DataArt

Smart Import Tool

Developer Guide

Contents

1. Overview

The Smart Import Tool exposes an API over Lightning Message Service (LMS). This API enables you to build a custom LWC Extension component that:

- **Observes** app events (one-way), perfect for logging, analytics, or state tracking.
- **Influences** app behavior (two-way), where your Extension can intercept certain events and reply with data or decisions that shape how the Smart Import Tool proceeds.

In a typical setup, the Extension lives in a Lightning tab (or App Page), renders the Smart Import Tool component, and uses LMS to exchange messages with it. The API centers around a single SmartImportToolEvent message channel that carries different types of events. Messages can be handled asynchronously (fire-and-forget) or synchronously (Smart Import Tool waits for Extension response). In the meantime, the Extension can access its Apex controller to interact with the database for logging, data validation and enrichment.

Prerequisites

- The Smart Import Tool package is installed in the org (Sandbox / Scratch Org / Developer Org where you can deploy and run LWC).
- 'LMS ENABLED' setting is 'TRUE' in "Smart Import Tool Setting" Custom Metadata Type (see Admin Guide) to allow the app to emit LMS messages.
- Lightning Web Security is enabled. See Setup -> Session Settings -> “Use Lightning Web Security for Lightning web components and Aura components” checkbox is checked.
- Working knowledge of Lightning Web Components (LWC) and familiarity with Lightning Message Service basics (publish/subscribe).

2. Event Message Schema

All Smart Import Tool messages share one SmartImportToolEvent__c channel and use a type discriminator:

- **type** — Message type. Examples: AppReady, OperationSelected, ObjectSelected, FileSelected, DocumentLoaded, and *.Response for synchronous replies.
- **source** — Who sent the message, e.g., SmartImportTool or extension.
- **payload** — A JSON object with type-specific data (can be empty for some responses).
- **correlationId** — A GUID used to tie a request to its response (critical for synchronous flows).

Because multiple event types travel over one channel, always inspect type and branch your handler accordingly.

3. Subscribing to events

Subscribing to events can be implemented in two ways: asynchronously (by default) or synchronously, so that the app waits for *.Response message. You can combine both ways for different types of events.

Asynchronous (Observe-Only)

You don’t have to do anything special - just subscribe LMS channel and react to whatever the app publishes. The app never waits on you, so use this mode for logging, analytics, and status updates.

Events that run in asynchronous mode have an empty correlationId as it is not used.

Synchronous (Influence Behavior)

To influence specific app events, you must first register interest by sending a Subscribe message to the channel. The app will reply with Subscribe.Response indicating success or failure. For the events you subscribe to, the Smart Import Tool will pause during those events and expect your *.Response message with the same correlationId. If no response is received within 10 seconds, the application will continue to run.

Send a Subscribe request (example content):

```
{
  type: 'Subscribe',
  source: 'extension',
  correlationId: '5497f7c4-d0c5-4aba-b7c9-dad2af46b0f2',
  payload: {
    events: ['FileSelected', 'DocumentLoaded']
  }
}
```

Key points:

- Use a unique correlationId (e.g., generate with crypto.randomUUID).
- payload.events lists the event types you want to influence (not just observe).

Smart Import Tool response format:

```
{
   type: 'Subscribe.Response',
   source: ' SmartImportTool',
   correlationId: '5497f7c4-d0c5-4aba-b7c9-dad2af46b0f2',
   payload: {
      success: true
   }
}
```

On success subscription payload.success is true, otherwise payload.success is false and payload.errorMessage contains the details.

Important: Only subscribe to events you genuinely plan to handle synchronously. You must reply to every subscribed event with *.Response message. Doing otherwise can slow down the app and degrade user experience. Because the Smart Import Tool will wait for a response each time until the 10 seconds timeout.

Optionally, you can unsubscribe from events that you no longer need. Send an Unsubscribe event with a list of events you want to unsubscribe from. If the payload is empty, it will be interpreted as you want to unsubscribe from all previously subscribed events.

```
{
   type: 'Unsubscribe',
   source: 'extension',
   correlationId: null,
   payload: {
      events: ['FileSelected', 'DocumentLoaded']
   }
}
```

The request is always considered successful. So, the Unsubscribe.Response event's payload.success is always true, even if you were not subscribed to any of the events listed.

There's really no need to unsubscribe when closing the extension. Consider this option only for complex scenarios with dynamic subscription/unsubscription based on condition.

4. Event Types

Below is presented the payload for each event type, as well as the response format for synchronous communication.

AppReady

The event is sent when application is fully initialized and ready to work. Payload is empty. AppReady is always asynchronous. That is, there is no point in subscribing to the event and sending AppReady.Response message. You can use the event to send your Subscribe request to make sure that the application is ready to process it.

OperationSelected

The event is sent when user selected an insert/update operation, or it was selected automatically if there is only one option. Payload contains the selected operation.

OperationSelected payload:

```
{
  operation: "insert"
}
```

With a synchronous response, you can optionally cancel the operation, thus returning the user back to the operation selection. To allow the app to continue, send reject: false or empty payload.

OperationSelected.Response payload:

```
{
  reject: true
}
```

Cancel operation can be used if the user is actually allowed to perform the operation according to their profile, but you do not want to allow it to be performed using the Smart Import Tool.

ObjectSelected

The event is sent when user selected an object to import, or it was selected automatically if there is only one option. Payload contains the selected object, that is JSON object with name (API Name) and label attributes.

