Merative Social Program Management 8.1

IEG in the Universal Access Responsive Web Application

Note

Before using this information and the product it supports, read the information in Notices on page 37

Edition

This edition applies to Merative™ Social Program Management 8.0.0, 8.0.1, 8.0.2, 8.0.3, and 8.1.

Contents

1 IEG in the Universal Access Responsive Web Application

Universal Access uses forms to gather information about citizens, such as when they apply for benefits. Merative ™ SPM Universal Access Responsive Web Application forms that gather data as evidence are implemented in IEG, as in the classic Universal Access citizen application. However, forms are now rendered in the browser by the IEG React Player, rather than the IEG Java™ player, and in some cases, the IEG behavior is different.

Due to the technology and user interface changes, your existing IEG scripts must be tested before use, and in most cases, at least some minor changes are needed for existing scripts to work in the new application.

The default connectivity handling in the Universal Access Responsive Web Application helps to prevent citizens losing data in IEG forms by preventing them from leaving pages with unsaved changes. For more information about data loss prevention in IEG, see the *Merative™ SPM Universal Access Responsive Web Application Guide*.

1.1 IEG elements and attributes specific to the design system and Universal Access Responsive Web Application

The following IEG elements and attributes apply to the design system and Universal Access Responsive Web Application only.

- **Display elements and attributes**
  - The `combo-box` element, which is a child element of the `question` element.
  - The `explainer` element, which is a child element of the `cluster`, `question-page`, and `relationship-page` elements.
  - The `hint-text` element, which is a child element of the `container`, `list-question`, and `question` elements.
  - The `next-button-label` element, which is a child element of the `question-page`, `relationship-page`, and `summary-page` elements.
  - The `relationship-detail-header` element, which is a child element of the `relationship-summary-list` element.
  - The `quick-add-list` element, which is a child element of the `relationship-page` element.
- **Display element attributes**
  - The `grouping-id` attribute of the `cluster` element.
- **Flow-control element attribute**
  - The value 'hidden' for the loop-type attribute of the loop element.
- **Meta-display elements**
  - The `class-names` element, which is a child element of the `layout` element.
  - The `date-picker` value for the `type` child element of the `layout` element.

For more information about IEG elements, see the *Authoring Scripts using Intelligent Evidence Gathering Guide*.

1.2 IEG configuration not currently supported for the Universal Access Responsive Web Application

The following IEG configuration is not currently supported by the Merative ™ SPM Design System or the Universal Access Universal Access Responsive Web Application.

- **Question matrices**
  Question matrices display a list of questions that are based on a code table and, for each of the code table values and each entity, a check box is displayed for you to select the values that apply to a particular entity.
- **Three-field date picker**
  The three-field date picker is no longer supported. Dates either default to a single-field date input field or can be configured with a date picker component by using the `layout` element.
- **Grouping individual question help at cluster level**
  Cluster-level help is supported, however, the `compile.cluster.help` property, which groups the help text for each of the questions in a cluster into the cluster help panel is not supported.
- **Display elements and attributes**
  - The `custom-output` element, which renders custom HTML on summary pages only.
  - The `show-page-elements` attribute on the `edit-link` element for editing specific clusters.
  - The `footer-field` element, which displays values that are calculated from expressions in the `footer-row` element of a list.
  - The `footer-row` element, which adds an extra row at the end of a list to display total or summary information.
  - The `help-text` element, which displays help text, is not supported for pages.
  - The `icon` element, which is used to add images to either the title area of a page or the sections panel.
  - The `label-alignment` element, which is used in the `layout` element for a cluster to control the text alignment of the labels in the cluster.
  - The `label-width` element, which is used in the `layout` element for a cluster to control the width of the labels in the cluster.
  - The `num-cols` element, which is used in the `layout` element for a cluster to control the number of columns in the cluster.
  - The `type` element, which is used in the `layout` element for a cluster to control the layout of labels in relation to input controls.
  - The `width` element, which is used in the `layout` element for a cluster to control the width of the cluster on the page.

    - The `legislation` element, which creates legislation links at page and question level to point to relevant legislative information.
    - The `policy` element, which creates policy links at page and question level to point to relevant policy information
    - The `skip-field` element, which enables a more flexible layout of elements within clusters or footer rows in lists where no visible display element is needed.
    - The `row-help` element, which specifies help for rows in a list.
    - The `set-focus` attribute of the `question-page` element, which sets focus for a page.
- **Meta-display elements**
    - The `codetable-hierarchy-layout` element, which is used in questions with a code table hierarchy type to control different aspects of the layout.
- **Structural, administrative, and other elements and attributes**
    - The `hide-for-control-question` attribute on the `ieg-script` element, which hides the label and value of control questions for loops when the loop is entered.
    - The `highlight-validation` attribute on the `ieg-script` element. Validations are now always displayed with the failing input field.
    - The `show-progress-bar` attribute on the `ieg-script` element. Progress through sections is now indicated by text and the section title. For example, **STEP 2 OF 4 · HOUSEHOLD**.

For more information about IEG elements, see the the *Authoring Scripts using Intelligent Evidence Gathering Guide*.

1.3 Customizing the Back button in IEG forms

You can customize the behavior of the Back button in IEG forms to suit your applications.

For the best user experience, set the behavior of the **Back** button in IEG according to whether you have a single form or multiple forms in your application. Where you have multiple forms, you typically want to navigate back to the previous form.

- Where you have a single form, always disable the **Back** button on the first page of the IEG form. The **Back** button goes back one page in the form, not in the application, so you don't need one on the first page. For more information about the `show-back-button` element, see the *Authoring Scripts using Intelligent Evidence Gathering Guide*.
- Typically, an application has multiple forms. By default, a feature with two forms, Apply and Submit is provided in the `universal-access-ui` package. The default feature has two instances of `BaseFormContainer`, `ApplicationFormContainer` and `SubmissionFormContainer`.

  By default, the Apply form has the **Back** button disabled on its first page.

  In Universal Access Responsive Web Application 3.0.4 or later, the **Back** button of the Submit form goes to the Apply form in the `SubmissionFormContainer` component by default.

If you are customizing or overriding `SubmissionFormContainer` component, or using an earlier version, you must add some code to the `SubmissionFormContainer` component to ensure that the **Back** button goes to a previous form.

