



Integration Guide | PUBLIC

Document Version: 1.0.0 – 2026-08-01

SAP IBP Inbound Integration

Reusable Integration Flow to Integrate External Systems to SAP Integrated Business Planning in SAP Cloud Integration

Content

1	Document History.	3
2	Introduction.	4
3	Prerequisites.	5
4	Configuring the Integration Flow Sequence.	6
4.1	Defining Parameters.	6
4.2	Configuring Extension Points.	7
	Loop Control.	10
4.3	Configuration Scenario Examples.	11
	Scenario 1 - Static One-Time Data Load Without Extensions.	11
	Scenario 2 - One-Time Body Enrichment via BeforeDynamicDataLoadExtensionPoint.	12
	Scenario 3 - Paginated Data Load via BeforePostDataExtensionID.	14
	Scenario 4 - Audit Transfer Completion via AfterDynamicDataLoadExtensionPoint.	17
	Scenario 5 - Build a Package Audit Entry via AfterPostDataExtensionID.	19
5	Calling the Generic Process.	21
5.1	Option 1 - With Externalized Parameters.	21
5.2	Option 2 - Headers Only.	22
5.3	Option 3 - Flat Body.	23
5.4	Option 4 - Full Structure.	24
5.5	Advanced Data Load Patterns.	24
6	Automatic Input Completion.	28
6.1	Structure Completion.	28
6.2	Attribute Inheritance.	29
6.3	Attribute Blocking.	29
6.4	Attribute Placement Pattern.	30
6.5	Extension Attribute Preservation.	31
7	Response and Error Handling.	32
7.1	Output.	32
7.2	Error Handling.	33
8	Troubleshooting.	35

1 Document History

The following table provides an overview of the most important changes.

Version	Date	Description
1.0.0	August 1, 2026	Initial version

2 Introduction

You can transfer data from external systems to SAP Integrated Business Planning (SAP IBP) using the [SAP IBP Write - Inbound Integration](#) iFlow sequence from the SAP IBP - Reusable Integration Flows package. The iFlow sequence runs on SAP Cloud Integration and consists of three co-dependent reusable integration flows (iFlows) that provide a standardized, extensible framework:

iFlow	Purpose
SAP IBP Write - Inbound Integration Generic Process integration flow	Orchestrates the entire flow: creates the batch, calls extensions, invokes the SAP IBP Write - Inbound Integration Data Transfer integration flow, and finalizes with the SAP IBP Write - Process Posted Data integration flow.
SAP IBP Write - Inbound Integration Data Transfer integration flow	Iterates through packages sequentially or in parallel and calls the SAP IBP Write - Inbound Integration Dynamic Post Data integration flow for each package.
SAP IBP Write - Inbound Integration Dynamic Post Data integration flow	Posts a single package of data to SAP IBP via the SAP IBP Write - Post Data integration flow, with optional before and after extension points.

You can call the iFlow sequence by providing the complete data payload upfront, or by providing only the required parameters and letting the iFlow sequence retrieve the data dynamically through extension flows that you implement.

3 Prerequisites

Before you use this integration flow (iFlow) sequence, make sure you have the following:

- You have enabled the [Planning - Integration Suite - Cloud Integration Integration \(SAP_COM_0931\)](#) communication scenario that allows a connection between SAP IBP and SAP Cloud Integration.
- You have deployed the [SAP IBP - Reusable Integration Flows](#) package. For more information, see [Setting Up the Integration for Using SAP IBP - Reusable Integration Flows](#).

4 Configuring the Integration Flow Sequence

4.1 Defining Parameters

You can configure the integration flow (iFlow) sequence by setting parameters on the input XML or as externalized integration flow parameters. Externalized parameters act as defaults when the corresponding XML attribute or message header is missing or empty.

Parameters for SAP IBP Write - Inbound Integration Generic Process

The following parameters are required:

Parameter Name	Parameter ID	How to Configure the Parameter
Batch Command	BatchCommand	Enter the operation type. Accepted values: DELETE , INSERT_UPDATE , REPLACE .
Batch Name	BatchName	Enter the name for the batch in SAP IBP.
Destination for SAP IBP	DestinationforSAPIBP	Enter the destination of the SAP IBP system that you previously created in the SAP BTP subaccount.

The following parameters are optional:

Parameter Name	Parameter ID	How to Configure the Parameter
After Dynamic Data Load Extension Point	AfterDynamicDataLoadExtensionPoint	Enter the <i>ProcessDirect</i> address of an extension integration flow. When set, this integration flow is called after the <i>SAP IBP Write - Inbound Integration Data Transfer</i> integration flow completes and before Process Posted Data runs.
After Post Data Extension ID	AfterPostDataExtensionID	Enter the <i>ProcessDirect</i> address of an extension integration flow. When set, this integration flow is called after each <i>SAP IBP Write - Post Data</i> integration flow call.

Parameter Name	Parameter ID	How to Configure the Parameter
Before Dynamic Data Load Extension Point	BeforeDynamicDataLoadExtensionPoint	Enter the ProcessDirect address of an extension integration flow. When set, this integration flow is called after Create Batches completes and before the SAP IBP Write - Inbound Integration Data Transfer integration flow starts.
Before Post Data Extension ID	BeforePostDataExtensionID	Enter the ProcessDirect address of an extension integration flow. When set, this integration flow is called before each SAP IBP Write - Post Data integration flow call.
Planning Area	PlanningArea	Enter the SAP IBP planning area ID. Required if you set <code>PlanningAreaVersion</code> .
Planning Area Version	PlanningAreaVersion	Enter the planning area version. If you set this parameter, you must also set <code>PlanningArea</code> .
Process Packages in Parallel	ProcessPackagesinParallel	Set to true to process packages in parallel. The default is false .
Process Unchanged Data	ProcessUnchangedData	Set whether unchanged data is processed.

Externalized Integration Flow Parameter

The following parameter can be configured at the integration flow level as externalized parameter.

SAP IBP Write - Inbound Integration Data Transfer

Parameter Name	Required	Default Value	How to Configure the Parameter
Number of Concurrent Processes	Yes	2	Enter the number of threads for parallel processing.

4.2 Configuring Extension Points

You can use extension points to customize the data flow by calling your own integration flows (iFlows) at specific stages of processing. The iFlow sequence provides four extension points.

Batch-Level Extension Points

Set batch-level extension points on the root `IBPInboundIntegrationGenericProcess` element or as message headers. Set `BeforeDynamicDataLoadExtensionPoint` to call a [ProcessDirect](#) endpoint before Data Transfer, and `AfterDynamicDataLoadExtensionPoint` to call one after Data Transfer.

Before Dynamic Data Load Extension Point

Trigger	After Create Batches completes and a batch ID has been assigned, before Data Transfer starts.
Purpose	Modify or enrich the entire batch body once before it is split into packages.
Input	The full normalized <code>IBPInboundIntegrationGenericProcess</code> body with the batch ID injected.
Output	Your modified body replaces what is passed to the SAP IBP Write - Inbound Integration Data Transfer integration flow.

