STIBO SYSTEMS
MASTER DATA MANAGEMENT

USER GUIDE

Data as a Service

Release 11.0-MP4 (September 21, 2022)

Table of Contents

Data as a Service

Data as a Service, also known as DaaS, is a cloud-based software tool that allows data to be accessed on demand.

STEP supports data publication with DaaS. Data is published from STEP via a single event processor to a Cosmos database on the DaaS service side using a generic STEP schema. On the service side, it is possible to define multiple services that draw upon the same data, but expose it to consumer systems in different ways via different GraphQL APIs.

Data can be published from different STEP contexts, allowing consumers to query specific context-dependent versions of the data. For information on configuring STEP to publish to the Stibo Systems' Data as a Service (DaaS) service, refer to the **Publishing to Data as a Service** topic.

Publishing to Data as a Service

This topic describes how to configure STEP to publish to the Stibo Systems' Data as a Service (DaaS) service, available as a cloud native service for Microsoft Azure.

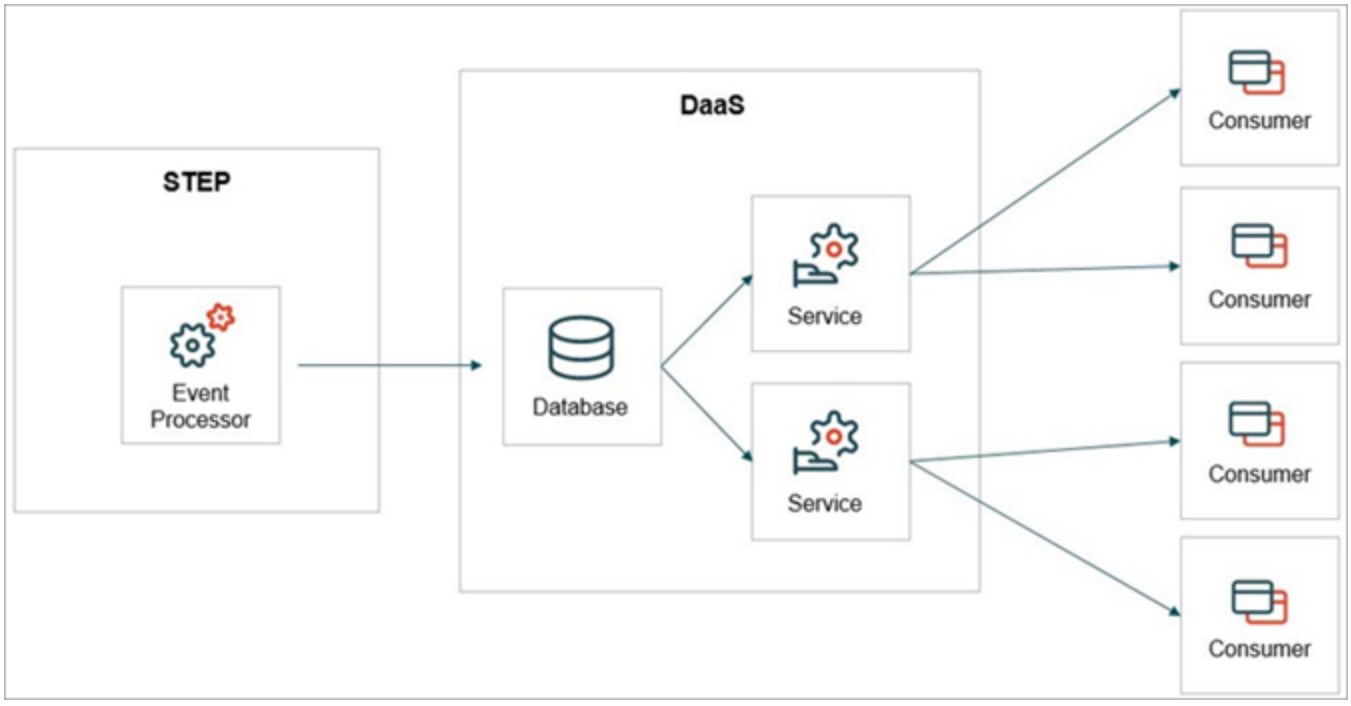
Documentation for the DaaS service is available directly from the service. Thus, this topic focuses on the STEP configuration required to publish data to DaaS.

Note: The Data as a Service commercial license must be enabled to use this functionality. DaaS is still in a ramp-up phase, so not all customers will have access to it. To learn more about the ramp-up phase / status, refer to the **License and Component Lifecycle** in the **System Release and Patch Notes** section of online help.

Data is published to DaaS via the 'DaaS Event Processor' event processor (EP) plugin, which uses the 'Azure DaaS' GIEP plugin. Subsequent sections describe how to configure the plugins. A basic understanding of how the DaaS solution works is required to successfully configure the plugins.

DaaS Basics

The diagram below illustrates the overall DaaS concept. With DaaS, data is published from STEP via a single EP to a Cosmos database on the DaaS service side using a generic STEP schema. On the service side, it is possible to define multiple different services that draw upon the same data but expose it to consumer systems in different ways via different GraphQL APIs. Data can be published from multiple different STEP contexts, allowing consumers to query specific context dependent versions of the data.



