IBM Cúram Social Program Management
Version 7.0.3

IBM Cúram Universal Access 1.1.0

Note

Before using this information and the product it supports, read the information in “Notices” on page 120

Edition

This edition applies to IBM® Cúram Social Program Management v7.0.3 and to all subsequent releases unless otherwise indicated in new editions.

Contents

List of Figures

List of Tables

Chapter 1. IBM Cúram Universal Access (New)

7.0.3.0

IBM Cúram Universal Access (New) enables citizens to access services in a browser from both desktop, tablet, and mobile devices. Universal Access uses modern technologies such as React to provide a working reference application that you can customize to provide your own citizen-facing web application. In comparison, Universal Access, delivered with versions 7.0.2 and earlier, use traditional technologies to customize the citizen-facing web application.

Documentation versions

The responsive Universal Access client uses an asset that is called "IBM Universal Access Responsive Web Application". The asset is updated at more regular intervals than the underlying IBM Cúram Social Program Management platform and therefore has its own version numbering scheme.

The online documentation applies only to the most recent version of Universal Access. To read the documentation in PDF format for the earlier versions, see the IBM Cúram Social Program Management PDF library.

IBM Cúram Universal Access release notes

Read the release notes for the latest release of Universal Access Responsive Web Application.

1.1.0 release notes

Read about updates and changes in Universal Access Responsive Web Application version 1.1.0., which is compatible with IBM Cúram Social Program Management version 7.0.3.

General

922: More flexible internationalization

How internationalization is configured has changed from using AppConfig.json to a more flexible and extensible intl.config.js. For more information, see: https://www.ibm.com/support/knowledgecenter/en/SS8S5A_7.0.3/com.ibm.curam.universalaccess.doc/CitizenEngagement/c_CECUST_Cssartifacts1Translation1.html

Removing AppConfig.json. Migrate the current configuration in AppConfig.json to config/intl.config.js. For more information, see the schema and examples in config/intl.config.js.sample.md.

perf: Bundle size focus

A number of dependencies ave been replaced or removed to reduce the size of the application bundle. This reduction improves the performance of the initial application download.

spm-mock-server

631 feat(mock-server): updates for HTTP errors

Better handling of HTTP error codes for GET requests to mock server using mockapis.js.

spm-universal-access

596 Fix lint warning type 'unexpected console no-console'

The console statement *'console.info(state);'* was removed from universal-access/src/modules/Application/selectors.jS.

612 Fix lint warning type 'Missing JSDoc comment require-jsdoc' and update functions to EC6 arrow functions

- Updated *'mapDispatchToProps'* functions to ES6 arrow functions within `'universal-access-ui\src\features\ApplicationForm\submission-form\SubmissionFormContainer.js'`

  Updated from:

```
createSubmissionForm(applicationFormId, callback) {
   SubmissionFormActions.createSubmissionForm(
     dispatch, applicationFormId, callback); },
 getCurrentPageDetails(submissionFormId, params, callback) {
   SubmissionFormActions.getCurrentPageDetails(
    dispatch, submissionFormId, params, callback); },
 sendCurrentPageAnswers(submissionFormId, data, callback) {
   SubmissionFormActions.sendCurrentPageAnswers(
     dispatch, submissionFormId, data, callback); },
```

  Updated to:

```
createSubmissionForm: (applicationFormId, callback) =>
  SubmissionFormActions.createSubmissionForm(
     dispatch, applicationFormId, callback),
getCurrentPageDetails: (submissionFormId, params, callback) =>
  SubmissionFormActions.getCurrentPageDetails(
    dispatch, submissionFormId, params, callback),
sendCurrentPageAnswers: (submissionFormId, data, callback) =>
  SubmissionFormActions.sendCurrentPageAnswers(
    dispatch, submissionFormId, data, callback),
```

- Updated the function *'getCombinedTranslationMessages'* in `'universal-access-ui\src\utils\Translations\TranslationUtils.js'` to an EC6 arrow function:

  Updated from:

```
pluralRuleFunction(arg1, arg2) {return arg1 && arg2 === 1 ? 'one' : 'other';},
```

  Updated to:

```
pluralRuleFunction: (arg1, arg2) => (arg1 && arg2 === 1 ? 'one' : 'other'),
```

- Added JSDoc for the functions *'createMarkup'* and *'sanitize'* in the *RichTest* component within `'intelligent-evidence-gathering\src\components\RichText.js'`
- Added JSDoc for the component *'AuthenticatedRoute'* in `'universal-access-ui\src\router\AuthenticatedRoute.js'`

spm-universal-access-ui

477 Disabling Next button on Intelligent Evidence Gathering applications to avoid multiple clicks.
To avoid multiple user clicks on the **Next** button, causing multiple requests during an Intelligent Evidence Gathering flow, a new feature is added that disables the **Next** button after the first click and re-enables it after the next page is loaded.

576 Update the Organization page card and secondary menu buttons content that are read by the JAWS accessibility tool

- Updated the organization home page to work with the accessibility tool JAWS:
  - The user collapsible menu has been updated to read the secondary menu **user name**, **profile**, and **log out** buttons.
  - The card descriptions have been updated.
- Updated the Organization card content:

- Two new message properties are added for each card item on the organization home page that is read by the JAWS in `universal-access-ui/src/features/Organisation/components/OrganisationComponentMessages.js`

```
organisationApplyForBenefitsLabel: {
      id: 'Organisation_ApplyForBenefitsLabel',
       defaultMessage: 'Apply for benefits start your application now.',
     },

    organisationViewYourAccountLabel: {
      id: 'Organisation_ViewYourAccountLabel',
      defaultMessage: 'View your account see your next payment and much more.',
    },
```

- Updated the *'Card'* component label reference to the new message properties in `universal-access-ui/src/features/Organisation/components/OrganisationComponent.js`

  Updated from:

```
'aria-label={intl.formatMessage(messages.organisationApplyForBenefits)}'
```

  Updated to:

```
'ariaLabel={intl.formatMessage(messages.organisationApplyForBenefitsLabel)}'
```

  Updated from:

```
'aria-label={intl.formatMessage(messages.organisationViewYourAccount)}'
```

  Updated to:

```
'ariaLabel={intl.formatMessage(messages.organisationViewYourAccountLabel)}'
```

- Updated the secondary menu user name, profile and log out button links.

  JAWS previously read **Profile** for the secondary menu, this function has been updated to read the logged-in user name. When the menu is expanded the **Profile** and**Log out** menu options are read.

  The following changes have been added to `universal-access- ui/src/features/ApplicationHeader/components/ProfileMenuComponent.js`:

  Updated the *'HeaderOverflowMenu' ariaLabel* to read the user name, which previously read the profile menu item.

  - The aria label has been updated from:

```
'ariaLabel={props.intl.formatMessage(translations.headerProfileText)}'
```

    To:

```
'aria-label={props.userFullName}'
```

  - Added aria-label:

```
'aria-label={props.intl.formatMessage(translations.headerProfileText)}'
```

    and:

```
'role="button"'
```

    for the secondary menu item profile.

  - Added aria-label

```
'aria-label={props.intl.formatMessage(translations.headerLogOutText)}'
```

And

```
'role="button"'
```

for the secondary menu item logout.

506 Application title update for accessibility

The application title dialog **'Leave this application?'** was previously implemented using a *'span'* element and has been updated to a *'Link'* element, making the link selectable by JAWS.

The following changes have been added to `universal-access-ui/src/features/ApplicationForm/ApplicationFormHeaderComponent.js`:

- Updated the trigger property from a 'Span' element to 'Link' element in the function 'renderApplicationTitle'.

  Updated code from:

```
<span data-testid="apply-for-benefits-application-title-trigger">{title}</span>
```

  To:

```
const isExitButtonVisible =
   metadata !== undefined && this.state.metadata.exitButton
     ? this.state.metadata.exitButton.visible : false;
 const to = 'javascript:void(0)';
 trigger={
    <Link  data-testid="apply-for-benefits-application-title-trigger"
      id="button-save-and-exit-leave" to={to}> {title} </Link>
 }
```

- Moved the const *'isExitButtonVisible'* from the *'render'* function to the *'renderApplicationTitle'* function.
- Removed the link property *'to={to}'* from the *'Header'* component and added to the trigger link in the function *renderApplicationTitle*.

652 Removed the deprecated lifecycle method 'componentWillMount' adding the initialization of the state 'isLiveValidationEnabled' to the constructor

The component lifecycle method *'componentWillMount'* is called before the component is mounted but has being deprecated. The initialization of the state *'isLiveValidationEnabled'* has been added to the constructor.

The following changes have been added to `'universal-access-ui\src\features\ApplicationForm\application-form\steps\ApplicationFormQuestions.js'`

- Removed the deprecated lifecycle function:

```
componentWillMount() {
  this.enableLiveValidate(false);
  this.props.onRef(undefined);
};
```

- Updated the state for *'isLiveValidationEnabled'* in the constructor:

```
this.state = {
  ..............
  isLiveValidationEnabled: false,
};
```

- Updated the function *'enableLiveValidate'* to an EC6 arrow function:

```
enableLiveValidate(isLiveValidationEnabled) {
```

Updated to:

```
enableLiveValidate = isLiveValidationEnabled => {
```

The following changes have been added to 'universal-access-ui\src\features \ApplicationForm\submission-form\steps\SubmissionFormQuestions.js':

- Removed the deprecated lifecycle function:

```
componentWillMount() {
  this.enableLiveValidate(false);
  this.props.onRef(undefined);
};
```

- Updated the state for both props in the constructor:

```
this.state = {
  ..............
  isLiveValidationEnabled: false,
};
```

- Updated the function *'enableLiveValidate'* to an EC6 arrow function, from:

```
enableLiveValidate(isLiveValidationEnabled) {
```

To:

```
enableLiveValidate = isLiveValidationEnabled => {
```

461 Update accessibility text of the "Leave this application?" dialog

Update the ariaLabel for the application **"Leave this application?"** dialog to describe each dialog button and link when read by JAWS.

The *'leaveDialogDescription'* default message was updated from

```
"Beginning of dialog window. Escape will cancel and close the window.
Save the application link."
```

to:

```
"Beginning of dialog window leave this application. Save will save the application information
entered and close the window. Escape will cancel and close the window. Leave will delete the
information
entered and you will have to start your application again. Save the application link."
```

in 'universal-access-ui/src/features/AccountHome/components/ AccountHomeMessages.js'

452 Update dashboard pod title 'TODOS'

Update the dashboard pod title from 'TO-DOS' to 'TO DO'S'. JAWS previously read this string as *to dash dos'* but is now read correctly.

The *'todoPodTitle'* default message was updated from "'TO-DOS'" to "TO DO'S" in 'universal-access-ui/src/features/AccountHome/components/AccountHomeMessages.js'

632 Replaced the deprecated lifecycle method 'componentWillReceiveProps' with the lifecycle method componentDidUpdate

The lifecycle method *'componentWillReceiveProps'* has been replaced with *'componentDidUpdate'*. This lifecycle function passes in the parameter *'prevProps'* to compare with *'this.props'* and where different the *'changePage'* function is called.

The following code changes have been added:

- universal-access-ui\src\features\ApplicationForm \ApplicationFormHeaderComponent.js

Updated from:

```
componentWillReceiveProps(props) {
  if (props.schemaFormQuestions !== this.props.schemaFormQuestions) {
    this.changePage(props.schemaFormQuestions);
  }
}
```

To:

```
componentDidUpdate(prevProps) {
   if (prevProps.schemaFormQuestions !== this.props.schemaFormQuestions) {
     this.changePage(this.props.schemaFormQuestions);
   }
}
```

- universal-access-ui\src\features\ApplicationForm\application-form\steps\ApplicationFormQuestions.js

  Updated from:

```
componentWillReceiveProps(props)
  { if (
       props.currentPage !== this.props.currentPage ||
       props.schemaFormQuestions !== this.props.schemaFormQuestions
   ) {
    this.changePage(props.currentPage, props.schemaFormQuestions);
    }
}
```

  To:

```
componentDidUpdate(prevProps) {
  if (
     prevProps.currentPage !== this.props.currentPage ||
     prevProps.schemaFormQuestions !== this.props.schemaFormQuestions
   ) {
     this.changePage(this.props.currentPage, this.props.schemaFormQuestions);
   }
}
```

- universal-access-ui\src\features\ApplicationForm\submission-form\steps\SubmissionFormQuestions.js

  Updated from:

```
componentWillReceiveProps(props) {
   if (
     props.currentPage !== this.props.currentPage ||
     props.schemaFormQuestions !== this.props.schemaFormQuestions
   ) {
     this.changePage(props.currentPage, props.schemaFormQuestions);
  }
}
```

  To:

```
 componentDidUpdate(prevProps) {
   if (
     prevProps.currentPage !== this.props.currentPage ||
     prevProps.schemaFormQuestions !== this.props.schemaFormQuestions
   ) {
     this.changePage(this.props.currentPage, this.props.schemaFormQuestions);
   }
 }
```

universal-access-ui

562 feat(universal-access-ui): Introduce route-based code splitting

Introduces route-based code splitting of the built app JavaScript to allow for a smaller initial bundle size. This pattern is achieved through use of react-loadable and routes like the following example:

```
const Organization = Loadable({
  loader: () => import('../features/Organisation'),
  loading: LoadingPage,
});
...
    <Route component={Organization} exact path={PATHS.HOME} />
```

This feature also introduces a *LoadingPage* component that is consumed by react-loadable to handle any error or timeout conditions during the chunk load.

470 feat(universal-access-ui): Updating application title on the browser tab

The application title within the browser tab is updated when a page loads to provide better context to users to indicate where they are within the application.

333 feat(universal-access-ui): move alert notification into benefits selection pane

Previously, the alert notification was outside of the benefits selection pane. Now the alert notification has been moved into the top of the pane section.

554 feat(universal-access-ui): refactoring to fix flicker between pages

The spinner component is now moved into the page it was currently rendered on. Previously, the spinner rendered instead of the page. Now only sections of the page that are still loading will show the spinner. This is to address the flickering issue which is a result of either the page being rendered OR the spinner based on requests to the server or other similar loading processes.

spm-universal-access-mocks

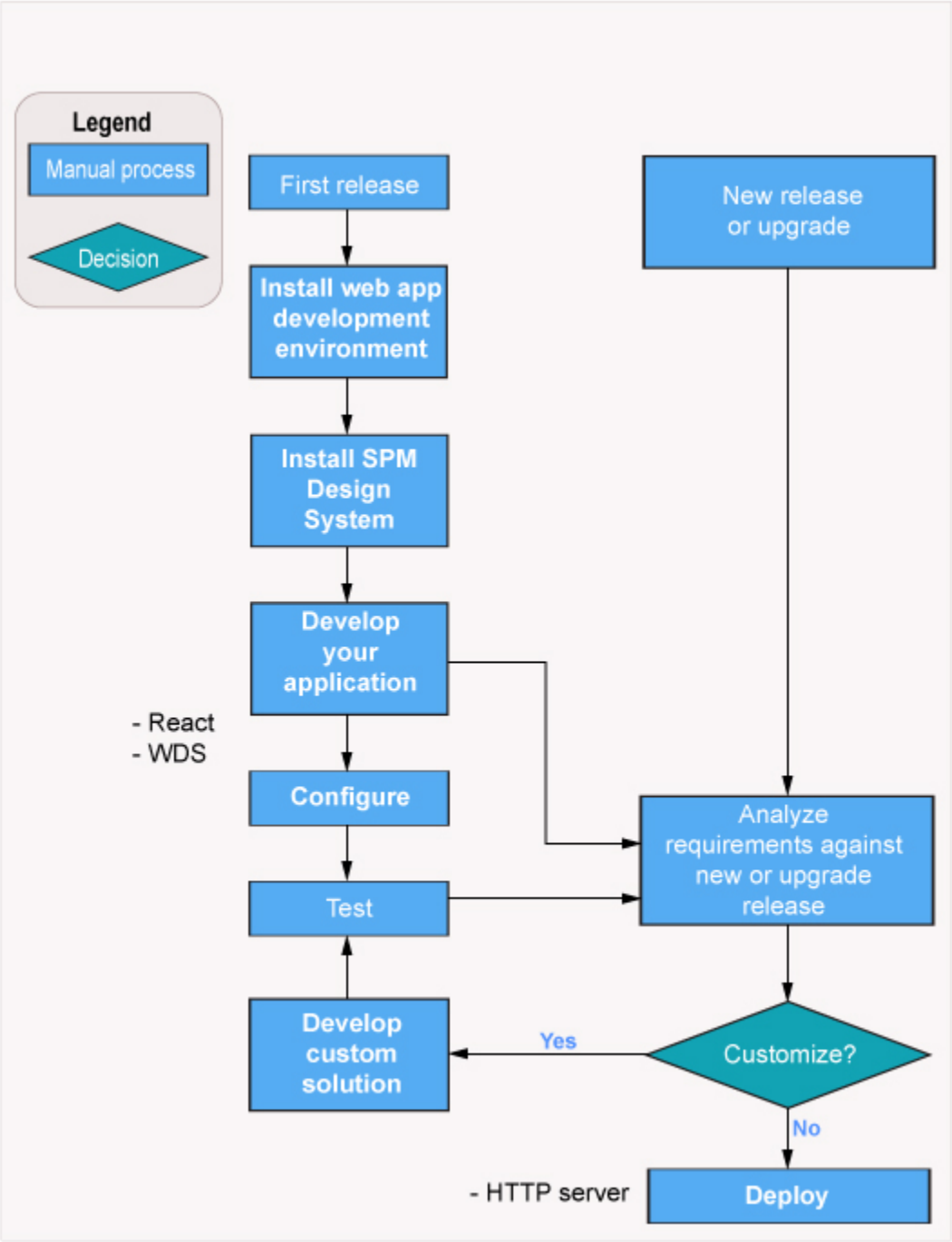
473 feat(universal-access-mocks): iegform submission

Previously, the mock server could not handle submission form requests. The mock server has now been updated to handle and simulate IEGform submissions to support end-to-end testing.

Getting started

IBM Cúram Universal Access enables governments to provide citizens with a single point of access to all social programs and services for which they are eligible. Universal Access connects citizens to programs, streamlines applications for those programs, and reduces administrative work, allowing caseworker to spend more time interacting with citizens.

You can quickly develop and deploy Universal Access by using the Design System. Hover over some of the process steps in bold text, for example, "Develop your application" to find out more about how to build and deploy your Universal Access:



1. “Installing the IBM Cúram Universal Access development environment” on page 20
2. Installing the IBM Social Program Management Design System
3. “Configuring the IBM Cúram Universal Access server” on page 65
4. Developing applications using the IBM Social Program Management Design System
5. “Customizing the IBM Cúram Universal Access application” on page 24
6. Deploying a built application

IBM Cúram Universal Access business overview

A business overview of the online facilities that are provided by Universal Access enterprise module. Use this information to map existing features and capability to your business requirements during a business analysis.

IBM Cúram Universal Access is a citizen-facing web application that provides citizens with online facilities. Read this business overview of the online facilities that are provided by the Universal Access. Use this information to map existing features and capability to your business requirements during business analysis.

Citizen account

When citizens create a secure citizen account, they can access a range of relevant information. Citizens can also use the citizen account to track and manage interactions with the agency.

Creating a citizen account and logging in

Citizens can create a citizen account during the application processes.

Creating an account

Citizens can select **Sign up** on the organization **Home** page to create an account. Citizens then enter their first and last names, an optional email address and account password. If citizens select **I don't have an email address**, they can specify a user name instead.

For more information about the application process, see *Completing and submitting benefit applications*.

Administration configurations

- Number of login attempts before the account is locked out: 5
- Number of remaining login attempts before a user warning is displayed: 3
- Number of break-in attempts before an account is locked: 3

Logging in

To log in to the citizen account, citizens select **Log in** on the organization **Home** page. Depending on how they created their account, citizens enter either an `Email or username` and `password` and then select **Next**. You can configure the number of login attempts citizens have before their account is locked out. For example, if you set the number of login attempts to three, citizens who make three unsuccessful login attempts have their accounts locked out.

In the next page, if the user name and password authentication is successful, the **Citizen account** dashboard is displayed.

Browsing the organization home page

Citizens can browse the home page to find out how the organization can help them, how to apply for benefits, or manage an existing benefit.

Check what you might get

Citizens can select **Check what you might get** on the organization **Home** page to check their eligibility for benefits.

Apply for benefits

Citizens can select **Apply for benefits** on the organization **Home** page to start the application process.

View your account

Citizens can select **View your account** on the organization **Home** page and either view a dashboard of applications and eligibility checks or view their benefits.

Browsing the dashboard

When the citizen logs in, they see the **Dashboard** and the **Your benefits** tabs.

Dashboard

The **Dashboard** is laid out in a series of panes as outlined in Table 1.

*Table 1: **Dashboard** panes*

User interface pane	Description
System messages	System messages are broadcast to all logged-in citizens. System messages inform citizens about, for example, planned system outages.
In-progress applications	Citizens can either continue or delete in-progress applications. **Note:** In-progress applications are also known a draft applications.
Benefits Checker	Lists any in-progress eligibility checks. Citizens can either **Recheck** or **Delete** eligibility checks.
PAYMENTS	Lists the latest payment made to citizens. Citizens can also view payment details or see their payment history.
TO-DOs	Lists actions that citizens must take to complete an application.
MEETINGS	Outlines details of meetings that citizens have been invited to. A date is included for all meetings. The latest meeting is shown first.
NOTIFICATIONS	Shows acknowledgments for all the applications that citizens make. A date is included for all notifications. The latest notification is shown first.

For more information on configuring messages, see *Customizing specific message types*.

Related concepts

Customizing specific message types
Organizations can customize the default message to create a referral message or a service delivery message.

Browsing the Your benefits page

When the citizen logs in, they see the **Dashboard** and the **Your benefits** tabs.

Your benefits

Logged-in citizens who select **Your benefits** on the **Dashboard** are brought to the **Your benefits** page. Citizens who are not logged in are redirected to the **Log in** page, when they log in they are brought to the **Your benefits** page, which displays all types of applications, these are in-progress, pending, withdrawn, denied, and active applications.

If a submitted application is approved by the caseworker and a product delivery case is created for that application, the application also appears on the **Your benefits** page.

The **Your benefits** page displays applications that can be in one of the following states:

- **Application in progress**. The application is in progress but is not yet submitted. Citizens can either continue or delete applications in this category.
- **Pending decision**. The application is awaiting a decision from the case worker. Citizens can either download or withdraw applications in this category.
- **Active**. The caseworker has authorized the application.
- **Denied** The caseworker has rejected the application.
- **Authorization failed**. Citizens can download applications in this state.
- **Withdrawn**. Citizens can withdraw the application if it is **Pending decision** or the caseworker has **Denied** the application.

Viewing payments
The **PAYMENTS** pane on the **Dashboard** lists payments that are made to the citizen. The messages associated with these payments can be retrieved from IBM Cúram Social Program Management or another remote system. Canceled or expired payments are also displayed.

A payment can be made by check, electronic funds transfer (EFT), cash, or voucher.

Depending on the payment type, different details are displayed. The following details can be displayed on for each payment:

Check
Payee address and check number

EFT
Bank sort code and bank account number

Cash
Payee address

Voucher
Payee address and voucher number

Note: Citizens do not see these payment details on the dashboard itself. Instead, citizens must select **All payments** in the **PAYMENTS** panel and then select **>** in a specific payment to see payment details for that payment.

Viewing TO DOs
The **TO DO's** pane on the **Dashboard** lists verifications and action messages that the caseworker creates for the citizen.

A to do could be, for example, a request to provide supplementary information to support a benefit application.

Displaying contact information
The **Contact us** tab, and **Profile** link display the citizen's contact information and the contact information of the agency caseworker.

Citizen information

Citizens can select **Citizen Name** > **Profile** to display their contact information including address, phone number, and email address. A configuration setting determines whether the citizen's contact information is displayed on the citizen account. For example, an agency can set the `curam.citizenaccount.contactinformation.show.client.details` property to `false` to disable citizen contact information. For more information, see *Configuring contact information*.

Caseworker contact information

The **Contact us** tab displays information for the agency caseworker of each case that the citizen is associated with is displayed. Caseworker contact information from IBM Cúram Social Program Management and remote systems can be displayed. The following information can be displayed for the caseworker:

- Name
- Business phone number
- Mobile phone number
- Pager
- Fax
- Email

Use configuration settings to specify the contact details to display and hide on the contact information page. For example, an agency can display an caseworker's business phone number and email address only. Similarly, an agency can hide contact information. For more information about configuring the display of citizen contact information, see *Configuring contact information*.

Related concepts

Configuring contact information
Configure contact information for citizens and caseworkers.

Citizen account messages

The **PAYMENTS**, **TO DO'S**, **MEETINGS**, and **NOTIFICATIONS** panes on the **Dashboard** display citizen account messages. Messages can be about meetings the citizen is invited to, or activities that are scheduled for the citizen. By using web services, messages from remote systems can also be displayed.

Displaying a message

Each message has a title and an icon. In addition, the **TO DO'S** and **NOTIFICATIONS** messages have an effective date and time that specifies when the message is displayed. Usually the effective date of a message is set to the current date, but in some circumstances configuration settings can specify the effective date. For example, when a service is scheduled for the citizen, you might not want to display the message immediately if the service is scheduled for two months in the future. In this case, a configuration setting is provided to specify the number of days before the start date of the service that the message must appear in the citizen's account. For example, the system uses these days to populate the effective date. Messages from remote systems are displayed based on the effective date that is specified in the web service.

Prioritization and ordering

You can assign a priority to a message so that it is displayed at the top of the **MEETINGS** listing.

You can also configure the order of messages types in the administration system. For example, you can configure payment messages to be displayed first and meeting messages to be displayed second.

Message duration

The message type determines the length of time that the message is displayed. The message duration can be set either by start and end dates or by replacing one message with another.

Some messages relate to items that have start and end dates that the agency can use to specify the duration for which a message is displayed. For example, service messages are displayed until the start date of the service has passed. In other cases, it might be appropriate for a message to be replaced by another message. The agency can use a configuration setting to determine whether the agency wants to:

- Specify the duration for when a message is replaced.
- Specify the number of days after which the message is removed.

The duration of messages from remote systems is based on the expiry date that is defined in the web service.

System messages

Agencies use system messages to send a message to everyone who has a citizen account. For example, if an agency wants to provide information and help line numbers to clients who were affected by a natural disaster, such as a flood, hurricane, or earthquake. System Messages can be configured in the Administration application by using the **New System Message** page.

The **Title** and **Message** fields define the title of the message and the message body that is displayed to a client in the **My Messages** pane. The message can be defined with a priority by using the **Priority** field, which means that the message appears at the top of the messages listing.

The **Effective Date and Time** field defines an effective date for the message, such as when the message is displayed in the **Citizen Account** page. The **Expiry Date and Time** field define an expiry date for the message, for instance, when the message no longer is to be displayed in the citizen account.

When the message is saved, it has a status of **In-Edit**. Before the message is displayed in the citizen account, it must be published. When it is published, the message is active and is displayed in the citizen account based on the effective and expiry dates defined.

Predictive Response Manager

The Predictive Response Manager (PRM) is the infrastructure that is used to build and then generate and display messages on the Citizen Account home page.

A number of default messages are provided and are described in this information along with their associated configurations

Applying for benefits

Citizens can apply for benefits from the organization home page or the **Dashboard**. Citizens must submit an application that includes personal details like income, expenses, employment, education. This information is the evidence of the citizen's case. Agencies can use this information to determine eligibility for benefits. Citizens can also apply offline by downloading the application form, filling it in and sending it to the agency. Citizens can also contact their local agency office.

Before you begin

Citizens can apply for benefits by logging in to their account. Citizens who log in can save an application for a benefit before they submit it and then return later to complete the application. Citizens can also partially apply for benefits without logging in. If the configuration option *submit on completion* is set to **No**, citizens can submit a partial submitted application. Citizens do not have to be logged in to submit the partial application.

Note: The terms "benefit" and "program" are synonymous. An application might consist of one or more benefits. For example, the "Income Support" application might contain the "Food Assistance" and "Cash Assistance" benefits.

Procedure

1. Citizens click **Apply for benefits** on the organization **Home** page, the **Dashboard**, or the **Your benefits** tab.

   Note: Benefits are displayed in alphabetical order by default, but you can override this order.
2. For each benefit type, citizens can take the following actions:
   a) Click **Learn more** to find out more about the benefit. If the *More Info URL* setting is configured for the application, **Learn more** is conditionally displayed.

b) Click **Print application** to print the application form, complete it by hand and mail it to the agency. If the *PDF Application Form* setting is configured for the application, **Print application** is conditionally displayed.

c) Click **Apply** to start the application process for the benefit. **Apply** is conditionally displayed if **multiple applications** is set to **Yes** or if **multiple applications** set to **No** and the citizen has no existing, pending decision applications.