After Dynamic Data Load Extension Point

Trigger	After the SAP IBP Write - Inbound Integration Data Transfer integration flow completes successfully, before Process Posted Data.
Purpose	Logging, notifications, or post-processing based on transfer results.
Input	The SAP IBP Write - Inbound Integration Data Transfer response body.
Output	Not used by the main flow. The SAP IBP Write - Process Posted Data integration flow uses the stored batch ID. Design this extension to perform actions such as logging, audit records, notifications, or queue messages.

Package-Level Extension Points

Set package-level extension points on `IBPInboundIntegrationDynamicPostData`. You can also provide them as root or header defaults. They are inherited to packages where appropriate. Set `BeforePostDataExtensionID` to call a [ProcessDirect](#) endpoint before posting a package, and `AfterPostDataExtensionID` to call one after posting.

Before Post Data Extension ID

Trigger	Before the SAP IBP Write - Post Data integration flow is called for each package in the SAP IBP Write - Inbound Integration Dynamic Post Data integration flow.
Purpose	Paginated data loading, dynamic data enrichment, or transformation per package.

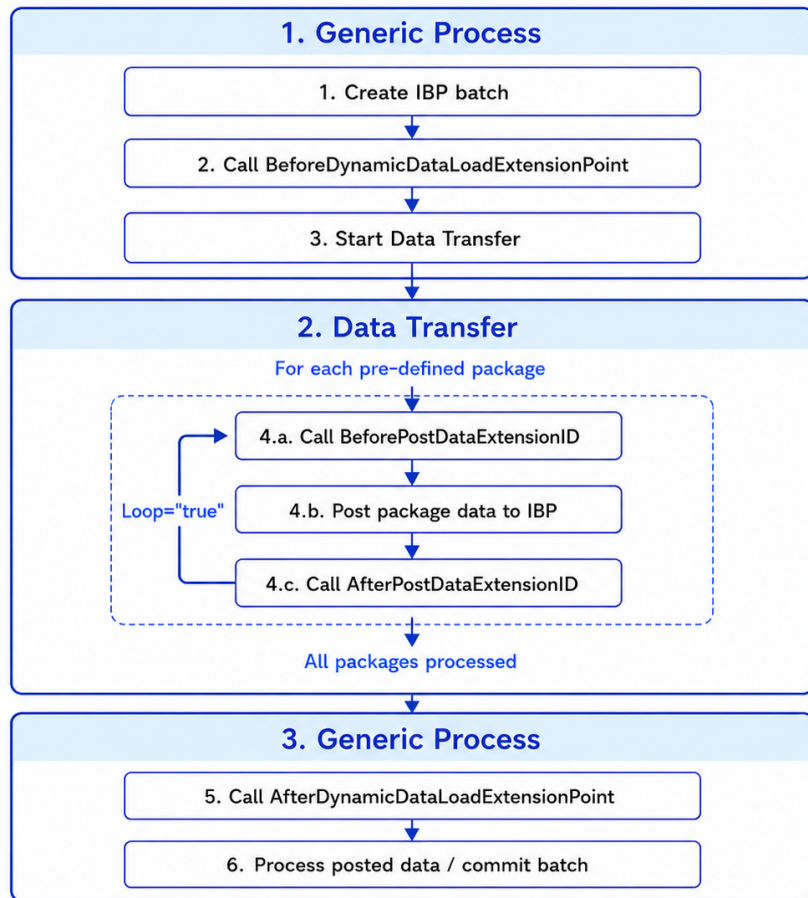
Input	The <code>IBPINboundIntegrationDynamicPostData</code> element with its <code>PackageContent</code> node.
Output	Your modified body replaces what is posted to SAP IBP.
Loop Control	Return <code>Loop="true"</code> on the root element to continue iterating. Return <code>Loop="false"</code> or omit it on the last page.
Note	The <code>BeforePostDataExtensionID</code> and <code>AfterPostDataExtensionID</code> attributes are not inherited to <code>PackageContent</code> elements. They are blocked at the Dynamic Post Data normalizer level.

After Post Data Extension ID

Trigger	After the <i>SAP IBP Write - Post Data</i> integration flow completes successfully for each package.
Purpose	Logging, notifications, or conditional logic based on post results.
Input	The raw <i>SAP IBP Write - Post Data</i> response body, preserved and restored from an internal property.
Output	Not used by the main flow. Design this extension to perform actions such as logging, audit records, notifications, or queue messages.

Processing Sequence with Extension Points

The extension points run in the following order if they are configured.



Keep the following in mind when you design your extension flows:

Behaviour	Implication
BatchID is available to all extension points	The IBP batch has already been created before the first extension runs.
BeforeDynamicDataLoadExtensionPoint runs once	Use it for decisions that affect the whole transfer.
BeforePostDataExtensionID may run more than once for the same package	Use Loop for pagination.
AfterPostDataExtensionID runs after one package post attempt	Use it for package-level result handling.
AfterDynamicDataLoadExtensionPoint runs after package transfer	Use it for batch-level completion handling.

4.2.1 Loop Control

The Loop attribute controls whether BeforePostDataExtensionID is called again for the same package after posting. Only BeforePostDataExtensionID can control iteration - the other three extension points

do not support looping. Set `Loop` on `IBPInboundIntegrationDynamicPostData`, or return it from your extension.

Value	Behavior
true	After posting, the integration flow (iFlow) calls <code>BeforePostDataExtensionID</code> again for the same package to fetch the next chunk.
false or omitted	This is the last or only chunk. The iFlow continues with the next package.

The loop also stops in the following situations:

- Your extension returns no child data element under `IBPInboundIntegrationDynamicPostData`. The loop ends with a no-data condition.
- An error occurs during processing. Processing stops according to the error handling rules.

Note

An empty `PackageContent` element is still treated as a data element. The iFlow does not count the number of `item` rows to decide whether to stop looping.

4.3 Configuration Scenario Examples

This section describes common configuration scenarios for the integration flow sequence.

4.3.1 Scenario 1 - Static One-Time Data Load Without Extensions

Use this scenario when your wrapper integration flow (iFlow) already has the complete dataset.

Configure the following in the [Content Modifier](#) or as root XML attributes:

Header	Value
<code>BatchCommand</code>	DELETE, INSERT_UPDATE, or REPLACE
<code>BatchName</code>	Descriptive batch name
<code>DestinationforSAPIBP</code>	The SAP BTP destination pointing to the target SAP IBP system
Data attributes	<code>DataType</code> , <code>FieldList</code> , <code>FileName</code> , and others

Request Body Example

Example request body (Option 3 - Flat Body):

❧ Example

```
<IBPInboundIntegrationGenericProcess
BatchCommand="INSERT_UPDATE"
BatchName="Location Load"
DestinationforSAPIBP="MY_IBP_DEST">
<PackageContent
DataType="MasterData"
FieldList="LOCID, LOCTYPE, LOCDESCR"
FileName="Location File"
MasterDataPrefix="IBP"
MasterDataType="LOCATION">
<item><LOCID>1010</LOCID><LOCTYPE>P</LOCTYPE><LOCDESCR>Plant DE</LOCDESCR></item>
<item><LOCID>1020</LOCID><LOCTYPE>P</LOCTYPE><LOCDESCR>Plant UK</LOCDESCR></item>
</PackageContent>
</IBPInboundIntegrationGenericProcess>
```

Result: data is loaded in a single package and a single iteration.

