.1.1.4 Canonical Functions
 - Section 5.1.1.4.1 contains
 - Section 5.1.1.4.2 endswith

- Section 5.1.1.4.3 startswith
- Section 5.1.1.6 Literals
- Section 5.1.1.6.1 Primitive Literals
- Section 5.1.3 System Query Option \$select
- Section 5.1.4 System Query Option \$orderby
 - You can use \$orderby only with a JDBC connector.
- Section 5.1.5 System Query Options \$top and \$skip
 - You can use \$skip only with a JDBC connector.
- Section 5.1.6 System Query Option \$count
- Section 5.1.7 System Query Option \$search.
 - You can do simple searches like `http://host/service/Products?$search=blue`
 - You cannot use \$search and \$filter at the same time
- 5.1.8 System Query Option \$format

See [here](#) for the complete OData V4 specification for URI conventions.

Supported OData Version 2 Conventions

Informatica recommends that you use OData V4. However, you can use the following OData V2 URI conventions to access data:

- Section 1. URI Components
- Section 2. Service Root URI
- Section 3. Resource Path
- Section 3.1. Addressing Entries
 - You cannot use relationships and complex entities
- Section 3.3. Addressing Service Operations
- Section 4. Query String Options
- Section 4.1. System Query Options
- Section 4.3. Top System Query Option (\$top)
- Section 4.5. Filter System Query Option (\$filter)
 - With the \$filter option, you can use only 'endswith' and 'startswith'
- Section 4.7. Format System Query Option (\$format)
- Section 4.8. Select System Query Option (\$select)
- Section 4.9. Inlinecount System Query Option (\$inlinecount)

See [here](#) for the complete list of OData V2 specification for URI conventions.

Custom Composite Keys

When you use OData to access a JDBC connector, you can define and edit custom composite keys. You define a custom composite key when a database entity does not have a primary key. With this custom composite key that you create, you can perform read operations on the JDBC database.

To use custom composite keys, you must use OData V4.

Primary Keys and Custom Composite Keys

A primary key is an object field (a column in a database table or a database view) or a set of object fields (columns in a database) that can uniquely identify each record in a database table or database view. You use primary keys to query objects in the database.

If an entity does not have a primary key, you can define a custom composite key for the entity. To create a custom composite key, manually select a field or multiple fields for that key. The custom composite key takes the place of the primary key, and you can use it to read data from a JDBC database.

Example: You connect to a JDBC database of employee records and you see that the entity *empaccount* does not have a primary key. You can manually select the fields *email*, *ID*, and *name* to define a custom composite key for *empaccount*.

You can also edit custom composite keys that you created. If you defined a custom composite key that references an entity that no longer exists, you can remove the key. If a custom key contains fields that no longer exist, you can remove these fields.

Viewing the Custom OData Entity Keys Section

1. Enable OData and publish a JDBC connection.
2. Click the **Published Metadata** tab.

You see a new section, Custom ODataEntity Keys, with the following information:

- Error messages, if any, with information about entities without a primary key and custom keys assigned to entities that no longer exist.
- A list of entities and their corresponding key fields.

3. Click **Edit Custom Keys**.

The Custom OData Entity Keys dialog box opens.

The following image shows the Custom OData Entity Keys dialog box:

Custom OData Entity Keys

Search:

Entity: booknopk Search:

Is Key	Name	Label	Type
☐	Author	Author	string
☐	Price	Price	int
☐	Size	Size	string
☒	Title	Title	string
☒	id	id	int

Key fields: Title, id

Apply Cancel

On the left side, you see three types of entities:

- Entities that do not have a primary key.
- Entities that do not exist but still have a custom key mapped to them.
- Entities that have a primary key defined, and do not require a custom composite key.

On the right side, you see a list of fields that you can map to create a custom composite key.

Creating OData Custom Composite Keys

To create OData custom composite keys, you select entities that you want to use as primary keys.