Results

If citizens quit the application without saving it, the application displays a warning dialog so that citizens can return to the application if this option is selected in error.

Note: Citizens must click the application name on the page in to see the **Leave this application** dialog. The application name is also conditionally enabled depending on whether the **quit and delete** option is enabled in the IEG script.

Clicking **Leave** brings citizens to the dashboard if they are logged in or the organization home if they are not logged in.

Clicking **Cancel** returns citizens to the point at which they left the application script with the previously entered data available. Citizens can cancel an application without saving at any point before they submit. Citizens can only cancel when the application is in progress, if they **Save** and **Exit** they can only **Delete** the application.

Citizens can also:

- Resume an application by selecting the **Continue** link on the **Your benefits** page, or by selecting **Continue** on any in-progress application alerts in the **Dashboard**.
- **Withdraw** an application. If available, the withdraw option is displayed for the pending decision application on the **Your benefits** page.
- **Delete** an application. Citizens can only delete an *in progress* application that they did not submit to the agency.

Starting and selecting an application
Citizens can select the benefits they want to apply for.

Citizen start an application by selecting **Apply for benefits** on the **Organization** home page or selecting the **Benefits** navigation item. Citizens are then brought to the **Apply for benefits** page.

The **Apply for benefits** page describes each of the available applications. To make it easier for administrators to find the required application, they are grouped into categories, for example "unemployment services". The applications, and their categorization, are defined in the Universal Access Administration section of the Administration Application. Citizens can also **Learn more** about each application or can **Print application** to a PDF file.

Citizens can **Apply** for a benefit. Citizens start an application for a benefit they have already applied for, they can resume the application or they can **Start again**.

Citizens might use an application to apply for one or more programs. Typically, the system prompts citizens to select the programs they want to apply for. However, in two situations the system does not prompt the citizen to select programs:

- A single program is defined for the application.
- Each application is configured so that the citizen can select a program or automatically select all of the programs that are associated with the application.

Configuring the application process

You can configure the application process as follows:

- Each configured application is displayed. If an application has more than one associated program, it is displayed in the second column of the **Apply for benefits** page.

- A configuration property **program selection** is available at the application level. If the property is set to **Yes**, an **Include benefits** page is displayed allowing the citizen to select all, or a subset of the configured programs.
- If an application only contains one program and the configuration property program selection is set to **Yes**, the **Include benefits** page is not displayed.
- If the program selection is set to **No** and the application contains multiple programs, all the programs are automatically applied for and the **Include benefits** page is not displayed.
- A configuration property multiple application is available at the program level. If this property is set to **No** there is an existing pending decision for the program, the **Apply** option is visible but disabled.

When citizens select the applications and the programs they want to apply for, the system starts the associated IEG script. Citizens use the script to complete the selected applications.

Managing existing applications

When a citizen logs in, any existing applications are listed and the citizen is presented with different options that depend on the state of an application.

The agency can configure the system to specify whether citizens need to be authenticated before they apply for benefits:

- If authentication is enabled, citizens must either create a new user account or log in to an account before they start the application process.
- If authentication is disabled, citizens can proceed with the application without authentication.

The configuration property *curam.citizenworkspace.authenticated.intake* specifies whether citizens must log in to apply for benefits. If the property is set to **NO**, citizens do not have to log in to apply for benefits. If the property is set to **YES**, citizens must create an account or log in to an existing account to apply for benefits.

Depending on how authentication is configured, applications are managed in one of the following ways: Citizens can log in to their account, or they can sign up from the application overview page. Citizens can also be prompted to log in, sign up, or send application without an account at the end of the IEG application script.

If citizens create an account, they are automatically logged in to the system and the intake process starts. The system also checks whether they have any existing applications.

The configuration property *curam.citizenworkspace.authenticated.intake* is available at the application level. If this property is set to **No**, citizens can submit a partially completed application, if this property is set to **Yes**, citizens cannot submit a partially completed application.

Existing applications are in one of the following categories:

- **Application in progress**. The application is in progress but is not yet submitted. Citizens can either continue or delete applications in this category.
- **Pending decision**. The application is awaiting a decision from the case worker. Citizens can either download or withdraw applications in this category.
- **Active**. The caseworker has authorized the application.
- **Denied** The caseworker has rejected the application.
- **Authorization failed**. Citizens can download applications in this state.
- **Withdrawn**. Citizens can withdraw the application if it is **Pending decision** or the caseworker has **Denied** the application.

The application lists are displayed only if there are items in the list, that is, if there are no saved applications. If applications are listed, the citizen is presented with different options that depend on the state of an application. The citizen might resume or delete an incomplete application, withdraw a submitted application, or start a new application.

Related concepts

Securing the IBM Cúram Universal Access server

The IBM Cúram Universal Access web application is gives citizens access to their most sensitive personal data over the internet. Security must be a primary concern in the development of citizen account customizations. All projects that are built on Universal Access must focus on delivering security from beginning to end.

Saving an application
By default, applications are automatically saved for citizens who are logged in. Citizens can also manually save applications, including in-progress applications.

During a timeout or the accidental closure of the browser window, the application is automatically saved each time that citizens click **Next** in the IEG script. When citizens click **Next**, the information on the previous page is saved. Citizens can also use the **Benefits** page to resume or to delete each in-progress screening. Automatic saving works for logged-in citizens only. Applications for citizens who are not logged are not saved.

A system property specifies whether applications are automatically saved. By default, this property is enabled. For more information, see *Configuring applications*.

When citizens quit an application, three options are displayed. The options the system displays depends on how the intake application is configured. Citizens can take one of the following actions:

- **Save the application**
- **Leave the application without saving**
- **Cancel** the application

If citizens save the application and they are not logged in, the save application screen is displayed. Citizens can create an account, log in, or send the application without logging in.

The configuration property *curam.citizenworkspace.authenticated.intake* is available at the application level. If the property is set to **No**, citizens cannot submit a partially completed application, so the option to **Send application without account** is displayed when citizens select **Save and exit**. If the property is set to **Yes** citizens cannot submit a partially completed application, so the option to **send application without account** is not displayed when citizens select **Save and exit** .

Related concepts
Configuring applications
The administration system allows you to define different types of applications. Once defined, citizens can submit an application for programs to the agency. For each application, you can configure the available programs and an application script and data schema. You can also configure the remaining applications details, including application withdraw reasons.

Resuming an application
Logged-in citizens can resume an application by selecting the **Continue** link on either the **Dashboard** or the **Your benefits** page.

Selecting the **Continue** link in the citizen's **Dashboard** resumes the application from where the application was last saved. When an application is resumed, the data that is entered is automatically saved as citizens moves from page to page through the script.

When citizens resume an application, they are brought to where they left off when the application was saved.

Submitting an application
To allow citizens to submit an application to the agency, you must specify a submission script for the application in the administration system. After citizens submit an application, the way the script is processed depends on the configuration of the programs for which the citizen is applying.

The application might be submitted when citizens complete the intake script or when they exit a script before it completes. An intake application can be configured so that an agency can dictate whether an application script can be submitted before it is complete or not.

If citizens send an application to the agency, either by exiting or completing a script, the screen that is displayed depends on:

- Whether citizens are logged in
- Whether citizens must either create or log in to an account before the application is submitted.

If citizens are not logged in, they are prompted to log in or create a new account. If the property is enabled, citizens must log in to an existing account or create a new account before the application can be sent to the agency. For more information, see *Managing existing applications*.

Specifying log in requirements

The system can be configured so that:

- Citizens are not required to identify themselves to the system AND
- Citizens can send the application to the agency without logging in or creating an account.

Alternatively, the system can be configured so that citizens must create an account or log in. For more information, see *Managing existing applications*.

Managing in-progress and submitted applications

If citizens log in before they send the application to the agency, the system can determine whether:

- There is an in-progress application of the same type OR.
- Citizens previously submitted applications for the same programs that are still pending disposition, that is, awaiting a decision by the agency.

For an in-progress application of the same type, a page is displayed. From here, citizens can send the new application to the agency or keep the saved application, thus discarding the new application. The options available are to **Start again** or **Resume** the in-progress application.

If citizens submit applications for the same programs, the system determines whether they can still submit any of the programs to the agency for processing. Programs can be configured so that multiple applications can be submitted for the program at any time. For example, submitting a new application for cash assistance for a different household unit than a previously submitted application that the agency is processing. This screen indicates that the application cannot be submitted for all of the programs for which the citizen wants to apply. However, the application might still be sent to the agency. There are three options: continue to submit the application for the programs for which the citizen can apply, save the application, or delete the application.

The configuration property **Multiple application** is available at the program level. If this property is set to **No** and there is a pending decision for the program, the **Apply** option is visible but disabled.

Specifying a submission script

To submit an application to the agency, a submission script must be specified for the application in administration. The submission script is required because applications require additional information, which does not form part of the application, to be captured before the applications can be submitted. For example, a Cash Assistance application requires information that relates to the citizen's ability to attend an interview. This information would not be appropriate for another type of application that does not require an interview to be conducted, for example, unemployment insurance. Electronic signatures are another example of the type of information that would typically be captured by using a submission script. This data might not be captured as part of the script, as citizens can submit the application before completing the script.

Processing a submitted script

The processing that happens on completion of the submission script depends upon the configuration of the programs for which citizens are applying. Program eligibility can be configured such that it might be determined by using IBM Cúram Social Program Management or a remote system. If IBM Cúram Social Program Management is specified as the eligibility system, an application case creation process is started. The application case creation process includes a search and match capability, which attempts to match citizens on a new application to registered persons on the system based on configured search criteria.

When search and match finishes, one or more application cases are created. If the programs that are applied for are configured for different application case types, multiple application cases are created. If the application was submitted within the business hours of the root location for the organization, the application date on the application case is set to today's date. If the application is submitted outside of the business hours of the organization, the application date is set to the next business date.

Mapping the application data to case evidence tables

The data that is entered for the application might be mapped to case evidence tables. The mappings are configured for a particular program by using the Cúram Data Mapping Editor. For the appropriate evidence entities to be created and populated in response to an online application submission, a mapping configuration must be specified for a program.

Associating requested programs with application cases

When the application case is created, the programs that are requested by the citizen are associated with the relevant application case. Some organizations might impose time limits within which an application for a program must be processed. A number of timer configuration options are available for a particular program. These timers are set when a program is associated with an application case.

If the eligibility is determined by a remote system, configurations are provided to allow a web service to be started on a remote system.

Displaying submission confirmation

The submission confirmation page is displayed upon successful submission of an application to the agency. The submission confirmation page displays the reference number that is associated with the submitted application. Citizens can use this reference number in any further correspondence about application with the agency.

Configuring intake applications for PDFs

The citizen might also open and print a PDF. The configuration of the intake application determines the actual PDF that opens. The application can be configured to use a PDF designed specifically by the agency with the intake application, or, if no PDF form is specified, to use a generated generic PDF. If an agency-designed form is specified, this form is opened when the citizen clicks the **PDF** link. For programs with associated mapping configurations of type **PDF Form Creation**, the data that is entered during the online application is copied to the PDF form. The data is copied for each of the programs for which the citizen is applying with this mapping configuration. If a mapping configuration is not associated with a program, the information that is entered during the online application for that program is not copied to the PDF form. If a PDF form is not specified, a generic generated form opens instead. This form contains a copy of the information that is entered by the citizen when the citizen is completing the online application.

The agency can define additional information to be displayed on the generic generated form. Typically, the information that might be required by the citizen is to help the agency process the application timely and effectively for both the agency and citizen. Proof of identity is an example of this additional information. This additional information is configurable for each type of application.

Submission confirmation

When citizens successfully submit an application, going through the sign and submit screen, they are brought to an updated version of the **Overview**. The stages specific to the application process are now updated with a confirmation message to indicate that the application was successfully submitted:

- A customizable icon
- An application reference number
- Informational message for the citizen
- A **Save submitted application PDF** link that allows citizens to download the information entered as part of the application, in PDF format.

Related concepts

Managing existing applications

When a citizen logs in, any existing applications are listed and the citizen is presented with different options that depend on the state of an application.

Printing an application

Citizens can open and print an application form in two ways.

- Citizens are directed to a PDF that they can open, complete, and print.
- Citizens are taken through a script. After citizens complete or exit the script, they can open a PDF containing the information they entered.

PDF forms can be configured to provide versions in all supported languages. The programs that can be applied for using the PDF form can also be configured.

Each PDF form that is defined in the administration system is displayed on the **Apply for Benefits** page. The **Apply for benefits** page is displayed when **Apply For Benefit** is selected from the organization **Home** page.

If **PDF Application Form** is configured for the application, **Print application** is displayed.

To open the PDF form, citizens click **Print application**. Citizens can also identify the address of the local office to which to send the form. A system property sets whether the system uses postal codes or counties for this function.

Withdrawing an application

Citizens can withdraw successful applications from the **Your benefits** page. If the application did not successfully submit, the **Withdraw** option is not displayed.

Citizens can withdraw a successfully submitted application or they can also withdraw applications for all or any one of the programs.

Citizens can withdraw each program individually. The reasons for withdrawing the program application can be configured for the intake application in the administration system.

The **Reason** field contains a list of configurable code table values that are defined by the administrator. The list of values is configured at application level.

The **First name**, **Last name**, and **Reason** fields are mandatory.

The submit action on the page withdraws the application. The system automatically updates the status of the programs that are associated with the application case to **Withdrawn** and sends a notification to the application caseworker.

The difference between deleting and withdrawing an application

The **Withdraw** action is different from the **Delete** action in that only a submitted application can be withdrawn and only an in-progress application can be deleted. Also, **Delete** physically deletes the application record, **Withdraw** changes the status of the application to *Withdrawn* after the citizen goes through a workflow.

Related concepts

Citizen account

When citizens create a secure citizen account, they can access a range of relevant information. Citizens can also use the citizen account to track and manage interactions with the agency.

Deleting an application

Citizens can delete applications that are not yet submitted to the agency.

Citizens can delete applications from the **Dashboard** or the **Your benefits** pages. When citizens click the **Delete application** link for an in-progress application, a confirmation dialog is displayed.

Installing the IBM Cúram Universal Access development environment

Before you install IBM Cúram Universal Access, install the prerequisites.

Prerequisites and supported software

The supported software for accessibility, HTTP servers, development tools, and web browsers.

Development tools

Node.js is a prerequisite for installation.

Supported software	Version	Prerequisite minimum	Product minimum	Components	Operating system restrictions	Details
NodeJS	8 and future fix packs	8	1.0.0		No	

HTTP servers

These HTTP servers are prerequisites for deployment.

Supported software	Version	Prerequisite minimum	Product minimum	Components	Operating system restrictions	Details
IBM HTTP Server	9.0	9.0.0.5	1.0.0		No	
	8.5.5	8.5.5.9	1.0.0		No	
Oracle HTTP Server	(12.1.3) and future fix packs	(12.1.3)	1.0.0		No	

Accessibility

This accessibility software is supported.

Supported software	Version	Prerequisite minimum	Product minimum	Components	Operating system restrictions	Details
Freedom Scientific JAWS screen reader	18 and future fix packs	18	1.0.0		No	
Apple VoiceOver	Any version and future fix packs	Any version	1.0.0		No	

Web browsers

The IBM Cúram Universal Access is developed for public facing applications. While every effort is made to ensure that the pages specified for the application use standard web technologies and formats that are compatible with all browsers, the listed browsers are the only supported browsers.

Chrome, Firefox, Edge and Safari release new versions more frequently than Internet Explorer, and they install updates automatically by default. Releases of Universal Access are tested on the latest browser versions that are available at the start of the IBM development cycle.

Note: Only stable Chrome releases are tested.

If no issues result from the tests, IBM certifies the browser version.

For each new product release, the prerequisites list the version that is certified. If, for any reason, IBM cannot certify that version, you might need to revert to a version that is previously fully certified. While IBM supports customers who use newer versions of these browsers than the last certified version, customers must understand that the versions are not fully tested.

Supported software	Version	Prerequisite minimum	Product minimum	Components	Operating system restrictions	Details
Apple Safari	11 and future fix packs	11	1.0.0		No	Not applicable
Google Chrome	64 and future fix packs	64	1.0.0		No	Not applicable
Microsoft Edge	41 and future fix packs	41	1.0.0		No	Not applicable
Microsoft Internet Explorer	11 and future fix packs	11	1.0.0	Not applicable	No	Not applicable
Mozilla Firefox	58 and future fix packs	58	1.0.0	Not applicable	No	Not applicable

Installing the IBM Cúram Universal Access node packages

You can install the Universal Access and IBM Social Program Management Design System node packages into a lightweight or a full development environment.

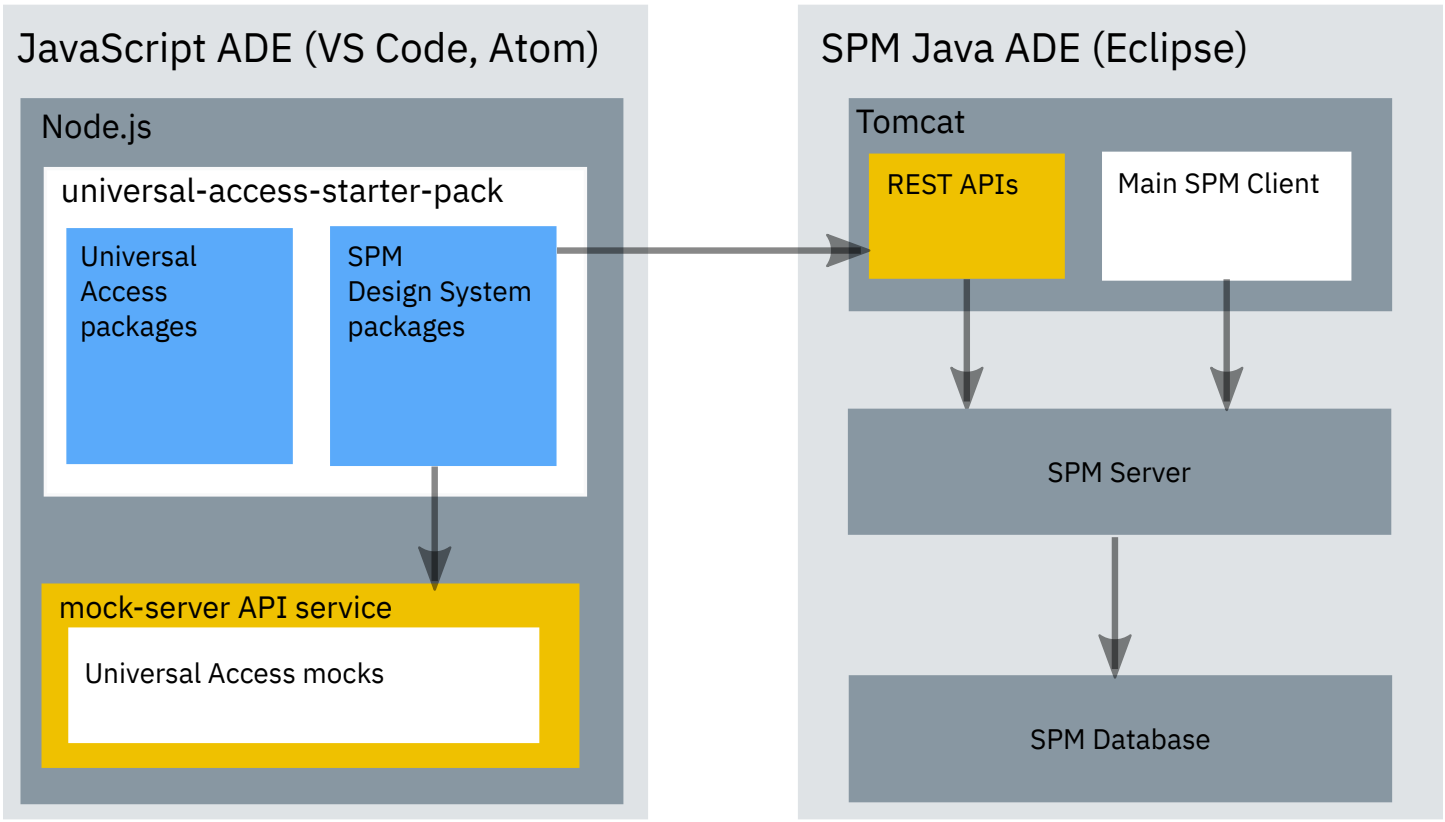
Before you begin

Lightweight development environment

For quick and easy installation, install the Universal Access React application. Then use the `universal-access-mocks` package to provide mock data specific to Universal Access business scenarios for testing purposes. `universal-access-mocks` is consumed by the mock server to provide mock APIs in the development environment so that you do not have to host an IBM Cúram Social Program Management server during development.

Full development environment

Install the Universal Access React application. Then, instead of using the `universal-access-mocks` package, install the SPM Java Application Development Environment (ADE) to develop and test your APIs. For more information, see *Installing an SPM React ADE*.



About this task

The Universal Access `@spm/universal-access-starter-pack` package provides a starter React application into which you install the other packages, its installation differs from that of other packages.

- `@spm/universal-access-starter-pack`
- `@spm/mock-server`
- `@spm/universal-access-mocks`
- `@spm/universal-access`
- `@spm/universal-access-ui`

Procedure

1. Download the Universal Access Responsive Web Application and IBM Social Program Management Design System Node packages.
   a) Open IBM Fix Central, select **Cúram Social Program Management**, your installed version and platform, and click **Continue**.
   b) Ensure that **Browse for fixes** is selected, and click **Continue**.
   c) Select the check boxes for `IBMUniversalAccessResponsiveWebApplication` and `IBMSocialProgramManagementDesignSystem` and click **Continue**.
   d) Only versions that are compatible with your IBM Cúram Social Program Management version are shown. Download SPM_DS_*<version>*.zip and UA_Web_App_*<version>*.zip and extract the packages in the archive files to any directory.
2. Extract the `spm-universal-access-starter-pack_`*`version`*`.tgz` file.

   The extracted `package` directory forms the React App, all other packages are installed into this directory.
3. Rename the extracted `package` directory to reflect your project. For example, `universal-access-custom-app`.

4. From the renamed, extracted package directory, install the IBM Social Program Management Design System packages. Enter the following commands.

```
npm install <path>/govhhs-govhhs-design-system-core-<version>.tgz
npm install <path>/govhhs-govhhs-design-system-react-<version>.tgz
npm install <path>/spm-core-<version>.tgz
npm install <path>/spm-intelligent-evidence-gathering-<version>.tgz
```

   Where *<path>* is the download path, and *<version>* is the package version.

   Note: Ignore any Node package dependency warnings for now. If needed, you can resolve them later.
5. Enter the following commands from the renamed, extracted package directory to install IBM Cúram Universal Access packages. Ignore any warnings you might see. <path> and <version> refer to the download path and package version.

```
    npm install <path>/spm-universal-access-<version>.tgz
    npm install <path>/spm-universal-access-ui-<version>.tgz
```

6. Run the following command to install the package dependencies.

```
npm install
```

7. You can run the Universal Accessreference application by entering the following command from your application directory.

```
npm start
```

   If the local host does not start automatically, browse to http://localhost:3000/ to see the running application.

Results

You can now begin to customize the reference Universal Access application for your organization in different ways depending on the installation option you chose in *Installation alternatives*.

Related information

Installing an SPM React ADE

Upgrading to later versions of IBM Cúram Universal Access

You can upgrade your packages to later versions of IBM Cúram Universal Access.

Procedure

1. Go to IBM Fix Central and search for your product and version to locate the latest version for your installation.
2. Download and extract the new image.
3. Read the 1.0.1 release notes. Take note of any pre-installation steps, requirements, restrictions, installation steps, and post-installation steps that might apply between the new version and your current version.
4. Read the `readme` file that is downloaded with the version. Take note of any pre-installation steps, requirements, restrictions, installation steps, and post-installation steps.
5. Extract the `universal-access-starter-pack` to a temporary directory and compare it to your working project directory. Apply any differences you find to your working directory.
6. Enter the following commands from the renamed, extracted package directory to install IBM Cúram Universal Access packages. <path> and <version> refer to the download path and package version.

Note: Ignore any warnings that you might see.

```
npm install <path>/spm-mock-server-<version>.tgz
npm install <path>/spm-universal-access-<version>.tgz
npm install <path>/spm-universal-access-ui-<version>.tgz
npm install <path>/spm-universal-access-mocks-<version>.tgz
npm install <path>/spm-universal-access-starter-pack-<version>.tgz
```

7. Upgrade your design system, for more information, see *Upgrading to a new version of the design system*.

Related tasks

Installing the IBM Cúram Universal Access node packages
You can install the Universal Access and IBM Social Program Management Design System node packages into a lightweight or a full development environment.

Related information

Upgrading to a new version of the design system

Customizing the IBM Cúram Universal Access application

Customize the reference application and build your custom Universal Access application by using the development resources supplied.

Planning for development

Review the supported prerequisites, download the required software, and review the release notes.

Planning steps

When you complete the following steps, you are ready to start developing your app:

- Choose an Integrated Development Environment (IDE) to develop your application, for more information, see *Development environment*.
- Decide which installation alternative you want to use. You can install IBM Cúram Universal Access in two ways, as a lightweight installation or as a production-like installation. For more information, see *Installing IBM Cúram Universal Access*.
- Review the supported prerequisites to identify the supported versions of your selected software. For more information, see *Installation prerequisites*.
- Download the software that you need from IBM Passport Advantage® or from another software vendor websites as appropriate.
- Install Universal Access, for more information, see *Installing IBM Cúram Universal Access*.
- Deploy IBM Cúram Universal Access. For more information, see *Deploying IBM Cúram Universal Access.*
- Review the latest “IBM Cúram Universal Access release notes” on page 1 and complete any relevant post-installation steps.

Related tasks

Installing the IBM Cúram Universal Access node packages
You can install the Universal Access and IBM Social Program Management Design System node packages into a lightweight or a full development environment.

Building IBM Cúram Universal Access for deployment
Build Universal Access for deployment on an HTTP server.

Development environment

Choose an Integrated Development Environment (IDE) to develop your application.

Development Tools

There are many IDEs that you can use to develop your application, for example:

- Visual Studio Code
- Atom
- Sublime
- Vim
- Webstorm

There is no dependency on any specific IDE, so you can choose your own environment. However, IBM uses VSCode, which supports many plug ins that make development faster and easier, for example:

- Linting tools (ESLint)
- Code formatters (Prettier)
- Debugging tools (Debugger for Chrome)
- Documentation tools (JSDoc)

IBM does not own, develop, or support these tools.

Development resources

IBM Cúram Universal Access includes resources that you can use with the IBM Social Program Management Design System to customize and extend Universal Access.

universal-access-starter-pack

A development environment and a fully functional and deployable reference application. The application uses the IBM Cúram Social Program Management modules (core, web-design-system, universal-access, universal-access-ui) to provide a client that can interact with Universal Access.

You can rename, modify, and extend the starter pack to customize the reference application to suit the needs of your organization. The pack demonstrates how a modern and responsive Universal Access client can be built by using React, Redux and the IBM Social Program Management Design System.

universal-access

This module connects the client application to the IBM Cúram Social Program Management server. *universal-access* makes HTTP requests to the IBM Cúram Social Program Management server to allow the client to interact with a Universal Access installation. Unlike the *universal-access-ui*, this module does not render content. This module uses Redux as a storage mechanism for requests and responses. For more information, see *Working with Redux*.

universal-access-ui

universal-access-ui contains a set of features that presents the views to the user, it depends on *universal-access* to provide the data it needs for those views.

universal-access-mocks

This module provides mock data specific to Universal Access business scenarios for testing purposes. It is consumed by the mock server to provide mock APIs in the development environment so that developers are not required to host an IBM Cúram Social Program Management server during development.

mock-server

The *mock-server* module is a lightweight server that can serve HTTP requests and return mock data as a response. Use *mock-server* during client development as a substitute for a real server to test features.

For more information, on IBM Social Program Management Design System packages, see *Design system packages*.

Related information

Design system packages

Developing compliantly

Follow these guidelines to protect your project from making customization changes that are incompatible with the base product, or have the potential to incur upgrade impacts.

Never use undocumented APIs

JavaScript does not have access modifiers such as public/private/protected. It is possible to call functions in SPM modules that are not intended for public use. Calling these functions is not supported as those APIs can change in a future release and break your code.

The only JavaScript APIs that are intended for public use are documented in the `docs` folder of the SPM `node_modules`. For example, `node_modules/@spm/core/docs/index.html`.

Observe the reducer namespace

If you use Redux, your Reducer names must not infringe on the namespace for universal access reducers. All universal access reducers are prefixed with UA, for example. UA`BenefitSelection`. When universal access and custom reducers are combined, clashing names override the universal access reducers. Customizing universal-access reducers is not supported.