4.3.2 Scenario 2 - One-Time Body Enrichment via BeforeDynamicDataLoadExtensionPoint

Use this scenario when you need to modify the entire batch body once before processing begins.

Use this extension point in the following situations:

- Removing packages that should not be posted based on the complete incoming transfer
- Setting package-level defaults before processing starts
- Validating consistency across all packages
- Assigning different BeforePostDataExtensionID values to different packages based on routing logic

Header	Value
BatchCommand	INSERT_UPDATE
BatchName	Enriched Load
BeforeDynamicDataLoadExtensionPoint	My_Enricher
DestinationforSAPIBP	MY_IBP_DEST

Your extension integration flow receives: the full body with the batch ID already assigned, including the nested IBPInboundIntegrationDataTransfer content. Packages may have empty PackageContent at this stage if a BeforePostDataExtensionID is configured to fill them later.

Your extension integration flow returns: the modified body in the same structure. The returned body replaces what is passed to the *SAP IBP Write - Inbound Integration Data Transfer* integration flow.

Example: Prepare or Filter the Transfer Body

In this example, you can inspect the transfer body, assign a package-specific extension ID to one package, and return the updated body. Your extension integration flow receives the full body and works through the packages as follows:

1. Read the full body. The batch ID is already assigned.
2. For each package, read its metadata such as **DataType**, **FileName**, and **FieldList**.
3. Remove packages that should not be processed, or assign a **BeforePostDataExtensionID** to packages that need a specialized data fetcher.
4. Apply any package-level defaults you need.
5. Return the updated body.

Input your extension receives:

Sample Code

```
<IBPInboundIntegrationGenericProcess
  BatchCommand="INSERT_UPDATE"
  BatchID="BATCH-DEMO-001"
  BatchName="Daily Demand Load"
  BeforePostDataExtensionID="Fetch_Demand_Page"
  DestinationforSAPIBP="MY_IBP_DEST"
  PlanningArea="DEMAND_PA">
  <IBPInboundIntegrationDataTransfer
    BatchID="BATCH-DEMO-001"
    BatchName="Daily Demand Load"
    BeforePostDataExtensionID="Fetch_Demand_Page"
    DestinationforSAPIBP="MY_IBP_DEST">
    <Packages>
      <PackageDefinition
        DataType="KeyFigure"
        FileName="Demand_Adjustments"
        FieldList="I_LOCID, I_PRDID, I_TSPCUSTID, KEYFIGUREDATE, I_ACTUALSQTY"
        TimeDisaggregationLevel="2">
      <PackageContent />
    </PackageDefinition>
  </Packages>
</IBPInboundIntegrationDataTransfer>
</IBPInboundIntegrationGenericProcess>
```

Output your extension returns:

Sample Code

```
<IBPInboundIntegrationGenericProcess
  BatchCommand="INSERT_UPDATE"
  BatchID="BATCH-DEMO-001"
  BatchName="Daily Demand Load"
  BeforePostDataExtensionID="Fetch_Demand_Page"
  DestinationforSAPIBP="MY_IBP_DEST"
  PlanningArea="DEMAND_PA">
  <IBPInboundIntegrationDataTransfer
    BatchID="BATCH-DEMO-001"
```

```

BatchName="Daily Demand Load"
BeforePostDataExtensionID="Fetch_Demand_Page"
DestinationforSAPIBP="MY_IBP_DEST">
<Packages>
  <PackageDefinition
    DataType="KeyFigure"
    FileName="Demand_Adjustments"
    FieldList="I_LOCID,I_PRDID,I_TSPCUSTID,KEYFIGUREDATE,I_ACTUALSQTY"
    TimeDisaggregationLevel="2"
    BeforePostDataExtensionID="Fetch_Demand_Page">
    <PackageContent />
  </PackageDefinition>
</Packages>
</IBPInboundIntegrationDataTransfer>
</IBPInboundIntegrationGenericProcess>

```

Extension integration flow setup: create an integration flow with a [ProcessDirect](#) sender adapter. Set its address to match `BeforeDynamicDataLoadExtensionPoint`.

4.3.3 Scenario 3 - Paginated Data Load via BeforePostDataExtensionID

Use this scenario when your data source requires pagination, for example, a REST API with a cursor or a large JDBC result set.

Use this extension point in the following situations:

- Fetching package data from an external system such as an API, database, or file storage
- Loading large data sets in package-sized chunks
- Transforming or automatically completing package fields before posting
- Adding package-specific metadata
- Using different fetching logic for different packages

Configure the following in the Content Modifier (Header tab). No request body is needed - the [SAP IBP Write - Inbound Integration Generic Process](#) integration flow builds an empty structure:

Header	Value
BatchCommand	INSERT_UPDATE
BatchName	KF Paginated Load
BeforePostDataExtensionID	My_KF_Pager
DestinationforSAPIBP	MY_IBP_DEST
PlanningArea	MYPA

Your extension integration flow receives the following input:

Sample Code

```
<IBPInboundIntegrationDynamicPostData
  BatchID="BATCH-001"
  BeforePostDataExtensionID="My_KF_Pager"
  DestinationforSAPIBP="MY_IBP_DEST"
  Loop="false">
  <PackageContent DataType="KeyFigure" .../>
</IBPInboundIntegrationDynamicPostData>
```

Your extension integration flow returns the following output:

Sample Code

```
<IBPInboundIntegrationDynamicPostData
  BatchID="BATCH-001"
  DestinationforSAPIBP="MY_IBP_DEST"
  Loop="true">
  <PackageContent
    DataType="KeyFigure"
    FieldList="PROID, KF_SUPPLY, TIMEPERIOD"
    FileName="KF Page 3"
    PlanningArea="MYPA">
    <item>...</item>
  </PackageContent>
</IBPInboundIntegrationDynamicPostData>
```

- Set Loop="true" while more pages remain.
- Set Loop="false" or omit it on the last page.
- Preserve all input attributes on the output root element.

Extension integration flow setup: create an integration flow with a *ProcessDirect* sender adapter. Set its address to match BeforePostDataExtensionID.

Key difference from BeforeDynamicDataLoadExtensionPoint: this extension runs per-package iteration, not once for the entire batch.

Example: Paginated Key Figure Load

In this example, you can load a large key figure data set in package-sized chunks. Your extension integration flow fetches one page of records per call and signals whether more pages remain. Work through the following steps:

1. Read the current page range from headers or extension state.
2. Call your source system for the current range.
3. Map the source fields to the target IBP field names. If a required planning level attribute is missing in the source, provide a valid constant or dummy member that exists in the planning area, for example, a valid customer placeholder for I_TSPCUSTID.
4. Build the PackageContent with the mapped rows.
5. If more pages remain, set Loop="true". On the last page, set Loop="false".