1. Click **Edit Custom Keys**.
2. Select an entity from the list of entities.
3. Select the fields that you want to use as a custom key. You can select multiple fields.

Important: Ensure that you analyze the data and select fields that uniquely identify each record in the database table or view. For example, if you want to use the fields **FirstName** and **LastName** to create a custom composite key, you cannot have two employees with the same first and last names.

4. Click **Apply**.

You see the custom composite key that you created in the list under Custom ODataEntity Keys.

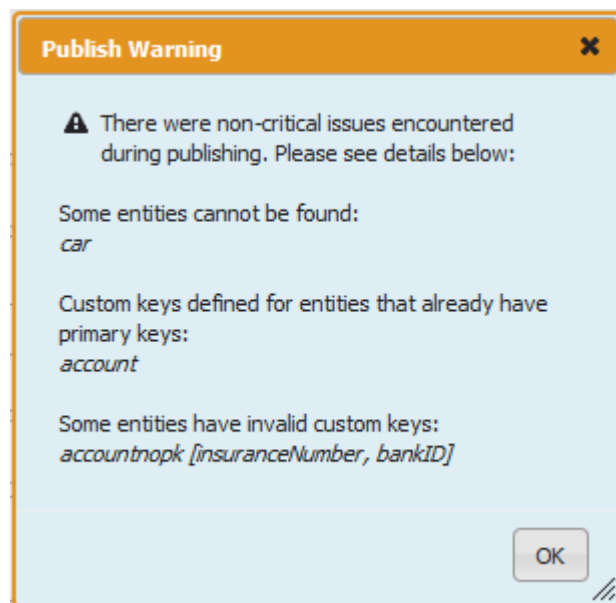
Editing Custom Composite OData Keys

You can edit and delete custom composite OData keys.

Publish Validation

When you publish a connection, Informatica Process Designer checks the custom composite keys you created and displays a warning if it encounters any potential issues.

The following image shows a sample warning message:



Using Salesforce System Services Listeners and Connectors

Salesforce Outbound Messages

Use a Salesforce outbound message (OBM) to trigger an Application Integration process.

For example, design an OBM such that a change to the Account Salesforce object triggers the CreateNewID process.

To create a Salesforce OBM, perform the following tasks:

1. Download and install the Informatica Cloud Real Time for Salesforce package (the 'managed package'). For more information, see the *Installing the Salesforce Managed Package* topic in the *Salesforce Managed Package* section.
2. Log in to Informatica Intelligent Cloud Services and open the Application Integration service.
3. Create and publish the process you want the Salesforce OBM to trigger.
4. Log in to your Salesforce developer account and create a Salesforce OBM.