Don't modify the starter pack files

While you can modify the starter pack files in place, it is better to copy the files and change the copy. Your upgrades will then be easier as you can compare files between the current and previous version of the product without the added complexity of your customization changes. Where upgrade changes are needed, manually apply the changes to your custom version.

Developing with routes

Routes define the valid endpoints for navigation in your application. Your application consists of a network of routes that can be traversed by your users to access the application's pages.

IBM Cúram Universal Access uses the *react-router* and *react-router-dom* packages to manage navigation. React Router defines and works with routes. For more information, see the React Router documentation at https://reacttraining.com/react-router/web/guides/philosophy.

The Routes component

The module for Universal Access exports the *Routes* component, which exposes the routes defined by the module. The defined routes are the suite of pages that are prebuilt and available for reuse in Universal Access.

Routes component

You can import and reuse the Routes component in your application. The code example shows how import and reuse the Routes component in a sample application.

```
import React from 'react';
import { injectIntl, intlShape } from 'react-intl';
 import { HashRouter } from 'react-router-dom';
import '@spm/web-design-system/js/govhhs-design-system-core.min';
 import { Routes } from '@spm/universal-access';

const App = (props) => {
  return (
    {/** You must define your routes controller (Hash vs Browser) */}
     <HashRouter>
      <div className="app">
        <div className="my-header-navigation">
          <a href="#/">Home</a> | <a href="#/faq">Faq</a>
        </div>
         <Routes />
      </div>
     </HashRouter>
  );
};

App.propTypes = {
  intl: intlShape.isRequired,
};

export default injectIntl(App);
```

Note: In the example application, the Routes are wrapped in a *HashRouter*. However, the module for Universal Access does not currently support the *BrowserRouter* component. Routes do not resolve if you use *BrowserRouter*. For more information, see https://reacttraining.com/react-router/web/guides/philosophy.

Adding routes

You can add a route by including a new route anywhere inside your Router component.

The following code example adds a route to `MyNewPageComponent` into the router component:

```
import { HashRouter,  Route  } from 'react-router-dom';
…
<HashRouter>
 <div className="app">
  <div className="my-header-navigation">
   <a href="#/">Home</a> | <a href="#/my-new-page">New Page</a>
  </div>
  <UARoutes />
   <Route path="/my-new-page" component={MyNewPageComponent} />
 </div>
</HashRouter>
```

Replacing routes

You can replace existing paths from the Universal Access module's Routes component with your preferred component.

Wrap your routes in a *<Switch>* component

You can replace existing paths from the Routes component with your preferred component. To achieve this, you must first wrap your routes in a *<Switch>* component from react-router. This action ensures that the first match of the requested path that is found in your application is used to resolve the path. For more information on Switch, see https://reacttraining.com/react-router/web/guides/philosophy.

Add a route with the same path

When you have wrapped in *Switch*, you add a route with the same path as the page you are overriding.

Note: This route must come before the *<Routes/>* component to ensure it is matched first.

The following code example shows a replacement route to *MyHomePageComponent* enclosed in a *<Switch>*:

```
import { HashRouter, Route, Switch } from 'react-router-dom';
…
<HashRouter>
 <div className="app">
  <div className="my-header-navigation">
   <a href="#/">Home</a> | <a href="#/my-new-page">New Page</a>
  </div>
  <Switch>
   <Route path="/" component={MyHomePageComponent} />
   <Routes />
   <Route path="/my-new-page" component={MyNewPageComponent} />
  </Switch>
 </div>
</HashRouter>
```

Redirecting routes

You can redirect existing paths by using the react-router Redirect component.

Redirecting a route

The following code example imports the *Redirect* component and redirects the path *'/bring-me-home'* to *"/"*.

```
import { HashRouter, Route, Switch, Redirect } from 'react-router-dom';
…
<HashRouter>
 <div className="app">
  <div className="my-header-navigation">
   <a href="#/">Home</a> | <a href="#/my-new-page">New Page</a>
  </div>
  <Switch>
   <Route path="/" component={MyHomePageComponent} />
   <Redirect path"/bring-me-home" to="/"
   <Routes />
   <Route path="/my-new-page" component={MyNewPageComponent} />
  </Switch>
 </div>
</HashRouter>
```

Removing routes

You can remove unwanted routes from IBM Cúram Universal Access.

You might want to reuse some but not all of the Universal Access <Routes/>. For those routes that you want to remove instead of replacing, use the react-router <Redirect> component to send users to a '404' style page, or some other valid end point.

You must declare the redirect before the <Routes/> component. You must also wrap the redirect in a *<Switch>* component. The following code example removes the route to "FAQ" by redirecting to a 404 page:

```
<HashRouter>
 <div className="app">
  <div className="my-header-navigation">
   <a href="#/">Home</a> | <a href="#/faq">FAQ</a>
  </div>
  <Switch>
   <Redirect path="/faq" to="/404page" />
   <Routes />
  </Switch>
 </div>
</HashRouter>
```

Advanced routing
IBM Cúram Universal Access is now code-split based on routes.

Code splitting

Code-split based on routes is achieved using *react-loadable* and the `@spm/universal-access-ui` package that is in the default *LoadingPage* component. For more information, see For more information, see https://github.com/facebook/create-react-app/blob/master/packages/react-scripts/template/README.md#code-splitting and https://github.com/jamiebuilds/react-loadable. The following example shows how to achieve the same split with the routes that you added:

```
import { LoadingPage } from '@spm/universal-access-ui';
…
const MyNewPageComponent = Loadable({
 loader: () => import(/* webpackChunkName: "MyNewPageComponent" */ '../features/
MyNewPageComponent'),
 loading: LoadingPage,
});
…
   <Route
    component={MyNewPageComponent}
    exact
    path='/my-new-page'
    />
```

Titled routes

Accessibility rules require that a web page should have a descriptive title. You can implement a descriptive title using the *TitledRoute* component of the *@spm/universal-access-ui* package. To localize the title, *TitledRoute* exposes a title prop that accepts a *react-intl message ()* and can be used with or without code-split routes as shown in the following example:

```
import { TitledRoute } from '@spm/universal-access-ui';
import { defineMessages } from 'react-intl';
…
const titles = defineMessages({
 myNewPage: {
   id: 'app.titles.myNewPage',
   defaultMessage: 'My New Page',
 },
});
…
   <TitledRoute
    component={MyNewPageComponent}
    exact
    path='/my-new-page'
    title={titles.myNewPage}
  />
```

Authenticated routes

You can protect parts of the application in two ways:

1. On access , handle authentication failures to a REST API and redirect to a login page.
2. Block access to specific routes to avoid any cost in running the REST API.

The following example shows how to block access to specific routes. The *@spm/universal-access-ui* package provides an *AuthenticatedRoute* component that accepts an *authUserTypes* array prop of the allowed user types to access this route. *AuthenticatedRoute* also wraps *TitledRoute* and therefore offers a title prop. The following is an example of using *AuthenticatedRoute*:

```
import { AuthenticatedRoute } from '@spm/universal-access-ui';
import { Authentication } from '@spm/universal-access';
import { defineMessages } from 'react-intl';
…
const titles = defineMessages({
 myNewPage: {
   id: 'app.titles.myNewPage',
```

```
    defaultMessage: 'My New Page',
  },
});
…
    <AuthenticatedRoute
     authUserTypes={[Authentication.USER_TYPES.STANDARD, Authentication.USER_TYPES.LINKED]}
     component={MyNewPageComponent}
     exact
     path='/my-new-page'
     title={titles.myNewPage}
    />
```

The example blocks access to the */my-new-page* routes for all users who are not of type STANDARD or LINKED, these users are redirected to the */login* route.

Connecting to Universal Access APIs

You must connect your web application to IBM Cúram Social Program Management Universal Access REST APIs. You can use the mock server API service and the RESTServices utility to help you to develop and test your REST API connections.

The mock server API service

The mock server is a mock API service that is provided to aid rapid development. The mock server serves APIs that simulate calling real web APIs. When you are developing your application, the mock server provides a lightweight environment against which the React components can be tested communicating with the services that provide their data.

Configuring the mock server

Configure the mock server location through the following properties in the `.env.development` file. You can change these values to suit your needs.

- `REACT_APP_REST_URL=http://localhost:3080`
- `REACT_APP_BASE_URL=http://localhost:3080`
- `REACT_APP_API_URL=http://localhost:3080`
- `MOCK_SERVER_PORT=3080`

Running the mock server

Run the mock server by using the following command from the root directory of your project:

```
npm run start:mock-server
```

However, when you are developing locally, you can use the following command that starts both the mock server and the client:

```
npm run start
```

See the `package.json` file in your project for the full list of commands.

Adding mock APIs

The universal-access project includes a number of mock APIs that simulate calling the SPM Universal Access APIs. These mock APIs support running some basic scenarios in development mode for the existing set of features.

As you develop your application, you typically create new APIs that you also want to mock. When the mock server starts, it looks to import the `/mock/apis/mockapis` file relative to the folder the command

was started from. In this file, the mock-server expects to find three objects, GET, POST, and DELETE, that it can query to serve API requests for those HTTP methods.

The format of the mock definition is a relative URL that is assigned a JavaScript object. For example, the following code assigns the object `user` to the URL `/user`, and the object `payments.json`, which is read from a file, to the `/payments` URL.

```
const user = {

  'firstname': 'James',

  'surname': 'Smith',

  'gender': 'male',

   ...

}

const mockAPIsGET = {
  // ADD YOUR GET MOCKS HERE

  // Example of providing mock data in response to an API request in
  // the format uri:mockobject
  '/user': user,

  '/payments': readFile('./payments/payments.json)
};
```

If you use mocking extensively, it is better to use separate files and folders to structure your mocks.

Using universal-access mock APIs

The `mockapis.js` file is preconfigured to import and use mock APIs defined and exported by the universal-access package. This allows your project to reuse and extend the set of universal-access mock APIs.

```
const mockAPIs = require('@spm/universal-access-mocks');

// Extract the existing universal access GET,POST and DELETE mocks for merging.
const UAMockAPIsGET = mockAPIs.GET;
const UAMockAPIsPOST = mockAPIs.POST;
const UAMockAPIsDELETE = mockAPIs.DELETE;

...

//create custom mocks

...

// Merge UA mocks with custom mocks
const GET = Object.assign({}, UAMockAPIsGET, mockAPIsGET);
const POST = Object.assign({}, UAMockAPIsPOST, mockAPIsPOST);
const DELETE = Object.assign({}, UAMockAPIsDELETE, mockAPIsDELETE);

module.exports = { GET, POST, DELETE };
```

Where the same URL is used by a custom mock that was previously assigned to a universal-access package mock, the custom mock replaces the universal access version.

The RESTService utility

The `@spm/core` package provide the RESTService utility, which you can use to connect your application to a REST API. You can fetch resources with alternatives such as Fetch API, SuperAgent,or Axios. However, the RESTService utility provides some useful functions for connecting to SPM REST APIs.

The RESTService utility supports the GET, POST, and DELETE HTTP methods through the following JavaScript methods:

- `RESTService.get(url, callback, params)`

- `RESTService.post(url, data, callback)`
- `RESTService.del(url, callback)`

The full RESTService class documentation is in the `doc` folder in the `@spm/core` package.

The RESTService utility hides details of calls, such as passing credentials, language, and errors. The callback that is passed to the GET, POST, or DELETE methods is started after the API calls return. API calls are asynchronous, so write your code to expect and handle a delay in receiving a response.

The RESTService utility provides the following functions during communications.

Authentication

Authentication of the user is handled transparently by the RESTService utility. After a user is authenticated, the REST APIs automatically send the required 'credentials', that is, the authentication cookies, with each request. For more on how authentication is handled for REST, see Cúram REST API security.

If a user's session is invalidated before a new request is made to a REST API, then the '401 unauthorised' response is returned by the server. The the RESTService utility relays the response to the callback function passed by the caller.

Handling responses

The RESTService utility formats the response from the server to ensure that callbacks receive the response in a consistent manner.

Each get, post, and delete method accepts a callback function from the caller. When invoked by the RESTService utility, the callback function receives a boolean that indicates the success or failure of the API call and the response. The callback function can then deal with the result. For example, a failure can be used to trigger your code to throw an error with the response data that can be used to trigger an error boundary. For more information on the callback function parameters, see the API documentation for the RESTService utility.

User Language

The 'Accept-Language' HTTP header is automatically set by the RESTService utilitye based on the user's selected language, which the user can select using the language picker in the reference application. This allows the server to respond in the correct locale where locale sensitive information is being handled on the server.

The locale passed in the header is set in the transaction that is initiated by that REST request, and is used for the duration of that transaction. For more on transactions, see Transaction control.

Handling Timeouts

The RESTService utility can manage unresponsive calls to the server. The following properties are set, and can be modified, in the `.env` files to set thresholds for timeouts.

- `REACT_APP_RESPONSE_TIMEOUT=10000 // Wait 10 seconds for the server to start sending`
- `REACT_APP_RESPONSE_DEADLINE=60000 // but allow 1 minute for the file to finish loading`

Simulating slow responses

During development, it is important to test that your application continues to operate in an acceptable way even when network responses are slow. You can simulate a slow network connection by setting a property in the `.env.development` file in the root of your project.

For example, setting `REACT_APP_DELAY_REST_API=2500` delays the response from all GET requests for 2.5 seconds.

The value can be set to any positive integer to adjust the delay.

Table 1 outlines the process environment variables that the API uses. Use the variables to configure the service.

Table 2: Process environment variables

Variable	Setting	Default	Description
REACT_APP_RESPONSE_TIMEOUT	Milliseconds	10 seconds	Sets the maximum time to wait for the first byte to arrive from the server, but does not limit how long the entire download can take. Set the response timeout to be a few seconds longer than the actual time it takes the server to respond. The lengthened response allows for time to make DNS lookups, TCP/IP, and TLS connections.
REACT_APP_RESPONSE_DEADLINE	Milliseconds	60 seconds	Sets a deadline for the entire request, including all redirects, to complete. If the response is not fully downloaded within REACT_APP_RESPONSE_DEADLINE, the request is aborted.
REACT_APP_DELAY_REST_API	Milliseconds		Use only for development testing to simulate a delay in the response from the API.

Developing authentication

The universal-access package exports the Authentication module, which can be used to log in and out of the application and to inspect the details of the current user. The login service is passed a user name and password, and optionally a callback function that is invoked when the authentication request is completed.

Authentication services

The Authentication API works in three modes:

- Simple Authentication (Development mode)
- SSO Authentication
- JAAS Authentication

Simple Authentication (Development Mode)

During client development, the authentication defaults to use a simple authentication that does not require an SPM server. This simple authentication bypasses proper authentication (JAAS or SSO) and instead accepts the user name `dev` without any password. The login process can be ran and allows access to the 'user account' password protected pages.

This simple authentication is sufficient to do most client development work and avoids the need to configure your client application to communicate with an SPM server. It is triggered by the `REACT_APP_SIMPLE_AUTH_ON=true` property in the `env.development` file.

You can set `REACT_APP_SIMPLE_AUTH_ON=false` if you want to trigger an SSO or JAAS login service.

SSO Authentication

The application supports single sign-on (SSO), which is a typical use case for many enterprises that serve multiple applications with a single user name and password for their clients. The client application can be configured to use SSO through the `REACT_APP_SSO_ENABLED=true` environment property.

For more information about configuring your universal access deployment to use SSO, see "Configuring single sign-on" on page 92.

JAAS Authentication

If not in development mode, and not using single sign-on, then the login process defaults to use the standard JAAS login module.

- `REACT_APP_SIMPLE_AUTH_ON=false`
- `REACT_APP_SSO_ENABLED=false`

The JAAS login module is exposed through the SPM universal access API at the `/j_security_check` end point and authenticates the user against the SPM database of users. For more information about JAAS login, see Authentication Architecture.

User Account Types

The universal access client supports three different user account types, Public, Generated, and Citizen. For more on user accounts and security see User Accounts. If you want to customize the log in and sign up process provided by the universal access starter pack, the Authentication module provides log in functions to support each of these three user account types.

```
Authentication.login

Authentication.loginAsPublicCitizen

Authentication.loginWithGeneratedUser
```

Tracking the logged in user

The universal access client application uses 'session storage' in the browser to store some basic details of the currently logged-in user after they are authenticated with the server. This session storage is typically used to inform the client application what views it should present, for example if no user is logged in, then the login and signup page buttons are presented on the home page.

The Authentication module provides functions that query who the current logged in user is and their account details, according to the session storage in the browser.

```
Authentication.getLoggedInUser

Authentication.getUserAccount
```

Logged in - Client vs Server

It is possible for the user to seem logged in on the client when they are not logged in on the server. This does not compromise the security of the application. The SPM server APIs use session tokens that are stored in cookies to determine whether the current user is authenticated. The cookies are transmitted with each API call, and only a valid token results in a successful response.

For example, if a user's session times out on the server, the next API request to the server results in a 401 unauthorized response, even if the user seems to be logged in to the client application. This behavior ensures that no matter what the client application says about the currently logged-in user, the server responds only to valid session tokens.

Developing with Redux

Redux is used as a client-side store to store data that is retrieved by IBM Cúram Social Program Management APIs and data that is used to present a consistent user experience.

What is Redux?

Redux is a client-side store that provides a mechanism for holding data in the browser.

- The store is typically used to manage state in the client application. State can include the following types of data:
  - System data that is returned from an API request.
  - User input data that is collected before it is posted to APIs.
  - Application data that is not sent from or to the server, but is created and maintained to control how the application works. For example, transient user selections like hiding or showing a side pane.
- Redux uses a unidirectional architecture, which simplifies the process of managing state.
- Redux can be used as a caching mechanism to avoid unnecessary network round-trips, although consider this usage carefully to ensure the data that is presented is always current.
- Redux proves to be beneficial as your application grows and becomes more complex. By centralizing state management and offering tools that give a holistic view of the application state, development can scale more easily.

Note: This topic assumes that you are familiar with Redux and using Redux with React components. If you are not familiar with these technologies and how they work together, you should complete tutorials from the official sources for these technologies.

How is Redux used in Universal Access

IBM Cúram Universal Access uses Redux to store the data that is retrieved by the IBM Cúram Social Program Management APIs.

Each GET API used by Universal Access has an associated ‘store slice’ where the response of the API is stored. React components can monitor the store for updates relevant to them and automatically update as data changes. The store is also used for collecting user input, such as user information that is requested while users sign up. This data can then be retrieved from the store and posted to the IBM Cúram Social Program Management server.

Other parts of the store are not tied to IBM Cúram Social Program Management APIs, and track data that is used to present a consistent user experience.

Creating a Redux store

By default, the Universal Access starter pack is configured to use a Redux store. This configuration is needed to allow it to use the `universal-access` and `universal-access-ui` packages. The store configuration is initiated from the `src/redux/ReduxInit.js` file in the starter pack.

```
...

import configureStore from './store';

...

// ==================================
// 1. Create the store and initialize the universal-access module.
// ==================================

// Create a Redux store
// This is optional, if you don't want to create your own Redux store you can remove this,
const appStore = configureStore();

// Configure the UA package
// 1. If you are using your own store, you must share it with UA
UAReduxStore.configureStore(appStore);

...
```

For more information on Redux, see https://redux.js.org/.

Configuring the store

Configure the store in the `src/redux/store.js` file, which exports the `configureStore` function that can be called to create a new Redux store. The configure store function can be modified to:

- Add Redux 'middleware'.
- Provide a custom set of reducers.

  Note: To work with the universal-access packages, the store must use the reducers that are exported from the `universal-access` package.

Adding reducers

If you decide to use Redux with your custom React components, you must create custom reducers and add them to the store. All Universal Access reducers are prefixed with **UA**, for example `UAPaymentsReducer`. Do not use the UA prefix in custom reducers to avoid overriding `universal-access` reducers. Overriding reducers is not supported, see “Developing compliantly” on page 26.

The `src/redux/rootReducer.js` file defines the set of reducers for the store, and combines them into a single *root reducer* that can be passed to the `configureStore` function in the `src/redux/store.js` file.

For convenience, the file defines an `AppReducers` object where you can add custom reducers. The custom reducers that are defined in the `AppReducers` object are combined with the `UAReducers` imported from the `universal-access` package, and the superset of reducers is returned.

The following code excerpt shows the `rootReducer` function that returns the combination of Universal Access reducers and custom reducers.

```
const AppReducers = {
  // Add custom reducers here...
  // customReducer: (state, action) => state,
};

/**
 * Combines the App reducers with those provided by the universal-access package
 */
const appReducer = combineReducers({
  ...AppReducers,
```

```
  ...UAReduxReducers,
});



/**
 * Returns the rootReducer for the Redux store.
 * @param {*} state
 * @param {*} action
 */
const rootReducer = (state, action = { type: 'unknown' }) => {
  ...
  return appReducer(state, action);
};
```

Clearing Redux store data

The Redux store is a JavaScript object that is stored in the global object for the browser window. The content of the store is visible through inspection, either programmatically or by browser plug-in tools, such as the developer tools. It is critical that the store is cleared for the current user when they log out to ensure that no sensitive user data is left on the device for malicious actors. The log out feature that is provided by the starter pack triggers a Redux action that clears the store.

Developing with universal-access modules

Universal Access modules provide a connection between React components and the Redux store. This design allows the React components to focus on presentation and reduces the complexity of the code in the presentation layer. The modules also manage the communication between the client application and the IBM Cúram Social Program Management APIs, including authentication, locale management, asynchronous communication, error handling, Redux store management and more.

Modules and APIs

Each universal-access module is responsible for handling the communication between a single API. For example, the Payments module is responsible for communicating with the `/v1/ua/payments` API. For more information about IBM Cúram Social Program Management APIs, see *Connecting to a Cúram REST API*.

Blackbox

Modules are blackbox so are not open to customization or extension. The modules expose actions and selectors to interact with the module. The actions and selectors are APIs that are documented in the `<your-project-root>/node_modules/@spm/universal-access/docs/index.html` file.

Actions

Module actions are used to modify the Redux store, like inserting, modifying, or deleting data from the store. For example, the `PaymentsActions` action modifies the payments slice of the store.

Some actions include calls to APIs. For example, `PaymentsActions.getData` action calls the `v1/ua/payments` API and dispatches the result to the payments slice of the store, or sets an error if the API call fails.

Selectors

Module selectors are used to query the Redux store. They provide the response to predefined store queries. For example, the `PaymentsSelector.selectData` selector returns the `/payments/data`

slice from the store, and the `PaymentsSelector.selectError` selector returns the value of the `/payments/error` slice of the store.

Reusing Universal Access modules in your custom components

You can use the actions and selectors from the universal-access package to connect your custom components to existing IBM Cúram Social Program Management APIs and the Redux store. You can use the `react-redux` module to connect your components. Examples of this technique can be found in the `universal-access-ui` features.

For example, the following code is from the `PaymentsContainer` file in the Payments feature. The code shows how the actions and selectors from the Payments module are connected to the properties of the Payments component.

This pattern is documented extensively in the official Redux documentation.

```
import { connect } from 'react-redux';
import React, { Component } from 'react';

...

/**
 * Retrieves data from the Redux store.
 *
 * @param state  the redux store state
 * @memberof PaymentsContainer
 */
const mapStateToProps = state => ({
  payments: PaymentsSelectors.selectData(state),
  isFetchingPayments: PaymentsSelectors.isProcessing(state),
  paymentsError: PaymentsSelectors.selectError(state),
});
/**
 * Retrieve data from related rest APIs and updates the Redux store.
 *
 * @export
 * @param {*} dispatch the dispatch function
 * @returns {Object} the mappings.
 * @memberof PaymentsContainer
 */
export const mapDispatchToProps = dispatch => ({
  loadPayments: () => PaymentsActions.getData(dispatch),
  resetError: () => PaymentsActions.resetError(dispatch),
});
/**
 * PaymentsContainer initiates the rendering the payments list.
 * This component holds the user's payment details list.
 * @export
 * @namespace
 * @memberof PaymentsContainer
 */
export default connect(
  mapStateToProps,
  mapDispatchToProps
)(PaymentsContainer);
```

Related information

Connecting to a Cúram REST API

Developing with headers and footers

IBM Cúram Universal Access contains a predefined header and footer. The header and footer contain content that is found in the header and footer of an application, such as links, **log in**, and **sign up** buttons, and menus for logged in users.

Headers and footers

You can customize your application headers and footers by replacing the sample components with your own custom versions.

The App.js file in the *universal-access-sample-app* module, reuses the sample *ApplicationHeader* and *ApplicationFooter* components that are provided by the universal-access module by placing them above and below the main content of the application:

App.js

```
    <HashRouter>
      <ScrollToTop>
        <div className="app">
          <a className="wds-c-skipnav" href="#main-content">
            {formatMessage(translations.appSkipLink)}
          </a>

          <Route path="/" component={ApplicationHeader} />
          <main id="main-content" className="main-content">
            <Content>{routes}</Content>
          </main>

          <ApplicationFooter />
        </div>
      </ScrollToTop>
    </HashRouter>
```

Header

Typically, an application header has two views. One view has items relevant to users who are not logged in or signed up, for example a **Sign Up** button. The second view shows items that are relevant to users who are signed up and logged in, for example an **Update your profile** button.

To facilitate the separate views, use a react-router-dom *Route* component. The App.js sample demonstrates wrapping the *ApplicationHeader* component in a Route component and passing Route information to the *ApplicationHeader*. This allows the *ApplicationHeader* to query the Route properties and decide what to display based on the current location in the application. For example, you might want to show a different view for the login page route (‘*my-app-domain*/#/login’) from the application home page route (‘*my-app-domain*/#/’).

The following code sample shows how the *ApplicationHeader* queries its location property to find out what page the application is displaying. The sample code then uses this information to decide what to show in the header.

```
 get isOnLoginPage() {
  return this.props.location.pathname === '/login';
}

render() {
 return (
  <Header
    title={this.pageTitle}
    type="scrollable"
    logo={<img src={logo}
    alt="agency"
    id={this.props.loggedInUser} />}>
    <PrimaryNavigation type="scrollable">
      <TabList scrollable>
        <Tab
         id="tab1"
         href="#/"
         text={
           this.props.intl.formatMessage(translations.headerHomeLabel)}/>
           <Tab
             id="tab2"
             href="#/my-applications"
             text={this.props.intl.formatMessage(
                translations.headerBenefitsLabel)}/>
        </TabList>
      </PrimaryNavigation>
      <SecondaryNavigation type="Scrollable"/>

       {/* Show signed out menu */}
```

```
          {!this.isOnLoginPage &&
           this.props.loggedInUser === null &&
           !this.isUserProfileLoaded &&
           this.signInMenu}

          {/* Show signed in menu */}
          {this.props.loggedInUser &&
           this.isUserProfileLoaded &&
           this.profileMenu}
        </SecondaryNavigation>
      </Header>
    );
  }
```

Login and sign up in the header

If you are building your own customer header, you must identify which page you are currently displaying the Header on, you must also differentiate between logged in and logged out users. Whether a user is logged in or out can be determined by using the authentication API provided by the universal-access module. The Authentication API provides functions to allow you to log in and out of the application, and also allows you to query if a user is logged in and who that user is. For more information, see the Authentication API documentation.

The following code sample shows how the *ApplicationHeader* uses the Authentication API. In this function, a check is made to see whether a user is logged in before it loads that user's profile. The user's profile is needed to display the user's full name in the header.

```
fetchProfile() {
  if (Authentication.isLoggedIn() && !this.isUserProfileLoaded) {
     this.props.loadProfile();
  }
}
```

Footer

You can add a footer to the bottom of the application page in the same way as you add the header to the top of the page. The universal-access module provides a sample application footer that is used in the universal-access-sample-app, see the `App.js` sample. The sample footer is static and does not change based on the location or the authentication state, however the footer can be made dynamic by following the example from the header.

Related tasks

Changing the application header or footer

Build on the styling scenario from *Using the Web Design System to style content* by adding a link to the application header or footer. For more information about the application header and footer, see *Developing with headers and footers*.

Providing the application in another language

IBM Cúram Universal Access is globalized, that is it can be translated into different languages. Universal Access also supports regionalization of currencies, calendar and date formats as defined by IBM Cúram Social Program Management on which the application depends, for more information, see *Developing for Regional Support*.

Related information

Developing for Regional Support

Selecting a language

Citizens can select a preferred language from the **language** drop-down in the footer of the application. When citizens select a preferred language, the application is displayed in that language. The application retains the preferred language setting based on a cached value in the browser.

Note: The **language** drop-down only appears when more than one language is configured for the application.

Note: A citizen's language preference is not saved if the browser is configured to block access to its local storage, the application reverts to the default language (English) when the page is reloaded.