With the current release of DaaS, the service can handle objects of the following STEP super types published from the Approved workspace:

- Product
- Entity
- Classification
- Asset (STEP object, not binary content)
- Attribute
- Unit
- Attribute Group (called Data Type Group in DaaS)
- Reference and Classification Product Link Type (grouped as Reference Type in DaaS)
- Data Container Type
- List of Values (a maximum of 5,000 list of values are published)

For product, entity, classification, and asset data node types, the following data is published:

- ID
- Name
- Parent ID
- Object type ID
- Attribute values
- Outgoing references (including metadata)
- Local outgoing attribute links (including metadata)

Note: Outgoing classification product links (including metadata) are grouped with references in DaaS.

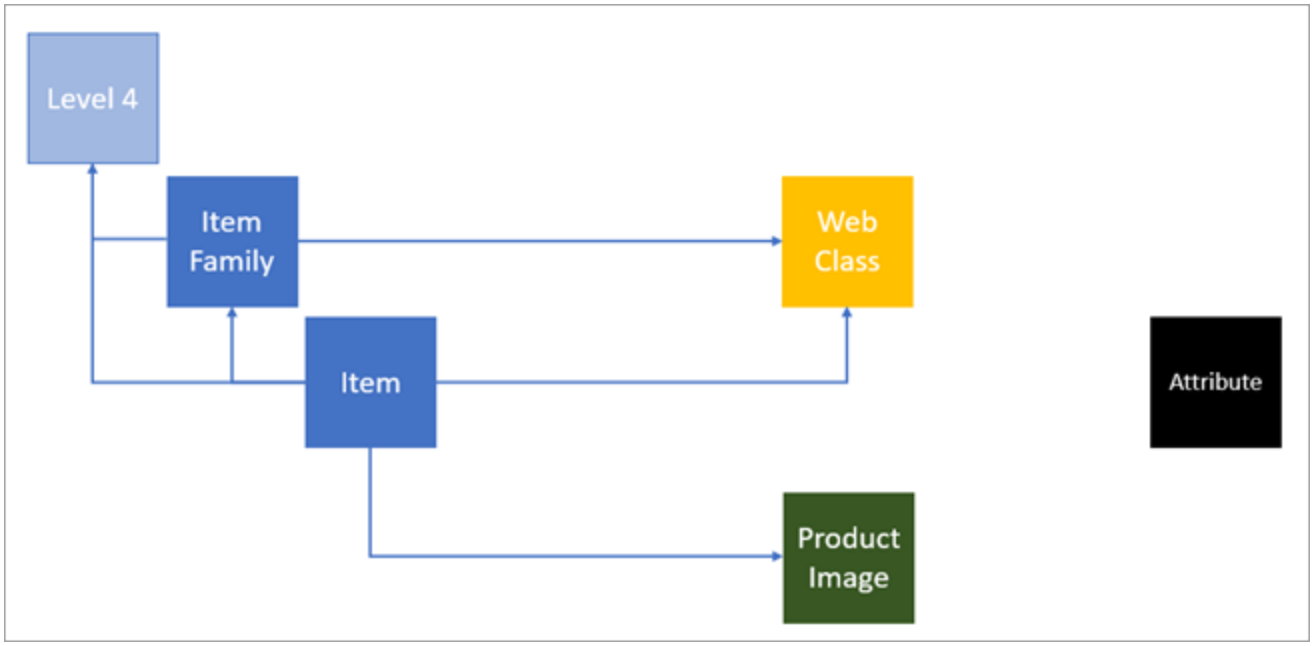
For the remaining configuration types, ID, Name, attribute values and attribute group membership are published.

Important: Only a single event processor should be configured to publish to a DaaS instance.

The DaaS service offers functionality for denormalizing data when exposed to consumers, but this denormalization is performed at runtime and data is, except for hierarchically inherited data that can be localized (detailed below), not denormalized during the publishing to DaaS. This generally means that if data owned by an object is to be exposed via DaaS, the object must be published.

The first step for configuring the publishing of data to DaaS is to define the subset of the STEP model that needs to be exposed. For many PMDM use cases, the data to expose via DaaS is centered around the Item level in the product hierarchy, potentially including data on the variant and/or family level.

For this document, the example model shown below is used.



In this example, the two relevant product object types are 'Item Family' and 'Item' and they both inherit data hierarchically from Level 4 instances, with 'Item' instances also inheriting from 'Item Family' instances. The Level 4 type is dimmed in the illustration to indicate that instances of this type are not published.

Both product object types can be linked to classifications of the 'Web Class' type that, for the sake of the example, are assumed to hold data that must be exposed in DaaS. 'Item' instances can additionally reference assets of the 'Product Image' type that also hold data that must be exposed.

Finally, when attribute values are exposed via DaaS, it is assumed that attribute names are also included. Since attribute names in STEP are owned by the attribute objects, these must be published as well.

The presented model is used in the following configuration examples in the 'DaaS Event Processor' section.

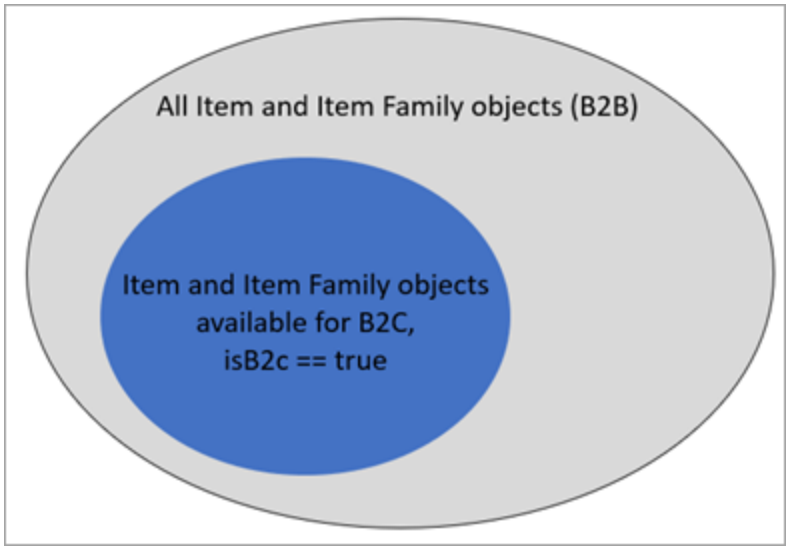
Horizontal Filtering (Assortments)