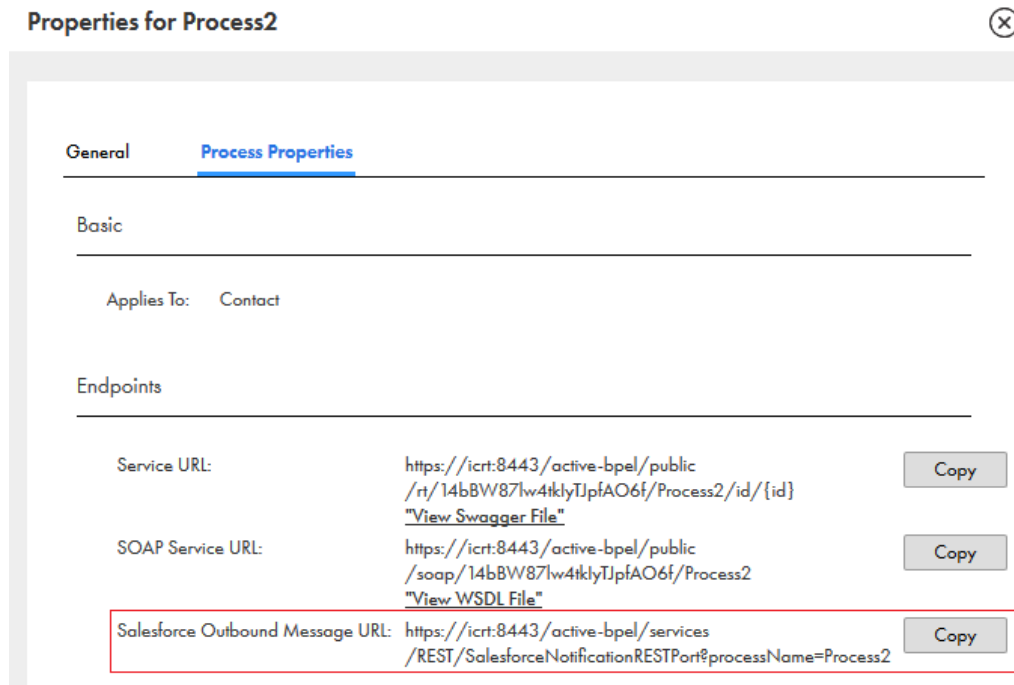
Salesforce OBM URL

When you invoke a process through a Salesforce OBM, Salesforce sends a message to a specific endpoint called the Salesforce OBM URL.

To view the Salesforce OBM URL for a process, perform the following steps:

1. In Application Integration, create a Salesforce connection and suffix the connection name with 'Salesforce'.
For example, name the connection 'AccountDetails-Salesforce' or 'TestSalesforce' and perform steps 2 through 5 to see the Salesforce OBM URL.
Note: You cannot view the Salesforce OBM if you use 'Salesforce' anywhere else in the name except at the end.
2. Save and publish the Salesforce connection.
3. Create, save, and publish a process that uses the Salesforce connection..
4. From the **Explore** page or from the **Process Designer** page of the process, click **Actions > Properties**.

5. In the **Properties** window that appears, click the **Process Properties** tab to view the Salesforce OBM URL. The following image shows a sample **Process Properties** tab with a Salesforce OBM URL visible:



Creating a Salesforce OBM

To create a Salesforce OBM, log in to your Salesforce developer account, create a workflow rule, and then define an OBM.

Before you create a Salesforce OBM to trigger an Application Integration process, you must perform the following tasks:

- Install the managed packaged in the Salesforce organization in which you want to create an OBM.
- Create and publish the Application Integration process you want the Salesforce OBM to trigger.

1. Log in to your Salesforce developer account.
2. Go to **Create > Workflow & Approvals > Workflow Rule**.
3. Click **New Rule**.

All Workflow Rules

Configure your organization's workflow by creating workflow rules. Each workflow rule consists of:

- Criteria that cause the workflow rule to run.
- Immediate actions that execute when a record matches the criteria. For example, Salesforce can automatically send an email that notifies the account team when a new high-value opportunity is created.
- Time-dependent actions that queue when a record matches the criteria, and execute according to time triggers. For example, Salesforce can automatically send an email reminder to the account team if a high-value opportunity is still open ten days before the close date.

View: **All Workflow Rules** [Create New View](#)

New Rule						
Action	Rule Name	Description	Object	Active		
Edit Del Activate	INFA_AccountRule	Triggers an Informatica Cloud Application Integration process when Account Annual Revenue exceeds \$500000	Account	☐		
Edit Del Activate	test		Account	☐		
Edit Del Deactivate	testRule		Account	☒		
Edit Del Activate	UserTest		User	☒		

4. On the **New Workflow Rule** page, select the object you want the rule to apply to and then click **Next**.

Workflow Rule
New Workflow Rule Help for this Page

Step 1: Select object Step 1 of 3

Select the object to which this workflow rule applies.

Object: Account

Next Cancel

- On the **Configure Workflow Rule** page, enter a rule name and description, set evaluation and rule criteria, and then click **Save & Next**.

Step 2: Configure Workflow Rule Step 2 of 3

Enter the name, description, and criteria to trigger your workflow rule. In the next step, associate workflow actions with this workflow rule.