Configuring the languages provided by the application

Add languages to the application or change the default language.

About this task

The application can provide a number of languages in the user interface. You can customize the application by adding languages or changing the default language.

Procedure

1. Create a `src/config/intl.config.js` file.

   Note: This file is read by the `src/intl/IntlInit.js` component, which handles storage of the configuration and creates the *react-intl IntlProvider*.
2. Review the following example `src/config/intl.config.js`:

```
export default {
  defaultLocale: "en",
  locales: [
    {
      locale: "en",
      displayName: "English",
      localeData: require("react-intl/locale-data/en")
      messages: require("../locale/messages_en”)
    },
    {
      locale: "de",
      displayName: "German",
      localeData: require("react-intl/locale-data/de"),
      messages: require("../locale/messages_de")
    },
    {
      locale: "ar",
      displayName: "Arabic",
      direction: "rtl",
      localeData: require("react-intl/locale-data/ar"),
      messages: require("../locale/messages_ar")
    },
    {
```

```
      locale: "ht",
      displayName: "Haitian",
      /*
```

3. If the locale you need to support is not in the *react-intl* locale data you can create your own locale data by creating an object with the locale property. You must also include at a minimum the *pluralRuleFunction*. For more information, see https://github.com/yahoo/react-intl/issues/1050. The following example includes a *pluralRuleFunction* for Haitian.

```
      */
      localeData: {
        locale: "ht",
        pluralRuleFunction(arg1, arg2) {
          return arg1 && arg2 === 1 ? "one" : "other";
        }
      },
      messages: require("../locale/messages_ht")
    }
  ]
};
```

Note: A `src/config/intl.config.js.sample.md` is provided which details the `intl.config.js` object schema.

Translating your application
Use *react-intl* and *babel-plugin-react-intl* to extract text from your application. You can then translate the text into another language and include that translation in the application.

Extracting translatable content
During development, IBM used *react-intl* (https://github.com/yahoo/react-intl) and *babel-plugin-react-intl* (https://github.com/yahoo/babel-plugin-react-intl) to globalize IBM Cúram Universal Access.

About this task

Follow the same method as used by IBM during development to extract the translatable content from your application.

Note: *react-intl* provides react components and an API to format dates, numbers, and strings, including pluralization, and handling translations. *babel-plugin-react-intl* extracts string messages from React components that use *react-intl*.

Procedure

1. Use the *react-intl* `defineMessages` API to define the default message string entry within the application.
2. Add *babel-plugin-react-intl* and its dependencies *babel-cli* and *babel-preset-react-app* to the application's *devDependencies*.
3. Add a `.babelrc` file in the root of your project. Use `.babelrc` to configure the settings for the *babel-plugin-react-intl*. The following is an example `.babelrc` file:

```
{
  "presets": ["react-app"],
  "plugins": [
```

```
    [
      "react-intl", {
        "messagesDir": "translations/messages",
      }
    ]
  ]
}
```

4. Add the following line to your `package.json` "scripts":

   UNIX:

   ```
   “extractTranslations”: "NODE_ENV=production babel ./src >/dev/null"
   ```

   Windows:

   ```
   "extractTranslations": "set NODE_ENV=production&&babel ./src > NUL"
   ```

5. Run the extraction command: `npm run extractTranslations`.

Results

This procedure extracts all translations to the `translations/messages` directory as specified in the `.babelrc` configuration.

The content of `translations/messages` along with the JSON content under the locale directories of the *@spm/universal-access-ui* and *@spm/intelligent-evidence-gathering* directory form what should be sent for translation.

What to do next
For more information, see *Including translated content in your application*.

Including translated content in your application
IBM Cúram Universal Access exposes a *src/intl/IntlInit* component. This component reads the configuration provided in the custom src/config/intl.config.js to seed your application with messages for all the languages you want your application to support.

About this task

Procedure

1. Translations must be returned for use in your product in the format of a single JSON file per locale. This JSON file should be in the format expected by *react-intl*, which is `{[id: string]: string}`, as shown in the following example:

   ```
   {
   “label1”: “Translated text1“,
   “label2”: "Translated text2”,
   }
   ```

   Where *id* is the id that is used in your *defineMessages* entry and subsequent extracted message id.

   Note: The id in this file format *{[id: string]: string}* must match the id that you define in your code as in the *defineMessages* structure. For more information, see https://github.com/yahoo/react-intl/wiki/API#definemessages.

   This single file and its location within the application forms the entry to the messages value with the `intl.config.js` for your configured locale, for example:

   ```
   {
         locale: "de",
   ```

```
      displayName: "German",
      localeData: require("react-intl/locale-data/de"),
      messages: require("../locale/messages_de")
},
```

2. *react-intl* also requires that its own locale configuration (localeData) is provided to support some of its internal functions. For more information, see https://github.com/yahoo/react-intl/wiki#loading-locale-data.

Results

When you have configured it correctly with the `src/config/intl.config.js` file, the *ApplicationFooter* language selection drop-down should expose your new locale selection, it should also load and apply the configured translation messages to the application.

Note: If your application does not find messages for the currently selected language at run time, *react-intl* defaults to the text of the *defaultMessage* entry that was used when the message was defined in the source code.

Regionalizing your application

User interface elements, such as date formats and currency symbols are defined in IBM Cúram Social Program Management, for more information, see *Developing for Regional Support*.

The universal-access module and its components respect the regional settings as defined by the IBM Cúram Social Program Management to ensure your application is synchronized with the configuration of the IBM Cúram Social Program Management instance on which it depends.

Related information

Developing for Regional Support

Customization Scenarios

Customize the IBM Cúram Universal Access web application.

The first scenario shows how to change default text on the **My Details** page. Each subsequent scenario adds to the previous one to build out new content in your Universal Access project.

Note: Follow the scenarios in sequence. If you start in the middle of the scenario list, you might have to go back through previous scenarios.

Changing the application text

You can change the default text in the application by providing custom text that overrides the default text for any language. In this scenario, an English language message is changed.

About this task

Each message or text string that citizens see in the app is provide using the *react-intl* package that supports the globalization of React apps. *react-intl* allows the messages to be extracted and translated to other supported languages, it also adds placeholders for data, for example.

To change the existing text of any of the languages that are provided by IBM, you must provide a custom version of the message that is mapped to the same *message id*.

Procedure

1. Find the ID of the message you want to replace. All product messages are defined in the *universal-access-ui* package. In your project, go to `/node_modules/@spm/universal-access-ui/locale`.
   a) The `locale` folder contains message files for each supported locale. For your chosen language, search the appropriate `message_xx.json` for the text string that you want to replace. For example, to change the English text **Apply for a benefit**, search `messages_en.json` for that string

as shown in the following example. If there is more than one instance of the string, you must find the correct message ID for the text you want to change. The simplest way to find the correct instance is to try replacing each ID one by one, reloading the page each tine to see if the new string is displayed.

```
"System_Messages_Alert_Description": "System messages alert description",

"Payments_NoPaymentMessages": "No payment messages",

"Payments_ApplyForABenefitLink": " Apply for a benefit ",

"TODO_NoTODOMessages": "No to-dos",

"TODO_CaseworkerMessage": "Your caseworkers can create to-dos for you.",

"Meetings_NoMessages": "No meetings",
```

b) For the **Apply for a benefit** string, use the associated ID *"Payments_ApplyForABenefitLink"* to override the message in your custom `messages_en.json`.

2. Create a custom message file by creating a `messages_en.json` file in the `src/locale` folder. Custom messages are injected into the application at application start. For more information, see *Localizing the application*. To help you get started, the starter pack provides a locale folder from where custom messages files are automatically loaded. Assuming this process has not been customized for your project, you can add your custom file to this folder: `src/locale`.
3. To replace the message, create a new *id:message* mapping in your custom message file by using the same ID value as shown in the following example.

```
"Payments_ApplyForABenefitLink": " Click here to apply for a benefit ",
```

Related concepts

Providing the application in another language

IBM Cúram Universal Access is globalized, that is it can be translated into different languages. Universal Access also supports regionalization of currencies, calendar and date formats as defined by IBM Cúram Social Program Management on which the application depends, for more information, see *Developing for Regional Support*.

Adding content to the application

Build on the text change scenario from *Changing application text* to add a new route. You also add content that is displayed when the route is loaded.

Before you begin

If you are not familiar with React and React Router, you must take a basic course in building a web application with React and React Router.

The term "feature" refers to the content that is displayed when a route is loaded, this content is what citizens see on the user interface. A feature is an abstraction that includes all the content that comes together to create the end user experience. A feature could be a collection of JavaScript files, json files, and APIs that work together to generate the end user experience. The term "feature" could be referred to as a page, view, or component in other application environments.

This scenario adds a new feature that presents a logged-in person's details in the main content area when a `/person` URL is loaded. This scenario is built on in later scenarios by calling APIs, using client side stores, error handling, or globalization.

About this task

When you extend the IBM Cúram Universal Access reference application you might want to introduce new content that is displayed when citizens click a link.

Procedure

1. Create the content for the feature, take the following steps:
   a) Create a folder called `features` under the `/src` folder in your project
   b) Add a subfolder called `person`, and add a file called `PersonComponent.js` to that folder as shown in the following example.

```
src/features/Person/PersonComponent.js
```

   c) Add some HTML to display when the component is loaded. The following example displays some data that is returned by an API:

```
import React from 'react';

const Person = () => { return (
    <div>
      <h1>James Smith</h1>
      <h2>Gender: Male</h2>
      <h2>Born: April 1st 1996</h2>
    </div>
)};
export default Person;
```

2. Add a route to link to your feature, take the following steps:
   a) Declare an associated URI for each feature in the application. The URI allows React to present the feature when the URI is requested in the browser. This technique is standard *'React Routing'* for displaying features. For more information on routes in the Universal Access client, see *Customizing with routes*. Add a simple component that displays when the route is loaded:
      1) Open `routes.js` in your project.
      2) Import a *Person* component from the folder `features/person` which you create in the next step.
      3) Add a new route "`/person`" that loads the *Person* component as shown in the following example:

```
import React from 'react';
import { Route, Switch } from 'react-router-dom';
import { Routes as UARoutes } from '@spm/universal-access-ui';
 import Person from './features/PersonComponent'

export default (
  <Switch>
    <Route path="/person" component={Person} />
    <UARoutes />
  </Switch>
);
```

3. Load the new feature by using the route, take the following steps:
   a) Run your application, enter the following command:

```
npm run start
```

   b) Start a browser and enter the full URL for the feature, for example: http://localhost:8888/#/person

Results

When the application loads, the person details are displayed in the main content area.

Related concepts

Developing with routes

Routes define the valid endpoints for navigation in your application. Your application consists of a network of routes that can be traversed by your users to access the application's pages.

Using the Web Design System to style content

Build on the route and person content scenario that you added in *Adding content to the application* by styling the content of a person's details.

Before you begin

The Web Design System is a design framework that enables developers to build a cohesive and consistent application. By selecting components from a design catalog and applying design principles, design and development is faster and user experience is improved.

About this task

The full catalog of Web Design System components, including descriptions of when and where to use them, is documented in the *govhhs-design-system-react* package. You can access these packages through `index.html` file in `/node_modules/@govhhs/govhhs-design-system-react/docs`. This scenario uses a number of Web Design System components to improve the person feature.

Procedure

1. Import contents from the Web Design System. Enter the following command to import the *Avatar* and *MediaObject* components from the package `@govhhs/govhhs-design-system-react`:

```
import {Avatar, MediaObject} from '@govhhs/govhhs-design-system-react'
```

2. Update `PersonComponent.js` to use the Grid, Column, Card, MediaObject, Avatar, and List components to display the person's details. You can also include an address in a separate card.

   Use the following code to replace the previous `PersonComponent.js`:

```
import React from 'react';
import {Grid, Column, Card,CardBody,CardHeader, List, ListItem, Avatar, MediaObject } from
'@govhhs/govhhs-design-system-react'

const avatarMediaJames = <Avatar initials="JS" size="medium" tooltip="profile photo" />;
const Person = () => {
    return (
        <Grid className="wds-u-p--medium">
            <Column width="1/2">
             <Card>
                <MediaObject media={avatarMediaJames} title="James Smith">
                <List>
                    <ListItem>Gender: Male</ListItem>
                    <ListItem>Born: April 1st 1996</ListItem>
                </List>
            </MediaObject>
            </Card>
            </Column>
            <Column width="1/2">
             <Card title="Address">
                 <CardHeader title="Address"/>
                 <CardBody>
                    <List>
                        <ListItem>1074, Park Terrace</ListItem>
                        <ListItem>Fairfield</ListItem>
                        <ListItem>Midway</ListItem>
                        <ListItem>Utah 12345</ListItem>
                    </List>
                </CardBody>
            </Card>
            </Column>
        </Grid>
)};
export default Person;
```

3. Save `PersonComponent.js`.

Results

Reload the application, the application should show the updated styling.

Changing the application header or footer
Build on the styling scenario from *Using the Web Design System to style content* by adding a link to the application header or footer. For more information about the application header and footer, see *Developing with headers and footers*.

Before you begin

To customize the header, you must create your own custom version. To keep this scenario brief, work on the header only and copy the existing header from *universal-access-ui*. Make some small changes to the header to show how it can be customized. Alternatively, completely replace the header or footer with your own version.

About this task

Change the application header to include a new link that to take you to the **My Details** page.

Procedure

1. Copy the Universal Access header by copying the `node_modules/@spm/universal-access-ui/src/features/ApplicationHeader` folder to `src/features`.
2. Fix any broken imports. Take the following steps:
   a) Use ESLint or a similar linting tool to find any errors where imports are not found.

      Note: If you do not use a linting tool, you get build errors.

   b) Errors are generated because the *universal-access-ui* uses relative paths when it imports dependencies from its own project. For imports that are within the *universal-access-ui module*, but outside the `features/ApplicationHeader` folder, you must change the imports to reference the official exported version of those dependencies from the *universal-access-ui* node module.

   c) For each import that is not resolved, find the equivalent export in the *universal-access-ui* package. Inspect `node_modules/@spm/universal-access-ui/src/index.js` to find the list of exported artifacts and their exported names.

      The *Paths* module is referenced in the *ApplicationHeader* by using the default import from a relative path as shown in the following example: `import PATHS from '../../router/Paths'` Amend module as shown in the following example: `import { Paths } from 'universal-access-ui'`

   d) Repeat this procedure for all the files in the `ApplicationHeader` folder, some of the imports of *'Paths'*, and for some other references such as *'ErrorBoundary'* and *'AppSpinner'*.
3. Replace the existing header with your custom version, take the following steps:
   a) Open `src/App.js file` and remove the imported *ApplicationHeader* from *universal-access-ui*.
   b) Import your cloned version from `./features/ApplicationHeader` as shown in the following example:

      ```
      import ApplicationHeader from './features/ApplicationHeader';
      ```

      Import *ApplicationHeader* as a default import, without curly brackets, rather than a named import. Alternatively, you can add a named export to your *ApplicationHeader* feature.
4. Update the header feature to include a tab that loads the */person* page take the following steps:
   a) Open `constants.js` in `src/features/ApplicationHeader/components`. `constants.js` defines an object that represents a navigation item for the header.
   b) Add and entry for the new page **My Details** as shown in the following example:

```
/**
 * Application navigation header tabs.
```

```
 */
const NAVIGATION_HEADER_TABS = {
    ...

  PROFILE: { NAME: 'PROFILE', ID: 'navigation-profile' },
  CHANGE_PASSWORD: { NAME: 'CHANGE_PASSWORD', ID: 'navigation-change-password' },
   MYDETAILS: { NAME: 'MYDETAILS', ID: 'my-details' } ,

};
```

c) Open `ApplicationHeaderLogic.js`.. `ApplicationHeaderLogic.js.` contains the logic that tracks which tabs are selected so they can be highlighted as active.

d) Update the *isTabActiveForUrlPathname* function to include the new **My Details** page in the **Your Account** section. For brevity, the value is hardcoded in the following example. However you can replicate the pattern that is used by the *universal-access* code to add it to *Paths*.

```
const isTabActiveForUrlPathname = (urlPathname, navigationTabName) => {

  ...
  switch (navigationTabName) {
    case FIND_HELP.NAME:
      return (
        urlPathname === Paths.HOME ||
        urlPathname === Paths.APPLY ||
        urlPathname === Paths.BENEFIT_SELECTION ||
        urlPathname === Paths.APPLICATION_OVERVIEW
      );
    case YOUR_ACCOUNT.NAME:
      return (
        urlPathname === Paths.ACCOUNT ||
        urlPathname === Paths.BENEFITS ||
        urlPathname === Paths.PAYMENTS.ROOT ||
        urlPathname === Paths.PAYMENTS.DETAILS ||
         urlPathname === '/person'
      );
```

Open `ApplicationHeaderComponent.js` and find the Web Design System *PrimaryNavigation* component. `ApplicationHeaderComponent.js` renders the header.

e) Add a tab called **'My Details'** with a link to the person feature inside `ApplicationHeaderComponent.js`. For brevity, the example is hardcoded values, but you can replace these values with variables. If you want, you can also localize the tab.

```
..

<PrimaryNavigation>
  <Tabs>
    ...

    <Tab
      id={NAVIGATION_HEADER_TABS.YOUR_BENEFITS.ID}
      href={HASH_SYMBOL + LOCATIONS.BENEFITS}
      label={formatMessage(translations.headerYourBenefitsLabel)}
    />
     <Tab
      id="person_tab"
      href="#/person"
      label="My Details"
    />
  </Tabs>
  ...
</PrimaryNavigation>

...
```

5. Save your file and restart the application.
6. You can modify the application footer in the same way by replacing the *universal-access-ui* version in `src/App.js` with your own custom version.

Results

Navigate to the home page. Note the new tab called **My Details** in the primary navigation area. When you select **My Details**, the person feature is loaded in the main content area.

Related reference

Developing with headers and footers
IBM Cúram Universal Access contains a predefined header and footer. The header and footer contain content that is found in the header and footer of an application, such as links, **log in**, and **sign up** buttons, and menus for logged in users.

Creating an IBM Cúram Social Program Management REST API

Build on the scenario from *Changing the application header or footer*, use a REST API to get data to your application.

About this task

The most common way to get data to your application is to use a web API to receive the requested data as a JSON string that your application then parses and renders. IBM Cúram Social Program Management provides development tools and the runtime infrastructure that you can use to build and deploy an API with your IBM Cúram Social Program Management server. The API can be called using the standard HTTP verbs such as GET, POST, and DELETE. The API returns data as a JSON string in the response body. For more information, see Developing Cúram REST APIs.

Related information

Developing Cúram REST APIs

Connecting to REST APIs from the application

Build on the IBM Cúram Social Program Management REST API that you created in the scenario *Creating an IBM Cúram Social Program Management REST API* by calling it from your application.

About this task

Features in your application rely on passing data to and from the IBM Cúram Social Program Management server or another service. The reference application already consumes a number of Universal Access APIs to support business features.

This scenario updates the person feature to read the data from an API instead of just displaying hardcoded values. The scenario shows you how to create and use the following items:

- Use the *RESTService* module to helps you call APIs.
- Use the mock server to show you how to create a mock API that allows you to rapidly develop your feature without spending time building and deploying the real APIs that it eventually uses.
- Connect your application to a IBM Cúram Social Program Management development environment that hosts the APIs by using Tomcat to enable real integration testing in the development environment.

Procedure

1. Create a mock API, take the following steps:
   a) In your project, open `/mock/apis/mockAPIs.js`.

   The mock server consumes mockAPIs.js, it contains the mappings from APIs to the mock data. The mock server uses this information to provide the correct data when an API call is made in development mode. `mockAPIs.js` also contains an import from the *universal-access-ui* package and assignments for GET, POST and DELETE APIs as shown in the following example:

```
const mockAPIs = require('@spm/universal-access-mocks');

// Extract the existing universal access GET,POST and DELETE mocks for merging.
const UAMockAPIsGET = mockAPIs.GET;
```

```
const UAMockAPIsPOST = mockAPIs.POST;
const UAMockAPIsDELETE = mockAPIs.DELETE;
```

Use these APIs to test the Universal Access application. For more information, see *Working with the mock server*.

b) To add more mock data, add your mocks to the placeholders provided. This scenario adds the person data for a person 'James Smith' that is returned when the `'/person'` path is loaded.

c) Add an object in `mockAPIs.js` to represent James Smith. For simplicity, do not normalize the dates, or use code tables, later scenarios show you how to globalize and handle code tables.

```
const user = {
 firstname: 'James',
 surname: 'Smith',
 dob: 'April 1st 1996',
 gender: 'male',
 address: {
   addr1:'1074, Park Terrace',
   addr2:'Fairfield',
   addr3:'Midway',
   addr4:'Utah 12345',
 }
}
```

d) Include a value for the URI *'/user'* in the *mockAPIsGET* object to return the mock object as shown in the following example:

```
const mockAPIsGET = {
  '/user': user,
}
```

The new *'/user'* mock API is merged with the mocks from *universal-access-ui* and is deployed by the mock server on port 3080.

e) Test that the new API is working, start the application using `npm start`.

f) Using the browser, load the */person* URL: http://localhost:3080/person. If successful, the browser displays the response.

2. Use the RESTService module from the core package to make an AJAX call to the API.

   There are many agents that can be used to achieve this. The RESTService uses Superagent to make the AJAX call. The RESTService handles the following functions:

   - Authentication credentials are automatically handled for each call, and users are redirect to log in when appropriate.
   - The user's locale is passed to ensure the response is in the correct locale.
   - Timeouts are managed using the settings from the `.env` file.
   - Errors are captured and thrown in a standard fashion so that the error handling infrastructure is invoked.

   For more information on the RESTService module, see *Working with the RESTService*.

3. Open `PersonComponent.js` file. Make the following changes, check that your application is still displaying the page after each step:

   a) To enable lifecycle methods that are required to manage the API calls, convert the old stateless component to a stateful *React.Component* class:

   `Old stateless Person component`

```
const Person = () => {
   return (
    <JSX code here>
   );
}
```

Updated stateful Person component

```
class Person extends Component {
    render(){
      return (
        <JSX code here>
     )};
}
```

b) Create local state to hold the API data.

The local state stores the values returned by the API that drive the render function. Whenever the state is updated, the component rerenders to reflect the state change. For this scenario, hardcode the values for the state in your class constructor so that something is displayed on the page. To differentiate between this temporary default data and the API data, change the *firstName* to 'Roger'. Later, when you introduce the API, the data for 'James' is returned from the API and not the default state as shown in the following example:

```
constructor(props) {
  super(props);
    this.state= {
      user : {
        firstName:'Roger',
        surname:'Smith',
        dob:'April 1st 1996',
        gender: 'Male',
        address: {
          addr1:'1074, Park Terrace',
          addr2:'Fairfield',
          addr3:'Midway',
          addr4:'Utah 12345',
        }
     }
  }
}
```

c) Convert all hardcoded references to use the values from the state.

Now that you have a state object, replace all hardcoded values with references to the state. Replace each hardcoded piece of data with a state reference *{this.state.user.X}*. Examples are as follows:

```
...
class Person extends Component {
    render(){
      return (
        ...
            <Card>
               <MediaObject media={avatarMedia} title= {this.state.user.firstName} >
               <List>
                   <ListItem>Gender:  {this.state.user.gender} </ListItem>
                   <ListItem> {this.state.user.gender} </ListItem>
               </List>
           </MediaObject>
    )};
        ...
}
...
```

d) Import the *RESTService* module.

To call an API you must to invoke one of the methods of the *RESTService* module. First you must import it from the core package:`import { RESTService } from '@spm/core'`

e) Create a *componentDidMount* method to invoke the API call.

When your component is mounted by React, the *componentDidMount* function is invoked. In *componentDidMount* the API call can be made to populate the component state. Update your constructor to set the user values to blank when initializing, this setting ensure that your data is being loaded from the API. Then, add the following code to your Person component. The root

location of the API is taken from the values set in your `.env.development` file when in development mode. In production mode, it is taken from the `.env` file.

The `.env.development` file specifies the mock server URL as REACT_APP_API_URL, which has the value http://localhost:3080/ where the mock server is deployed. You can use this environment variable to prepend the */user* API.

The RESTService API accepts a URL and a callback function as parameters. In the following code, the callback function is passed as an anonymous function in the second parameter. Here the 'success' is checked, before the state is updated with the response.

Note: Error scenarios are not handled in this code. The scenario *Handling failures in the application* contains details about failure responses, 'Error Boundaries', and failure handling.

```
componentDidMount() {

    const url = `${process.env.REACT_APP_API_URL}/user`;
    const user = RESTService.get(url, (success, response) => {
      if (success) {
        this.setState((user: response));
      }
    });

  }
```

Results

Start your application, log in and select the **My Details** tab. The tab loads using data pulled from the */user* API. The API that you use in development mode is served from the mock server. In production mode, the 'real' API called using the *REACT_APP_API_URL* defined in the `.env` file. Assuming the contract remains the same between your mock and 'real' APIs, that is, the JSON structure matches in both, you can seamlessly switch between development and production, allowing for a much faster development process.

Related reference

Handling failures in the application
Handle any failures you find when you performed integration testing in the *Developing with IBM Cúram Social Program Management APIs by using Tomcat* scenario.

Testing REST API connections by using Tomcat

Build on the scenario in *Calling an API from the application*. Perform integration testing with the real IBM Cúram Social Program Management APIs instead of using the mock APIs in your Universal Access client.

Before you begin

You must be familiar with the IBM Cúram Social Program Management development environment, the development of REST APIs, and the IBM Cúram Universal Access development environment.

This scenario uses IP address 192.1.1.1 to represent the development computer for the IBM Cúram Social Program Management server, and 192.9.9.9 for the computer that hosts the Universal Access client. Therefore, however, these could be the same computer, and you could use the same IP address. Replace this address with the IP address of your development computer.

About this task

The mock server is hosted on the same domain as the application during development http://localhost. However, when your APIs are served from a different domain, you might encounter Cross Origin Resource Sharing (CORS) issues. You can use Tomcat to configure your Universal Access client and IBM Cúram Universal Access server to allow Cross Origin requests. To overcome the CORS issues, the REST toolkit

uses a filter that provides the required HTTP headers to allow browsers to accept responses from a different domain. In this scenario, the domain is where the REST application is deployed.

Procedure

1. Configure the IBM Cúram Social Program Management server, take the following steps:
   a) In your development environment, add the following properties to *Bootstrap.properties* and set the *hostname/ipaddress* of the computer where the Universal Access client is to be deployed:
      - *curam.rest.refererDomains = 192.9.9.9*
      - *curam.rest.allowedOrigins = 192.9.9.9*

      Note: If you develop the server and client on the same computer, you can use *"localhost"*.

      The property *curam.rest.allowedOrigins* is the *Origin* value in the CORS headers. Both properties can have comma-delimited domain names, for example, `curam.rest.allowedOrigins = 192.9.9.9, 192.9.9.8, mymachine.mycorp.com` to allow multiple domains to access the IBM Cúram Social Program Management application.
   b) Set the *CATALINA_HOME* environment variable to the location of your Tomcat installation. For example, on Windows set the following variable: `'set CATALINA_HOME=C:\DevEnv \7.0.1\tomcat'`
   c) Build IBM Cúram Social Program Management by using the appbuild server, database, client, and other components.
   d) Run an extra target `appbuild rest` to create the REST project in your `EJBServer\build \RestProject\devApp` directory.
   e) Copy `Rest.xml` into your Tomcat `conf/localhost` folder. For more information about building Cúram APIs, see *Developing Cúram REST APIs*.
   f) Start the server, RMILoginClient, and Tomcat in the normal way for IBM Cúram Social Program Management.

      The REST client starts automatically. When the client is running, the APIs are accessible in the `/Rest` base path, for example: `http://192.1.1.1:9080/Rest/<myapi>`.
2. Configure the Universal Access client, take the following steps:
   a) Modify the `.env.development` file that is located in *universal-access-starter-pack* to point to the REST URL on `Eclipse/Tomcat` as shown in the following example:

```
REACT_APP_REST_URL=http://192.1.1.1:9080/Rest
REACT_APP_BASE_URL=http://192.1.1.1:9080/Rest/v1
REACT_APP_API_URL=http://192.1.1.1:9080/Rest/v1/ua
```

      Note: If you develop the server and client on the same computer, you can use *"localhost"*.

      If you do want to connect to an application on WebSphere®, you must change *"http"* to *"https"* and update to the correct port. 9044 is the default port.
   b) Build the application, enter the following command: `npm run build`.
   c) Start the application, enter the following command: `npm run start`.

Results

Your Universal Access client application now communicates with the REST deployed on Eclipse with Tomcat.

Note: Run the application on in debug mode to allow it to stop at the breakpoints in the application code.