- Add a function to the component logic, for example:

```
  handleBackForFirstPage = () => {
    const { history, submissionFormDetails } = this.props;
    const { applicationFormId } = submissionFormDetails;
    history.push({
      pathname: `${PATHS.APPLY}/${applicationFormId}`,
    });
  };
```

- Then, inside the render function, pass the function to the `BaseFormContainer` component by using the `onBackForFirstPage` prop, for example:

```
  render() {
    const { submissionFormDetails, match } = this.props;


 RESTService.handleAPIFailure(this.props.createApplicationUsingFormDetailsError);
    RESTService.handleAPIFailure(this.props.createSubmissionFormError);
    RESTService.handleAPIFailure(this.props.deleteApplicationFormError);
    RESTService.handleAPIFailure(this.props.getSubmissionFormDetailsError);

    if (match.params.submissionFormId && submissionFormDetails) {
      return (
        <BaseFormContainer
          iegFormId={match.params.submissionFormId}
          iegHookBindingKey={HookBindings.SUBMISSION}
          onBackForFirstPage={this.handleBackForFirstPage}
          onExit={this.handleExit}
          onFinish={this.handleFinishScript}
          onSaveAndExit={this.handleSaveAndExit}
          title={(submissionFormDetails &&
 submissionFormDetails.applicationTitle) || ''}
        />
      );
    }

    return <AppSpinner />;
  }
}
```

For more information about the `onBackForFirstPage` property, see 1.15 Customizing script behavior with BaseFormContainer on page 28.

1.4 Configuring section navigation for forms

If you are developing scripts in IEG, you can enable section navigation to guide people through forms.

About this task

You can use section navigation on any forms, but it is particularly useful for longer forms. If you enable section navigation, it is a good idea to use section summary pages so that users can review their changes regularly.

Procedure

In your IEG script, add the `show-section` element to the `ieg-script` element.

1.5 Configuring progress information for forms

If you are developing pages in IEG, you can show progress text and the section title so citizens can see where they are in the script, for example, **STEP 2 OF 4 · HOUSEHOLD**.

Add the following IEG configuration property to the *`ieg-config.properties`* file to configure the text. The section title is added automatically.

```
# Text progress bar indicator
progress.bar.indicator.text=Step %1s of %2s
```

Where *`%1s`* is the current step number and the *`%2s`* is the total number of steps on the script. The message is calculated based on the total number of sections and the current section.

The IEGPageMetadata(JSON); component contains all of the metadata for each IEG form. The text progress indicator is displayed if IEGPageMetadata finds the `metadata['ieg-config']['progress-indicator']` element in the JSON.

1.6 Configuring dynamic titles on forms

If you are developing pages in IEG, you can configure the relationship pages with more relevant titles that are based on the user's responses.

The relationship page title accepts an International Components for Unicode (ICU) message template. Page titles and subtitles accept a specific formatting syntax based on ICU. It should be used in loops and will give more context to the users.

These six keywords are defined:

- `index`
- `innerIndex`
- `outerIndex`
- `ordinal`
- `innerOrdinal`
- `outerOrdinal`

You can use `index` and `ordinal` in simple non-nested loops. If they are used in a nested loop, it is synonymous to `outerIndex` and `outerOrdinal`.

Refer to these examples.

`"Add {ordinal} member"` displays **Add first member**, **Add second member**, ...

`"Add the {innerOrdinal} income for the {outerOrdinal} member"` displays **Add the first income for the first member** ...

`"{index, select, 0 {Add your {innerOrdinal} income} other {Add %1s's {innerOrdinal} income}}"` displays **Add your first income** or **Add Jane's first income** depending on the value of index (this is equal to ordinal - 1).

`"Ajouter la {ordinal}#feminine# personne"` displays **Ajouter la première personne**.

`"Ajouter la {innerOrdinal}#feminine# recette du {outerOrdinal}# %spellout-ordinal-masculine# membre"` displays **Ajouter la première recette du premier membre**.

You can define the title as follows:

```
{index, select, 0 {Your relationships} other {{personName}'s relationships}}
```

The outcome of this message template on the first relationship question page is **Your relationships**. On the following relationship question pages, it shows **[personName]'s relationships**. The reserved word `personName` displays the person's first name on the title of the page.

1.7 Configuring rich text on forms

You can configure rich text to display with a number of IEG display elements in IEG forms. You can also configure external links in rich text to open in a new tab or window.

About this task

Rich text is supported in the following IEG display elements that support text:

- `cluster` title, help, and description
- `container` title, help, and description
- `display-text`
- `divider`
- `list` title, help, and description
- `question` label and help
- `subtitle`

For more information about IEG elements, see the *Authoring Scripts using Intelligent Evidence Gathering Guide*.

Configuring external links to open in a new tab or window

You can configure external links to open in a new tab or window in IEG forms. By default, links open in the current tab.

About this task

For security reasons, HTML in rich text is sanitized to remove certain attributes before display, including the HTML target attribute. You must configure the rich text to leave the target attribute in the sanitized content so that the link opens in a new tab or window.

For example, the `<a href="myurl">my link</a>` link in rich text opens in the current tab as intended. The `<a href="myurl" target="_blank">my link</a>` link is intended to open in a separate tab or window. However, because the rich text is sanitized with DOMPurify before display, the target attribute is removed and the link opens in the current tab by default.

To configure DOMPurify to leave specific attributes, you must add `dompurify` to the dependencies and specify a DOMPurify persistent configuration in any JavaScript or JSX code that runs when the app is loaded. For example, `App.js`. For more information about DOMPurify, see https://github.com/cure53/DOMPurify#persistent-configuration.

Only one active configuration at a time is allowed. After you set the configuration, any extra configuration parameters that are passed to `DOMPurify.sanitize` are ignored. The

DOMPurify configuration persists until the next call to `DOMPurify.setConfig`, or until `DOMPurify.clearConfig` is called to reset it.

Procedure

1. Add `dompurify` to the dependencies in the *`package.json`* file.

```
npm install dompurify
```

2. To configure DOMPurify to leave the target attribute, specify the following DOMPurify persistent configuration in any JavaScript or JSX code that runs when the app is loaded.

```
import DOMPurify from 'dompurify';
DOMPurify.setConfig({ ADD_ATTR: ['target'] });
```

1.8 Configuring hint text for forms

You can use short sentences of hint text to explain the expected input format or content in IEG forms. For example, you can explain the expected format for a telephone number.

About this task

Hint text is suitable for short sentences and does not support HTML tags. If you want to add more text or format text with HTML tags, use the `help-text` or `explainer` elements instead. For more information, see the *Authoring Scripts using Intelligent Evidence Gathering Guide*.