Edit Rule Required Information

Object: Account

Rule Name: INFA_AccountRule

Description: Triggers an Informatica Cloud Application Integration process when Account Annual Revenue exceeds \$500000

Evaluation Criteria

Evaluate the rule when a record is:

- ☐ created
- ☐ created, and every time it's edited
- ☒ created, and any time it's edited to subsequently meet criteria

How do I choose?

Rule Criteria

Run this rule if the criteria are met

Field	Operator	Value	
Account: Annual Revenue	greater than	500000	AND
--None--	--None--		AND
--None--	--None--		AND
--None--	--None--		AND
--None--	--None--		

[Add Filter Logic...](#)

Previous Save & Next Cancel

- Go to **Add Workflow action > New Outbound Message**.
- On the **Configure Outbound Message** page, enter the required information, including the **Endpoint URL**. See the [Salesforce OBM URL on page 152](#) topic for information on how to obtain the Endpoint URL.

Configure Outbound Message

Enter the details of your outbound message and select the fields you want included in this message. Note that the fields available depend on the type of record previously selected.

Edit Outbound Message: Account Required Information

Name: SendAccountDetails

Unique Name: SendAccountDetails

Description: Update billing and account information

Endpoint URL: https://ai-pod1.staging1.informaticacloud.com/active-bpel/services/REST/SalesforceNotificationRES

User to send as: John Smith

Protected Component: ☐

Send Session ID: ☐

Account fields to send

Available Fields	Selected Fields
SicDesc	Id
Site	BillingLatitude
SystemModstamp	BillingLongitude
TickerSymbol	BillingPostalCode
TradeStyle	BillingState
Type	BillingStreet
UpsellOpportunity__c	AccountNumber
YearStarted	AnnualRevenue
NumberOfEmployees	AccountSource
Ownership	
Website	
BillingCity	
BillingCountry	
BillingGeocodeAccuracy	

Add Remove

- Click **Save**.

The Salesforce OBM creation is complete. The Salesforce OBM you created triggers an Application Integration process according to the criteria you set.

For detailed information about configuring a Salesforce OBM, see the Salesforce documentation.

Using a Salesforce Connection

If you are using the Salesforce Managed Package, the Salesforce Connector allow you to define a connection to your Salesforce organization using one of these authentication methods:

- OAuth Connection
- Password/Security Token Authentication

When you create a new Salesforce connection, choose the **Authentication Type** and provide the related connection properties.