Related information

Developing Cúram REST APIs

Handling failures in the application

Handle any failures you find when you performed integration testing in the *Developing with IBM Cúram Social Program Management APIs by using Tomcat* scenario.

Before you begin

You should build fault-tolerant web applications because, for example, web services such as a REST API are never fully reliable. When handling the expected response, the application must also allow for failures, such as network outages, timed out responses, internal server errors, or software bugs.

Universal Access ErrorBoundary component

According to React, "Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed."

An error boundary component is a React component that implements the *componentDidCatch* lifecycle method. For more information about error boundaries, see https://reactjs.org/

The *universal-access-ui* package exports a reusable ErrorBoundary component. The component has a default behavior to handle error scenarios by replacing the failing component with a generic message.

Note: Authentication errors have a specific handler in the ErrorBoundary component. If the error object that is received by the componentDidCatch method contains a status attribute with a value of '401' (Unauthorized error), then the client forces a log-out in the client application. Citizens are automatically redirected to the **Log in** page, so they can re validate and return to the page they were previously on. This situation typically happens if the session times out or has been invalidated on the server. The source code for the ErrorBoundary component is available in the *universal-access-ui* package.

This scenario shows API error handling in the **My Details** page where the API call fails. This scenario also shows how to use the Universal Access ErrorBoundary component to provide a better user experience when failures occur.

Error boundaries in the Universal Access application

The Universal Access starter pack contains the following two error boundaries:

- The first wraps the entire application to capture errors that might occur when loading the header or footer.
- The second wraps the main content to capture errors that are raised from components that are loaded in the main content section.

The error boundaries are shown in the following example:

```
/**
 * App component entry point.
 */
const App = () => (
  <HashRouter>
    <ScrollToTop>
      <ErrorBoundary>
        <ApplicationHeader />
        <ErrorBoundary>
          <Main pushFooter className="wds-u-bg--page">
            {routes}
          </Main>
        </ErrorBoundary>
        <ApplicationFooter />
      </ErrorBoundary>
    </ScrollToTop>
  </HashRouter>
);
```

The error boundary on the main section allows the application context to be retained. That is, the header and footer continue to be displayed when the error is raised from the main section. This continuity provides a better user experience.

You can replace these error boundaries with your own error boundaries.

Faking an API error

This API failure scenario uses a 404 response as the error, you trigger this failure by temporarily changing the API call to a non-existent API.

Take the following steps:

1. Open `PersonComponent.js`
2. Update the API to call in the *componentDidMount* method to the non-existent '/user1' as shown in the following example:

```
componentDidMount() {
   const url = `${process.env.REACT_APP_API_URL}/user1`;
   RESTService.get(url, (success, response) => {
     if (success) {
       this.setState({user: response});
   });
 }
```

3. Save your code and wait for the application to reload.

Provided you followed the previous scenarios, when the application reloads it displays the person and address cards but with no details. The values default to be the values that are created in the constructor of the `PersonComponent.js` file. Use the developer tools in your browser to verify the status of the network call that is made for the '/user1' API. You should see that the response status is a 404 indicating that the network call failed.

Catching an API failure

Using the failure scenario *Faking an API error*, you can modify the code to cater for this failure. The API call is asynchronous, and the callback runs outside the context of the Component tree. This execution mode means that the error that thrown in the call-back function is not caught by the *componentDidCatch* method of the *ErrorBoundary*. Therefore, instead of throwing an error in the callback, you update the state of the component. You can then use the *lifecycle* methods of the React component to react to the updated state when it arrives. Use a state attribute 'apiCallFailed' to hold the response.

In the componentDidMount method, add a branch to the callback passed to the RestService.get method. The failure branch sets the apiCallFailed value to the response value returned by the API as shown in the following example.

```
componentDidMount() {
    const url = `${process.env.REACT_APP_API_URL}/user1`;
    RESTService.get(url, (success, response) => {
      if (success) {
        this.setState({user: response});
      } else {
        this.setState({apiCallFailed: response})
      }
    });
  }
```

When the response is returned it updates the state, and triggers a rerender of the application. You can validate that the state was updated by printing the value in the console from the render method. An example response is as follows:

```
render() {
    console.log(`state -> ${this.state.apiCallFailed}`);
    return (
      ...
    )
 };
```

The render method should print the following error in the console: `state -> Error: cannot GET http://localhost:3080/user1 (404)`

Throwing an error

Now that you have control of the failure, throw an error with an appropriate value for the ErrorBoundary component to catch. As indicated, the API call is asynchronous, so you cannot throw the error from the componentDidMount. The throw could be placed in the render function which will execute when the state updates, but this pollutes the rendering method with code that is not dedicated to rendering. Instead, use the componentDidUpdate lifecycle method. This method is called when the state is updated, which happens when the callback updates the 'apiCallFailed' value.

The error object thrown can be anything that you choose so that the error as useful as possible to the citizen. In this instance, throw the string object that is returned by the response because it describes the issue.

Using a loading mask

Build on the scenarios you have completed up to now. Use a loading mask to indicate that the application is working on rendering a page.

About this task

Response times vary when using REST APIs over a network. In a many cases, the time it takes to receive the response is longer than the time it takes for React to render for the first time. This delay leads to a poor user experience when the page draws the components, but the data is missing.

To avoid poor user experience, use a loading mask to indicates to the user that the application is working on rendering their page.

This scenario uses the *AppSpinner* component from the *universal-access-ui package* to include a loading mask to the **My Details** page to demonstrate how your components can handle slow response times.

API response delay

During development, you must often replicate real world response times for APIs. You can configure the *RestService* to set a delay using the `env.development` file in your environment. By default this value is already set to 2.5 seconds. You should notice this delay when navigating the application in development mode, where you see spinners while components wait for the data to be returned from the mock server by way of the *RestService* module. You can increase or decrease this value to meet your application's needs.

The AppSpinner component

The *universal-access-ui* package includes the *AppSpinner* component, which you can reuse in your project. The *AppSpinner* component wraps the Spinner component from the `govhhs-design-system-react` package and includes a label for accessibility purposes. You can also create your own loading mask in the same manner. You can view the source code for AppSpinner in the *universal-access-ui* package.

Procedure

1. Waiting for the API

The AppSpinner is displayed while the application waits for the API to respond, so you need a mechanism to notify you when the data is, and is not loaded. Use the state to indicate when data is loaded and when it is not. Take the following steps:

a) Open the `PersonComponent.js` file.

b) In the constructor add an attribute called 'loading' to the state, with a value of true.

```
...
 constructor(props) {
   super(props);
   this.state = {
     user: {
       firstName: "",
       surname: "",
       dob: "",
       gender: "",
       address: {
         addr1: "",
         addr2: "",
         addr3: "",
         addr4: ""
       }
     },
     loading: true,
   };
 }
 ...
```

2. Display the loading mask

   Now you have a value that indicates whether the data is loading, take the following steps to display the loading mask based on the value:

   a) Import the AppSpinner loading mask from *universal-access-ui*:

   ```
   import {AppSpinner} from '@spm/universal-access-ui';
   ```

   b) In the render function, add a check that renders the AppSpinner if the loading value is true:

   ```
   render() {
       if (this.state.loading){
         return <AppSpinner/>
       }
       return (
         <Grid className="wds-u-p--medium">
           <Column width="1/2">

           ...

        )

     }
   ```

   When you save and reload the application, you should see the spinner in the main section area. However, the spinner continues to display after the data is returned.

3. Remove the loading mask.

   When the data is returned from the API, remove the mask by updating the state to indicate that loading is finished. Take the following steps:

   a) In the *componentDidMount* function, update the state to set the loading value to false when a successful response is returned as shown in the following example:

   ```
   componentDidMount() {
       const url = `${process.env.REACT_APP_API_URL}/user`;
       RESTService.get(url, (success, response) => {
         this.setState({loading: false})
         if (success) {
           this.setState({user: response});
         } else {
           this.setState({apiCallFailed: response})
   ```

```
      }
    });
  }
```

b) Save and reload the application. Now, when the API response is received, the loading mask is removed and the user's data is displayed.

Customizing an existing feature

The reference application available when you install IBM Cúram Universal Access satisfies a number of general business scenarios such as creating an account, logging in, and applying for benefits, for example. The scenarios are provided both as working software and as examples of how to construct the product. Where small customizations are desired you can reuse the elements of the Universal Access packages to construct your own version of the feature.

About this task

Features

The `universal-access-ui` package is structured by feature. Each feature is typically mapped to a single route. For example, when the `/profile` route is loaded, the Profile feature is displayed. The feature folder is a collection of files that work together to present that feature. Below is an example from the Profile feature.

```
/universal-access-ui

--/src

----/Feature

------/Profile

--------/components

----------/ContactInformationComponent.js

----------/PersonalInformationComponent.js

----------/ProfileComponent.js

----------/ProfileComponentMessages.js

--------/index.js

--------/ProfileContainer.js
```

The feature uses a commonly used pattern to move the data retrieval and management into a 'container component', and the rendering logic into stateless 'presentation components'. This pattern is widely documented and used extensively when working with React and Redux. The pattern is not covered in detail here, just to note that this is how features are structured.

Modifying how a feature is presented

Features can be cloned and modified. You can copy the entire code base for the feature in to your custom project and replace the route that served that feature with your version. You can then modify the code base to include your customizations.

Procedure

1. Find the feature that you want to replace in the `universal-access-ui` package.
   a) Inspect the URL end point that you want to change.
   For example, `/myapp/universal/#/faqs` uses the `faqs` path.
   b) Open the `/node_modules/@spm/universal-access-ui/src/router/Path.js` file to find the variable referencing the route.
   For example, `Paths.FAQS`.

```
const Paths = {
  HOME: '/',
```

```
  ...
  FAQS: '/faqs',
  ...
  SIGNUP: '/signup',
};
export default Paths;
```

c) Open the file `/node_modules/@spm/universal-access-ui/src/router/Path.js` to find the variable referencing the route.
For example, `Paths.FAQS`.

```
...
import FAQ from '../features/FAQ';
...
export default () => (
  <Switch>
    ...
    <Route component={FAQ} exact path={PATHS.FAQS} />
    ...
  </Switch>
);
```

2. Copy the feature folder into your custom app.

   Using the location of the feature folder from step 1, copy the entire folder to your custom app. In the example, the contents of `/node_modules/@spm/universal-access-ui/src/features/Profile` would be copied to `src/features/FAQ`.

3. Replace the route with your custom version.

   a) In your project, open the `src/routes.js` file.

   b) Add a new route at any point before the UARoutes entry to ensure that your path supersedes the same path in UARoutes.

```
import React from 'react';
import { Switch, Route } from 'react-router-dom';
import { Routes as UARoutes } from '@spm/universal-access-ui';
import FAQ from './features/FAQ';

export default (
  <Switch>
    <Route component={FAQ} exact path='/faqs' />
    <UARoutes />
  </Switch>
);
```

4. At this point, you can verify whether your custom version of the feature is being used. Make an obvious change to the feature and reload the application to verify that the change is being picked up and displayed.
5. Customize the feature. Now that you have a exact copy of the feature you can make changes to the code to introduce your customizations.

   Note: You now have full ownership of the feature. On upgrade of the `universal-access-ui` package you will not receive any changes that have been applied to the product version of the feature. Instead, you must manually apply any upgrade features that you wish to retain.

   Note: Most features in the `universal-access-ui` package depend on the modules in the universal-access package to work with the data required by the feature. On upgrade, you must validate that your feature has not been impacted by any changes to modules your feature depends on. See *Working with universal-access modules*.

Deploying your web application to a web server

You can deploy your web application on a web server in a production-like environment as part of your development process. Deployment in a production environment is outside the scope of this documentation, but you can use the instructions in this section for guidance.

Building IBM Cúram Universal Access for deployment

Build Universal Access for deployment on an HTTP server.

About this task

Procedure

1. To quickly configure the `universal-access-starter-pack` application, edit the `.env` configuration file that is located in `universal-access-starter-pack` and modify the following properties to point to the server that hosts the REST services:

   ```
   REACT_APP_REST_URL=<ServerHostName>:9044/Rest
   REACT_APP_BASE_URL=<ServerHostName>:9044/Rest/v1
   REACT_APP_API_URL=<ServerHostName>:9044/Rest/v1/ua
   ```

   Replace the *<ServerHostName>* and the port number in the properties with the host name and port of the server where the REST services are deployed, for example:

   ```
   REACT_APP_REST_URL=https://192.168.1.1:7002/Rest
   ```

2. Enter the following command to install dependent packages:

   ```
   npm install
   ```

3. Enter the following command to generate a build folder within the `universal-access-starter-pack` and build the application:

   ```
   npm run build
   ```

4. Copy and deploy the build folder to either IBM HTTP Server or Oracle HTTP Server. For more information on deploying the built application, see *Deploying your application*.

Related information

Deploying your application

Install and configure IBM HTTP Server with WebSphere Application Server

Install and configure IBM HTTP Server either on the same server as WebSphere Application Server or on a remote server. To enable cross-origin resource sharing (CORS), you can set the curam.rest.allowedOrigins property for the REST application on your application server, or install the IBM HTTP Server plug-in for WebSphere Application Server.

Before you begin

WebSphere Application Server must be installed and configured.

Install IBM Installation Manager. For more information, see the IBM Installation Manager documentation. You can download IBM Installation Manager from Installation Manager and Packaging Utility download documents.

About this task

To enable cross-origin resource sharing (CORS), choose one of the following options:

- Set the curam.rest.allowedOrigins property for the REST application that is deployed on the application server. For more information about setting the curam.rest.allowedOrigins property, see Cúram REST configuration properties.
- Install and configure the IBM HTTP Server plug-in for WebSphere Application Server to enable IBM HTTP Server to communicate with WebSphere Application Server. WebSphere Customization Toolbox is needed to configure the plug-in.

Procedure

1. Install IBM HTTP Server. For more information, see Migrating and installing IBM HTTP Server.
2. Optional: If you don't set the curam.rest.allowedOrigins property, you must install the following software:
   a) Install the IBM HTTP Server plug-in for WebSphere Application Server.
      For more information, see Installing and configuring web server plug-ins.
   b) Install the WebSphere Customization Toolbox.
      For more information, see Installing and using the WebSphere Customization Toolbox.
3. Start IBM HTTP Server. For more information, see Starting and stopping the IBM HTTP Server administration server.
4. To secure IBM HTTP Server, see Securing IBM HTTP Server.

Generating an IBM HTTP Server plug-in configuration

This task is needed only if you install the IBM HTTP Server plug-in for WebSphere Application Server. Use WebSphere Customization Toolbox to generate a plug-in configuration.

Before you begin

Start WebSphere Application Server. For more information, see Starting a WebSphere Application Server traditional server.

Procedure

To generate the IBM HTTP Server plug-in configuration, complete the steps at the WebSphere Application Server Network Deployment plug-ins configuration topic.

Configuring the IBM HTTP Server plug-in

Configure the IBM HTTP Server plug-in for WebSphere Application Server and WebSphere Customization Toolbox. This task is necessary only if you have chosen to install the IBM HTTP Server plug-in, instead of setting the curam.rest.allowedOrigins property for the REST application that is deployed on the application server.

About this task

You can run the `configurewebserverplugin` target to complete the following tasks:

- Add the web server virtual hosts to the client hosts configuration in WebSphere Application Server.
- Propagate the plug-in key ring for the web server.
- Map the modules of any deployed applications to the web server.

Procedure

1. Start IBM HTTP Server.
   For more information, see Starting and stopping the IBM HTTP Server administration server.
2. On the remote WebSphere Application Server, run the following command.

   ```
   build configurewebserverplugin -Dserver.name=server_name
   ```

   The `configurewebserverplugin` target requires a mandatory `server.name` argument that specifies the name of the server when the target is invoked. For more information about the `configurewebserverplugin` target, see Configuring a web server plug-in in WebSphere Application Server.
3. Consider adding extra aliases to the *client_host*, as shown in the following examples:
   - For WebSphere Application Server, add port number 9044.
   - For the default HTTP port, add port number 80.
   - For HTTPS ports, add port number 433.

For more information about client host setup, see step 19 in the WebSphere Application Server port access setup topic.

4. To avoid port mapping issues from web applications, restart WebSphere Application Server and IBM HTTP Server.

   For more information, see Starting and stopping the IBM HTTP Server administration server.

Install and configure Oracle HTTP Server with Oracle WebLogic Server

Install and configure Oracle HTTP Server on either the same server as Oracle WebLogic Server or on a remote server.

Before you begin

Oracle WebLogic Server must be installed and configured. For more information, see Installing and Configuring Oracle WebLogic Server and Coherence.

Installing Oracle HTTP Server and its components

Install and configure Oracle HTTP Server in either a stand-alone domain, or in an Oracle WebLogic Server domain. If Oracle HTTP Server and Oracle WebLogic Server are on different computers, you must install and configure an Oracle web server plug-in for proxying requests.

About this task

The Oracle web server plugin allows requests to be proxied from Oracle HTTP Server to Oracle WebLogic Server. If you install and configure the Oracle web server plug-in, requests that are delegated to Oracle WebLogic Server still appear to originate from the Oracle HTTP Server, even if Oracle HTTP Server and Oracle WebLogic Server are hosted on two different servers.

Because of the web browser same-origin policy, cross-origin resource sharing (CORS) is restricted in many browsers by default. The web server plug-into enables CORS where Oracle HTTP Server and Oracle WebLogic Server are installed on different computers.

CORS enables an instance of your web application that is deployed on Oracle HTTP Server in one domain to request the REST services that are deployed on Oracle WebLogic Server in another domain.

Procedure

1. Install Oracle HTTP Server for Oracle WebLogic Server. For more information, see Installing and Configuring Oracle HTTP Server.
2. To configure Oracle HTTP Server, choose one of the following options:
   - To configure Oracle HTTP Server in a stand-alone domain, follow the instructions at Configuring Oracle HTTP Server in a Standalone Domain.
   - To configure Oracle HTTP Server in an Oracle WebLogic Server domain, follow the instructions at Configuring Oracle HTTP Server in a WebLogic Server Domain.
3. If Oracle HTTP Server and Oracle WebLogic Server are installed in different domains, to enable CORS, install a web server plug-in.

   For information about configuring an Oracle WebLogic Server proxy plug-in, see Configuring the Plug-In for Oracle HTTP Server.
4. To secure Oracle HTTP Server, follow the procedure at Managing Application Security.

Results

The Oracle HTTP Server instance is now ready to for you to deploy the application. The default location for deploying the application is *OHS_INSTANCE*`/config/fmwconfig/components/${COMPONENT_TYPE}/instances/${COMPONENT_NAME}/htdocs`. However, you can configure the default location value to a different location.

What to do next

Start Oracle HTTP Server. For more information, see Starting the Servers.

Configuring the Oracle HTTP Server plug-in

If a web server such as Oracle HTTP Server is configured in the topology, you must configure a web server plug-in in Oracle WebLogic Server. The web server plug-in enables Oracle WebLogic Server to communicate with Oracle HTTP Server.

About this task

To enable an Oracle HTTP Server web server plug-in in Oracle WebLogic Server, you can run the `configurewebserverplugin` target.

Procedure

1. Start Oracle HTTP Server.

   For more information, see Starting the Servers.
2. On the remote Oracle WebLogic Server, run the following command.

   The `configurewebserverplugin` target requires a mandatory `server.name` argument that specifies the name of the server when the target is invoked.

   ```
   build configurewebserverplugin -Dserver.name=server_name
   ```

   For more information about the `configurewebserverplugin` target, see Configuring a web server plug-in in Oracle WebLogic Server.

Deploying your application

To test your application against an existing IBM Cúram Social Program Management that is deployed on an enterprise application server, you can deploy the application on IBM HTTP Server or Oracle HTTP Server. Both web servers are based on Apache HTTP server so the deployment procedure is similar.

Before you begin

You must have built your application for deployment.

About this task

The built deliverable comes with a preconfigured `.htaccess` configuration file for the Content-Security-Policy (CSP) header. When configured in the web server, this file is detected and executed by the web server to alter the web server configuration by enabling or disabling additional functionality. For more information about CSP, see the *Content Security Policy Quick Reference Guide* related link.

Procedure

1. Copy and paste the `build` directory contents to the appropriate directory for your HTTP server.

   For more information about the `<directory>` directive, see the related links.
2. Configure the web server to call the `.htaccess` file.

   For more information on how to configure `.htaccess` files in a web server, see the *Apache HTTP Server Tutorial: .htaccess files* related link.

Related information

GitHub documentation: npm run build
Content Security Policy Quick Reference Guide
Apache core features V2.0: <Directory> Directive
Apache core features V2.4: <Directory> Directive
Apache HTTP Server Tutorial: .htaccess files

Configuring the IBM Cúram Universal Access server

System administrators use the following configuration options to configure and maintain IBM Cúram Universal Access features such as applications and online categories.

Prerequisites

You must enable cookies and JavaScript in the browsers to access the application by configuring the appropriate browser preferences.

The following table lists the browser preferences that you must configure for the application to work, and shows the errors that are displayed if the prerequisites are not met.

Table 3: Information messages for browser preferences

Browser preference	Information message
When cookies are disabled	Cookies are currently disabled and are required for the application to work. Please enable cookies and retry.
When JavaScript is disabled	JavaScript is currently disabled and is required for the application to work. Please enable JavaScript and retry.
When cookies and JavaScript are disabled	Cookies and JavaScript are currently disabled and are required for the application to work. Please enable and try again.

Configuring service areas and PDF forms

You can define a service area by configuring the counties or ZIP codes that are associated with the service area. You can also specify a PDF form that citizens can use to apply for programs.

Configuring service areas

Service areas are defined in the **Service Areas** section of the administration application. When defining a service area, you must specify a service area name. You can associate counties and zip codes with the service area, these represent the areas covered by the service area. Service areas can be associated with a local office which represents the office that services the service areas associated with it. Local offices identify where citizens can apply in person for a program or where they can send an application. For more information on associating service areas with local offices where a citizen can apply in person for a program, see *Defining local offices for a program*.

Configuring PDF forms

PDF forms are defined in the **PDF Forms** section of the administration application. When defining a PDF form, you must specify a name and language. You can also add a version of the form for each language that is configured. The forms are accessible from the **Print and Mail Form** page.

You can associate a local office with a PDF form. Associating a local office with a PDF form allows an administrator to define the local office and associated service areas where citizens can send their completed application.

Enabling citizens to search for a local office

A search page allows citizens to search for a local office. Citizens can either search by county or by zip code. The system property *curam.citizenworkspace.page.location.search.type* determines how the search works. If you set *curam.citizenworkspace.page.location.search.type* to **Zip**, citizens can search for a local office using a zip code. If you set this property to **County**, citizens can select from a list of counties to get a list of local offices.

Related concepts

Defining local offices for a program

Citizens might be able to apply for a program in person at a local office. A local office must be first defined in the *LocalOffice* code table in system administration.

Configuring programs

You can configure different types of programs. To configure a program, you configure display and system processing information, local offices, mappings to PDFs, and evidence types.

Configured programs can be associated with applications. The main aspects to configuring a program are as follows:

- Configure programs and associated display and system processing information.
- Configure local offices where an application for a program can be sent.
- Configure mappings that allow information gathered during application intake to be mapped to a PDF form.
- Configuring evidence types that allows for expedited authorization of programs that may need to be processed before other programs within a multi-program application.

Related concepts

Defining programs for a screening

Configuring a Program

Programs are configured on the administration **New Program** page. Details and specifications of the program are required to be defined when the program is created.

Defining a name and reference

The name that you define is displayed in the administration application.

Define a name and reference when creating a new program. The name that is defined is displayed both to the citizen and in the internal application. The reference is used to reference the program in code.

Defining an intake processing system

Define an intake processing system for each program.

Two options are available:

- **Cúram**
- Select from the list of preconfigured remote systems.

If intake is managed by IBM Cúram Social Program Management, select **Cúram**. If intake is managed by an external system, the program application is sent to the remote system by using the `ProcessApplicationService` web service, select a remote system.

If **Cúram** is specified as the intake system, an application case type must be selected. An application case of the specified type is created in response to a submission of an application for the program. An indicator is provided which dictates whether a **Reopen** action is enabled on the programs list on an application case for denied and withdrawn programs of a particular type. A workflow can be specified that is initiated when the program is reopened. For more information on configuring application cases, see *Cúram Intake overview*.

When an application case type is selected, the program can be added manually to that type of application case by a worker in the internal application as part of intake processing. A configuration setting specifies whether the program is a coverage type. Coverage types are automatically evaluated by program group rules in the context of healthcare reform applications, such as insurance affordability. Coverage types cannot be applied for directly by a citizen or manually added to an application case by a worker and authorized. If the program is a coverage type, select **Yes**. The program is filtered out of the list of programs available to be added to online and internal applications in administration and the list of programs available to be manually added to an application case by a worker. If the program is not a

coverage type, select **No**. The program will be available to be manually added to online and internal applications in administration and to an application case by a worker.

A remote system must be configured in the administration application before it can be selected as the case processing system. For more information about remote systems, see *Configuring Remote Systems.*

Related concepts

Configuring remote systems

Applications and life events data can be sent through web services for processing by a remote system. To enable remote processing, specify a remote system and the required web services. Remote systems can be configured allowing applications and life event data to be sent to them for processing via associated web services.

Related information

Cúram Intake overview

Defining case processing details

Define a case processing system for each program.

Two options are available:

- **Cúram**
- Select from remote systems.

If the program eligibility is determined and managed by using a Cúram-based system, select **Cúram**. If eligibility is determined and managed by an external system, select a remote system.

If you select **Cúram** as the case processing system, more options are available to allow you to configure program level authorization. Program level authorization means that if an application case contains multiple programs, each program can be authorized individually, and a separate case is used to manage the citizens on an ongoing basis.

Defining the integrated case strategy

Define the integrated case strategy so that the system can identify whether a new or existing integrated case is used when program authorization is successful.

The integrated case strategy identifies whether a new or existing integrated case is used when program authorization is successful. The integrated case hosts any product deliveries created as a result of the authorization. If a new integrated case is created, all of the application case clients are added as case participants to the integrated case. If an existing integrated case is used, any additional clients on the application case are added as case participants to the integrated case. Any evidence captured on the application case that is also required on the integrated case is copied to the integrated case upon successful authorization. The configuration options for the integrated case strategy are as follows:

New
: A new integrated case of the specified type is always created when authorization of the program is successful.

Existing (Exact Client Match)
: If an integrated case of the specified type exists with the same citizens as those cases present on the application case, the existing case is used automatically. If multiple integrated cases that meet these criteria exist, the caseworker is presented with a list of the cases and must select one to proceed with the authorization. If no existing cases match the criteria, a new integrated case is created.

Existing (Exact Client Match) or New
: If one or more integrated cases of the specified type exist with the same citizens as those cases present on the application case, the caseworker is presented with the option to select an existing case to use as the ongoing case, or to create a new integrated case. If no existing cases match the criteria, a new integrated case is created.

Existing (Any Client Match) or New
: If one or more integrated cases of the specified type exist, where any of the clients of the application case are case participants, the caseworker is presented with the option to select one of the existing

cases to use as the ongoing case, or to create a new integrated case. If no existing cases match the criteria, a new integrated case is created.

Specifying the Integrated Case Type
The administrator must specify the type of integrated case to be created or used upon successful program authorization as defined by the Integrated Case strategy listed.

Specifying a client selection strategy
Specify a client selection strategy to define how clients are added from the application case to the product delivery.

The client selection strategy defines how clients are added from the application case to the product delivery created as a result of authorization of a program. If a product delivery type is specified, a client selection strategy must be selected. The configuration options are as follows:

All Clients
All of the application clients are added to the product delivery case. The application case primary client is set as the product delivery primary client. All other clients are added to the product delivery as members of the case members group.

Rules
A rule set determines the clients to be added to the product delivery if a product delivery is configured. At least one client must be determined by the rules for authorization to proceed.

User Selection
The user selects the clients who are added to the product delivery. The caseworker must select both the primary client and any other clients to be added to the case member group on the product delivery.

Specifying a Client Selection Ruleset
A Client Selection Ruleset must be selected when the Client Selection Strategy is **Rules**.

Specifying a product delivery type
Specify a product delivery type.

The **Product Delivery Type** drop-down specifies the product delivery that is used to make a payment to citizens in respect of a program. **Product Delivery Type** displays all active products configured on the system.

Note: This field applies to both program and application authorization processing. That is, program and application authorization can result in the creation of the product delivery type that is specified.

Submitting a product delivery automatically

The **Submit Product Delivery** indicator specifies if the product delivery created as a result of program authorization should be submitted automatically for approval. If selected, the product delivery created as a result of authorization of this program is submitted automatically to a supervisor for approval.

Note: This field applies to both program and application authorization processing. That is, program and application authorization can result in the automatic submission of a product delivery.