6. Return the package body.

Input your extension receives (first call):

Sample Code

```
<IBPInboundIntegrationDynamicPostData
  BatchID="BATCH-DEMO-001"
  BatchName="Daily Demand Load"
  BeforePostDataExtensionID="Fetch_Demand_Page"
  DataType="KeyFigure"
  DestinationforSAPIBP="MY_IBP_DEST"
  FieldList="I_LOCID,I_PRDID,I_TSPCUSTID,KEYFIGUREDATE,I_ACTUALSQTY"
  FileName="Demand_Adjustments"
  TimeDisaggregationLevel="2">
<PackageContent
  BatchID="BATCH-DEMO-001"
  BatchName="Daily Demand Load"
  BeforePostDataExtensionID="Fetch_Demand_Page"
  DataType="KeyFigure"
  DestinationforSAPIBP="MY_IBP_DEST"
  FieldList="I_LOCID,I_PRDID,I_TSPCUSTID,KEYFIGUREDATE,I_ACTUALSQTY"
  FileName="Demand_Adjustments"
  TimeDisaggregationLevel="2" />
</IBPInboundIntegrationDynamicPostData>
```

Output your extension returns (not the last page):

Sample Code

```
<IBPInboundIntegrationDynamicPostData
  BatchID="BATCH-DEMO-001"
  BatchName="Daily Demand Load"
  DataType="KeyFigure"
  DestinationforSAPIBP="MY_IBP_DEST"
  FieldList="I_LOCID,I_PRDID,I_TSPCUSTID,KEYFIGUREDATE,I_ACTUALSQTY"
  FileName="Demand_Adjustments"
  TimeDisaggregationLevel="2"
  Loop="true">
<PackageContent
  DataType="KeyFigure"
  FieldList="I_LOCID,I_PRDID,I_TSPCUSTID,KEYFIGUREDATE,I_ACTUALSQTY"
  FileName="Demand_Adjustments"
  TimeDisaggregationLevel="2">
  <item>
    <I_LOCID>LOC_DEMO_01</I_LOCID>
    <I_PRDID>PROD_DEMO_01</I_PRDID>
    <I_TSPCUSTID>DUMMY_CUSTOMER</I_TSPCUSTID>
    <KEYFIGUREDATE>2026-07-16</KEYFIGUREDATE>
    <I_ACTUALSQTY>125</I_ACTUALSQTY>
  </item>
  <item>
    <I_LOCID>LOC_DEMO_02</I_LOCID>
    <I_PRDID>PROD_DEMO_02</I_PRDID>
    <I_TSPCUSTID>DUMMY_CUSTOMER</I_TSPCUSTID>
    <KEYFIGUREDATE>2026-07-16</KEYFIGUREDATE>
    <I_ACTUALSQTY>240</I_ACTUALSQTY>
  </item>
</PackageContent>
</IBPInboundIntegrationDynamicPostData>
```

Output your extension returns (last page):

Sample Code

```
<IBPInboundIntegrationDynamicPostData
  BatchID="BATCH-DEMO-001"
  BatchName="Daily Demand Load"
  DataType="KeyFigure"
  DestinationforSAPIBP="MY_IBP_DEST"
  FieldList="I_LOCID,I_PRDID,I_TSPCUSTID,KEYFIGUREDATE,I_ACTUALSQTY"
  FileName="Demand_Adjustments"
  TimeDisaggregationLevel="2"
  Loop="false">
  <PackageContent
    DataType="KeyFigure"
    FieldList="I_LOCID,I_PRDID,I_TSPCUSTID,KEYFIGUREDATE,I_ACTUALSQTY"
    FileName="Demand_Adjustments"
    TimeDisaggregationLevel="2">
    <item>
      <I_LOCID>LOC_DEMO_99</I_LOCID>
      <I_PRDID>PROD_DEMO_99</I_PRDID>
      <I_TSPCUSTID>DUMMY_CUSTOMER</I_TSPCUSTID>
      <KEYFIGUREDATE>2026-07-16</KEYFIGUREDATE>
      <I_ACTUALSQTY>90</I_ACTUALSQTY>
    </item>
  </PackageContent>
</IBPInboundIntegrationDynamicPostData>
```

4.3.4 Scenario 4 - Audit Transfer Completion via AfterDynamicDataLoadExtensionPoint

Use this scenario when you need to react once at batch level after all packages have been transferred, before the final process-posted step runs.

Use this extension point in the following situations:

- Sending one completion notification after all packages transfer
- Writing batch-level audit information
- Preparing a follow-up trigger before process-posted handling runs
- Storing batch metrics based on transfer status and messages

Configure the following in the *Content Modifier* (Header tab):

Header	Value
AfterDynamicDataLoadExtensionPoint	Notify_Batch_Done
BatchCommand	INSERT_UPDATE
BatchName	Daily Demand Load
DestinationforSAPIBP	MY_IBP_DEST

Your extension integration flow receives: the [SAP IBP Write - Inbound Integration Data Transfer](#) response body, including transfer-level attributes, `ExceptionOccurred`, and messages. Detailed package rows are not available in this body.

Your extension integration flow returns: a response body, but it is not used by the main flow. The useful work is the log entry, notification, stored trigger, or queue message.

Extension integration flow setup: create an integration flow with a [ProcessDirect](#) sender adapter. Set its address to match `AfterDynamicDataLoadExtensionPoint`.

Example: Audit Transfer Completion

In this example, you can persist a transfer-level audit entry when Data Transfer completes. Your extension integration flow works through the following steps:

1. Extract `BatchID`, `BatchName`, `DestinationforSAPIBP`, and `ExceptionOccurred` from the input body.
2. Count messages by severity.
3. Write a transfer audit entry, for example to a data store, log, or external audit endpoint - with the batch ID, destination, exception flag, message summary, and timestamp.
4. Return the response body. The return value is not used by the main flow.

Input your extension receives:

Sample Code

```
<IBPInboundIntegrationDataTransfer
  BatchID="BATCH-DEMO-001"
  BatchName="Daily Demand Load"
  BeforePostDataExtensionID="Fetch_Demand_Page"
  DestinationforSAPIBP="MY_IBP_DEST"
  ExceptionOccurred="false">
  <Messages />
</IBPInboundIntegrationDataTransfer>
```

Output your extension returns - not used by the main flow:

Sample Code

```
<IBPInboundIntegrationDataTransfer
  BatchID="BATCH-DEMO-001"
  BatchName="Daily Demand Load"
  BeforePostDataExtensionID="Fetch_Demand_Page"
  DestinationforSAPIBP="MY_IBP_DEST"
  ExceptionOccurred="false">
  <Messages />
  <TransferAudit>
    <BatchID>BATCH-DEMO-001</BatchID>
    <Destination>MY_IBP_DEST</Destination>
    <ExceptionOccurred>false</ExceptionOccurred>
    <Summary>Package transfer completed without reported errors.</Summary>
    <Timestamp>2026-07-16T09:30:00Z</Timestamp>
  </TransferAudit>
</IBPInboundIntegrationDataTransfer>
```

4.3.5 Scenario 5 - Build a Package Audit Entry via AfterPostDataExtensionID

Use this scenario when you need to react to the result of each individual package post. At this point the body is the SAP IBP write response for one package, not the original package input.