OAuth Connection

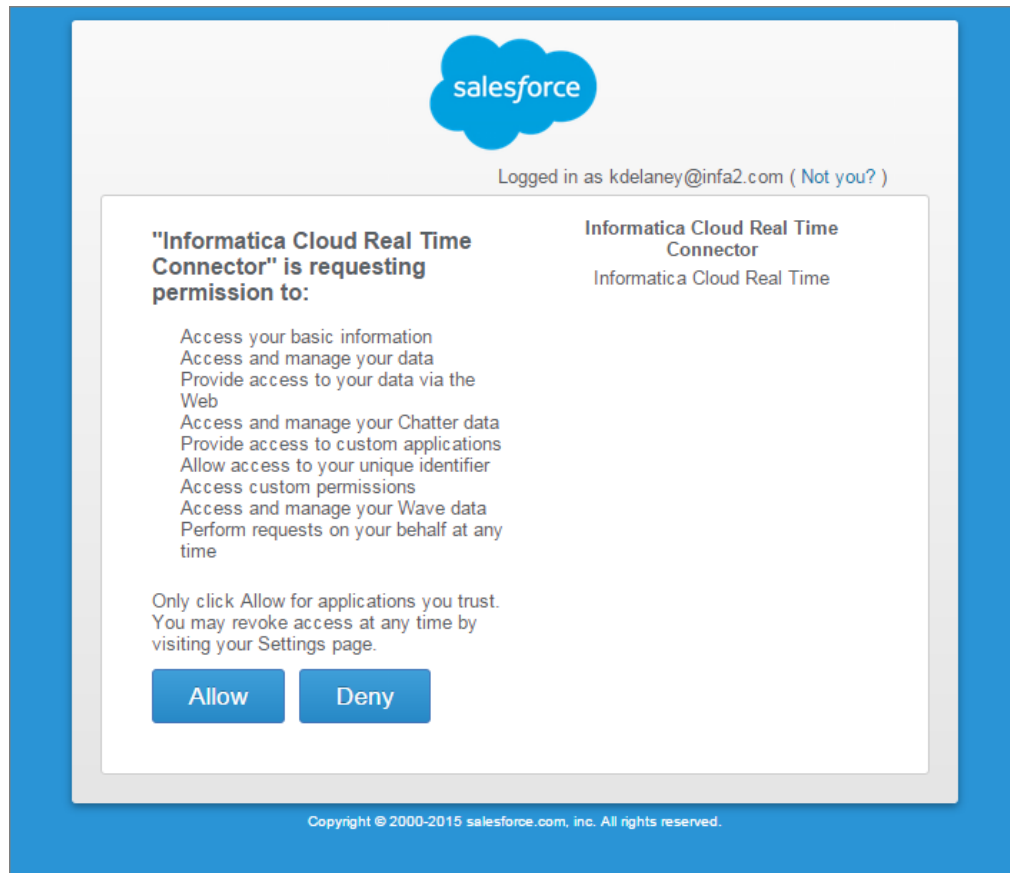
Salesforce supports the use of OAuth to allow access to Salesforce.com through its API. OAuth is a standard protocol that allows for secure API authorization. One benefit of OAuth is that users do not need to disclose their Salesforce credentials and the Salesforce administrator can revoke the consumer's access at any time.

To enable OAuth as the authentication type for a Salesforce connection, follow these steps:

Authentication:		
Authentication Type: OAuth		
Name	Value	Description
Authorization URL *	https://login.salesforce.com/services/oauth2/authorize	Enter the Salesforce OAuth authorization URL. Default value is https://login.salesforce.com/services/oauth2/authorize. For Sandbox, use https://test.salesforce.com/services/oauth2/authorize.
Token Request URL *	https://login.salesforce.com/services/oauth2/token	Enter the OAuth token request URL. For production, use https://login.salesforce.com/services/oauth2/token. For sandbox, use https://test.salesforce.com/services/oauth2/token.
Session Duration:	60	Enter the number of minutes to wait before refreshing the session. Default: 60 minutes.
Authorization Status:	Not yet authorized. Indicates the current status and the last time that authorization was completed.	
Authorize Access:	Authorize	Click to initiate the authorization workflow using OAuth.

1. Enter the production or test URL as instructed on the screen for these required fields:
 - **Authorization URL.** Salesforce provides dedicated URL that handles authorization. Enter the URL for either the production or test environment to access with this connection.
 - **Token Request URL.** Salesforce provides a dedicated URL that handles token requests. Enter the URL for either the production or test environment to access with this connection.
2. For the **Session Duration**, enter the number of minutes you want to maintain each session opened for this connection.
3. Click **Authorize** to start the authorization process.
4. **Note:** To successfully launch the authorization process, popups must be enabled in your browser. After you start the authorization process, you can cancel it prior to completion (by closing the dialog box. In that case, the previous state (authorized or not authorized) is unchanged.
5. If prompted, enter the Username and Password at the Salesforce login page and click **Log In to Salesforce**.

6. After your login is validated, the following window appears:



7. Click **Allow** to complete the authorization.
8. In the Salesforce Connection properties, an **Authorization Status** message similar to the following appears:

Note: You must complete the authorization process within 3 minutes. Otherwise, it times out. If that occurs, return to the connection properties and click **Authorize** again.

The authorization remains active as long as the connection is published, provided that OAuth access was not revoked in Salesforce. After a session duration expires, it is automatically renewed in the background.