Configuring timers
Agencies can impose time limits within which an application for a program must be processed. You can configure application timers for each of these programs.

For example, an agency might want to specify that food assistance applications are authorized within 30 business days of the date of application.

The following configuration options are available, including the duration of the timer, whether the timer is based on business or calendar days, a warning period, and timer extension and approval.

Duration
The length of the timer in days. This value, along with the fields **Start Date** and **Use Business Days** (and the configured business hours for the organization) calculate the expiry date for the timer. This

value is used as a number of business days if **Use Business Days** is set. If **Use Business Days** is not set, this value is used as calendar days.

Start Date
Specifies whether the timer starts on the application date or the program addition date. The options available are **Application Date** and **Program Addition Date**.

Note: In most cases, these dates are the same. That is, the programs are added at the same time as the application is made. However, when a program is added later to the application, after initial submission, the dates differ.

Warning Days
Specifies a number of warning days to warn citizens that the timer deadline is approaching. If configured, the **Warning Reached** workflow is enabled when the warning date is reached and the timer is still running (for example, the program is not completed).

End Date Extension Allowed
Specifies whether citizens can extend the timer by a number of days.

Extension Approval Required
Specifies whether a timer extension requires approval from a supervisor. If approval is required, the supervisor either approves or rejects the extension. After the extension is approved, or if approval is not required, the timer expiry date is updated to reflect the extension.

Use Business Days
Specifies if the timer should not decrement on non-working days. If this indicator is set, the system uses the **Working Pattern Hours** for the organization to determine the non-working days when it is calculating the expiry date for the timer.

Resume Timer
Specifies whether the program timer must be resumed when the program is reopened.

Resume From
If a timer is resumed, the **Resume From** field specifies the dates from which a program can be resumed. The values include the date that the program was completed, denied, or withdrawn, and the date that the program was reopened.

Timer Start
Specifies a workflow that is started when the timer starts.

Warning Reached
Specifies a workflow that is started when the warning period is reached.

Deadline Not Achieved
Specifies a workflow that is enacted if the timer deadline is not achieved; that is, the program is not being withdrawn, denied, or approved by the timer expiry date.

Configuring multiple applications
Configure multiple applications so that citizens can apply for a program while they have a previous application pending.

The **Multiple Applications** indicator dictates if citizens can apply for a program while they have a previous application pending. If set to true, citizens can have multiple pending applications for the given program. That is, citizens can submit an application for this program while they already have a pending application in the system. If it is set to false, this program is not offered if logged in citizens have pending applications for this program.

This configuration is not applicable to Health Care Reform Applications.

Defining a PDF form
Defining a PDF form for a program enables citizens to print an application for that program and either post it to the agency or bring it to a local office.

When a PDF Form is specified for a program, the PDF form is displayed on the **Print Out and Mail** section of the **What you might get** page that is displayed when citizens complete a screening. PDF Forms must be

defined before they can be associated with a program. When they are defined, they are displayed on the **Print and Mail Application Form** page.

Defining a URL
If a URL is defined, a **More Info** link is displayed with the program name so that citizens can find out more information about the selected program.

Defining description and summary information
When a program is displayed on the **Select Programs** page, a description can be displayed which gives a description of the program. The **Online Program Description** field defines this description.

A description summary of the program can also be defined using the **Online Program Summary** field. The field is a high-level description of the program displayed on the **What you might get** page that is displayed when citizens complete a screening.

Defining local office application details
Citizens can apply for programs at a local office. If this is the case, the **Citizen Can Apply At Local Office** indicator indicates that local office information is displayed for a program.

Additional information can also be defined, for example, citizens might need to bring proof of identity if they want to apply at the local office. An administrator can define this information in the **Local Office Application Information** field.

Defining local offices for a program
Citizens might be able to apply for a program in person at a local office. A local office must be first defined in the *LocalOffice* code table in system administration.

Associating a local office with a program allows an administrator to define the local offices and their associated service areas where a particular program can be applied for in person. This information is displayed on the **What you might get** page that is displayed to citizens when they complete a screening. Service areas must be defined before they can be associated with a local office.

Defining PDF mappings for a program
The information that citizens enter during an application can be mapped to a PDF form which citizens can then print.

To map the application data to the PDF Form for all programs a citizen is applying for, there must be a mapping configuration of type *PDF Form Creation* for each of the programs. The PDF Form is the form specified for the Online Application the program is associated with.

Defining program evidence types
Associate evidence types with a program.

Evidence types can support applications for multiple programs where a program must be authorized more quickly than other programs for which citizens might have applied. Using this type of configuration, only the evidence required for the program to be authorized is used and copied to the ongoing cases. This allows benefits for the authorized program to be delivered to citizens, while the caseworker continues to gather the evidence required for the other programs applied for.

Configuring applications

The administration system allows you to define different types of applications. Once defined, citizens can submit an application for programs to the agency. For each application, you can configure the available programs and an application script and data schema. You can also configure the remaining applications details, including application withdraw reasons.

There are five aspects to configuring an application:

- Configuring information about an application and associated display information.
- Configuring the script and schema that collects and stores the information specified during the application.
- Configuring the programs for which an application can be used to apply.

- Configuring reasons that citizens can select withdrawing an application.
- Configuring additional application system properties.

Related concepts

Saving an application

By default, applications are automatically saved for citizens who are logged in. Citizens can also manually save applications, including in-progress applications.

Configuring an application

Configure applications on the **New Application** page using the these application configurations.

Name

The name of the application displayed in the online portal.

Program selection

Indicates whether citizens can select specific programs to apply for or whether they are brought directly into an application script. That is, citizens can apply for all programs associated with the application.

URL

If a URL is defined, a **More Info** link is displayed with the application name so that citizens can find out more information about the selected application.

Summary information

Allows an administrator to define a high-level description of the application to be displayed.

Description information

Allows an administrator to define a description of the application to be displayed.

Configuring an application script

Define an IEG for the application to collect the answers to the application questions.

Specify a script name in the **Question Script** field. A data store schema must be specified to store the data entered in the script. A schema name must be specified in the **Schema** field. On saving the application, an empty template for both the script and schema is created by the system based on the question script and schema specified. You can update these templates from the **Application** tab by selecting the hyperlinks provided on the page. Click the **Question Script** link to start the IEG editor so you can edit the question script. Click the **Schema** link to start the Datastore Editor and edit the schema.

Configuring a submission script

Configure a submission script for an application so that citizens can submit an application to the agency. The script defines additional information which does not form part of the application script to be captured, for example, a TANF typically requires information regarding the citizen's ability to attend an interview.

Specify an IEG submission script in the **Submission Script** field. On saving the application, an empty template for the submission script is created by the system based on the Submission Script that you specify. You can update this from the **Application** tab by selecting the hyperlink on the page. Clicking on the link starts the IEG editor which lets you edit the question script.

Defining a PDF form

Define a PDF form that is displayed when citizens complete and online application.

The data that is collected during the online application is copied by the system into this PDF, which citizens can print. The PDF form can be selected from the **PDF Forms** drop down menu. If a PDF form is not specified for an application, a default generic PDF form can be used. You can get the default template from the **XSL Templates** section of the system administration application.

The data passed to the XSL template reads directly from the data store. Instead of displaying the datastore labels in the PDF, define a property file to specify user-friendly names for entities and attributes and to hide entities and attributes that you do not want to display in the PDF. Upload the property file to **Application Resources** in the **Intelligent Evidence Gathering** section of the administration application.

Name the property file using the following convention: `<application schema name>PDFProps`. The contents of the property file is as follows:

Name an entity
`<Entity Name=<Name To Be Displayed in the PDF>`, for example, *Application=Intake Application*

Hide an attribute
`<Entity Name.Attribute Name.hidden=true`, for example, *Application.user Name.hidden=true*

Specify a label for an attribute
`<Entity Name.Attribute Name=PDF Label`, for example, *Submission.dig FirstName=First Name*

Configuring client registration
Use the **Client Registration** field to decide whether clients are registered as prospects or persons.

To determine whether to register the client as a prospect or a person, the system checks the client registration configuration in the following two scenarios:

- If **Person Search and Match** is configured, and no match can be found for the client.
- If **Person Search and Match** is not configured, that is, the clients on an application are always registered without the system automatically searching and matching them.

If the **Client Registration** field is not set, the system checks the system property **Register as Prospect Person** to identify whether a client is registered as a prospect or a person.

Configuring submission confirmation page details
Additional information can be configured on the **Submission Confirmation** page which is displayed when citizens submit an online application.

Use the **Title** and **Text** fields to define a title and text to be displayed on the **Submission Confirmation** page.

Associating programs with applications
Associate programs with the application so that citizens can apply for particular programs.

Any program described in **Configuring Programs** can be associated with an application. When associating programs with an application, you can set the display order of the selected program relative to other programs associated with the application.

Defining mappings for an application
Applications can be processed by IBM Cúram Social Program Management or a remote system.

If the application is processed by IBM Cúram Social Program Management the information entered in an application is mapped to the evidence tables associated with the application case defined for the programs associated with the application. The mappings are configured for an application by creating a mapping using the Data Mapping Editor. A mapping configuration must be specified in order for the appropriate evidence entities to be created and populated in response to an online application submission.

For more information about the Data Mapping Editor, see the *Data Mapping Editor Guide*.

Configuring withdrawal reasons
Citizens can withdraw the application for all or any one of the programs for which they applied.

When withdrawing an application, a withdrawal reason must be specified. Withdrawal reasons can be defined for a particular application in the **Intake Application** section of the administration application. Before associating a withdrawal reason with an application, withdrawal reasons must be defined in the **WithdrawalRequestReason** code table.

Mandating authentication before applying
The agency can configure the system to specify whether citizens must create an account or log in to make an application.

The system property *curam.citizenworkspace.authenticated.intake* indicates if authentication is switched on. If this property is switched on, citizens must create an account or log in before starting an application. If *curam.citizenworkspace.authenticated.intake* is switched off, citizens are taken directly to the application selection page.

If *curam.citizenworkspace.authenticated.intake* is set to '**YES'**, citizens are navigated to the following components:

- The **Apply for benefits** page.
- The login page when citizens click **Apply**.

Optional authenticated application
The agency can configure the system to specify whether, before applying, citizens can choose to be authenticated.

The system property *curam.citizenworkspace.intake.allow.login* indicates if authentication is switched on or off. If this property is switched on, citizens will be given the option to log in before starting an application. If *curam.citizenworkspace.intake.allow.login* is switched off, citizens are taken directly to the application selection page.

Displaying a confirmation page on quit
The agency might want to display a confirmation page to citizens when they quit the application process.

The system property *curam.citizenworkspace.display.confirm.quit.intake* indicates if a confirmation page is displayed. If this property is switched on, a confirmation page is displayed when quit is selected while making an application. If *curam.citizenworkspace.display.confirm.quit.intake* is switched off, a confirmation page is not displayed when citizens quit an application. This property is only used when the property *curam.citizenworkspace.intake.allow.login* is set to **NO**.

Mandating authentication before submission
The agency might want to mandate that citizens log in before submitting an application.

The system property *curam.citizenworkspace.intake.submit.intake.mandatory.login* indicates that citizens must log in before submitting an application. If *curam.citizenworkspace.intake.submit.intake.mandatory.login* is switched on, citizens must create an account or login before they can submit an application. If *curam.citizenworkspace.intake.submit.intake.mandatory.login* is switched off, citizens can submit an application without logging in.

Enabling applications link
Use the system property *curam.citizenworkspace.intake.enabled* to indicate whether citizens can start the application process from the **Home** page.

If *curam.citizenworkspace.intake.enabled* is switched on, the **Apply For Benefits** link is displayed on the **Home** page. If *curam.citizenworkspace.intake.enabled* is switched off the applications link is not displayed.

Prepopulating the application script
When authenticated citizens apply from benefits from their accounts, information already known about the citizen performing the application can be prepopulated.

The system property *curam.citizenaccount.prepopulate.intake* indicates whether the IEG script is prepopulated. The default value of this property is true which means that the script is prepopulated.

The application auto-save property

The auto-save Intake property dictates if applications are auto-saved in the citizen account.

By default, this property is set to true. All applications irrespective of type are automatically saved. Each application is auto-saved when citizens click **Next** as they progress through the IEG script. If this property is set to false, applications are not automatically saved in the citizen account.

Configuring online categories

Online categories group different types of applications together to make it easier for citizens to find the ones that they need. You must define online categories for screenings and applications to be displayed. After you define online categories, you must associate each application to a category.

Defining online categories

When defining an online category a name and URL must be defined. If a URL is defined a **More Info** link is displayed with the name of the online category allowing citizens to find out more information about the selected category. An order can be assigned to a category which dictates the display order of the selected category relative to other categories.

Associating applications

Applications must be associated with an online category so they can be displayed in the application. When associating an application with an online category an order can be applied which dictates the display order of the application relative to other applications within the same category.

Configuring the citizen account

Although customization is required to modify some citizen account information, you can configure information on the citizen account and the **Contact Information** tab.

Messages can originate as a result of transactions in IBM Cúram Social Program Management or a remote system. Most of the configuration options apply to all messages but there are a some configuration options that do not apply to messages originating from a remote system.

Configuring the citizen account

Configure the citizen account.

Many aspects of the citizen account are configurable:

- The text displayed in the participant messages in the **Messages** panel
- The system messages displayed in the **Messages** panel
- The display order of the messages in the **Messages** panel

Configuring messages

The **Messages** panel of the organization **Home** page displays messages to logged-in citizens. For example, a message that informs citizens when their next benefit payment is due or the amount of the last payment.

Messages can be displayed which relate to meetings, activities, and application acknowledgments. Messages can be displayed as a result of transactions in IBM Cúram Social Program Management or they can originate from remote systems by way of a web service.

The links that follow outline the aspects of the **Messages** section, which are configurable.

Account messages

Adding a message or changing a dynamic element of an account message requires customization. The text that is defined for existing messages that are provided in the initial application configuration can be updated by using a set of properties for each type of message.

Properties are as follows:

- `CitizenMessageMyPayments` - the messages about payments.

- `CitizenMessageApplicationAcknowledgement` - messages about application acknowledgments.
- `CitizenMessageVerificationMessages` - messages about verification messages.
- `CitizenMessageMeetingMessages` - messages about meetings.
- `CitizenMessagesReferral.properties` - messages about referrals.
- `CitizenMessagesServiceDelivery` - messages about service deliveries.
- `CitizenAppealRequestMessage` - messages about appeal requests.

Property files are stored in the **Application Resources** section of the administration application. To update the message, each file needs to be downloaded, updated, and uploaded again. The icons that are displayed in the citizen account for each type of message can be configured in the **Account Messages** section of the **administration** application.

Adding a message that originates from a remote system requires that a code table entry to be added to the `ParticipantMessageType` code table and an associated entry in the **Account Messages** listing in the administration application. Messages then can be sent by way of the `ExternalCitizenMessageWS` web service.

Creating application acknowledgments
Create messages to acknowledge an application.

Table 4: Application acknowledgment

Message Area	Description
Title	<Icon> TANF Application Acknowledgment
Message	`We have received your TANF Application form. The status of this application is pending. We will contact you when the application has been processed.`
Effective Date	Current® date
Duration	An administrator can use a configuration setting to define the number of days (from the effective date) that the message is displayed.
Notes	None.

Creating meeting messages
Create messages for a meeting invitation, a meeting cancellation, and a meeting update. An administrator can use a configuration setting to set the number of days (from the effective date) that the meeting messages are displayed.

Table 5: Meeting invite

Message Area	Description
Title	<Icon> Meeting Invitation - Meeting with Case Worker
Message 1 (Not an all day meeting and the meeting start and end date are on the same day)	`You are invited to attend a meeting from 9.00AM until 5.00PM on 12/04/2010 in Meeting Room 1, Block C. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.`

Table 5: Meeting invite (continued)	
Message Area	**Description**
Message 2 (All day meeting for one day only)	You are invited to attend an all day meeting on 12/04/2010 in Meeting Room 1, Block C. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.
Message 3 (All day meeting for multiple days)	You are invited to attend an all day meeting each day from 12/04/2010 until 15/04/2010 in Meeting Room 1, Block C. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.
Message 4 (Non-all day meeting for multiple days)	You are invited to attend a meeting from 9.00AM until 5.00PM from 12/04/2010 to the 13/04/2010 in Meeting Room 1, Block C. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.
Notes	When the case worker is setting up a meeting, the location is an optional field. Therefore, if a meeting location is not specified, the preceding messages are displayed without a location. Also, the meeting organizer's contact details are optional. Therefore, if no contact details are found, the preceding message is displayed without the organizer's contact details.

Table 6: Meeting cancellation	
Message Area	**Description**
Title	<Icon> Cancellation - Meeting with Case Worker
Message 1 (Not an all day meeting and the meeting start and end date are on the same day)	The meeting that you were scheduled to attend from 2.00PM until 6.00 PM on 12/04/2010 is canceled. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information.
Message 2 (All day meeting for one day only)	The all day meeting that you were scheduled to attend on 12/04/2010 is canceled. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information.

Table 6: Meeting cancellation (continued)	
Message Area	**Description**
Message 3 (All day meeting for multiple days)	The all day meeting that you were scheduled to attend from 12/04/2010 until 15/04/2010 is canceled. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information.
Effective Date	Current Date.
Notes	The meeting organizer's contact details link opens a page that shows the organizer's contact details.

Table 7: Meeting update	
Message Area	**Description**
Title	<Icon> Cancellation - Meeting with Case Worker
Message 1 (Date and Time change of a non-all-day meeting)	The meeting that you were scheduled to attend from 2.00PM until 6.00 PM on 12/04/2010 is rescheduled to 3.00PM until 7.00 PM on 13/04/2010 in Meeting Room 1, Block C. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.
Message 2 (Location change of a non-all-day meeting)	The location of the meeting you are scheduled to attend from 2.00PM until 6.00 PM on 12/04/2010 is changed. This meeting is now scheduled for Meeting Room 1, Block D. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.
Message 3 (Date, time, and location change of non-all-day meeting)	The meeting that you were scheduled to attend from 2.00PM until 6.00 PM on 12/04/2010 is rescheduled to 3.00PM until 7.00 PM on 13/04/2010. It is rescheduled for Meeting Room 2, Block C. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.
Message 4 (Date change of all day meetings for multiple days)	The all day meeting that you are scheduled to attend from 12/04/2010 until 15/04/2010 is rescheduled. This meeting will now take place from 13/04/2010 until 16/04/2010. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.

Table 7: Meeting update (continued)

Message Area	Description
Message 5 (Location change for all day meeting for multiple days)	The location of the all day meeting you are scheduled to attend from 12/04/2010 until 15/04/2010 is changed. This meeting is rescheduled for Meeting Room 1, Block D. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.
Message 6 (Date and location change for all-day meeting for multiple days)	The all day meeting that you are scheduled to attend from 12/04/2010 until 15/04/2010 is rescheduled. This meeting will now take place from 13/04/2010 until 16/04/2010 in Meeting Room 1, Block D. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.
Message 7 (Date change for an all-day meeting)	The all day meeting that you are scheduled to attend on 12/04/2010 is rescheduled. This meeting will now take place on 13/04/2010. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.
Message 8 (Location change for an all-day meeting)	The location of the all day meeting you are scheduled to attend on 12/04/2010 is changed. This meeting is rescheduled for Meeting Room 1, Block D. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.
Message 9 (Date and location change for an all-day meeting)	The all day meeting that you are scheduled to attend on 12/04/2010 is rescheduled. This meeting is rescheduled for 13/04/2010 in Meeting Room 1, Block D. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.
Message 10 (Date and time change of a non-all-day meeting for multiple days)	The meeting that you are scheduled to attend from 2.00PM until 6.00 PM on 12/04/2010 until 15/04/2010 is rescheduled. This meeting is rescheduled for 2.00PM until 6.00 PM on 13/04/2010 until 16/04/2010. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.

Table 7: Meeting update (continued)	
Message Area	**Description**
Message 11 (Location change of a non-all-day meeting for multiple days)	`The location of the meeting you are scheduled to attend from 2.00PM until 6.00 PM on 12/04/2010 until 15/04/2010 is changed. This meeting is rescheduled for Meeting Room 1, Block D. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.`
Message 12 (Date, time, and, location change of non-all-day meeting for multiple days)	`The meeting that you are scheduled to attend from 2.00PM until 6.00 PM on 12/04/2010 until 15/04/2010 is rescheduled. This meeting is rescheduled for 2.00PM until 6.00 PM on 13/04/2010 until 16/04/2010 in Meeting Room 1, Block D. Please contact Joe Bloggs at 014567832 or joe@SemAgency.com if you need more information or cannot attend.`
Notes	When the case worker is setting up a meeting, the location is an optional field. Therefore, if a meeting location is not specified, the preceding messages are displayed without a location. Also, the meeting organizer's contact details are optional. Therefore, if no contact details are found, the preceding message is displayed without the organizer's contact details.

Creating payment messages

Create messages for an issued payment, a canceled payment, a due payment, a stopped payment, an unsuspended payment, an issued overpayment, and an issued underpayment. An administrator can use a configuration setting to set the number of days (from the effective date) that the payment messages are displayed.

Table 8: Payment issued	
Message Area	**Description**
Title	<Icon> Latest Payment
Message 1	`Your latest payment of $22.00 was due on 22/07/2009. Click here to view the payment details. Your next payment is due on 29/07/2009. Click My Payments to view your payment history.`

Table 8: Payment issued (continued)	
Message Area	**Description**
Message 2 (Payment previously canceled)	`Your latest payment of $22.00 was due on 22/07/2009. Click here to view the payment details. This payment was originally canceled on 23/07/2009. Click here to view details of the canceled payment. Your next payment is due on 29/07/2009. Click My Payments to view your payment history.`
Effective Date	Current Date.
Notes	A payment can be issued, then canceled, and then reissued. The **here** hyper link opens a page that shows payment details. The **My Payments** link opens the **My Payments** page in the Citizen Account. **Note:** If no more payments are due, the `Your next payment is due on 29/07/2009` part of the messages is not displayed.

Table 9: Payment canceled	
Message Area	**Description**
Title	<Icon> Payment Canceled
Message	`Your payment of $22.00, due on 22/07/2009, has been canceled. Click here to view the details. Click Contact Information to contact your caseworker if you need more information. Your next payment is due on 29/07/2009. Click My Payments to view your payment history.`
Effective Date	Current Date.
Notes	If no more payments are due, the `Your next payment is due on 29/07/2009` part of the message is not displayed. The **Contact Information** link opens the **Contact Information** tab in the citizen account. The **My Payments** link opens the **My Payments** page in the Citizen Account.

Table 10: Payment due	
Message Area	**Description**
Title	<Icon> Next Payment Due
Message	`Your next Cash Assistance payment is due on 29/07/2011.`

Table 10: Payment due (continued)	
Message Area	**Description**
Effective Date	Current Date.
Notes	This message is appropriate when it is the first payment that a citizen receives.

Table 11: Case suspended	
Message Area	**Description**
Title	<Icon> Payments Stopped
Message	`Your Cash Assistance payments have been stopped from 29/07/2009. Click Contact Information to contact your caseworker if you need more information.`
Effective Date	Current Date.
Notes	The **Contact Information** link opens the **Contact Information** tab in the Citizen Account.

Table 12: Case unsuspended	
Message Area	**Description**
Title	<Icon> Payments Unsuspended
Message	`Your Cash Assistance payment suspension has been lifted from 29/07/2009. Your next payment is due on 31/07/2009.`
Effective Date	Current Date.
Notes	None.

System messages

Agencies use system messages to send messages to citizens who have a citizen account. For example, if an agency wants to provide information and help line numbers to citizens who were affected by a natural disaster. System messages can be configured in the administration application by using the **New System Message** page.

Use the **Title** and **Message** fields to define the title of the message and the message body that is displayed in the **My Messages** pane. Define the message as a priority by using the **Priority** field, the message appears at the top of the messages listing.

Note: If multiple priority messages exist, the effective date of the message and the message type is used to dictate the message order. For more information, see *Ordering and Enabling/Disabling Messages*.

Use the **Effective Date and Time** to define an effective date for the message, such as when the message is displayed in the citizen account. Use the **Expiry Date and Time** field to define an expiry date for the message, for instance, when to remove the message from the Citizen Account.

When the message is saved, it has a status of **In-Edit**. Before the message is displayed in the Citizen Account, it must be published. After it is published, the message is active and is displayed in the Citizen Account based on the effective and expiry dates defined.

Configuring message duration
System properties set the length of time a type of message is displayed in the citizen account. For example, a payment message can be configured to be displayed for 10 days. These configuration options apply only to messages that originate as a result of transactions on IBM Cúram Social Program Management.

The following system properties are provided:

- `curam.citizenaccount.payment.message.expiry.days` - sets the number of days from the effective date that a payment message is displayed in the citizen account. A payment message is displayed for this duration unless another payment message is created which replaces it. The default value is 10.
- `curam.citizenaccount.intake.application.acknowledgement.message.expiry.days` - sets the number of days from the effective date that an application acknowledgment message is displayed in the citizen account. An acknowledgment message is displayed for this duration unless another acknowledgment message is created which replaces it. The default value is 10.
- `curam.citizenaccount.meeting.message.effective.days` -sets the number of days from the effective date that a meeting message is displayed. A meeting message is displayed for this duration unless another meeting message is created which replaces it. The default value is 10.

Switching off messages
An agency might not want to display messages in the Citizen Account. To cater for this choice, the system property `curam.citizenaccount.generate.messages` enables an agency to switch all messages *on* or *off*. The default value is `true`, which means that messages are generated and displayed in the Citizen Account.

Configuring last logged in information
The text displayed in the welcome message and last logged on information can be updated using the properties that are stored in the `CitizenAccountHome` properties file stored in the **Application Resource** section of the Administration Application.

The following properties are provided:

- *citizenaccount.welcome.caption* - updates the welcome message.
- *citizenaccount.lastloggedon.caption* - updates the last logged on message.
- *citizenaccount.lastloggedon.date.time.text* - updates the date and time message.

Configuring contact information
Configure contact information for citizens and caseworkers.

Contact information displayed in the citizen account displays contact details (phone numbers, addresses and email addresses) stored for the logged in citizen and also caseworker contact details (business phone number, mobile phone number, pager, fax and email) of the case owners of cases associated with the logged in citizen in IBM Cúram Social Program Management and on remote systems.

Citizen contact information

The following system property is provided that sets whether contact information is displayed to a citizen.

`curam.citizenaccount.contactinformation.show.client.details`
If the property is set to `true`, citizens' address, phone number, and email address are displayed. If this property is set to `false`, contact information is not displayed. The default value for this property is true.

Caseworker

The following system properties are provided to set whether agency worker contact information is displayed to a citizen, and if displayed, additional system properties are provided to dictate the type of contact information displayed:

curam.citizenaccount.contactinformation.show.caseworker.details
Sets whether worker contact details are displayed in the citizen account. If this property is set to true, worker contact details of cases associated with the logged in citizen are displayed. If this property is set to false, worker contact information is not displayed. The default value for this property is true.

curam.citizenaccount.contactinformation.show.businessphone
Sets whether the worker's business phone number is displayed. The default value of this property is true.

curam.citizenaccount.contactinformation.show.mobilephone
Sets whether the worker's mobile number is displayed. The default value of this property is true.

curam.citizenaccount.contactinformation.show.emailaddress
Sets whether the worker's email address is displayed. The default value of this property is true.

curam.citizenaccount.contactinformation.show.faxnumber
Sets whether the worker's fax number is displayed. The default value of this property is true.

curam.citizenaccount.contactinformation.show.pagernumber
Sets whether the worker's pager is displayed. The default value of this property is true.

curam.citizenaccount.contactinformation.show.casemember.cases
Sets whether the worker's contact information is displayed for cases where the citizen is a case member. If this property is set to true, cases where the citizen is a case member are displayed. If this property is set to false, then only cases where the citizen is the primary client are displayed. Note: this property only applies to cases originating from IBM Cúram Social Program Management. The types of product deliveries and integrated cases to be displayed can be configured in the Product section of the Administration Application. For more information on administering this see the *Cúram Integrated Case Management Configuration Guide*.

Configuring remote systems

Applications and life events data can be sent through web services for processing by a remote system. To enable remote processing, specify a remote system and the required web services. Remote systems can be configured allowing applications and life event data to be sent to them for processing via associated web services.

Specifying a name and root URL

Remote systems are configured in the Administration application. A name and Root URL must be specified. The Root URL represents the root Uniform Resource Locator (URL) of the remote system. It consists of the Protocol (http or https), Host Name (for example, shell) and Port (for example, 9082) . An example for root URL is http://shell:9082/. A Display Name can be configured which is used to display the name of the agency associated with the remote system. This is used in the citizen account when a list of remote systems are displayed to a citizen. It should be used to represent a more meaningful agency name to a citizen rather than using the remote system name. The Source User Name represents the user name that the remote system uses when invoking inbound Cúram web services.