Note: Specific globalization considerations apply to the date format when it is used in hint text and messages. Ensure that you have the same date format in the `REACT_APP_DATE_FORMAT` environment variable, and in the`DateAdapter_DateFormat` and `Errors_date` messages in the `intelligent-evidence-gathering-locales` package.

Procedure

In your IEG script, you can add the `hint-text` element to any `container`, `question` or `list-question` element.

For example:

- Container

```
<container show-container-help="true">
        <title id="primaryPhoneNumber">primaryPhoneNumber</title>
    <hint-text id="PhoneNumber.Hint">PhoneNumber.Hint</hint-text>
    <help-text id="PhoneNumber.Help">Telephone number must only contain numbers,
 parentheses, or dashes and be 10 digits. For example, (212) 555-0010 or
 2125550010.</help-text>
    <question id="primaryPhoneType" mandatory="true">
                <help-text id="PhoneNumber.Help">Telephone number must only contain
 numbers, parentheses, or dashes and be 10 digits. For example, (212) 555-0010 or
 2125550010.</help-text>
        <label id="PrimaryPhoneType.Label">Primary Phone Type</label>
    </question>
</container>
```

- Question

```
<question id="firstName" mandatory="true">
        <hint-text id="FirstName.Hint">FirstName.Hint</hint-text>
    <label id="FirstName.Label">First Name</label>
</question>
```

- List question

```
<list-question entity="Person" id="currentlyWorking" mandatory="false">
    <label id="CurrentlyWorking.Label">Please select the people that have a job:</
label>
    <hint-text id="CurrentlyWorking.Hint">CurrentlyWorking.Hint</hint-text>
    <item-label>
        <label-element attribute-id="firstName" />
    </item-label>
</list-question>
```

1.9 Configuring explainer text for forms

You can use the `explainer` element to provide extra text in IEG forms that is initially hidden and that can be expanded to show further explanation. For example, you can provide background information that a user can choose to expand only if needed.

About this task

You can use the `explainer` element to provide a large amount of text without cluttering up the form. For more information, see the *Authoring Scripts using Intelligent Evidence Gathering Guide*.

Procedure

In your IEG script, add the `explainer` element to any `cluster`, `question-page`, or `relationship-page` element.

For example:

- `cluster`

```
<cluster>
        <explainer>
          <title id="ExplainerCluster.Title">Why do we ask for your Social Security
 Number?</title>
          <description id="Explainer.Description">Your Social Security Number
 ensures that your application is unique to you and reduces processing time.</
description>
        </explainer>
        <question control-question="false" id="isSSN" mandatory="true" multi-
select="false" show-field-help="false">
          <label id="IsSSN.Label">What is your Social Security Number?</label>
        </question>
      </cluster>
```

- `question-page`

```
<question-page>
<explainer>
 <title id="ExplainerSSN.Title">Why do we ask for your Social Security Number?</
title>
 <description id="ExplainerSSN.Description">Your Social Security Number ensures that
 your application is unique to you and reduces processing time.</description>
 </explainer>
</question-page>
```

- `relationship-page`

```
<relationship-page>
<explainer>
 <title id="ExplainerSSN.Title">Why do we ask for your Social Security Number?</
title>
 <description id="ExplainerSSN.Description">Your Social Security Number ensures that
 your application is unique to you and reduces processing time.</description>
 </explainer>
</relationship-page>
```

1.10 Configuring the 'Help' label for forms

You can change or remove the 'Help' label from the help icon for input controls in the application by overriding the default text. To remove the label, override the default text with a single space character in a custom messages file.

Procedure

1. Create a `src/locale/messages_en.json` messages file with a single space character as the value for the help label message ID, `WidgetHelp_helpToggleText`.

```
{
  "WidgetHelp_helpToggleText": " "
}
```

2. Update the `src/config/intl.config.js` file in the English locale to point to the custom messages file.

```
// [...]    {
      locale: 'en',
      displayName: 'English',
      localeData: () => {
        require('@formatjs/intl-pluralrules/locale-data/en');
        require('@formatjs/intl-relativetimeformat/locale-data/en');
      },
      messages: require('../locale/messages_en'),
    },// [...]
```

3. Rebuild and deploy the application to see your changes.

1.11 Configuring required or optional labels for form fields

You can choose whether to indicate the required fields or the optional fields in IEG forms. As the majority of questions in a typical form should be required, indicating the optional questions rather than the required questions typically results in a less cluttered form. By default, optional fields are highlighted in IEG forms.

About this task

By default, fields that are not configured as required in the IEG script are labeled as **Optional** and required fields are not labeled. If you choose to indicate required fields instead, fields that are configured as required in the script are labeled **Required** and optional fields are not labeled.

Procedure

Show labels for required questions only by adding the `REACT_APP_DISPLAY_REQUIRED_LABEL` environment variable to your *`.env`* file with a value of `true`.
For example:

```
REACT_APP_DISPLAY_REQUIRED_LABEL=true
```

1.12 Configuring input formats and constraints for form fields

You can customize field inputs and constraints on IEG forms, such as phone numbers, social security numbers (SSN), dates, currencies, and percentages. You can adjust the width of form fields to match the length of the expected input, and choose to use a date picker for dates where appropriate.

About this task

Where users need to type confidential information, you can obscure the input values to ensure privacy. This configuration is done in the data store schema by setting a new data type and cannot be used with masks. Instead of using a mask, you can also implement any extra constraints, such as the number of characters, in the data store schema by creating a custom domain, see Configuring inputs to be obscured for privacy on page 22.

Masked input fields increase input field readability by formatting or constraining typed data. You can apply input masks with the IEG `class-names` element, which is a child element of the `layout` element. The `class-names` element adds the content of the element to the HTML that is generated for the component, this element accepts multiple values that are separated by a space. For more information about the IEG `layout` element, see the *Authoring Scripts using Intelligent Evidence Gathering Guide*.

You might need a custom mask that is not supported by the `class-names` element, such as variants of the Social Security Number (SSN) or Social Insurance Number (SIN). To create a custom mask, use the `mask-format` element, which is a child element of the `layout` element, to set custom masks with Cleave.js format.