Publishing to DaaS is handled via an EP plugin, where it is possible to use the standard EP functionality for filtering events, enabling objects with specific characteristics to go into the associated event queue, while also filtering away other objects (i.e., object type and business rule-based filters that are configured on the workbench Event Triggering Definitions tab). However, it is sometimes a requirement that different DaaS service consumer systems are exposed to different subsets of data.

For example, if only a subset of the full product assortment can be sold directly to consumers (B2C) while the full assortment (or a different subset) is available to businesses (B2B). In DaaS, it is possible to handle such a setup via a concept known as 'assortments.'

Assortments can be defined in the publishing setup and the DaaS service can be configured so that different endpoints (also known as services) have access to specific assortments. Refer to the DaaS service

documentation for details on how to configure the endpoints on the DaaS side. On the publishing side, the assortment filtering can be configured to only apply for objects of specific types, filtering on values for a configured attribute. The diagram below illustrates the concept.



Additional information about how to configure assortments can be found in the **DaaS Event Processor** section below.

Vertical Filtering

'Vertical filtering' is configuring what data for the individual objects should be exposed to consumers. The options for configuring vertical filtering include:

- On the DaaS service side, define a GraphQL schema for a service that does not expose fields for data that should not be exposed to consumers (refer to the DaaS service documentation for details on configuration).
- On the STEP side, for data that should never be exposed to any consumer via DaaS, run event processor from a system user account that is not privileged to view / read that specific data.

Azure DaaS GIEP Configuration

Communication between STEP and DaaS is handled via the 'Azure DaaS' gateway integration endpoint (GIEP) plugin.

To configure a DaaS GIEP:

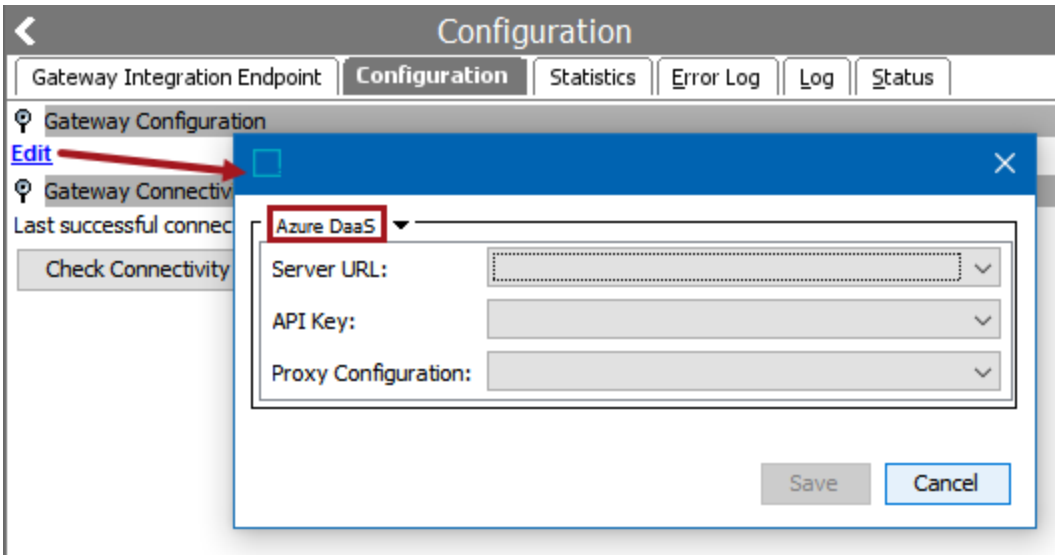
1. Obtain a management API key from Stibo Systems.
2. Set the following configuration properties in the sharedconfig.properties file:

```
DaaS.Azure.Endpoint.<integer>=<endpoint-url>
DaaS.Azure.FunctionKey.<integer>=<api-key-name>,<mgmt-api-key>
```

For example:

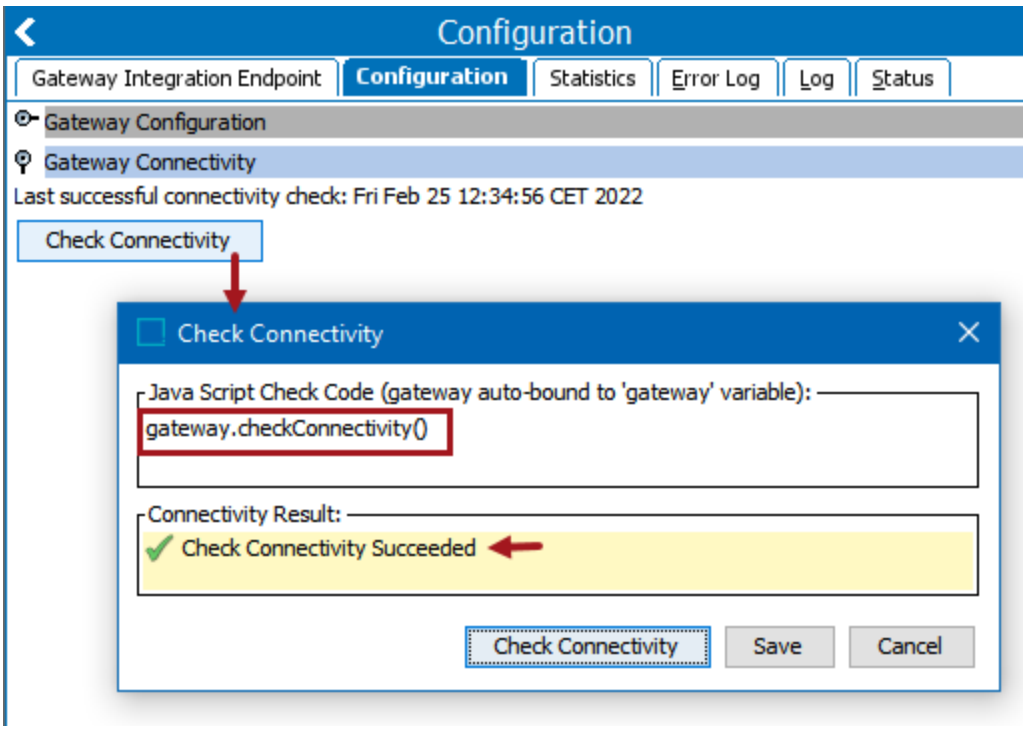
```
DaaS.Azure.Endpoint.1=https://my-endpoint.com
DaaS.Azure.FunctionKey.1=APIKey1,MyAPIKey
```

3. Restart the STEP system.
4. In System Setup, create a new GIEP. For details, refer to the **Creating a Gateway Integration Endpoint** topic in the **Data Exchange** documentation.
5. On the Configuration tab, open the Gateway Configuration flipper, click the **Edit** link, and select the following elements:



   - From the dropdown, select the 'Azure DaaS' plugin.
   - Select the appropriate 'Server URL' from the dropdown.
   - select the appropriate 'API Key' from the dropdown.
6. Before proceeding, it is strongly recommended to test the connection from the gateway to Microsoft Azure.
   - On the Gateway Connectivity, click the **Check Connectivity** button.
   - In the Check Connectivity dialog, in the Java Script Check Code section, add:

     ```
     gateway.checkConnectivity()
     ```
7. Click the **Check Connectivity** button and verify that the check passes with a green checkmark, as shown below.



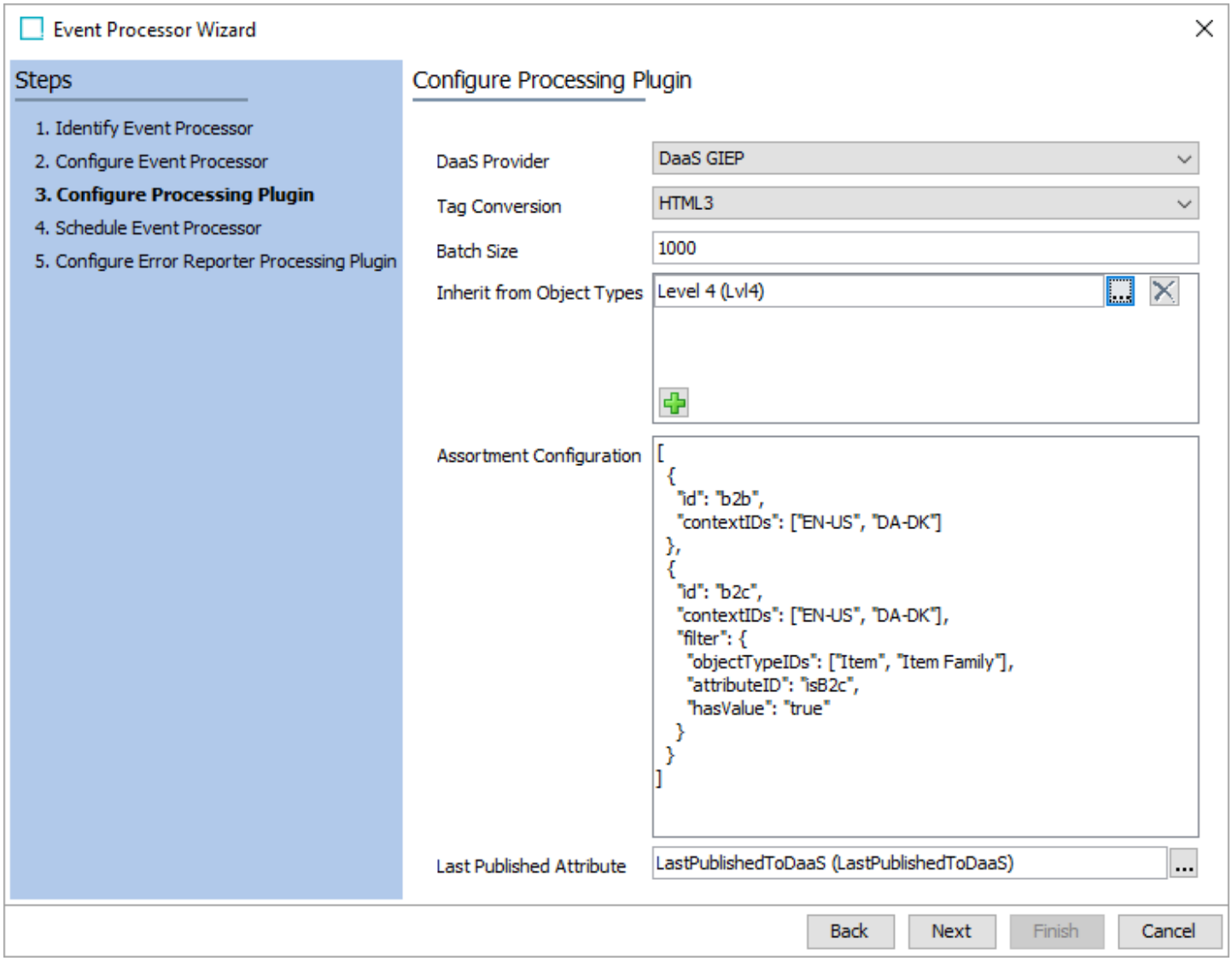
8. Right-click the new GIEP (with a yellow icon) and click the **Enable Integration Endpoint** option to update it with a green icon ().