Adding a service to a remote system

A target system can have multiple services associated with it. A URL must be defined for every service associated with a remote system. The URL is used for identifying and interacting with the service in the remote system. The URL is based on the combination of the root URL of the remote system, consisting of the system host name and port, and the extension URL for the associated service. For example, a URL http://shell:9082/ProcessApplicationService for a process application web service on a remote system is the combination of the root URL(http:// shell:9082/) of the remote system and the extension URL(ProcessApplicationService) of the associated process application service. The Invoking User Name and Password define the user name and password required to communicate with the web service on the remote system.

Securing the IBM Cúram Universal Access server

The IBM Cúram Universal Access web application is gives citizens access to their most sensitive personal data over the internet. Security must be a primary concern in the development of citizen account customizations. All projects that are built on Universal Access must focus on delivering security from beginning to end.

It is recommended that all projects take at least the following steps to ensure the security of the project delivery:

- Ensure that the project team are familiar with the principles of secure application development, and common vulnerabilities such as the OWASP Top Ten.
- Develop and apply a Threat Model
- Employ security experts to test everything from requirements to the finished deployment.
- Plan for how the application is used in public spaces like libraries and kiosks.

The security model

The IBM Cúram Universal Access security model implements different account types to support both anonymous and registered citizens. As citizens use Universal Access, they transition through the account types.

IBM Cúram Universal Access has the following user types:

Public citizen account

When citizens view the organization **Home** page they are automatically logged in under the *publiccitizen* account. This account only has access to the home page and pages that allow citizens to enter or reset passwords.

Anonymous account

When the user clicks a link to perform intake, they are logged out as *publiccitizen* and logged back as an *anonymous* account with a random user name. A principle of Universal Access is that users do not have access to the data of other users. If all intakes and screenings are performed using a single user account, *publiccitizen*, for example, one citizen might see data that has been entered by another citizen.

Registered accounts

Standard accounts created by citizens. Citizens can create accounts when they first use the application, or during processes like applying for benefit. These accounts differ from anonymous accounts in that they allow citizens to continue previously saved applications, restart applications that were previously unfinished, and review or withdraw previously submitted applications.

Linked accounts

Linked accounts are accounts that have been linked with an underlying Concern Role ID for a Person entity.

Some typical scenarios for linking are presented. These are examples, the actual processes for linking is unique to each citizen. A citizen requests a Citizen Account. The citizen is asked to present themselves at their local Social Welfare office with drivers license and other personal identification. The caseworker, uses custom developed functions to enter details for the new linked account after verifying the identity of the citizen.

A citizen creates a user account for Universal Access and submits an Intake Application. They are contacted by their caseworker who asks them if they want access to more services. The citizen agrees and presents themselves at the local office with identification such as a passport. The caseworker is able to link the citizen to the account they used to submit the Intake Application.

In both of these cases the caseworker does not have access to the citizen's password. Instead, the linking process triggers a batch job that generates a letter, sent to the citizen's home address. The letter contains the password and a separate letter then contains an electronic code card. All of this functionality is developed by the customer however it is supported by Universal Access APIs that allow a user name to be linked to a Concern Role ID.

Authorization roles and groups

The account types are assigned different authorization roles. The roles limit the methods that can be invoked. No additional permissions should be granted to authorization roles except for Linked Accounts, which use the LINKEDCITIZENROLE. If adding additional custom methods to citizen account, additional permissions will be required.

For more information about adding additional custom methods to citizen account, see *Customizing the citizen account*.

If only a subset of the functionality offered by IBM Cúram Universal Access is being used, permission to invoke the unused methods should be removed from the database. For example, if citizen account is not used, the LINKEDCITIZENROLE and other related authorization artifacts should be removed, as they are not needed. Projects not using citizen account should also consider the deployment implications. For more information, see *Customizing the citizen account*.

Authorization roles should be configured only for the functionality that is being used. It is recommended that unused Security IDentifiers (SIDs) should be removed from the database. For example, if citizen account is not being used, the LINKEDCITIZENROLE and other related authorization artifacts should be removed, as they are not needed. Projects not using citizen account should also consider the deployment implications. For more information, see *Citizen Account Security Considerations*.

Proper use of the authorization roles and groups ensure that no user can access functions for which they have no permission. It will not however, prevent users from using these functions to access data belonging to user users. This is the preserve of Data-based Security. Universal Access provides a framework for Data-based Security and all customizations should use this framework. For more information, see *Citizen Account Security Considerations*.

Related concepts

Customizing the Citizen Account
Users can use the Citizen Account to log in to a secure area where users can screen and apply for programs.

Security and the Citizen Account
Security must be a primary concern when you customize the citizen account customizations. All public-facing applications must be analyzed and tested before they are deployed. Users must contact IBM support to discuss unusual customizations that might have specific security issues.

Configuring user accounts

With the IBM Cúram Universal Access security model, you can configure three types of user account: a generic public account, a system generated account, and a citizen generated account.

Generic public account
Use a generic public account when citizens first access the organization **Home** page.

System generated account
After citizens navigate from the **Home** page, the system generates a temporary user account and automatically logs them on by using the generated credentials. Use this system generated account to secure any data that the citizen enters before the citizen either creates their own user account or logs in to an existing account. After the citizen logs in, ownership of the data is transferred to the citizen's account.

Citizen created account
Citizens can either create a user account or log in to an existing account. After either option is selected, the citizen's data is secured under this account. This account might be linked to a participant on the system of record that provides the citizen with access to all information that is accessible from the Citizen Account. For more information about creating an account, see the *Creating a Citizen*

Account related link. For more information about authentication, see the *logging in to a Citizen Account* related link.

A number of configuration settings apply to user name and password. The following parameters can be defined:

- User name and password length.
- The number of special characters of which a password must consist.
- The minimum and maximum password length.
- The maximum number of login attempts before an account is locked out.
- The period after which citizens must change their password. This configuration can be specified in days or can use a date.

The terms and conditions URL also can be defined.

Related concepts

Creating a citizen account and logging in

Citizens can create a citizen account during the application processes.

Integrating external security

By default, IBM Cúram Universal Access uses its own authentication system that is backed up by a database of registered users. Universal Access can be configured to integrate with external security systems.

As government agencies increasingly provide online services, there is a drive to ensure that citizens can be authenticated for any of these services by using a single set of credentials. This approach provides benefits for the government in streamlining the authentication process and also for the citizen because citizens do not have to remember user names and passwords.

This process, in turn, increases security for the following reasons:

- It makes it less likely that citizens write down their user names and passwords.
- It focuses security efforts on implementing best practice in a single enterprise security system.

Universal Access can be deployed in *Identity Only* mode for registered users so that creating accounts occurs externally and user accounts defer externally for authentication.

When citizens can access government services online, downloading sensitive personal information on a public computer is a security risk as the downloaded files are cached by the browser. To notify citizens of the security risks, a warning message is displayed before citizens download the PDF. The warning message is configurable in system administration.

The system also displays a citizen timeout message when the application is left idle for a specific period. The time after which the timeout message is displayed is configurable. The citizen can either continue to use the application or quit the application by responding to the timeout message. If the citizen does not respond, the system automatically logs the citizen out of the session.

Customizing account creation and management

You can customize account creation and management.

Account management events

Events are raised at key points during account processing. The events can be used to add custom validations to the account management process.

For more information about adding custom validations to the account management process, see the *Cúram Server Developer* section. The following table shows the events that are in the `curam.citizenworkspace.security.impl.CitizenWorkspaceAccountEvents` class:

Table 13: Account events	
Event Interface	**Description**
`CitizenWorkspaceCreateAccountEvents`	Events raised around account creation. For more information, see the related Javadoc information in the WorkspaceServices component.
`CitizenWorkspacePasswordChangedEvent`	Event raised when a user is changing their password. For more information, see the related Javadoc information in the WorkspaceServices component.
`CitizenWorkspaceAccountAssociations`	Events raised when a user is linked or unlinked from an associated Person Participant. For more information, see the related Javadoc information in the WorkspaceServices component.

Related information

Cúram Server Developer

PasswordReuseStrategy API

Use the `curam.citizenworkspace.security.impl.PasswordReuseStrategy` API to add your own password change validations.

As part of the password reset function, there is a default validation that prevents a user from entering a new password that is the same as the user's current password. Using the `PasswordReuseStrategy` API, custom validations can be added to restrict users from changing their passwords to current or previous values if required. For example, a customer might want to implement a password reuse strategy that prevents users from reusing a previous password until after six password changes.

For further details, see the API Javadoc.

CitizenWorkspaceAccountManager API

Use the `curam.citizenworkspace.security.impl.CitizenWorkspaceAccountManager` API to create and link citizen accounts. Use the API to build out custom functionality to support caseworkers who want to link accounts and create accounts on behalf of the citizen.

The API offers the following methods:

- Creating standard accounts
- Creating linked accounts
- Removing links between participants and accounts.
- Retrieving account information

For more information, see the API Javadoc.

Data caching

Minimize the risk of citizens accessing each others' data from browser and server data caches. Cached data can be accessed when citizens use the browser back button or browser history to retrieve data entered by other users, or when PDF files are cached locally on the computer that was used to make the application.

Server caching

HTTP servers like Apache can set cache-control response headers to not store a cache. Use this approach to prevent access to data using the browser back button or history.

Browser caching

Browsers can be configured not to cache content. If citizens can access the web portal in a "kiosk", then the browser should be configured never to cache content.

Advise citizens to clear their cache and close all browser windows they have used when they are finished using the web portal. Also tell citizens to remove PDF documents that they download from the browser's temporary internet files.

External security authentication example

Ensure that citizens can be authenticated for any of your services by using a single set of credentials, which provides the benefits of a streamlined authorization process for both governments and citizens. An example outlines the implementation of a set of customization requirements for a team that is deploying Universal Access.

Universal Access uses its own authentication system that is backed up in a database of registered users. You can also configure the system to integrate with external security systems. You can improve security by enabling the use of a single set of credentials, because citizens do not have to remember lists of user names and passwords and, hence, are less likely to write down their user names and passwords. Also, security efforts are focused on implementing best practice in a single Enterprise Security System.

Consider an example analysis of requirements to integrate with an external security system. Any analysis of requirements for external security integration should consider the following minimum questions:

- Does your deployment support anonymous screening, anonymous intake, or both?
- Is account management supported in IBM Cúram Universal Access or in the external security system?
- Is single sign-on (SSO) required?

Example customization requirements

The topics in this section describe the configuration and development tasks to implement the following set of customization requirements for a team that is deploying Universal Access. The topics refer to the requirements as appropriate.

1. Users can access Universal Access and perform anonymous screening or intake.
2. Users who want to access their saved screening or intake information must first create an account on a system called `CentralID`.
3. Users who log in to Universal Access can use their CentralID username and password to authenticate.
4. Users perform all of their account management using an external system that is named CentralID, for example, resetting a password, creating a new account, changing account details.
5. CentralID stores all user records in a secure LDAP server.
6. Because all account management is now performed in CentralID, the account creation screens and password reset screens are to be removed from Universal Access.
7. Users should be able to log in as soon as they have registered with CentralID, and there should be no delay while waiting for an ID to propagate to Universal Access.

Configuring an alternative login ID

By default, you cannot change user names cannot be changed after they are created. However, you can configure an alternative login ID that can be updated.

For information about configuring alternative login IDs, see the related links. If you configure an alternative login ID for a user name that is case-sensitive, then the alternative login ID is also case-sensitive.

Related information

Cúram Regionalization Guide
Alternate Login IDs
Configuration of the Apache log4j Java-based logging utility

Deploying in identity-only mode for registered users

You must configure the application server to use LDAP for authentication if a user is in `Identity-Only` mode. Also, configure the necessary properties to deploy in identity-only mode for registered users.

Configuring the application server to use LDAP for authentication in `Identity–Only` mode

If a user is in `Identity–Only` mode, it is necessary to match the login IDs that are stored in LDAP with the login IDs that are stored in the `ExtendedUsersInfo` table.

For information about how to configure your application server to use LDAP for authentication, see the relevant application server documentation.

Configuring properties to deploy in identity-only mode for registered users

Add the following properties to the *`AppServer.properties`* file:

```
curam.security.check.identity.only=true
curam.security.user.registry.disabled.types=EXT_AUTO,EXT_GEN
curam.citizenworkspace.enable.usertypes.for.temporary.users=true
public.user.type=EXT_AUTO
```

To reconfigure the application server, run the following command:

```
appbuild configure
```

The `curam.security.check.identity.only` property ensures that application security is set to work in Identity Only mode. For more information about Identity Only authentication mode, see either *Deployment Guide for WebSphere* or *Deployment Guide for WLS*. In Identity Only mode, authentication uses only the internal user table to check for the existence of the user. The validation of the password is left to a subsequent module, either a JAAS module (Oracle WebLogic) or the User Registry (IBM WebSphere).

Take the example of a user, "johnsmith", who has been registered with the CentralID LDAP server. For John Smith to be able to use Universal Access, there must also be a "johnsmith" entry in the ExternalUser table. When John Smith logs in, his authentication request is passed to the Cúram JAAS Login Module. The Cúram JAAS Login Module checks that the user `johnsmith` exists in the Cúram ExternalUser table but does not check the password. The authentication then proceeds to the User Registry (WebSphere) or LDAP JAAS Module (WebLogic) where the user name and password are checked against the contents of the CentralID LDAP server. For the authentication to work correctly, it is necessary to configure the application server with the connection details for the secure LDAP server.

The Identity Only configuration allows the application to defer to an external security system such as an LDAP-based directory service for the authentication of user credentials. However, when an anonymous user accesses the organization **Home** page for the first time, the user is automatically logged in as a `publiccitizen` user. Subsequently, if the user chooses to screen themselves or to perform an intake, Universal Access creates a new "generated" anonymous user. Each generated user is unique, which ensures that the data that belongs to that user is kept confidential. Public citizen users and generated users are not inserted into the LDAP directory, so they cannot be authenticated by using the Identity Only mechanism. The following line ensures that users with the user type EXT_AUTO (public citizen users) and EXT_GEN (generated users) are authenticated against the External User table:

```
curam.security.user.registry.disabled.types=EXT_AUTO,EXT_GEN
```

After the previous configuration has been applied to the server and the server has been started, perform the following configuration steps:

1. Log in as sysadmin.
2. **Select Application Data** > **Property Administration**.
3. Select category **Citizen Account - Configuration**.
4. Set the property **curam.citizenaccount.public.included.user** to EXT_AUTO.
5. Set the property **curam.citizenaccount.anonymous.included.user** to EXT_GEN.
6. Set the property **curam.citizenworkspace.enable.usertypes.for.temporary.users** to TRUE.

7. **Publish** the property changes.

   You need another configuration entry so that Universal Access operates correctly with respect to authentication as shown in the following steps:
8. Select **Select Application Data** > **Property Administration**.
9. Select category **Infrastructure – Security parameters**.
10. Set **curam.custom.externalaccess.implementation** to **curam.citizenworkspace.security.impl.CitizenWorkspacePublicAccessSecurity**.
11. **Publish** the property changes.
12. Log out and restart the server.

Disabling the Create Account screens

Configure the necessary properties to disable the screens for creating an account that Universal Access provides by default. Requirement 4 in the example requirements indicates that all account management functions are handled by the external system, CentralID, including the creation of a new account and performing a password reset.

Configure Universal Access to disable the screens that are related to account management:

1. Log in as sysadmin.
2. Select **Application Data** > **Property Administration**.
3. Select Category **Citizen Portal - Configuration**.
4. Set the property *curam.citizenworkspace.enable.account.creation* to **NO**.
5. **Publish** the property changes.

The previous steps remove references to **Account Creation** pages from Universal Access. The Login screen still contains a link to a page for changing passwords. In this example, the implementation team can use the following steps to retain the link but change it to open a new browser window on the CentralID password reset page:

1. Log in as sysadmin.
2. Select **Application Data** > **Property Administration**.
3. Select Category **Citizen Portal - Configuration**.
4. Set the property *curam.citizenworkspace.forgot.password.url* to , for example **http:// www.centralid.gov/resetpassword**
5. **Publish** the property changes.

To completely remove the reset password link, use the following steps:

1. Log in as sysadmin.
2. Select **Application Data** > **Property Administration**.
3. Select Category **Citizen Portal - Configuration**.
4. Set the property *curam.citizenworkspace.display.forgot.password.link* to **NO**.
5. **Publish** the property changes.

Redirecting users to register with an external system

Replace the message that is displayed in the log in page so that non-registered users are directed to the CentralID page for registration.

Universal Access invites users to log in with a log in message. You can replace the message so that the log in page displays a message that is similar to the following example:

```
"<p>If you are registered with CentralID enter your user name
   and password to log in. To register, go to
   <a href="http://www.centralid.gov/register"> The CentralID
   registration page.</a></p>"
```

The properties for controlling the login page message are contained in the <CURAM_DIR>/EJBServer/components/Data_Manager/Initial_Data/blob/prop/Logon.properties file.

To customize the message that is displayed, follow the procedure in *Customizable IBM Cúram Universal Access Page Content*.

Enabling users to log on immediately after registration with CentralID

Users should be able to log in as soon as they have registered with CentralID. Some configuration is required to prevent a delay in the propagation of a user's ID to other systems.

To function correctly, each user must have an entry in the ExternalUser table. The customer could build a batch process to import users from the LDAP directory into the ExternalUser table. However, requirement 7 in the example requirements would not be satisfied, which states that users must be able to register with CentralID, and then immediately use Universal Access. Another option would be to build a web service or similar mechanism that would be launched when a new user registers with CentralID. The implementation of the web service would create the appropriate entry in the ExternalUser table.

A simpler option is to override the default log-in behavior to create new accounts as needed, after the completion of checks to ensure that the relevant entry exists in the LDAP server. You can override the default log-in behavior in Universal Access by extending the `curam.citizenworkspace.security.impl.AuthenticateWithPasswordStrategy` class and overriding the `authenticate()` method. The following code outlines how to use the `AuthenticateWithPasswordStrategy` and other security APIs to meet the previous requirements:

```
public class CustomSecurityStrategy extends AuthenticateWithPasswordStrategy {
  @Inject
  private CitizenWorkspaceAccountManager cwAccountManager;
  ...
  @Override
  public String authenticate(final String username,
      final String password)
      throws AppException, InformationalException {
    final String retval = null;
    if (username.equals(PUBLIC_CITIZEN)) {
      return super.authenticate(username, password);
    }
    // Authenticate generated accounts as normal
    if (cwAccountManager.isGeneratedAccount(username)) {
      return super.authenticate(username, password);
    }
    // Check that the user exists in LDAP
    // This prevents hackers from registering many bogus
    // accounts that exist in Curam but not in LDAP
    if (!isUserInLDAP(username)) {
      return SECURITYSTATUS.BADUSER;
    }
    // If there's no account for this user
    if (!cwAccountManager.hasAccount(username)) {
      createUserAccount(username);
    }
    return SECURITYSTATUS.LOGIN;
  }
  private void createUserAccount(final String username)
      throws AppException, InformationalException {
    final CreateAccountDetails newAcctDetails;
    ...
    cwAccountManager.createStandardAccount(newAcctDetails);
  }
}
```

This code checks to see whether the user is logging in is a public citizen user or a generated account. In both cases, authentication logic is delegated to the default `AuthenticateWithPasswordStrategy` API. In the case of a registered user, the Strategy checks the LDAP directory to ensure that the user exists in the LDAP directory. If the user exists in the LDAP directory and does not exist yet in Universal Access, then a new user account is created. Note, the custom code does not need to authenticate the user against LDAP since the authentication is handled by the User Registry in WebSphere or the LDAP JAAS Module in WebSphere. It is important to note that the password parameter of the `authenticate()` method is passed in clear text.

To install the `CustomSecurityStrategy` class, it must be bound in place of the Default Security Strategy class. Use a Guice Module to bind the implementation:

```
public class CustomModule extends AbstractModule {
  @Override
  protected void configure() {
    binder().bind(SecurityStrategy.class).to(
      CustomSecurityStrategy.class);
  }
}
```

You must configure the CustomModule at startup by adding a DMX file to the custom component as shown in the following example:

```
<CURAM_DIR>/EJBServer/custom/data/initial/MODULECLASSNAME.dmx

<?xml version="1.0" encoding="UTF-8"?>
<table name="MODULECLASSNAME">
    <column name="moduleClassName" type="text" />
    <row>
      <attribute name="moduleClassName">
        <value>gov.myorg.CustomModule</value>
      </attribute>
    </row>
</table>
```

Configuring single sign-on

Single sign-on (SSO) authentication enables users to access multiple secure applications by authenticating only once by using a single user name and password. Federated single sign-on that uses a SAML 2.0 IdP-initiated POST binding can be implemented through the Citizen Engagement app.

If a user authenticates to an SSO system, the user is no longer prompted for credentials when the user accesses multiple applications that are configured to work with the SSO system.

SSO systems usually maintain the user accounts on an LDAP (lightweight directory application protocol) server. If user accounts are stored at one location, it is easier for system administrators to safeguard the accounts. Also, when necessary, it is easier for users to reset their account passwords at one location instead of at multiple applications.

The following topics discuss the scenario where IBM Cúram Social Program Management is deployed on WebSphere. However, a similar process applies if IBM Cúram Social Program Management is deployed on another supported application server, such as Oracle Weblogic.

Related information

Oracle: Configuring SAML 2.0 Services

SAML web single sign-on profile initiation

An unauthenticated user can initiate a SAML web single sign-on (SSO) profile either through a service provider (SDP), or through an identity provider (IdP).

SP initiation

When an unauthenticated user first accesses an application through an SP, the SP directs the user's browser to the IdP to authenticate. To be SAML specification compliant, the flow requires the generation of a SAML `AuthnRequest` from the SP to the IdP. The IdP receives the `AuthnRequest`, validates that the request has come from a registered SP, and then authenticates the user. After the user has been authenticated, the IdP directs the browser to the Assertion Consumer Service (ACS) application that is specified in the `AuthnRequest` that was received from the SP.

IdP initiation

The IdP can send the assertion request to the service provider ACS in one of two ways:

- The IdP sends a URL link in a response to a successful authentication request. The user must click on the URL link to post the SAML response to the service provider ACS.
- The IdP sends an auto-submit form to the browser that automatically posts the SAML response to the service provider ACS.

The ACS validates the assertion and creates a JAAS subject, and then redirects the user to the SP resource, as shown in the following figure.

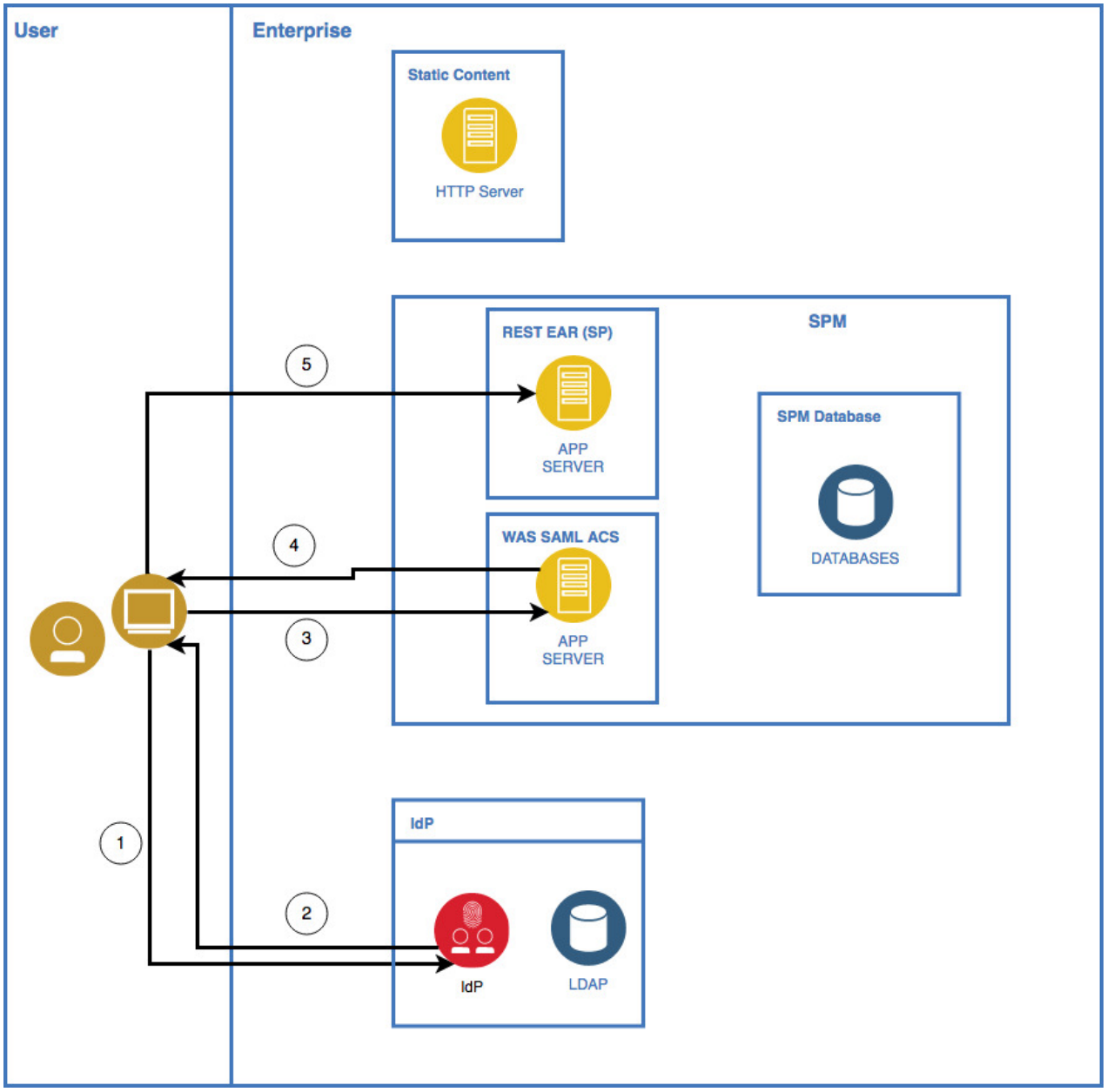


Figure 1: IdP initiated flow

Assertions and the SAML Response document

In all SAML web SSO profile flows, the binding defines the mechanism that is used to send information through assertions between the IdP and the SP. WebSphere supports HTTP POST binding for sending web SSO profiles. The browser sends an HTTP POST request, whose POST body contains a SAML response document. The SAML Response document is an XML document that contains information that includes the following items:

- – The logged in user's identity, which includes the user name, password, address and role, and how the user authenticated, and so on
  – The time period for which the assertion is valid
  – The identify provider that sent the assertion

- What audiences the assertion is meant to be valid for
- Group and role information about the user
- Other application-specific assertions and attributes that are related to the user

A simple assertion typically includes only the first three items from the previous list. To prove the authenticity of the information, the assertion is almost always digitally signed. To protect the confidentiality of parts of the assertion, the payload can be digitally encrypted.

The inclusion of extra attributes and assertions is limitless, which can cause the inclusion of a large amount of raw data in the document. The use of the XML format increases the size of the XML document through the inclusion of XML element tags, attribute names, and namespaces. When an assertion is digitally signed, the XML document includes the key information that is necessary for the receiver to validate the signature. If an assertion is encrypted, the XML document includes the public certificate that is used to encrypt the data.

The SAML Response XML document is then deflated and Base64 encoded, which further increases the size of the data. A typical SAML Response contains information that can be sent only through a login by a POST parameter. After login, an alternative mechanism is typically used to maintain the logged-in security context. Most systems use some cookie-based, server-specific mechanism, such as a specific security cookie, or the server's cookie tied to the user's HTTP session.

Related information

Oasis: SAML 2.0 Technical Overview

Oracle: JAAS Authorization Tutorial

The SAML 2.0 single sign-on flow in IBM Cúram Universal Access

To implement a more seamless SAML SSO flow,Universal Access only supports an identity provider (IdP) initiated web SSO flow. The SAML POSTs are controlled through the logic in Universal Access.

Browser-based single sign-on (SSO) through SAML v2.0 works well with many web applications where the SAML flow is controlled by HTTP redirects between the identity provider (IDP) and the service provider (SP). The user is guided seamlessly from login screens to SP landing pages by HTTP redirects and hidden forms that use the browser to POST received information to either the IdP or the SP.

In a single page application, all the screens are contained within the application and dynamic content is expected to be passed only in JSON messages through XMLHttpRequests. Therefore, the rendering of HTML content for login pages and the automatic posting of hidden forms in HTML content is more difficult. If the SP processes the content in the same way, it would be necessary to leave the application and hand back control to either the user agent or the browser, in which case the application state would be lost.