- **Input field masks**

  If the class name matches any of the reserved input mask class names, that class name is applied to the HTML control input. If the class name does not match a reserved input mask class name, the class name is applied to the `<div>` element that contains the HTML element (`cluster`, `question`, or `list-question`). You can use the following design system CSS classes as input masks to format and constrain input values for questions:

  - `wds-js-input-mask-currency`

    Masks input for currencies. The character limit is 21 characters. You can also set optional environmental variables for currency symbols, see Configuring currency symbols on page 21.
  - `wds-js-input-mask-numeral`

    Masks input for numerical input.
  - `wds-js-input-mask-yyyy-mm-dd`

    Masks input for the YYYY-MM-DD date format.

- `wds-js-input-mask-percentage`

  Masks input for percentage characters.
- `wds-js-input-mask-phone`

  Masks input for phone number fields according to the defined locale for the application. Configuring the phone number input mask requires some additional steps and you can also set optional environmental variables for delimiters and country codes, see Configuring phone numbers on page 20.
- `wds-js-input-mask-postal-code`

  Masks input for 2 groups of 3 characters that are separated by a space, XXX XXX, such as a Canadian postal code. Alphabetic characters are converted to uppercase.
- `wds-js-input-mask-sin`

  Masks input for 3 groups of 3 characters that are separated by spaces, XXX XXX XXX, such as a Canadian Social Insurance Number (SIN).
- `wds-js-input-mask-ssn`

  Masks input for digits that are separated by dashes and grouped as follows, XXX-XX-XXXX, such as a US social security number (SSN).
- `wds-js-input-layout-size--`*`field_size`*

  Adjusts the width of form fields to match the length of the expected input. Where *`field_size`* is one of the following sizes:

  - **`x-small`**
    Use for 2 - 3 characters, such as DD, MM, or title.
  - **`small`**
    Use for 4 - 6 characters, such as ZIP code, postal code, or CVV number.
  - **`medium`**
    Use for around 8 characters, such as SSN or DD/MM/YYYY.
  - **`large`**
    Use for around 16 characters, such as credit card numbers.
  - **`x-large`**
    Use for around 24 characters, such as email addresses.

- **Form field width**

  To avoid confusion about expected inputs, always match the width of form fields to the expected input. For example, use a form field that matches the length of the SSN.

- **Date picker**

  For date questions, in addition to the masked input, you can choose to add a date picker for dates by setting the value of the `type` child element of the `layout` element to `date-picker`. For those questions, you can then use the calendar or type the date. By default, date questions are displayed with the masked input field if no layout `type` is specified.

Procedure

1. In your IEG script, add the appropriate CSS classes to the `layout` element for the question. For example:

```
<question id="ssn" mandatory="true">
 <label id="SSN.Label">SSN</label>
 <layout>
  <class-names>custom-css-class1 wds-js-input-mask-ssn wds-js-input-layout-size--
medium
            </class-names>
 </layout>
</question>
```

2. If you want to add a custom mask, use a `mask-format` element in the `layout` element. Define the `mask-format` text value by using an XML CDATA section with a JSON object with reference to the Cleave.js documentation,
For example,

```
<layout><mask-format><![CDATA[{ "delimiter": " ",  "blocks": [2, 2, 2],
      "numericOnly": true  }]]></mask-format><layout>
```

Configuring phone numbers

You can configure an input mask class name to format phone number fields in IEG forms according to the defined locale for the application. You can also configure a phone number delimiter or a country prefix if needed.

Procedure

1. Add *`cleave.js`* as a dependency in your *`package.json`* file.

```
"cleave.js": "<version>"
```

   Where *version* is the version that you want to use.

2. Import the region-specific *`.js`* file in your initializing *`.js`* file.
   For example:

```
import 'cleave.js/dist/addons/cleave-phone.[country]';
```

   Where *country* is the locale that you want to use.

3. Add a REACT_APP_PHONE_MASK_FORMAT environment variable to your *`.env`* file.

```
REACT_APP_PHONE_MASK_FORMAT=[country]
```

   Where *country* is the locale that you want to use.

4. In your IEG script, add the `wds-js-input-mask-phone` class name to the question. For example:

```
<question id="primaryPhoneNumber" mandatory="true" show-field-help="true">
 <layout>
   <class-names>wds-js-input-mask-phone</class-names>
 </layout> <label id="PrimaryPhoneNumber.Label">Primary Phone Number</label>
</question>
```

5. Optional: You can set a custom delimiter for phone numbers by adding the `REACT_APP_PHONE_MASK_DELIMITER` environment variable to your *`.env`* file.

For example, to convert 1 636 5600 5600 to 1-636-5600-5600, set the environment variable as follows:

```
REACT_APP_PHONE_MASK_DELIMITER=-
```

6. Optional: You can set a fixed country code for phone numbers by adding the `REACT_APP_PHONE_MASK_LEFT_ADDON` environment variable to your *.env* file.
For example, to convert 1-636-5600-5600 to +1-636-5600-5600, set the environment variable as follows:

```
REACT_APP_PHONE_MASK_LEFT_ADDON=+
```

Configuring date formats

You can configure the date format in IEG forms by setting the `REACT_APP_DATE_FORMAT` environment variable.

About this task

By default, the date format is MM/DD/YYYY if you do not set a value for the `REACT_APP_DATE_FORMAT` environment variable. If you set an invalid value, the default date format is used.

The valid values are:

```
dd-mm-yyyy
mm-dd-yyyy
yyyy-mm-dd
```

Note: Specific globalization considerations apply to the date format when it is used in hint text and messages. Ensure that you have the same date format in the `REACT_APP_DATE_FORMAT` environment variable, and in the `DateAdapter_DateFormat` and `Errors_date` messages in the `intelligent-evidence-gathering-locales` package.

Procedure

Change the date format by adding the `REACT_APP_DATE_FORMAT` environment variable to your *.env* file.
For example, to change the date format to DD/MM/YYYY, set the environment variable as follows:

```
REACT_APP_DATE_FORMAT=dd-mm-yyyy
```

Configuring currency symbols

You can configure the currency symbol that is displayed for currency fields in IEG forms. Configure the `REACT_APP_CURRENCY_MASK_ADDON` environment variable to specify a currency

symbol to display either before or after the currency amount. The alignment of the currency symbol is based on the locale.

About this task

For more information about how the currency symbol is aligned based on locale, see the developer.mozilla.org documentation.

The value of the `REACT_APP_CURRENCY_MASK_ADDON` environment variable takes precedence over the deprecated `REACT_APP_CURRENCY_MASK_LEFT_ADDON` and `REACT_APP_CURRENCY_MASK_RIGHT_ADDON` environment variables.