DaaS Event Processor

After configuring a GIEP for handling the communication with DaaS, configure the following parameters on the DaaS Event Processor plugin.

Event Processor Configuration

On the DaaS EP, open the Configuration flipper and click the **Edit Configuration** link to open the Event Processor Wizard dialog. Add the selections defined below.



- **DaaS Provider** - The GIEP to handle the communication with DaaS. Only GIEPs that use the Azure DaaS plugin can be selected.
- **Batch Size** - The number of events to handle in each batch. Defaults to 1,000. Consult with Stibo Systems if the batch event number should be modified.
- **Inherit from Object Types** - As mentioned above, hierarchically inherited data can be localized when objects are published to DaaS. As with all event-based publishing, take care to ensure that relevant data objects are republished when hierarchically inherited data changes. Given that with DaaS, non-trivial configuration is required for this purpose (refer to **Event Triggering Definitions** section below), when configuring the EP plugin, a conscious decision is required about which hierarchy levels to inherit data from.

  The 'Inherit from Object Types' parameter allows the user to select the uppermost object type in the object type hierarchy from which data should be inherited from and localized during publishing. If no object types are selected, inherited data will not be published.

Note: It is strongly recommended to not localize and publish data inherited from upper levels in the data hierarchies, as changes to such data can require that vast amounts of objects need to be republished.

To illustrate the Inherit from Object Types logic, consider the diagram below.

In this example, 'Warranty' values owned by instances of the 'Level 4' object type and inherited by 'Item Family' and 'Item' instances are to be localized and published to DaaS. For this purpose, the 'Level 4' object type must be selected for the 'Inherit from Object Types' parameter although instances of this type are not themselves to be published. Since the selection is the uppermost type in the object type hierarchy, it is not necessary to also select 'Item Family for the parameter. When publishing an 'Item,' values from both the 'Item Family' and 'Level 4' level are localized.

A setup like this requires more configuration to ensure that descendant objects are republished when data inherited to these changes. This configuration is done via the event processor Event Triggering Definitions, which is described in the **Event Triggering Definitions** section below.

- **Assortment Configuration** - The text field holds a JSON data structure that defines assortments and, for each assortment, specifies which contexts to publish data for.

Note: You must specify an assortment and the list of contexts to publish data from.

If there is no requirement for assortment segregation, the configuration is simple, for example:

```
[
  {
    "id": "default",
    "contextIDs": ["EN-US", "DA-DK"]
  }
]
```

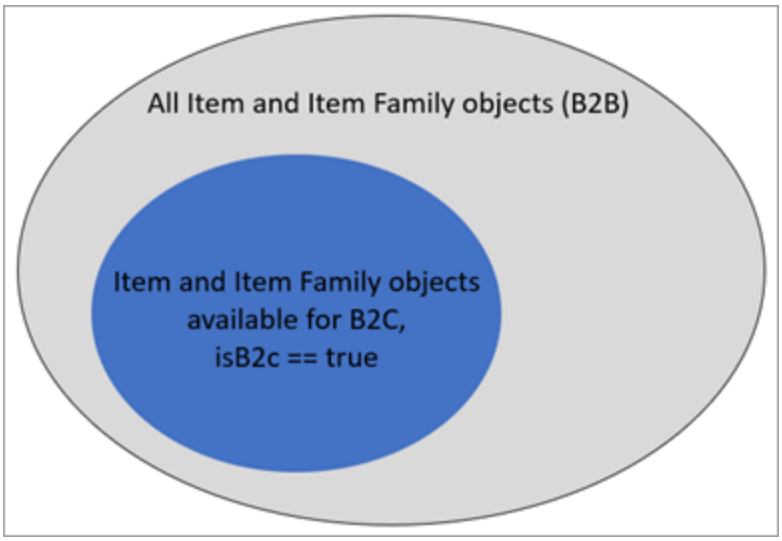
In this simple example, the data structure defines a single assortment with ID of 'default' and specifies that data for this assortment should be published from the 'EN-US' and 'DA-DK' contexts.

The more advanced example below has two assortments matching the assortment example from above.

```
[
  {
    "id": "b2b",
    "contextIDs": ["EN-US", "DA-DK"]
  },
  {
    "id": "b2c",
    "contextIDs": ["EN-US", "DA-DK"],
    "filter": {
      "objectTypeIDs": ["Item", "Item Family"],
      "attributeID": "isB2c",
      "hasValue": "true"
    }
  }
]
```

In this more advanced example, assortments 'b2b' and 'b2c' are defined.