This extension point is a good fit when you need to:

- Write an audit entry for each package post
- Send an alert when package errors exceed a threshold
- Collect per-package metrics
- Store troubleshooting context together with the SAP IBP response messages

Configure the following in the *Content Modifier* (Header tab):

Header	Value
AfterPostDataExtensionID	Audit_Package_Post
BatchCommand	INSERT_UPDATE
BatchName	Daily Demand Load
DestinationforSAPIBP	MY_IBP_DEST

Your extension integration flow receives: the SAP IBP write response for one package - either `IBPWriteKeyFigures` or `IBPWriteMasterData` - including status, request attributes, and messages.

Your extension integration flow returns: a response body, but it is not used by the main flow. The useful work is the side effect - a log entry, audit record, notification, or queue message.

Extension integration flow setup: create an integration flow with a *ProcessDirect* sender adapter. Set its address to match `AfterPostDataExtensionID`.

Example: Build a Package Audit Entry

In this example, you can record the file name, status, messages, and any response-provided counters for each package post. Your extension integration flow works through the following steps:

1. Extract the status and request attributes from the SAP IBP write response.
2. Read the batch ID and package context from message properties or headers.
3. Build an audit entry with the timestamp, batch ID, file name, status, and message summary.
4. Write the audit entry to a log, data store, or external audit endpoint.
5. Return the response body. The return value is not used by the main flow.

Input your extension receives:

Sample Code

```
<IBPWriteKeyFigures Status="OK">
```

```

<Request>
  <IBPWriteKeyFigures
    BatchId="BATCH-DEMO-001"
    Destination="MY_IBP_DEST"
    FileName="Demand_Adjustments"
    FieldList="I_LOCID,I_PRDID,I_TSPCUSTID,KEYFIGUREDATE,I_ACTUALSQTY"
    TimeDisaggregationLevel="2" />
  </Request>
<Messages>
  <Message Origin="/IBP/INTIS_WRITE_DATA" Type="Information">
    Initialization of file Demand_Adjustments is completed.
  </Message>
</Messages>
<Response>
  <Status>OK</Status>
</Response>
</IBPWriteKeyFigures>

```

Output your extension returns - not used by the main flow:

Sample Code

```

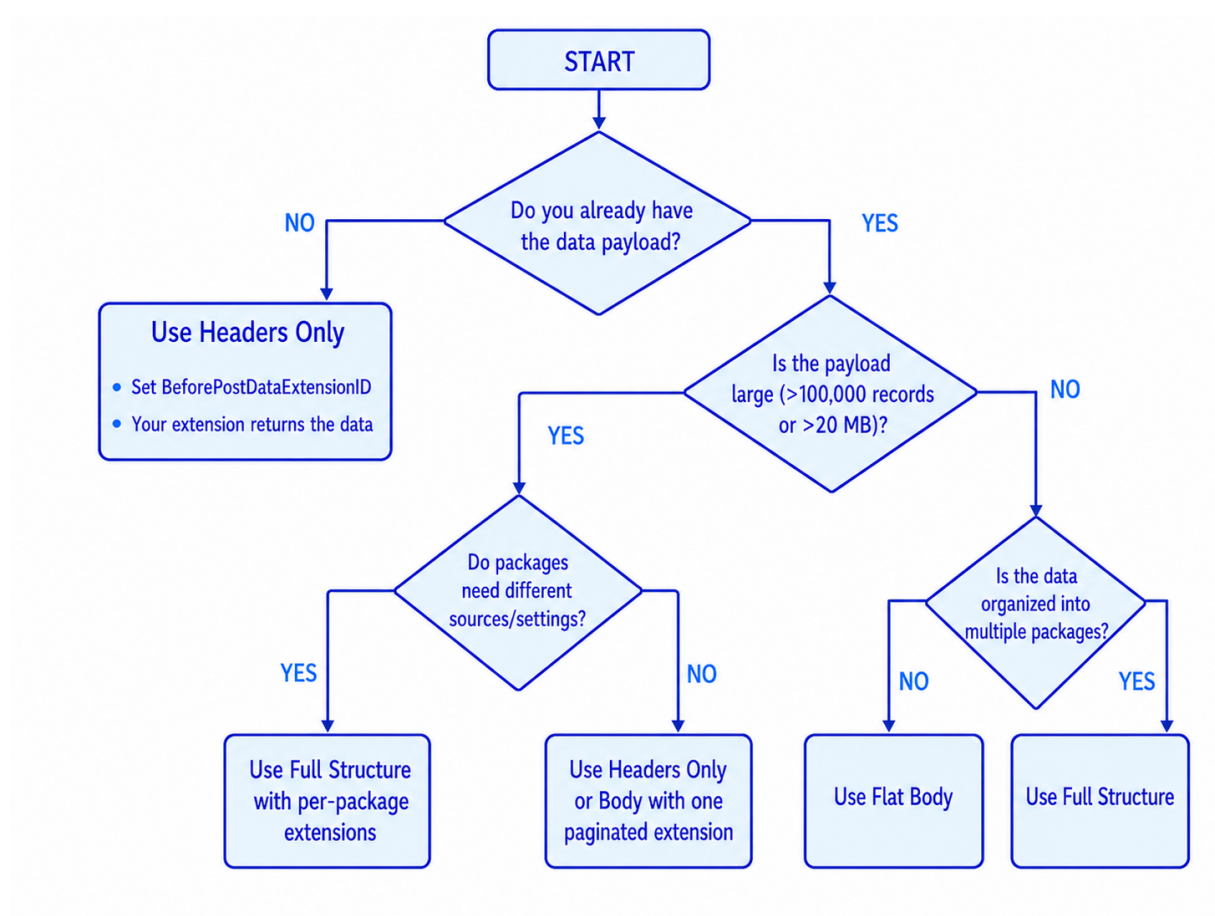
<IBPWriteKeyFigures Status="OK">
  <Request>
    <IBPWriteKeyFigures
      BatchId="BATCH-DEMO-001"
      Destination="MY_IBP_DEST"
      FileName="Demand_Adjustments"
      FieldList="I_LOCID,I_PRDID,I_TSPCUSTID,KEYFIGUREDATE,I_ACTUALSQTY"
      TimeDisaggregationLevel="2" />
    </Request>
  <Messages>
    <Message Origin="/IBP/INTIS_WRITE_DATA" Type="Information">
      Initialization of file Demand_Adjustments is completed.
    </Message>
  </Messages>
  <Response>
    <Status>OK</Status>
  </Response>
  <AuditEntry>
    <Timestamp>2026-07-16T09:18:41Z</Timestamp>
    <BatchID>BATCH-DEMO-001</BatchID>
    <FileName>Demand_Adjustments</FileName>
    <Status>OK</Status>
    <MessageSummary>Package posted successfully.</MessageSummary>
  </AuditEntry>
</IBPWriteKeyFigures>

```

5 Calling the Generic Process

There are four options for providing input to the *SAP IBP Write - Inbound Integration Generic Process* integration flow (iFlow).