Procedure

Use the following option to configure and align the currency symbol based on the locale by configuring the `REACT_APP_CURRENCY_MASK_ADDON` environment variable.

- Add the `REACT_APP_CURRENCY_MASK_ADDON` environment variable to your *.env* file. For example, to set the currency symbol to US dollars, enter the following command:

```
REACT_APP_CURRENCY_MASK_ADDON=$
```

Use the following deprecated option to explicitly align the currency symbol on either the left side or the right side.

- 

  Add a currency symbol for currency fields by adding the `REACT_APP_CURRENCY_MASK_LEFT_ADDON` or `REACT_APP_CURRENCY_MASK_RIGHT_ADDON` environment variables to your *.env* file. For example, to set the currency symbol for US dollars, enter the following command to set the environment variable:

```
REACT_APP_CURRENCY_MASK_LEFT_ADDON=$
```

  If both environment variables are set, `REACT_APP_CURRENCY_MASK_LEFT_ADDON` takes precedence.

Configuring inputs to be obscured for privacy

Where users need to type confidential information, you can obscure the input values to ensure privacy. Users can show or hide the text as they type. The user input is obscured when they type the confidential information, such as their Social Security Number (SSN). By default no constraints are applied, but you can create a custom domain to apply custom constraints where needed. For example, you can restrict the number of characters.

About this task

You can obscure inputs by setting the data type for a specified attribute of an entity to `IEG_OBSCURED` in the data store schema. This configuration cannot be used with masks. Instead of using a mask, you can also implement any extra constraints, such as the number of characters, in the data store schema by creating a custom domain.

For more information about data types and IEG domains, see the *Authoring Scripts using Intelligent Evidence Gathering Guide*.

For more information about data store schemas, see the *Authoring Scripts using Intelligent Evidence Gathering Guide*.

Procedure

1. In the entity, identify the attributes for which you want to obscure the input.
   For example, the `ssn` attribute for the social security number.
2. Edit the data store schema *`.xsd`* file for the IEG script and in the entity, change the type of the attribute to *`IEG_OBSCURED`*.
   For example,

```
<xsd:attribute name="ssn" type="IEG_OBSCURED"/>
```

3. Optional: To apply further input constraints to the field, create a custom domain.
   For example, to constrain the user from typing more than 9 characters in the input field for an SSN, you can create a custom domain called `SSN_OBSCURED`.
   a) Create a custom domain like the following domain.

```
....
    <xsd:include schemaLocation="IEGDomains"/>
<!-- NEW  TYPE  BEGIN-->
 <xsd:simpleType name="SSN_OBSCURED">
        <xsd:restriction base="IEG_OBSCURED">
   <xsd:minLength value="8"/>
   <xsd:maxLength value="9"/>
  </xsd:restriction>
    </xsd:simpleType>

<!-- NEW  TYPE  END-->
 <xsd:element name="Application">
.....
```

   b) Edit the data store schema *`.xsd`* file for the IEG script and change the type of the `ssn` attribute to *`SSN_OBSCURED`*.
   For example,

```
<xsd:attribute name="ssn" type="SSN_OBSCURED"/>
```

1.13 Configuring code-table hierarchies for form fields

You can use code-table hierarchies to add two related questions in IEG forms. When you answer the first question, the second question is enabled.

About this task

Any question where the data type is defined as a code table hierarchy is displayed as two separate questions in vertically aligned drop-down menus. The first question menu corresponds to the root code table in the hierarchy, and displays the label that is specified for the question. The second question menu corresponds to the second-level code table in the hierarchy, and displays a label that corresponds to the code table display name. The second menu is disabled until a selection is made in the first menu. Summary pages display both questions.

Displaying a code-table hierarchy value in a list, or the `codetable-hierarchy-layout` options, are not supported.

Procedure

To ensure that the label is displayed correctly for the second question, you must ensure that, for each code table `name` element, there is a corresponding `locale` element within the `displaynames` element in your code-table definition.

For example, see the following code-table definition.

```
<codetables package="curam.codetable" hierarchy_name="CountyCityHierarchy">
    <!-- Parent codetable - County -->
    <codetable java_identifier="COUNTYCODE" name="CountyCode">
        <displaynames>
            <name language="en">County</name>
            <locale language="en">County</name>
        </displaynames>
        <!-- code items... -->
    </codetable>
    <!-- Child codetable - City -->
    <codetable java_identifier="CITYCODE" name="CityCode"
 parent_codetable="CountyCode">
        <displaynames>
            <name language="en">City</name>
            <locale language="en">City</name>
        </displaynames>
        <!-- code items... -->
    </codetable>
</codetables>
```

1.14 Implementing a combo box for form fields

You can implement a combo box question with an auto-complete search function to help you to complete form fields in IEG forms as you type. For example, known address fields can be automatically selected when you enter an address. You can implement the option to add new items if they are not found, for example, add an address.

About this task

You must implement a search function in the Universal Access Responsive Web Application and register the search function with `IEGRegistry`. The search function can point to an internal or external search service to provide the information. Then, update the datastore schema definition and your IEG script.

Implementing search functions for `ComboBox` components

You can implement the `ComboBox` component to search external data sources as you type in a form field, with a built-in filter function. Implement a search function and associated error handling, and make that search function available to the IEG form. If needed, you can implement an **Add New** option so that users can add an item if it is not found.

Procedure

1. Implement the search function. A search function is a JavaScript™ function that receives one parameter that contains the value of the `ComboBox`, and returns an array of items to be displayed by the `ComboBox`.

The response of `search-function` is an array of items, `{items}`. Each item is an object with the following structure:

```
{
  id:"key"
  value:"value"
  item: { "attribute1": "value1", "attribute2": "value2" },
}
```

Where:

- `id` is a mandatory attribute to store the ID in the data store.
- `value` is the value of the question to store in the data store and to render in the list of options of the `ComboBox`.
- `item` is an optional complex object with the structure of the `formData` to be populated if that element is selected in the `ComboBox` component.