- For the 'b2b' assortment, no filtering is configured which means that all objects handled by the EP plugin are published to this assortment.
- For the 'b2c' assortment, a filter is configured for objects of the object types 'Item' and 'Item Family' so that only instances of these types that have the value of 'true' for the attribute 'isB2c' are published to the assortment. The filter only affects objects of the specified types, which means objects of other types handled by the EP are published to the assortment. The attribute used for filtering can be calculated and the value matching uses 'contains' logic (meaning, it can match a single value in a multi-valued attribute).



- **Last Published Attribute** - If desired, select a description attribute that is externally maintained, non-dimension dependent, ISO Date and Time validated. If configured, upon successful publishing, nodes for which the selected attribute is valid are updated with the date and time of the publishing. This value is not

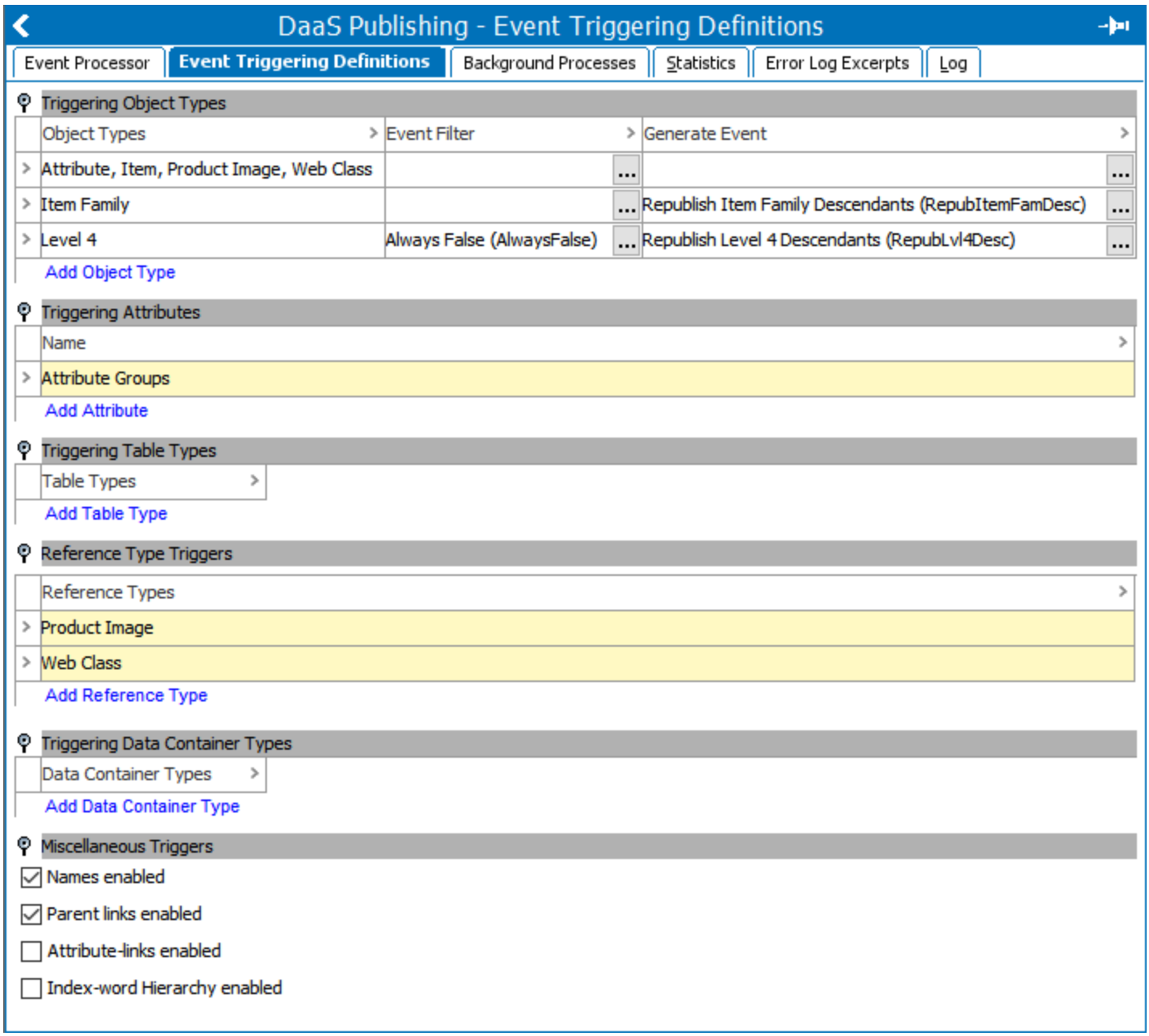
published to DaaS.

Important: The event processor must not be configured to listen for changes to the attribute identified in the Last Published Attribute parameter.

Event Triggering Definitions

On the DaaS EP, open the Event Triggering Definitions tab to determine which events go onto the EP event queue and which objects are published to DaaS.

The image below shows the required setup for the example model from the **DaaS Basics** section above.



In this example, the Triggering Object Types flipper includes:

- Row 1 - Object types for the object types for which no additional logic is necessary. For the Attribute type, events are queued whenever an attribute is created / modified / deleted, or when republish events are generated. For the product (Item), asset (Product Image), and classification (Web Class) types, events are queued when instances of these types are approved and data that matches the remaining triggering definitions has changed, or when publish / derived events are produced.