Salesforce Connected App Settings

Depending on the type of OAuth access enabled by the Salesforce administrator, you may be able to self-authorize or have pre-authorization. That option is determined by the Salesforce Connected App settings shown here:

The screenshot shows the 'Connected App Edit' interface for 'Informatica Cloud Real Time for Salesforce Connected App'. The page includes a header with the app name and a 'Help for this Page' link. Below the header, there's a 'Connected App Edit' section with a cloud icon and fields for 'Version' (20) and 'Description'. The main settings are organized into sections: 'Basic Information' with 'Start URL' and 'Mobile Start URL' fields; 'OAuth policies' with 'Permitted Users' (a dropdown menu showing 'All users may self-authorize', 'All users may self-authorize', and 'Admin approved users are pre-authorized') and 'IP Relaxation' (a dropdown menu showing 'Enforce IP restrictions'); 'Session Policies' with 'Timeout Value' (a dropdown menu showing '--None--') and 'High assurance session required' (a checkbox); and 'User Provisioning Settings' with an 'Enable User Provisioning' checkbox. At the bottom, there are 'Save' and 'Cancel' buttons.

To learn more, refer to the Salesforce documentation.

Password/Security Token Authentication

When you choose the Password authentication type, you supply these properties:

- **User Name:** The username, for API access, to be used for the connection.
- **Password:** The password of the user associated with this connection.
- **Security Token:** The security token that provides this user with API access.
- **Service URL:** The Salesforce URL to use for the login. Be sure to specify the Soap/c WSDL (and not Soap/u).

At runtime, each connection is enabled based on the password and security token you provide.

Using the Email Service

Email Connection Properties

In Process Designer, you can define an Email connection type that allows you to send email from one or more configured email servers as part of a published process.

Note: In some cases, you might want to define multiple connections. For example, you might use different SMTP hosts for each school within a university or have multiple domains that each require a unique SMTP configuration. Each SMTP host requires a separate connection.

In addition to the general connection properties (see *Design > Using Connectors > Creating a Connection > Defining Connection Properties*), you can configure the email-specific properties described below.

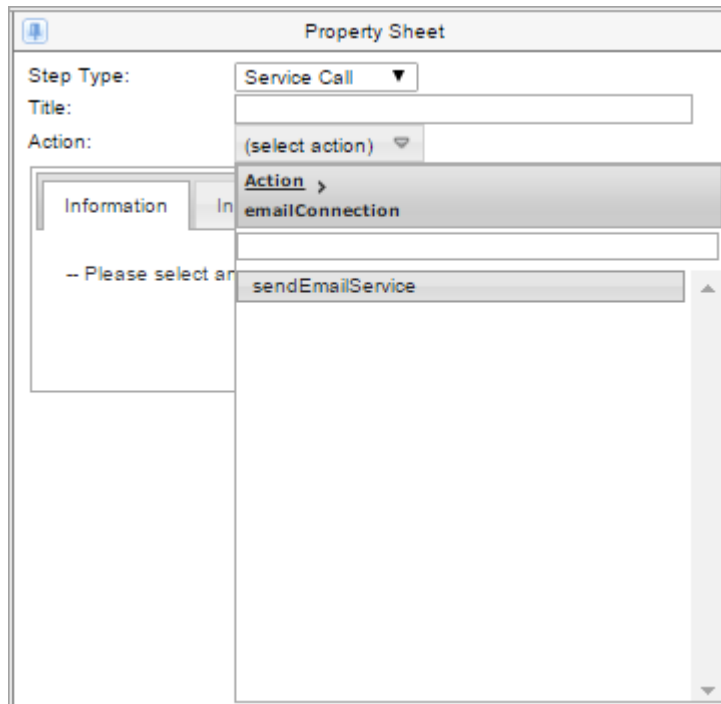
When you implement a Service step that includes an Email connection, note the following:

- You can configure the connection to run on either a specific agent or on Cloud Server or Any Secure Agent.
- You can publish the Email connection to run on a specific Secure Agent, and then call it in a Service step for another process, whether the process runs on the same agent, a different agent or on the Cloud Server. You do not need to wrap it in a process running on the agent where it was published.
- You can set Message headers in the process.
- You can send attachments in the email connection.