The structure of the item object must match the `formData` of the target entity. The following simple example populates the `ResidentialAddress` entity:

```
{
  'street1': 'street1',
  'street2': 'street2',
  'city': 'city',
  'zipCode':' zipCode',
  'state': 'state',
}
```

2. Register the search function with the `IEGRegistry` object. `IEGForm` has access to `IEGRegistry` and all registered functions. `IEGForm` reads the custom functions from `IEGRegistry` and stores them on its `formContext` so `IEGForm` can call custom functions.
   1. Implement the JavaScript™ function in any `.js` file.
   2. Import `IEGRegistry` in a JavaScript™ initial file, such as `App.js`, and add the custom function to the registry. For example:

```
 import { IEGRegistry } from '@spm/core';
 import { searchCity, customFunction } from './examples/playground/
customFunctions';
...

const App = () => {
  IEGRegistry.registerComboBoxSearchFunctions({ searchCity, customFunction });
    ....
};
```

Add New option

If you want to render an **Add New** option in the menu that is displayed by the `ComboBox`, the response of the JavaScript™ function must follow the structure:

```
{
   newItem: { id: '-1', label: 'Add New', value: ' ', position: 'top' },
   items,
 }
```

Where:

- `newItem` is a complex object with the definition of the **Add New** option.
- `id` is the ID of the new option.
- `label` is the label of the new option.

- `value` is the value of the new option.
- `position` is the position where the new option renders. The possible values are `bottom` and `top`.

Error messages

The search function must implement its own logic to handle errors if an error needs to be displayed on the UI, the response of the search function must be:

```
{errorMessage: 'Controlled Error Message'}
```

The error message is displayed underneath the `ComboBox`.

Configuring combo box scripts and schemas

Add the `combo-box` element to a question in your IEG script and configure the `combo-box` element attributes. Add a cluster after the question to display the information to the user when they select a menu item. Update the schema definition with the appropriate elements.

About this task

The `question` schema type must be a string. You cannot use a `question` with a `combo-box` child element as a control question.

You can review the design system usage guidance for the `ComboBox` component. In your development environment, open the Social Program Management Design System Storybook documentation at *`<path>@govhhs/govhhs-design-system-react/doc/index.html`* and search for `ComboBox`.

For more information about the IEG `combo-box` element, see the *Authoring Scripts using Intelligent Evidence Gathering Guide*.

Procedure

1. Add the `combo-box` child element to the `question` element. For example:

```
<question-page id="AboutTheApplicant_GB"  read-only="false" set-focus="false" show-
back-button="false" show-exit-button="true" show-next-button="true" show-person-
tabs="false" show-save-exit-button="true" entity="Person" >

<!-- ComboBox -->
<cluster entity="SearchAddress">
  <title id="SearchAddress.Title">Your address</title>
  <question id="fullAddress" mandatory="true" show-field-help="false">
  <label id="FullAddress.Label">Search for your address</label>
  <combo-box  key="id" search-function="searchAddress" target-
entity="ResidentialAddress" filter-items="true" />
</question>
</cluster>
 </question-page>
```

   Where:

   - `key` is the id to be stored in the data store and renders as a hidden widget on the front end. It is mandatory and the entity must define this property in the schema definition. The `key` schema type must be a string.
   - `search-function` is the name of the JavaScript™ search function to be called on each keydown event.

- `target-entity` is an optional attribute to show information to the user when they select a combo box menu item. In `target-entity`, specify the cluster entity to be populated with the value of the `search-function` result item attribute. Update the script to display the cluster entity on the page, the target entity must be shown on the same page as the combo box. If more than one cluster on the page is related to the same entity name, the first cluster that matches the entity attribute value with the `target-entity` value is populated.
- `filter-items` is an optional attribute that, if true, filters the items as you type with the built-in filter. By default, it is false.

2. Add a cluster to display the target-entity information when a user selects a menu item.

```
<question-page id="AboutTheApplicant_GB"  read-only="false" set-focus="false" show-
back-button="false" show-exit-button="true" show-next-button="true" show-person-
tabs="false" show-save-exit-button="true" entity="Person" >

<!-- ComboBox -->
<cluster entity="SearchAddress">
  <title id="SearchAddress.Title">Your address</title>
  <question id="fullAddress" mandatory="true" show-field-help="false">
  <label id="FullAddress.Label">Search for your address</label>
  <combo-box  key="id" search-function="searchAddress" target-
entity="ResidentialAddress" filter-items="true" />
</question>
</cluster>

<!-- ComboBox -->
<cluster entity="ResidentialAddress">
        <title id="Address.Title">Enter address</title>
        <help-text id="ADHelp">You must enter the address in which you physically
 reside (residential address).</help-text>
        <question control-question="false" id="street1" mandatory="true" multi-
select="false" show-field-help="false">
          <label id="Street1.Label">Street 1</label>
        </question>
        <question control-question="false" id="street2" mandatory="false" multi-
select="false" show-field-help="false">
          <label id="Street2.Label">Street 2</label>
        </question>
        <question control-question="false" id="city" mandatory="false" multi-
select="false" show-field-help="false">
          <label id="City.Label">City</label>
        </question>
        <question control-question="false" id="zipCode" mandatory="false" multi-
select="false" show-field-help="false">
          <label id="Zipcode.Label">ZIP code</label>
        </question>
</cluster>
 </question-page>
```

3. Edit the schema definition and add an element for the combo box and the target entity, for example:

```
<!-- ComboBox -->
<xs:element name="SearchAddress">
      <xs:complexType>
          <xs:attribute name="id" type="IEG_STRING" />
          <xs:attribute name="fullAddress" type="IEG_STRING"/>
      </xs:complexType>
</xs:element>
<!-- Target Entity -->
 <xs:element name="ResidentialAddress">
  <xs:complexType>
   <xs:attribute name="street1" type="IEG_STRING"/>
   <xs:attribute name="street2" type="IEG_STRING"/>
   <xs:attribute name="city" type="IEG_STRING"/>
   <xs:attribute name="zipCode" type="IEG_STRING"/>
  </xs:complexType>
 </xs:element>
2. Associate that new element to a Person entity.
<xs:element name="Person">
 <xs:complexType>
  <xs:sequence minOccurs="0">
   <xs:element ref="SearchAddress" minOccurs="0" maxOccurs="1"/>
   <xs:element ref="ResidentialAddress" minOccurs="0" maxOccurs="unbounded"/>
   ....
  </xs:element>
</xs:complexType>
```

1.15 Customizing script behavior with BaseFormContainer

The behavior of scripts in the application is controlled by the *BaseFormContainer.js* container component. Each form calls this container component, which controls script behavior such as whether partial submission is allowed, or where to go on exiting the script. You can customize the behavior for individual scripts by modifying `BaseFormContainer` properties.

About this task

The following `BaseFormContainer` properties are available:

- `iegFormId`. (Mandatory) This property corresponds to the IEG execution ID that is obtained from one of the following options:
  - An API that starts the script, by creating the execution with the necessary script ID and data store schema.
  - Existing executions that can be resumed.