- Row 2 - Events for the Item Family type are queued following the same logic as for the types in the first row. However, since Item Family objects can hold data that is inherited by Item objects, whenever there are changes to such data in the Approved workspace, descendant Item objects must be republished via a JavaScript-based business action selected in the Generate Event column. For the Item Family object for which an event is produced, the business action logic must find all descendant Item objects that exist in the Approved workspace and generate derived events for these.
- Row 3 - Includes logic that produces derived events for Item Family objects, in order not to have Item objects republished multiple times based on the same change, the business action must be configured so that it does not produce derived events for descendants when the current event itself is a derived event.

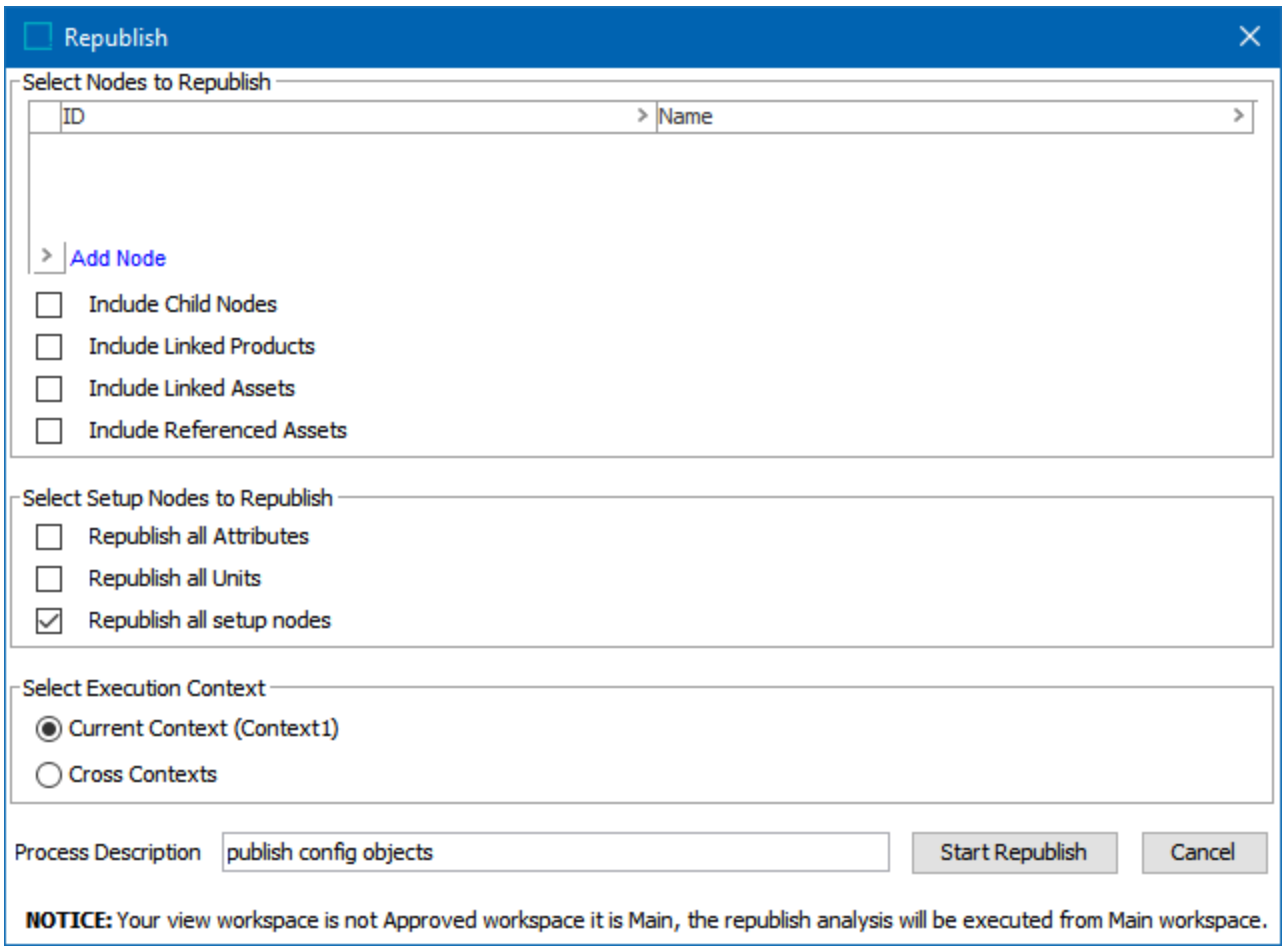
  Level 4 objects are not to be published to DaaS, but as they hold data inherited by objects that are published, logic must be configured to ensure that descendants are republished when said data changes via a JavaScript-based business action selected in the Generate Event column. Similar to the action configured for Item Family objects, this action must find all descendant Item and Item Family objects that exist in the Approved workspace and generate derived events for these. Additionally, since Level 4 objects are not to be published, a business condition that always returns 'false' must be configured in the Event Filter column.

JavaScript business rule examples are included in the **Event-Based Example Leaf Products with Inherited Data** topic and the **Event-Based Example Products Linked to Classification** topic, both in the **Data Exchange** documentation.

Publishing

Once the GIEP and EPs have been configured, data can be published to DaaS. For the initial publishing of data objects (products, classifications, entities, and assets), the recommended approach is to generate republish events via the Send Republish Event bulk update plugin. Bulk updates can be run on the contents of a collection or directly on a search result. For example, a simple approach is to go to the Approved workspace in the STEP workbench and run object type searches for the types that need to be published followed by a bulk update. The DaaS service is eventually consistent, meaning that the sequence in which data is published is not important.

From 10.2-MP2, the Send Republish Event bulk update plugin can be used for configuration objects like attributes and reference types. Alternatively, the Republish functionality is available from the EP context menu.