Property	Description
Authentication	Choose Disable if you want the connection to run with no authentication. For example, SMTP might be configured in your environment to work without authentication. Otherwise, choose Enable .
Host	Enter the email server's DNS name, such as mail.mydomain.com or an IP address, such as 192.168.1.1.
Port	Enter the port to use for communications between the Process Server and the email server. Default: 25
Username	Required if you enable authentication. Note: If SMTP is configured in your environment to work without authentication, do not enter a user name and password. If you disable authentication, any value entered here is ignored. Enter the user name used to login to your email server, usually the account name/email address. For example: notifyme@mydomain.com
Password	Required if you enable authentication. Not required if you disable authentication. Note: If SMTP is configured in your environment to work without authentication, do not enter a user name and password. If you disable authentication, any value entered here is ignored. Enter the password for the above email address.
Security	Select an option for the security protocol: None , TLS (Transport Layer Security), or SSL (Secure Sockets Layer). If necessary, change the Port to enable the appropriate security protocol.
Connection Timeout	Specify the connection timeout in milliseconds. Default: 30000 If you encounter a timeout while using an Email connection, review your network and firewall configuration.
Test Address	Enter the email address where you want to send a test message when you click Test to validate this connection.
From Address	Enter the email address you want to appear in the From field of emails sent from this connection. For example, no-reply@example.com.

Sending an Email from a Process

After you save and publish an Email connection, it displays as an Action in the Service step, so you can use it in a process:



Note the following when you create the process:

- Define email attachments as process objects with the following fields. This enables the process to handle multiple attachments. See the Published Metadata for the Email connection to review details:
 - Name
 - Content-type (mime type)
 - Content
- Set the contents of the email message in a temporary field or input field and use an Assignment step to set the content of each required field in the email message. For example, you might get the recipient addresses and attachments from upstream in the process.

Using Java Message Service (JMS)

A Java Message Service (JMS) enables messaging between loosely-coupled processes. "Loosely-coupled"-for purposes of messaging-means the messages are independent of one another. The interaction between the processes can be synchronous or asynchronous.

If the process sends a message while the message consumer is not running, the message waits in the JMS provider's queue until the message consumer is available. The contents of these messages can vary, but the data within the message is usually expressed as XML.

Taking advantage of the data flow that occurs with JMS, you can eliminate the need to invoke a service such as a database management system that runs on-premise.

Using the Shell Command Invoke Service Connector

You can create an invoke activity that calls a POJO service to execute a shell command script. The invoke activity is based on a shell command system service provided by Process Server. You can, for example, call a script that updates a database or adds a new user to an identity service. To use the shell command system service, you must create and locate your script in a working directory that is specified within the input message for the invoke activity.

The following message shows an example of input for the invoke activity.

```
<shl:execCommand xmlns:inv =  
  "urn:com:activee:rt:shellcmd:services:invoke"  
  xmlns:shl="urn:com:activee:rt:shellcmd:services:ishellcmdinvoker">  
  <inv:shellRequest>  
    <inv:command>string</inv:command>  
    <inv:workingDirPath>string</inv:workingDirPath>  
    <inv:charset>utf-8</inv:charset>  
    <inv:mergeErrorAndOutput>true</inv:mergeErrorAndOutput>  
    <inv:stdInput>string</inv:stdInput>  
  </inv:shellRequest>  
</shl:execCommand>
```

The elements of the message are described in the following table.

execCommand Input Element	Description
command	(Required) Name of the script to execute. Examples: cmd.exe /C myScript concat('shell.bat', \$requestVar)
workingPathDir	(Required) Working directory for the execution of the script
charset	Character encoding of the script. The default is UTF-8.
mergeErrorandOutput	True is the default setting. If true, directs errors as well as execution output to the same console.
stdInput	Data for the script