  Note: Later, the ID is used on the server to ensure that the current user matches the user who is associated with the execution in the `CitizenScriptInfo` table. The ID also ensures that the execution is not completed.
- `title`. (Mandatory) The title to be displayed in the header. You can convert the property by using the `formatMessage` for `react-intl`.
- `isLoginOrSignupAllowed`. If the property is `true` when **Save and exit** is clicked and the user is not logged in, the log-in screen is displayed. The default value is **True**.
- `isPartialSubmissionAllowed`. Specifies that partially completed scripts can be submitted. The corresponding option must be added to the header. The default value is **False**.

- `onExit`. Specifies what happens when a user exits the script without saving. By default, it goes to the home page.
- `onFinish`. Specifies what happens when the last page of the script is submitted. By default, it goes to the home page.
- `onPartialSubmission`. Specifies what happens when a partial script is submitted. By default, it saves the current page and then starts the `OnFinish` handler.
- `onSaveAndExit`. Specifies what happens when a user saves and exits the script. By default, it saves the current page and determines what page to go to. If the user is not logged in, the log-in page is displayed. If the user is logged in, the dashboard is displayed.
- `onRef`. A function that receives the instance of the current `BaseFormContainer` to provide access to its defined functions and props. You can use this function to customize the default `BaseFormContainer` functions. For an example of using the `onRef` function to customize the behavior of **Save and exit**, see the `SampleApplicationFormComponent` in the `sampleApplication`.
- `onBackForFirstPage`. A function that is called on the back-button click event of the first page of a form to redirect back to another form. The function contains the code responsible for the redirection. For example, you might want to go back to an application script from a submission script to change something before you submit an application.

Procedure

To modify the behavior for an existing form feature, follow the standard steps in Reusing existing features. For example, to customize the form that is loaded from the *`/eligiblity/form`* URL, do the following steps:

1. Find the path variable in the *`node_modules/@spm/universal-access-ui/routes/Paths.js`* file.
   For example, search for */`eligibilty/form`* to locate `PATHS.ELIGIBILITY.FORM`.
2. Search the *`Routes`* file for the path variable to find the location of the feature that it loads.
   For example, in the *`node_modules/@spm/universal-access-ui/src/routes/Routes.js`* file search for the `PATHS.ELIGIBILITY.FORM` path variable that you located in the previous step. The path variable maps to the *`feature/Forms/Eligibility`* location.
3. Copy the source code from the feature folder that you identified in the previous step to your custom folder.
   For example, copy *`node_modules/@spm/universal-access-ui/src/features/Forms/Eligibility`* to the *`your-custom-app/src/features/Forms/Eligibility`* folder.
4. Add a route in the *`your-custom-app/src/routes.js`* file with the same path as the original `PATHS.ELIGIBILITY.FORM` feature.
   a) Map the new route to your custom version of the form feature.
5. Update the properties of the form container according to your requirements.
   For example, use custom functions to change the behavior of the on-exit and on-finish flows, as shown in the following code sample:

```
<BaseFormContainer
          iegFormId={formId}
          iegHookBindingKey={HookBindings.SCREENING}
          onExit={this.myCustomHandleExitForm}
          onFinish={this.myCustomHandleFinishForm}
          title={myCustomTitle || ''}
        />
```

1.16 Merging clusters with the `cluster` element `grouping-id` attribute

If you are developing pages in IEG, you can merge several clusters on summary pages by using the **`cluster`** element **`grouping-id`** attribute. The **`grouping-id`** attribute is not supported for standard Merative™ Social Program Management web applications.

Related data fields can be defined within different clusters under the following conditions. You can use the **`grouping-id`** attribute to merge these related data fields into a single cluster on IEG pages.

- Data is defined within different schema entities but a single cluster can be defined for a single entity only.
- Data is defined within a conditional cluster but it must be included in a non-conditional cluster when the condition is met.

All clusters with a specific **`grouping-id`** attribute are merged into the first cluster with that **`grouping-id`** attribute. Aside from the questions, the cluster elements are shown as defined by the first cluster. Ensure that the other cluster elements in the first cluster, such as the title or buttons, are suitable for the merged cluster.

Where possible, do not have a conditional cluster as the first cluster if you are merging conditional and non-conditional clusters. If the first cluster is conditional and the condition is not met, then the merged cluster is not displayed. If a conditional cluster must be positioned before non-conditional clusters in a merged cluster, then add a non-conditional cluster with no questions as the first cluster with the **`grouping-id`**.

This sample XML snippet merges three clusters into a single cluster with the **`grouping-id`** attribute. The three clusters have data fields from three different entities and the last cluster is conditional.

```
<cluster entity="ResidentialAddress" grouping-id="100">
  <title id="Address.Title">Address</title>
  <edit-link
    skip-to-summary="false"
    start-page="AboutTheApplicant_GB"
  />
  <layout>
    <type>flow</type>
    <num-cols>2</num-cols>
    <label-alignment>left</label-alignment>
  </layout>
  <question
    id="street1"
  >
    <label id="Street1.Label">Street 1:</label>
  </question>
...
</cluster>
<cluster entity="Person" grouping-id="100">
  <question
    id="applyToMailingAddress"
  >
    <label id="ApplyToMailingAddress.Label">Mail to Same Address?</label>
  </question>
</cluster>
<condition expression="Person.applyToMailingAddress==&quot;N2OITYN2&quot;">
  <cluster entity="MailingAddress" grouping-id="100">
    <question
      id="street1"
    >
      <label id="Street1.Label">Street 1:</label>
    </question>
    ...
</cluster>
```

1.17 Configuring relationship pages questions

If you are developing pages in IEG, you can configure the text of the relationship questions on relationship pages.

By default, the question label is dynamic, in the first relationship question page, it renders as “What is [Name and Age of the Person related] to you?”. On the following relationship question pages, it renders “What is [Name and Age of the Person related] to [Name and Age of the Person]?

The attribute name for the start date must be `startDate`.

To show age in the relationship question label, you must populate the date of birth, which is defined as the `dateOfBirth` attribute of the `Person` entity.

You can use the following IEG configuration property to configure the default text.

```
# relationship question label on relationship page
relationship.question.label={index, select, 0 {What is %2s to you?} other {What is %2s
 to %1s?}}
```

The example ICU template does the following:

In the first iteration:

```
What is %2s to you?
```

Where *%2s* is the related person in the first iteration.