ObjectSelected payload:

```
{
  object: {
    name: "Account",
    label: "Account"
  }
}
```

With a synchronous response, you can optionally cancel the operation, thus returning the user back to the operation selection. To allow the app to continue, send reject: false or empty payload.

ObjectSelected.Response payload:

```
{
  reject: true
}
```

Cancel operation can be used if the user is actually allowed to perform the insert/update operation in the selected object according to their profile, but you do not want to allow it to be performed using the Smart Import Tool. Please provide clear Toast message to describe the reason of the cancelation.

FileSelected

The event is sent when user uploaded a file. Payload contains the file name.

FileSelected payload:

```
{
  name: "FileName.xlsx"
}
```

With a synchronous response, you can optionally cancel the operation, thus returning the user back to the file uploading step. To allow the app to continue, send reject: false or empty payload.

FileSelected.Response payload:

```
{
  reject: true
}
```

Cancel operation can be used if you're tracking unique file names and this one has already been imported.

DocumentLoaded

The event is sent when a file is loaded and processed. Smart Import Tool removes empty rows and columns. It recovers damaged files if a row has fewer columns than a longer row. All cells are represented as strings, including empty cells that become empty strings. Payload contains cleaned data presented as an array of arrays.

DocumentLoaded payload:

```
{
  data: [
    ["Column 1", "Column 2"],
    ["foo", "bar"]
  ]
}
```

With a synchronous response, you can optionally cancel the operation as other events.

DocumentLoaded.Response payload:

```
{
  reject: true
}
```

What's more interesting is that you can modify the data. You can implement your own data transformation that is not provided by the Smart Import Tool. Send modified data in the same format. For example, you can transform the cell values into uppercase.

DocumentLoaded.Response payload:

```
{
  data: [
    ["Column 1", "Column 2"],
    ["FOO", "BAR"]
  ]
}
```

You can also change the number of rows and columns, for example, automatically remove the document header.

ColumnsMatched

The event is sent when a document was loaded and the tool matched the Salesforce fields corresponding to the columns. The tool attempts to determine the most suitable field based on the column name. However, some columns may still have empty fields if no suitable field is found.

Payload contains array of columns. Each column contains an index - a number starting from 0, a value - the column name that is always in uppercase, and a field - the API Name of the Salesforce field matched.

ColumnsMatched payload:

```
{
  columns: [
    {index: 0, value: "NAME", field: "Name"},
    {index:1, value: "CITY", field: null}
  ]
}
```

With a synchronous response, you can change the assigned fields. Send a response in the same format with the fields changed. Please note that changing the value will have no effect, so it can be excluded from the response. You can also send only the changed columns to reduce the payload size.

ColumnsMatched.Response payload:

```
{
  columns: [
    {index:1, field: "BillingCity"}
  ]
}
```

Cancellation is not applicable for this event type. Therefore, sending `reject: true` will be ignored.

ImportStarted

The event is sent when the user clicked the “Import” button and the data is prepared for import. The specific operation insert or update is determined by the OperationSelected event.

Payload contains array of sObject records in JSON format:

```
{
  data: [
    {"sobjectType": "Account", "Name": "Account1", "Website": "www.example.com"}
  ]
}
```

With a synchronous response, you can cancel the import action.

ImportStarted.Response payload:

```
{
  reject: true
}
```

ImportFinished

The event is sent when import is completed. The specific operation insert or update is determined by the OperationSelected event.

Payload contains array of result objects in order of importing rows. Each object contains isSuccess attribute, Id if the record was inserted successfully, or message if there was an error.

```
{
  result: [
    {isSuccess: true, Id: "001W400000q6kv4IAA"},
    {isSuccess: false, message: "Some error"}
  ]
}
```

ImportFinished is always asynchronous. That is, there is no point in subscribing to the event. Just use the event for logging or other purposes.

5. Extension Development

The Extension is an LWC that embeds the Smart Import Tool and subscribes to the app's LMS channel. Start from this minimal example.

smartImportToolExt.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
   <apiVersion>65.0</apiVersion>
   <isExposed>true</isExposed>
   <targets>
      <target>lightning__Tab</target>
      <target>lightning__AppPage</target>
   </targets>
</LightningComponentBundle>
```

smartImportToolExt.html

```
<template>
   <dataart-smart-import-tool></dataart-smart-import-tool >
</template>
```

smartImportToolExt.js

```
import { LightningElement, wire } from 'lwc';
import { publish, subscribe, unsubscribe, APPLICATION_SCOPE, MessageContext } from
'lightning/messageService';
import CHANNEL from '@salesforce/messageChannel/DataArt__SmartImportToolEvent__c';

export default class SmartImportToolExt extends LightningElement {
   @wire(MessageContext) messageContext;

   _subscription = null;

   connectedCallback() {
      this.subscribeLMS();
   }

   subscribeLMS() {
      if (!this._subscription) {
         this._subscription = subscribe(
            this.messageContext,
```

```
            CHANNEL,
            (message) => this.handleIncoming(message),
            { scope: APPLICATION_SCOPE }
        );
    }
  }

  handleIncoming(message) {
    console.log('Incoming in extension: ', message);
  }
}
```

Place the Extension in a Lightning Tab or App Page. Add your code to handle different types of events. Have fun.

6. Contact us

If you have any questions or need help building your Extension, email smart.import.tool@dataart.com and our team will be happy to assist.