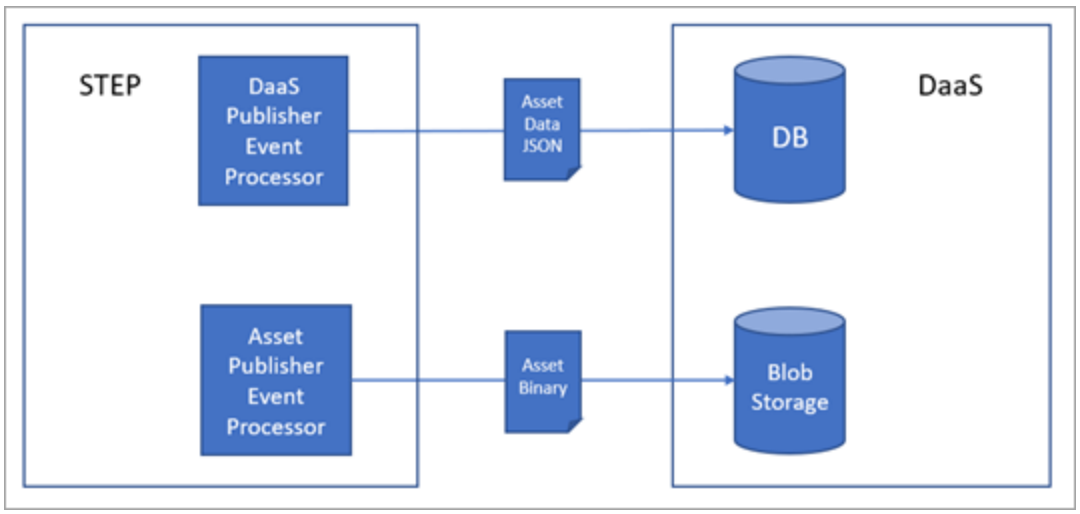
Note: For the initial publishing of data to DaaS, disable the event processor Generate Event business actions. If not disabled, there is the possibility of queuing a large number of duplicate events.

Asset Content via DaaS

Asset binaries (originals and conversions) can be published to DaaS using the Asset Publisher functionality described in the **Asset Publisher** topic of the **Digital Assets** documentation.

This section describes the STEP side of the solution while the DaaS functionality is described on the DaaS help page.

The diagram below illustrates the high-level concept.



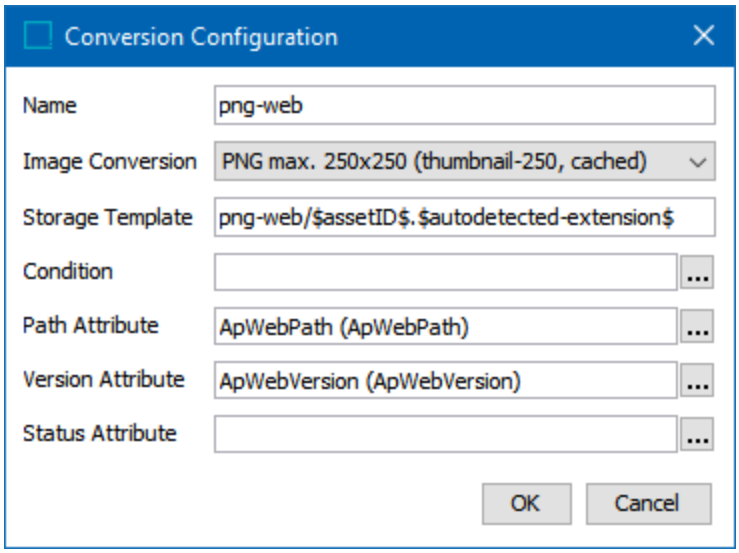
Asset binaries are published to a DaaS-managed blob storage account via the Asset Publisher, which is further responsible for writing information to the STEP asset objects, that when published to DaaS, allows the DaaS functionality to identify the content in blob storage and return it to consumers.

Configure

The Asset Publisher used for this purpose must be configured with a Microsoft Azure Blob Storage GIEP. Stibo Systems provides the connection string and container name for the configuration.

For details on configuring the Asset Publisher, refer to the **Asset Publisher Processing Plugin Parameters and Triggers** topic in the **System Setup** documentation.

When the Asset Publisher is used for DaaS, the following parameters are required:



- **Storage Template** - (required) Enter the path for the conversion configuration. The path must start with the name of the conversion configuration and be followed by a forward slash, as shown in the screenshot above

(png-web/).

- **Version Attribute** - (required) Select an attribute valid for the asset object types to be published.
- **Path Attribute** - (required) Select a description attribute that holds the information that the DaaS functionality uses to identify the appropriate binary in blob storage. The value for this attribute is crucial for the integration. The path attribute value is published to DaaS along with other asset object details.

  Note: For the asset content integration to work, the user account used for the DaaS publisher EP must have view permissions for the 'Path Attribute' description attribute.

  The Path Attribute is populated by the Asset Publisher upon successful publishing. To ensure DaaS has up-to-date information, the asset objects for which binary data is made available via DaaS must be published (or republished) to DaaS whenever content changes in the Approved workspace.

Publish

Choose one of the following methods to handle publishing the Path Attribute:

Important: Do not configure both of these options at the same time.

- Configure the DaaS publisher 'Event Triggering Definitions' to listen on changes to the Path Attribute and the Version Attribute configured in the Asset Publisher.
- Configure the DaaS publishing EP in the 'Notification Event Queues' parameter of the Asset Publisher

configuration as shown in the image below.