From the second iteration until the end:

```
What is %2s to %1s?
```

Where *%1s* is the new main person in the iteration and *%2s* is the related person in the iteration.

1.18 Configuring relationship starting dates on relationship summary pages

If you are developing pages in IEG, you can configure the start date of relationships for relationship summary pages. For example, Married since Jun 12, 2014.

You can use the following IEG configuration property to configure the default text.

```
# relationship type and start date label.
relationship.type.date.label=%1s since %2s
```

Where *%1s* is the relationship type and *%2s* is the relationship start date.

1.19 Configuring `quick-add-list`

The `quick-add-list` feature is enabled at the IEG script level. The `quick-add-list` component receives two parameters, `entity` with the Entity object is managed by the component and `criteria` with any specific criteria that the component might need to meet.

Common pattern

The code that follows is an example of a fully functional implementation of the `quick-add-list` component in a section of an IEG script:

```
<section>
    <question-page id="AnyMemberPage" show-back-button="true"  show-exit-button="false"
 show-save-exit-button="true" show-person-tabs="false">
        <title id="AnyMemberPage.Title">Household</title>
 <description id="AnyMemberPage.Description">Please enter details about the other
 people besides yourself who live in your home including those who are not related to
 you. Once you're finished please check the box to confirm the number of other people
 living in your home (not including yourself).
         </description>
         <condition expression="false">
     <cluster entity="Application">
  <question id="dummy" default-value-expression="householdCount()"/>
     </cluster>
        </condition>
        <quick-add-list entity="Person" criteria="isPrimaryParticipant==false">
            <title id="HouseholdList.Title">Household members</title>
     <quick-edit-link >
         <page-title id="Edit.PageTitle">Edit %1s (%2s)<argument id="Person.firstName"/
><argument id="Person.age"/>
                </page-title>
     </quick-edit-link>
     <quick-delete-link>
         <page-title id="Delete.PageTitle">Remove %1s %2s (%3s) from the household?
<argument id="Person.firstName"/><argument id="Person.lastName"/><argument
 id="Person.age"/></page-title>
         <confirm-message id="Delete.Message">Are you sure you want to remove %1s?
<argument id="Person.firstName"/></confirm-message>
         <confirm-button id="Delete.Button">Remove %1s<argument id="Person.firstName"/
></confirm-button>
     </quick-delete-link>
            <quick-add-link>
               <page-title id="Add.PageTitle">Add new person to household</page-title>
               <title id="Add.Title">Add new member</title>
            </quick-add-link>
            <page-content id="HouseholdMember"/>
        </quick-add-list>
        <condition expression="Application.householdCount != 0">
            <cluster>
                <question id="doneEditingHousehold" mandatory="true" control-
question="true" control-question-type="IEG_BOOLEAN">
                <label id="HasOtherMembers.Label">There are %1s other people in your
 home not including yourself<argument id="Application.householdCount"/></label>
                </question>
            </cluster>
        </condition>
        <condition expression="Application.householdCount == 0">
            <cluster>
                <question id="doneEditingHousehold" mandatory="true" control-
question="true" control-question-type="IEG_BOOLEAN">
                <label id="HasOtherMembers.Label">There are no other people in your
 household, just yourself</label>
                </question>
            </cluster>
        </condition>
    </question-page>
    <loop loop-type="hidden" entity="Person" criteria="isPrimaryParticipant==false">
        <question-page id="HouseholdMember">
            <title id="HouseholdMember.Title">Household</title>
            <cluster>
                <title id="HouseholdMember.Cluster.Title">Personal details</title>
                <question id="firstName" mandatory="true">
                    <label id="FirstName.Label">First Name</label>
                </question>
                <question id="lastName" mandatory="true">
                    <label id="lastName.Label">Last Name</label>
                </question>
                <question id="dateOfBirth" mandatory="true">
                    <label id="DateOfBirth.Label">Date of birth</label>
                </question>
            </cluster>
        </question-page>
    </loop>
</section>
```

The `quick-add-list` component uses a custom function `householdCount` that updates the number of household members. The logic for that custom function can be written as follows:

```
public Adaptor getAdaptorValue(final RulesParameters rulesParameters)
   throws AppException, InformationalException {

   final IEG2Context ieg2Context = (IEG2Context) rulesParameters;
   final long executionID = ieg2Context.getExecutionID();
   final long rootEntityID = ieg2Context.getRootEntityID();

   final IEGScriptExecution scriptExecution = IEGScriptExecutionFactory
     .getInstance().getScriptExecutionObject(executionID);
   Datastore ds = null;
   try {
     ds = DatastoreFactory.newInstance()
       .openDatastore(scriptExecution.getSchemaName());
   } catch (final NoSuchSchemaException e) {
     throw new AppException(IEG.ID_SCHEMA_NOT_FOUND);
   }

   final Entity rootEntity = ds.readEntity(rootEntityID);

   final Entity[] personEntities =
     rootEntity.getChildEntities(ds.getEntityType("Person"));

   rootEntity.setTypedAttribute("householdCount", personEntities.length - 1);
   rootEntity.update();

   return AdaptorFactory.getBooleanAdaptor(true);
 }
```

Notices

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the Merative website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of Merative

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of Merative.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

Merative reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by Merative, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

MERATIVE MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Merative or its licensors may have patents or pending patent applications covering subject matter described in this document. The furnishing of this documentation does not grant you any license to these patents.

Information concerning non-Merative products was obtained from the suppliers of those products, their published announcements or other publicly available sources. Merative has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-Merative products. Questions on the capabilities of non-Merative products should be addressed to the suppliers of those products.

Any references in this information to non-Merative websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this Merative product and use of those websites is at your own risk.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

The licensed program described in this document and all licensed material available for it are provided by Merative under terms of the Merative Client Agreement.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to Merative, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. Merative, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. Merative shall not be liable for any damages arising out of your use of the sample programs.

Privacy policy

The Merative privacy policy is available at https://www.merative.com/privacy.

Trademarks

Merative ™ and the Merative ™ logo are trademarks of Merative US L.P. in the United States and other countries.

IBM®, the IBM® logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Adobe™, the Adobe™ logo, PostScript™, and the PostScript™ logo are either registered trademarks or trademarks of Adobe™ Systems Incorporated in the United States, and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft™, Windows™, and the Windows™ logo are trademarks of Microsoft™ Corporation in the United States, other countries, or both.

UNIX™ is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.