Use the following decision tree to choose the right input option for your use case.



5.1 Option 1 - With Externalized Parameters

All parameters are configured as externalized parameters of the *SAP IBP Write - Inbound Integration Generic Process* integration flow. Use this option when your data is fetched by an extension integration flow and you don't want to create a wrapper integration flow.

How It Works

1. The *SAP IBP Write - Inbound Integration Generic Process* integration flow uses the configured externalized parameters to build the body.

2. The [Create Batches](#) step assigns a batch ID.
3. The [SAP IBP Write - Inbound Integration Data Transfer](#) integration flow calls the [SAP IBP Write - Inbound Integration Dynamic Post Data](#) integration flow.
4. Your `BeforePostDataExtensionID` extension integration flow fetches and returns data for each iteration.

Example Configuration

Example	
Externalized Parameter Name	Value
Before Post Data Extension ID	My_Data_Fetcher
Batch Command	INSERT_UPDATE
Batch Name	Daily KF Load
Destination for SAP IBP	MY_IBP_DEST
Planning Area	MYPA

5.2 Option 2 - Headers Only

All parameters are sent via message headers. Use this option when your data is fetched by an extension integration flow.

How It Works

1. The [SAP IBP Write - Inbound Integration Generic Process](#) integration flow reads the headers and builds an empty package structure.
2. The [Create Batches](#) step assigns a batch ID.
3. The [SAP IBP Write - Inbound Integration Data Transfer](#) integration flow calls the [SAP IBP Write - Inbound Integration Dynamic Post Data](#) integration flow.
4. Your `BeforePostDataExtensionID` extension integration flow fetches and returns data for each iteration.

Example Configuration

Example wrapper integration flow configuration (Content Modifier - Header tab):

❖ Example	
Header Name	Value
BatchCommand	INSERT_UPDATE
BatchName	Daily KF Load
BeforePostDataExtensionID	My_Data_Fetcher
DestinationforSAPIBP	MY_IBP_DEST
PlanningArea	MYPA

5.3 Option 3 - Flat Body

Your wrapper integration flow provides a body with `PackageContent` as a direct child of the root element. The *SAP IBP Write - Inbound Integration Generic Process* integration flow wraps it into the full nested structure.

Example Body

Sample Code
<pre><IBPInboundIntegrationGenericProcess BatchCommand="INSERT_UPDATE" BatchName="Location Load" DestinationforSAPIBP="MY_IBP_DEST" PlanningArea="MYPA"> <PackageContent DataType="MasterData" MasterDataPrefix="IBP" MasterDataType="LOCATION" FileName="Location File" FieldList="LOCID, LOCTYPE, LOCDESCR"> <item> <LOCID>1010</LOCID> <LOCTYPE>P</LOCTYPE> <LOCDESCR>Plant 1 DE</LOCDESCR> </item> </PackageContent> </IBPInboundIntegrationGenericProcess></pre>

Validation: when a body is present, all parameters must come from the root element attributes, not from headers. The `PackageContent` element requires the `DataType`, `FileName`, and `FieldList` attributes.

5.4 Option 4 - Full Structure

Your wrapper integration flow provides the complete nested structure with explicit packages.

Example Body

Sample Code

```
<IBPInboundIntegrationGenericProcess
  BatchCommand="INSERT_UPDATE"
  BatchName="Multi-Package Load"
  DestinationforSAPIBP="MY_IBP_DEST">
  <IBPInboundIntegrationDataTransfer>
    <Packages>
      <PackageDefinition>
        <IBPInboundIntegrationDynamicPostData>
          <PackageContent
            DataType="MasterData"
            MasterDataPrefix="IBP"
            MasterDataType="LOCATION"
            FileName="Location Package 1"
            FieldList="LOCID, LOCTYPE, LOCDESCR">
            <item>
              <LOCID>1010</LOCID>
              <LOCTYPE>P</LOCTYPE>
              <LOCDESCR>Plant 1</LOCDESCR>
            </item>
          </PackageContent>
        </IBPInboundIntegrationDynamicPostData>
      </PackageDefinition>
      <PackageDefinition>
        <IBPInboundIntegrationDynamicPostData>
          <PackageContent
            DataType="MasterData"
            MasterDataPrefix="IBP"
            MasterDataType="LOCATION"
            FileName="Location Package 2"
            FieldList="LOCID, LOCTYPE, LOCDESCR">
            <item>
              <LOCID>1020</LOCID>
              <LOCTYPE>P</LOCTYPE>
              <LOCDESCR>Plant 2</LOCDESCR>
            </item>
          </PackageContent>
        </IBPInboundIntegrationDynamicPostData>
      </PackageDefinition>
    </Packages>
  </IBPInboundIntegrationDataTransfer>
</IBPInboundIntegrationGenericProcess>
```

5.5 Advanced Data Load Patterns

Some data load scenarios require special handling beyond a standard single-batch call. This topic covers two patterns: managing large data volumes and loading the same data to multiple planning areas.

Large Data Volumes

Do not send very large payloads directly to the integration flow (iFlow). Large direct payloads can cause memory pressure, timeout failures, and unpredictable behavior under load. As a rule of thumb, use paginated extensions when the total payload would exceed 20 MB or 50,000 records.

Use one of the following patterns instead.

Headers Only with Paginated Extension

Use this pattern when all records come from one source and use the same settings. Set the following headers and let your extension control pagination with `Loop="true"` or `Loop="false"`:

Header	Value
BatchCommand	INSERT_UPDATE
BatchName	Large KF Load
BeforePostDataExtensionID	My_Paginated_Fetcher
DestinationforSAPIBP	MY_IBP_DEST

Body with Multiple Packages and Extensions

Use this pattern when packages have different sources, data types, file names, field lists, or extension logic:

Sample Code

```
<IBPInboundIntegrationGenericProcess
  BatchName="Multi-Source Load"
  BatchCommand="INSERT_UPDATE"
  DestinationforSAPIBP="MY_DEST"
  PlanningArea="MYPA">
  <IBPInboundIntegrationDataTransfer>
    <Packages>
      <PackageDefinition>
        <IBPInboundIntegrationDynamicPostData
          BeforePostDataExtensionID="Fetch_Locations_Paginated"
          DataType="MasterData"
          MasterDataPrefix="IBP"
          MasterDataType="LOCATION" />
        </PackageDefinition>
      <PackageDefinition>
        <IBPInboundIntegrationDynamicPostData
          BeforePostDataExtensionID="Fetch_Forecasts_Paginated"
          DataType="KeyFigure"
          FileName="Forecasts"
          FieldList="PRODID,LOCID,FCSTQTY" />
        </PackageDefinition>
      </Packages>
    </IBPInboundIntegrationDataTransfer>
  </IBPInboundIntegrationGenericProcess>
```

Loading to Multiple Planning Areas

All packages in one automatically managed SAP IBP batch should target the same planning area. If the same source data must be loaded to multiple planning areas, use one of the following patterns.

Solution A: Multi-Batch in a Single Call

The *SAP IBP Write - Inbound Integration Generic Process* integration flow creates and manages the primary batch. Additional planning areas use externally pre-created batches passed by **BatchID** at package level. The following requirements apply:

1. The primary planning area uses the normal Generic Process batch.
2. Additional planning areas use batches created by the caller.
3. Packages targeting external batches specify **BatchID**.
4. The caller manages the lifecycle of external batches.

Sample Code

```
<IBPInboundIntegrationGenericProcess
  BatchName="Multi-PA Load"
  BatchCommand="INSERT_UPDATE"
  DestinationforSAPIBP="MY_DEST"
  PlanningArea="PROD_PA">
  <IBPInboundIntegrationDataTransfer>
    <Packages>
      <PackageDefinition>
        <IBPInboundIntegrationDynamicPostData
          BeforePostDataExtensionID="Fetch_Data_Paginated"
          DataType="KeyFigure"
          FileName="Forecasts" />
        </PackageDefinition>
      <PackageDefinition>
        <IBPInboundIntegrationDynamicPostData
          BatchID="67890"
          BeforePostDataExtensionID="Fetch_Data_Paginated"
          DataType="KeyFigure"
          FileName="Forecasts" />
        </PackageDefinition>
      </Packages>
    </IBPInboundIntegrationDataTransfer>
  </IBPInboundIntegrationGenericProcess>
```

Advantages	Limitations
Single source read	Caller must create and manage additional batches
Single integration call	External batches are not tracked by the Generic Process
Data consistency from the same read	Requires SAP IBP batch API knowledge

Solution B: Data Store Caching

The first call reads the source data and stores it in the SAP Cloud Integration Data Store. Subsequent calls read from the Data Store instead of the source:

1. Call 1 (PA1): your extension reads the source, caches the data to the Data Store, returns the data, and loads it to planning area 1.

2. Call 2 (PA2): your extension reads from the Data Store, returns the data, and loads it to planning area 2.
3. Cleanup: delete the Data Store entry after all loads complete.

Advantages	Limitations
Standard batch handling	Requires multiple integration calls
Simpler orchestration	Data Store size limits apply
No SAP IBP batch API knowledge required	Timing gap between loads

6 Automatic Input Completion

The input completion process transforms valid but incomplete input into the complete internal structure required for reliable SAP IBP posting. It allows callers to use the simplest accepted input format.

The process completes missing structure, allows common attributes to be set at the root and blocks certain attributes from flowing to lower levels.

6.1 Structure Completion

The integration flow (iFlow) sequence accepts incomplete input structures and completes the missing wrapper elements automatically. You can send the simplest structure that fits your use case, and the integration flows add the missing layers before processing starts.

What You Send	What the System Adds
Headers only, no body	IBPINboundIntegrationGenericProcess > IBPINboundIntegrationDataTransfer > Packages > PackageDefinition > IBPINboundIntegrationDynamicPostData
PackageContent directly under root	Wraps it in IBPINboundIntegrationDataTransfer > Packages > PackageDefinition > IBPINboundIntegrationDynamicPostData
IBPINboundIntegrationDynamicPostData directly under root	Wraps it in IBPINboundIntegrationDataTransfer > Packages > PackageDefinition

The following example shows a flat input body and the completed structure the system produces from it:

Flat input

Sample Code

```
<IBPINboundIntegrationGenericProcess
  BatchName="Location Load"
  BatchCommand="INSERT_UPDATE"
  DestinationforSAPIBP="MY_DEST">
  <PackageContent
    DataType="MasterData"
    MasterDataPrefix="IBP"
    MasterDataType="LOCATION"
    FileName="Locations"
    FieldList="LOCID, LOCTYPE">
    <item>
      <LOCID>1010</LOCID>
      <LOCTYPE>P</LOCTYPE>
    </item>
  </PackageContent>
</IBPINboundIntegrationGenericProcess>
```

Completed structure

Sample Code

```
<IBPInboundIntegrationGenericProcess
  BatchName="Location Load"
  BatchCommand="INSERT_UPDATE"
  DestinationforSAPIBP="MY_DEST">
  <IBPInboundIntegrationDataTransfer
    DestinationforSAPIBP="MY_DEST"
    BatchName="Location Load">
    <Packages>
      <PackageDefinition>
        <IBPInboundIntegrationDynamicPostData
          DestinationforSAPIBP="MY_DEST"
          BatchName="Location Load">
          <PackageContent
            DataType="MasterData"
            MasterDataPrefix="IBP"
            MasterDataType="LOCATION"
            FileName="Locations"
            FieldList="LOCID,LOCTYPE">
            <item>
              <LOCID>1010</LOCID>
              <LOCTYPE>P</LOCTYPE>
            </item>
          </PackageContent>
        </IBPInboundIntegrationDynamicPostData>
      </PackageDefinition>
    </Packages>
  </IBPInboundIntegrationDataTransfer>
</IBPInboundIntegrationGenericProcess>
```

6.2 Attribute Inheritance

Attributes set on parent XML elements are automatically inherited by child elements in the SAP Integrated Business Planning (SAP IBP) inbound integration flow (iFlow) sequence, using nearest-ancestor-wins semantics. For example, if you set **DestinationforSAPIBP** on the root **IBPInboundIntegrationGenericProcess** element, it flows down to **IBPInboundIntegrationDataTransfer** and **IBPInboundIntegrationDynamicPostData** unless explicitly overridden at a lower level.

The inheritance uses a block-list approach where each element type blocks specific attributes from inheritance, see [Attribute Blocking \[page 29\]](#)

All other attributes not in the blocked list are inherited from the nearest ancestor that has them.

6.3 Attribute Blocking

The inheritance uses a block-list approach where each element type blocks specific attributes from inheritance. Without blocking, attributes set on parent elements would be copied to all child elements

regardless of their role. Some attributes are meaningful only at a specific processing layer and would be ambiguous or cause incorrect behavior if they appeared at a lower level.

Element	Blocked Attributes
IBPInboundIntegrationDataTransfer	AfterDynamicDataLoadExtensionPoint, BatchCommand, BeforeDynamicDataLoadExtensionPoint, PlanningArea, PlanningAreaVersion, ProcessUnchangedData
IBPInboundIntegrationDynamicPostData	ProcessPackagesinParallel
PackageContent	ProcessPackagesinParallel
PackageDefinition	ProcessPackagesinParallel

All other attributes not in the blocked list are inherited from the nearest ancestor that has them.

6.4 Attribute Placement Pattern

Attributes in the integration flow (iFlow) sequence work as defaults and overrides. You set common values once at the root element, and individual packages override only the attributes that differ. You don't need to repeat shared values on every package.

Level	Role	Typical Attributes
Root - IBPInboundIntegrationGenericProcess	Global defaults	DestinationforSAPIBP, BatchName, BatchCommand
Package - IBPInboundIntegrationDynamicPostData	Package-specific overrides	DataType, FileName, BeforePostDataExtensionID
Content - PackageContent	Data-specific settings	FieldList, MasterDataType

The following example sets common values at the root and uses two packages with different data types:

Sample Code

```
<IBPInboundIntegrationGenericProcess
  DestinationforSAPIBP="MY_DEST"
  BatchName="Daily Load"
  BatchCommand="INSERT_UPDATE"
  PlanningArea="MYPA">
  <IBPInboundIntegrationDynamicPostData
    DataType="MasterData"
    MasterDataType="LOCATION" />
  <IBPInboundIntegrationDynamicPostData
    DataType="KeyFigure"
    FileName="Forecasts" />
```

```
</IBPInboundIntegrationGenericProcess>
```

6.5 Extension Attribute Preservation

When a package extension returns a modified body, it may omit technical attributes that were present on the original package before the extension ran. The integration flow (iFlow) sequence restores those missing root-level attributes automatically so extensions can focus on the data without having to repeat package context.

The following steps describe how attribute preservation works:

1. Before the package extension runs, the iFlow stores the package root attributes.
2. The extension returns a modified package body.
3. The iFlow restores any missing package root attributes from the stored values.
4. Attributes explicitly returned by the extension are kept as-is.

Preserved attributes include **BatchID**, **BatchName**, **DestinationforSAPIBP**, **PlanningArea**, **DataType**, **FileName**, and **FieldList**, when those values were already part of the package context.

Preserved	Not Preserved
Missing attributes on the returned package root	Child elements or child attributes removed by the extension
Original package root attributes that still have a value	Attributes that were not part of the original package context
Attributes explicitly returned by the extension (returned values win)	Attributes on a response that no longer uses the expected package root

7 Response and Error Handling

This section describes the output format and error handling behavior of the integration flow sequence.

7.1 Output

Success Response

On successful completion, the response preserves the original root element with all attributes, includes any messages, and provides processing statistics:

❖ Example

```
<IBPInboundIntegrationGenericProcess
BatchCommand="INSERT_UPDATE"
BatchName="Location Load"
BatchID="BATCH-001"
DestinationforSAPIBP="MY_IBP_DEST"
ExceptionOccurred="false">
<Messages>
<!-- Any warnings from SAP IBP -->
</Messages>
<ProcessPostedDataResponse
FileCount="1"
SuccessfulFileCount="1"
TotalErrorRecordCount="0"
TotalRecordCount="2"
WorstStatus="OK">
<Raw><!-- Original output body of SAP IBP Write - Process Posted Data --></Raw>
</ProcessPostedDataResponse>
</IBPInboundIntegrationGenericProcess>
```

ProcessPostedDataResponse Attributes

Attribute ID	Description
FileCount	Number of files or packages processed.
SuccessfulFileCount	Number of files processed without errors.
TotalErrorRecordCount	Number of records that failed.

Attribute ID	Description
TotalRecordCount	Total number of records processed.
WorstStatus	Overall status. Accepted values: ERROR, OK, PROCESSED_WITH_ERRORS .

Error Response

On error, `ExceptionOccurred="true"` is set and error messages are included:

❧ Example

```
<IBPInboundIntegrationGenericProcess
  BatchCommand="INSERT_UPDATE"
  BatchName="Location Load"
  DestinationforSAPIBP="MY_IBP_DEST"
  ExceptionOccurred="true">
  <Messages>
    <Message Type="Error"
      Origin="SAP_IBP_Write_-_Inbound_Integration_Generic_Process_Stage_CreateBatches">
      Planning area INVALID not found
    </Message>
  </Messages>
</IBPInboundIntegrationGenericProcess>
```

Response Attributes

Attribute	Description
BatchID	Assigned batch identifier. Present on success.
ExceptionOccurred	true if any error occurred, false otherwise.
Original attributes	All input attributes are preserved in the response.

7.2 Error Handling

Standardized Error Structure

All three integration flows (iFlows) return errors in the same format. The root element name and original attributes are preserved for correlation. The `Origin` attribute on each message indicates where the error occurred.

Message Types

The following message types are available:

Type	Severity	Behavior
Abort	Highest	Stops processing immediately and triggers escalation.
Error	High	Included in the response. May stop processing.
Info / Information	Low	Filtered out of the final response.
Warning	Medium	Included in the response. Processing continues.

Error vs. Escalation

The integration flows distinguish between two error types:

Type	When	End Event
Error	Unhandled exception with no escalation headers.	Error End Event
Escalation	First exception with escalation headers (EndEvent_*EscalationCode or EndEvent_*EscalationMessage).	Escalation End Event

Escalation is used for controlled failures that should trigger specific alerting or retry logic.

8 Troubleshooting

Common Errors

The following errors may occur:

Error	Cause	Solution
"BatchCommand is required"	Missing or invalid batch command.	Use DELETE , INSERT_UPDATE , or REPLACE .
"Both BatchID and DestinationforSAPIBP must be provided"	The SAP IBP Write - Inbound Integration Dynamic Post Data integration flow was called directly without context.	Ensure the batch ID is available from the parent integration flow.
"DataType must be MasterData or KeyFigure"	Invalid DataType attribute.	Use KeyFigure or MasterData (case-insensitive).
"DestinationforSAPIBP is required"	Missing destination on the root element.	Add the DestinationforSAPIBP attribute to the root element.
"Invalid BatchCommand 'X'. Valid values: DELETE, INSERT_UPDATE, REPLACE"	Typo in BatchCommand .	Check the exact spelling.
"Planning area not found"	Invalid PlanningArea in SAP IBP.	Verify the planning area exists in SAP IBP.
"'PlanningAreaVersion' requires 'PlanningArea' to be set"	Version set without planning area.	Add the PlanningArea attribute.

Extension Integration Flow Not Called

If your extension integration flow isn't called, check the following:

- The [ProcessDirect](#) address matches exactly (case-sensitive).
- The extension integration flow is deployed.
- The address format is valid: alphanumeric characters, underscores, and hyphens only.
- The extension attribute is set on the correct XML element level.

Interpreting Error Origins

The `Origin` attribute in error messages tells you which integration flow and stage failed:

Origin Contains	Failed In
<code>Data_Transfer_Stage_DynamicPostData</code>	<i>SAP IBP Write - Inbound Integration Dynamic Post Data</i> integration flow
<code>Dynamic_Post_Data_Stage_PostData</code>	<i>SAP IBP Write - Post Data</i> integration flow
<code>Generic_Process_Stage_CreateBatches</code>	<i>SAP IBP Write - Create Batches</i> integration flow
<code>Generic_Process_Stage_DataTransfer</code>	<i>SAP IBP Write - Inbound Integration Data Transfer</i> integration flow

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.

About the icons:

- Links with the icon  : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
 - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
 - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon  : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

© 2026 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <https://www.sap.com/about/legal/trademark.html> for additional trademark information and notices.