Therefore, Universal Access supports only an IdP initiated web SSO flow. Any attempt to connect to a protected resource without first authenticating through IdP results in a 403 HTTP response from IBM Cúram Social Program Management web API. Therefore, an authentication request that is initiated through SP will results in a 403 HTTP response, and the application will then redirect the user to the login page that is contained in Universal Access.

The following figure illustrates the IdP initiated flow that is supported by Universal Access in a default installation.

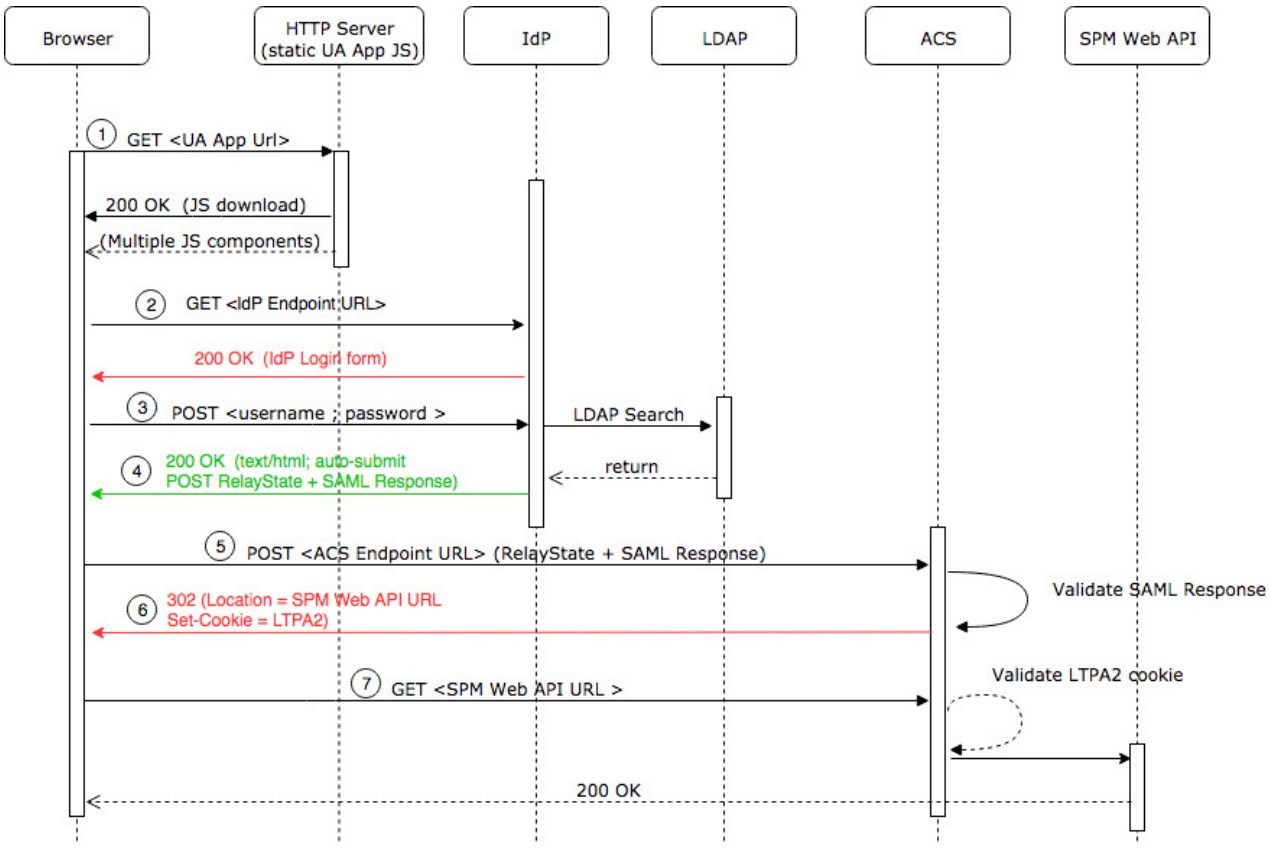


Figure 2: IdP initiated flow in IBM Cúram Universal Access

The following list references the numbered items in the image and provides a description of each step in the flow.

1. A user browses to the HTTP server that contains Universal Access.
2. The user can browse as normal by interacting with IBM Cúram Social Program Management as either a public or a generated user (which is not shown in the diagram). The user then opens the login page to access protected content, which triggers an initial request to the IdP endpoint. In most IdP configurations, an HTML login form responds to the request. Universal Access ignores the response.
3. To authenticate, the user completes the login form and clicks **Submit**. The form submission triggers an HTTP POST request that contains login credentials to the IdP.
4. After successful validation of the user credentials at the IdP, the IdP populates the SAML Response and returns it in an HTML form that contains hidden input fields. Several redirects might occur before the 200 OK HTTP response that contains the SAML information is received. Universal Access does not respond to the redirects.
5. Universal Access extracts the `RelayState` and `SAMLResponse` values, and inserts them in a new POST request to the application server Assertion Consumer Service (ACS).
6. The application server ACS validates the signature that is contained in the SAML Response. WebSphere Application Server also ensures that the originator is a Trusted Authentication Realm. If the validation is successful, the ACS sends an HTTP redirect that points to the configured IBM Cúram Social Program Management target landing page, along with an LTPA2 Cookie that will be used in any subsequent communication. The browser automatically sends a new request to the target URL, but Universal Access does not respond to the request.
7. Universal Access begins its standard user setup by requesting account and profile information from the relevant web API endpoints.

Configuring single sign-on properties

To enable IBM Cúram Universal Access to work with SAML single sign-on (SSO), configure the appropriate properties in the `.env` environment variable file that is in the root of the starter pack. Then, rebuild Universal Access.

Procedure

- Edit the following properties:

  REACT_APP_SSO_ENABLED
  This property takes a Boolean value. To enable SO authentication in Universal Access, set the value to `true`. If the value is false, SSO is disabled.

  REACT_APP_IDP_LOGIN_URL
  This property value specifies the identity provider (IdP) login page URL, for example:

  ```
  https://192.168.0.1:12443/pkmslogin.form
  ```

  REACT_APP_ACS_URL
  This property value specifies the Assertion Consumer Service (ACS) application server URL, for example:

  ```
  https:// 192.168.0.2:9443/samlsps/acs
  ```

  REACT_APP_SAML_INITIAL
  This property value specifies the SSO SAML initial request URL as defined by the IdP, for example:

  ```
  REACT_APP_SAML_INITIAL=https://192.168.0.1:12443/isam/sps/saml20idp/saml20/logininitial?
    RequestBinding=HTTPPost&PartnerId=https://192.168.0.2:9443/samlsps/
  acs&NameIdFormat=Email)
  ```

Related information

Cúram REST configuration properties

Configuring cross-origin resource sharing

For security reasons, browsers restrict cross-origin HTTP requests, including XMLHttpRequest HTTP requests, that are initiated inside IBM Cúram Universal Access. When the Universal Access application and the Universal Access web API are deployed on different hosts, extra configuration is required.

About this task

Universal Access can request HTTP resources only from the same domain that the application was loaded from, which is the domain that contains the static JavaScript. To enable Universal Access to support cross-origin resource sharing (CORS), enable the use of CORS headers.

Procedure

1. Log on to the IBM Cúram Social Program Management application as a system administrator, and click **System Configurations**.
2. In the Shortcuts panel, click **Application Data** > **Property Administration**.
3. Configure the curam.rest.allowedOrigins property with the values of either the host names or the IP addresses of the IdP server and the web server on which Universal Access is deployed.

Single sign-on configuration example

The example outlines a single sign-on (SSO) configuration for IBM Cúram Universal Access that uses IBM Security Access Manager to implement federated single sign-on by using the SAML 2.0 Browser POST profile.

Universal Access SSO configuration components

The following figure shows the components that are included in a Universal Access SSO configuration.

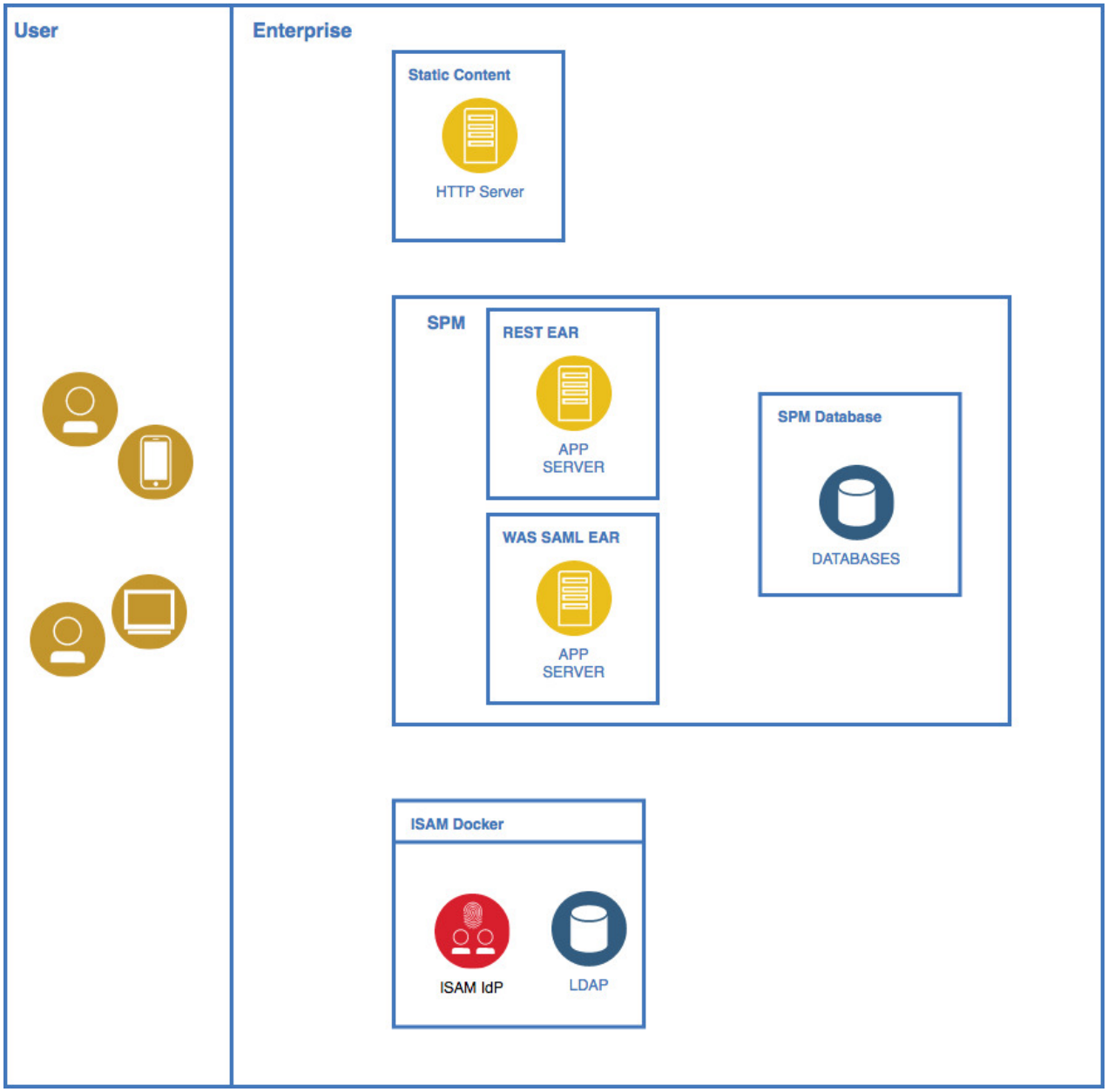


Figure 3: Universal Access SSO configuration components

Web browser
A user sends requests from the web browser for applications in the SSO environment.

Web server
Universal Access ReactJS static content that is deployed on a web server, for example, IBM HTTP Server, Apache.

IBM Security Access Manager server
The IBM Security Access Manager server includes the identity provider (IdP).

OpenLDAP server (user directory)
Among other items, OpenLDAP server contains the user name and password of all the valid users in the SSO environment.

IBM WebSphere Application Server
Among other applications, WebSphere Application Server contains deployed IBM Cúram Social Program Management, Citizen WorkSpace, and REST enterprise applications.

WebSphere SAML EAR
WebSphere package that contains the packages to run the SAML Assertion Consumer Service (ACS).

Database
Storage for the IBM Cúram Social Program Management, Citizen WorkSpace, and REST enterprise applications.

Configuring single sign-on through IBM Security Access Manager

Use the IBM Security Access Manager management console to configure single sign-on (SSO) in IBM Cúram Universal Access.

Before you begin

1. Start IBM Security Access Manager.
2. In the management console, log on as an administrator.
3. Accept the services agreement.
4. If required, change the administrative password.

About this task

In the IBM Security Access Manager management console, complete the steps that are outlined in the following procedure:

Procedure

1. Configure the IBM Security Access Manager database:
   a) In the top menu, click **Home Appliance Dashboard** > **Database Configuration**.
   b) Enter the database configuration details, such as **Database Type**, **Address**, **Port**, and so on, and click **Save**.
   c) When the **Deploy Pending Changes** window opens, click **Deploy**.
2. To install all the required product licenses, repeat the following steps for each activation code, where each activation code corresponds to a product license:
   a) In the IBM Security Access Manager management console, click **Manage System Settings** > **Licensing and Activation**.
   b) To import the licenses for IBM Security Access Manager and the federation add-on, click **Import**.
3. Configure the LDAP SSL database:
   a) In the IBM Security Access Manager management console, click **Manage System Settings** > **SSL Certificates**.
   b) Click **New** and create an `Ex: ldap` entry.
   c) Select the new **Ex: ldap** entry in the list.
   d) Click **Manage** > **Edit SSL Certificate Database**.
   e) In the **Edit SSL Certificate Database** window, click **Manage** > **Load**.
   f) In the window that opens, enter the LDAP server host name or IP address, the port number, and a name.
   g) Click load to retrieve the signer certificate from the LDAP server.
   The retrieved signed certificate is displayed in the list.
   h) Close the window.
   i) Select the option to deploy the pending changes.
4. Configure the runtime component:

a) In the IBM Security Access Manager management console, click **Secure Web Settings** > **Runtime Component**.

b) Click **Configure to display Runtime Environment Configuration popup**.

c) Click the **Main** tab, then select **LDAP Remote for remote LDAP registry** and click **Next**.

d) For **Management Domain**, select the default value, and enter the relevant data in the remaining fields.

In IBM Security Access Manager Policy Server, you can retain the default value for `Management Domain`.

e) In the **LDAP** tab, enter the following values:

Hostname
`LDAP server host name or IP address`

Port
`LDAP server host name or IP address`

DN
`cn=root,secAuthority=Default`

Password
`LDAP password`

Enable SSL check box
Select the **Enable SSL** check box.

Certificate Database
From the list, select the LDAP SSL database that you created previously, `Ex: ldap`.

f) Click **Finish**.

Configuring IBM Security Access Manager as an IdP

To configure IBM Security Access Manager as an identity provider (IdP), see the IBM Security Access Manager 9.0 Federation Cookbook that is available from IBM Developer Works.

Before you begin

Download the IBM Security Access Manager 9.0 Federation Cookbook from IBM Developer Works, as shown in the related link. Also download the mapping files that are provided with the cookbook.

About this task

To set up the example environment are, complete the specified sections in the IBM Security Access Manager 9.0 Federation Cookbook in order.

Procedure

1. Complete *Section 5, Create Reverse Proxy instance*.
2. Complete *Section 6, Create SAML 2.0 Identity Provider federation*.

   In Section 6.1, if you are using the ISAM docker deployment, it is possible to re-use the existing keystore that is included in the container instead of creating a new keystore. It is important to reflect this change in subsequent sections where the myidpkeys certificate database is referenced.
3. Complete *Section 8.1, ISAM Configuration for the IdP*.

   In Section 8.1, use the host name of the IdP federation.
4. Optional: After completing Section 8.1.1, if you require ACLs to be defined to allow and restrict access to the IdP junction, then follow the instructions in *Section 25.1.3, Configure ACL policy for IdP*.
5. Complete *Section 9.1, Configuring Partner for the IdP*.

   The export from Websphere does not contain all the relevant data. Therefore, in Section 9.1, after you complete configuring partner for the IdP, you must click **Edit configuration** and complete the remaining advanced configuration.

Related information

IBM Security Access Manager 9.0 Federation Cookbook

Configuring WebSphere Application Server

The procedure outlines the high-level steps that are required to configure IBM WebSphere Application Server as a SAML service provider.

About this task

For more information, see the related link to the WebSphere Application Server documentation.

Procedure

1. Deploy the `WebSphereSamlSP.ear` file.

   The `WebSphereSamlSP.ear` file is available as an installable package. Choose one of the following methods:

   - Log on to the WebSphere Application Server administrative console, and install the `app_server_root/installableApps/WebSphereSamlSP.ear` file to your application server or cluster.
   - Install the SAML Assertion Consumer Service (ACS) application by using a Python script. In the `app_server_root/bin` directory, enter the following command to run the `installSamlACS.py` script:

     ```
     wsadmin -f installSamlACS.py install nodeName serverName
     ```

     In the previous command, *nodeName* is the node name of the target application server, and *serverName* is the server name of the target application server.
2. Configure the ACS trust association interceptor:
   a) In the WebSphere Application Server administrative console, click **Global security** > **Trust association** > **Interceptors** > **New**.
   b) For **Interceptor class name**, enter `com.ibm.ws.security.web.saml.ACSTrustAssociationInterceptor`.
   c) Under custom properties, enter the values that are shown in the following table:

   In a standard WebSphere Application Server configuration, you would also define a value for the `login.error.page` custom property. However, the preferred method is to log onto the IdP first. Therefore, if you do not define a value for `login.error.page`, WebSphere Application Server returns a 403 error if a user logs on without first logging onto the identity provider (IdP).

Table 14: ACS trust association interceptor custom properties

Custom property name	**Value**
sso_1.sp.acsUrl	https://*WAS_host_name:ssl port*//samlsps/acs
sso_1.idp_1.EntityID	https://*isam_hostname:isam_port*//*URL of ISAM*/*ISAM Junction*/*IdP endpoint*/*federation name*/saml20
sso_1.idp_1.SingleSignOnUrl	https:// *isam_hostname:isam_port*//*URL of ISAM*/*ISAM Junction*/*IdP endpoint*/*federation name*/saml20/login
sso_1.sp.targetUrl	https://*WAS_host_name*:*WAS_port*/Rest
sso_1.idp_1.certAlias	isam-conf
sso_1.sp.filter	request-url^=/Rest;request-url!=/Rest/ j_security_check

Table 14: ACS trust association interceptor custom properties (continued)	
Custom property name	**Value**
sso_1.sp.enforceTaiCookie	false

3. Add the IdP federation partner data. The following substeps describe how to add the IdP data by using the WebSphere Application Server administrative console.
   a) To add the IdP host name or IP address as a trusted realm, click **Global security** > **Trusted authentication realms - inbound** > **Add External Realm**.
   b) Enter either the IBM Security Access Manager host name or IP address.
   c) To load the IdP certificate from IBM Security Access Manager, click **Security** > **SSL certificate and key management** > **Key stores and certificates** > **NodeDefaultTrustStore** > **Signer certificates** > **Retrieve from port**
   d) Enter the IBM Security Access Manager IP address and listener port, for example, `12443, alias = isam-conf`.

      Note: When the browser first attempts to connect to the IBM Cúram Social Program Management web API, an LTPA2 cookie is sent as part of the request. If the WebSphere Application Server com.ibm.ws.security.web.logoutOnHTTPSessionExpire property is set to true, which is the default configuration in IBM Cúram Social Program Management, then authentication fails because an HTTP session does not exist on the application server. By setting the property to `false`, the check for a valid HTTP session is not completed and when the LTPA2 token is valid, authentication succeeds.

      To configure the property in the WebSphere Application Serveradministrative console, click **Security** > **Global security** > **Custom properties**, and set the value of com.ibm.ws.security.web.logoutOnHTTPSessionExpire to `false`.
4. Implement cross-origin resource sharing (CORS) from the HTTP server to the WebSphere Application Server SAML ACS.
   a) To add a CORS header, configure a servlet filter for the `WebSphereSamlSP.ear` file that is deployed by a Trust Association Interceptor (TAI). The servlet filter adds a CORS HTTP header to HTTP responses. You can archive the implemented servlet filter as a `jar` file, and then store it in the `WebSphereSamlSP.ear\WebSphereSamlSPWeb.war\WEB-INF\lib` directory that is in the `installedApps` directory of your project in WebSphere Application Server.
      See the following example of how to implement a servlet filter:

```
public class SampleFilter implements Filter {

       @Override
       public void doFilter(ServletRequest arg0, ServletResponse servletResponse,
            FilterChain arg2) throws IOException, ServletException {

HttpServletResponse response = (HttpServletResponse) servletResponse;
HttpServletRequest request = (HttpServletRequest) arg0;


response.setHeader("Access-Control-Allow-Origin",
     "http://dubxpcvm156.mul.ie.ibm.com:9880");     <hostname or IP address of IBM UA
server>
response.setHeader("Access-Control-Allow-Credentials", "true");
response.setHeader("Access-Control-Allow-Headers", "x-requested-with, Content-Type,
origin, authorization, accept, client-security-token");
response.setHeader("Access-Control-Expose-Headers", "content-length");
        arg2.doFilter(request, response);
     }
}
```

   b) Configure the `web.xml` file for the deployed TAI EAR file to use the servlet filter for all the requests. Add the filter element that is shown in the following sample to the `web.xml` file, with the actual fully qualified name of the filter.

      You can add the filter element as a sibling to any existing element n the `web.xml` file, such as `<servlet>`. The `web.xml` file is in the `WebSphereSamlSP.ear\WebSphereSamlSPWeb.war`

\WEB-INF\lib directory, which is in the installedApps directory of your project in WebSphere Application Server.

```
<filter>
    <filter-name> SampleFilter </filter-name>
    <filter-class> SampleFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name> SampleFilter</filter-name>
    <url-pattern>/*</url-pattern>
 </filter-mapping>
```

Related information

Enabling WebSphere Application Server to use the SAML web SSO feature

Configuring CORS for IBM Security Access Manager

To permit cross-origin requests from the HTTP server to the IBM Security Access Manager domain, configure the IBM Security Access Manager runtime environment.

Procedure

1. To create LDAP and IBM Security Access Manager runtime users, create an ldif file that can be used to populate OpenLdap, as shown in the following sample:

```
# cat UA_usersCreate_ISAM.ldif
dn: dc=watson-health,secAuthority=Default
objectclass: top
objectclass: domain
dc: watson-health

dn: c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: country
c: ie

dn: o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organization
o: curam

dn: ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organizationalUnit
ou: curamint

dn: cn=caseworker,ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: person
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
cn: caseworker
sn: caseworkersurname
uid: caseworker
mail: caseworker@curam.com
userpassword: Passw0rd

dn: ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organizationalUnit
ou: curamext

dn: cn=jamessmith,ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: person
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
cn: jamessmith
sn: Smith
uid: jamessmith
mail: jamessmith@curamexternal.com
userpassword: Passw0rd
```

2. Add users to the OpenLDAP database:

a) On the host server that is running the docker containers, enter the following command:

```
docker cp UA_usersCreate_ISAM.ldif idpisam9040_isam-ldap_1:/tmp
```

b) To log on to the OpenLDAP container, enter the following command:

```
docker exec –ti idpisam9040_isam-ldap_1 bash
```

c) To add the users to OpenLDAP, enter the following command:

```
ldapadd -H ldaps://127.0.0.1:636 -D cn=root,secAuthority=default -f /tmp/
Curam_usersCreate_ISAM.ldif
```

3. Import the users into IBM Security Access Manager:

a) To log on to the IBM Security Access Manager command line interface, enter the following commands:

```
docker exec -ti idpisam9040_isam-webseal_1 isam_cli
isam_cli> isam admin
pdadmin> login -a sec_master -p <password>
```

b) To import the users into IBM Security Access Manager, enter the following commands:

```
pdadmin sec_master> user import caseworker
cn=caseworker,ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
pdadmin sec_master> user modify caseworker account-valid yes
pdadmin sec_master> user import jamessmith
cn=jamessmith,ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
pdadmin sec_master> user modify jamessmith account-valid yes
```

4. To test the identity provider (IdP) flow, enter the following URL in a browser:

```
https://ISAM login initial URL?RequestBinding=HTTPPost
&PartnerId=webspherehostname:9443/samlsps/acs&NameIdFormat=Email
&Target=WAS hostname:WAS port/Rest/v1
```

Replace the following values in the URL with the appropriate values for your configuration:

- *IBM Security Access Manager login initial URL*
- *WebSphere host name*
- *WebSphere Application Server host name*
- *WebSphere Application Server port*; inIBM Cúram Social Program Management the default value is 9044

When the IBM Security Access Manager docker container starts, the IdP endpoints are initialized only when the first connection request is received. However, if the first connection request is triggered by IBM Cúram Universal Access, an XHR timeout occurs before the initialization finishes. Therefore, this test step is required to ensure that the initialization of the IdP endpoints is completed.

5. In a browser, go to the home page and log in.

Customizing the IBM Cúram Universal Access server

Use this information to customize the Universal Access server. Typical customizable features are security and the citizen account.

Error logging in the citizen account

When a citizen submits an application, when a citizen clicks **Submit** a deferred process starts. If a mapping failure occurs, an error is logged.

Application property

The application property *curam.workspaceservices.application.processing.logging.on* increases the level of detail of error messages.

When *curam.workspaceservices.application.processing.logging.on* is set to *true*, detailed error messages are written to the application log files if the submission process fails.

Error codes

Each error message is prepended with an error code. These error codes help to automatically scan application logs so that unexpected failures can be identified. The error codes that are returned by the application is defined in the code table file `CT_ApplicationProcessingError.ctx`.

The range of codes that are reserved for internal processing is **APROCER001 – APROCER500**. Customers can use the range **APROCER501 – APROCER999** to log errors in custom processing, for example error codes for extension-mapping handler class.

The list of error codes that are returned by the application, and a brief description of the problem, is listed in Table 1.

Table 15: Application error codes

Code	Description
APROCER001	An error occurred creating a person.
APROCER002	An error occurred creating a prospect person.
APROCER003	A relationship error occurred creating a person.
APROCER004	An error occurred creating a case.
APROCER005	An error occurred while performing a "map-attribute" mapping.
APROCER006	An error occurred while performing a "set-attribute" mapping.
APROCER007	An error occurred while performing a "map-address" mapping.
APROCER008	General mapping failure.
APROCER009	Error creating evidence.
APROCER010	More than one PDF form is registered against the program type.
APROCER011	Error setting the alternate id type for a Prospect Person.
APROCER012	Invalid alternate ID value.
APROCER013	Error the Evidence Application Builder has not been correctly configured.
APROCER014	Evidence type not listed in the Mapping Configuration.
APROCER015	No parent evidence entity found.
APROCER016	An error occurred when trying to unmarshal the application XML.

Table 15: Application error codes (continued)	
Code	**Description**
APROCER017	An error occurred when trying to set a field value.
APROCER018	An error occurred when trying to create the PDF document.
APROCER019	An error occurred when trying to create the PDF document. A form code could not be mapped to a codetable description.
APROCER020	An error occurred when trying a WorkspaceServices mapping extension handler.
APROCER021	Missing source attribute in datastore entity.
APROCER022	An attribute in an expression is not valid.
APROCER023	Application builder configuration error.
APROCER024	Failed creating *DataStoreMappingConfig*, no name specified.
APROCER025	Failed creating *DataStoreMappingConfig*, the name is not unique.
APROCER026	The mapping to datastore had to be abandoned because the schema is not registered.
APROCER027	There was a problem parsing the Mapping Specification.
APROCER028	General mapping error. Mapping XML included.
APROCER029	Cannot have multiple primary participants.
APROCER030	No programs have been applied for.
APROCER031	An error occurred while attempting to map to Person data.
APROCER032	An error occurred while attempting to map to Relationship data.
APROCER033	An error occurred while creating Cases.
APROCER034	An error occurred while creating evidence.
APROCER035	No programs have been applied for.
APROCER036	An error occurred reading data from the datastore.
APROCER037	Specified integrated case type does not exist.
APROCER038	Specified case type does not exist
APROCER039	Duplicate SSN entered for prospect person.
APROCER040	Duplicate SSN entered.
APROCER041	There was a problem with the workflow process.
APROCER042	No primary participant has been identified as part of the intake process.

Customizing submitted applications

Use customization points, for example, customizing the generic PDF for processed applications, to customize the application intake process when an intake application is submitted.

Customizing the intake application workflow

View a summary of the intake application workflow in a flowchart.

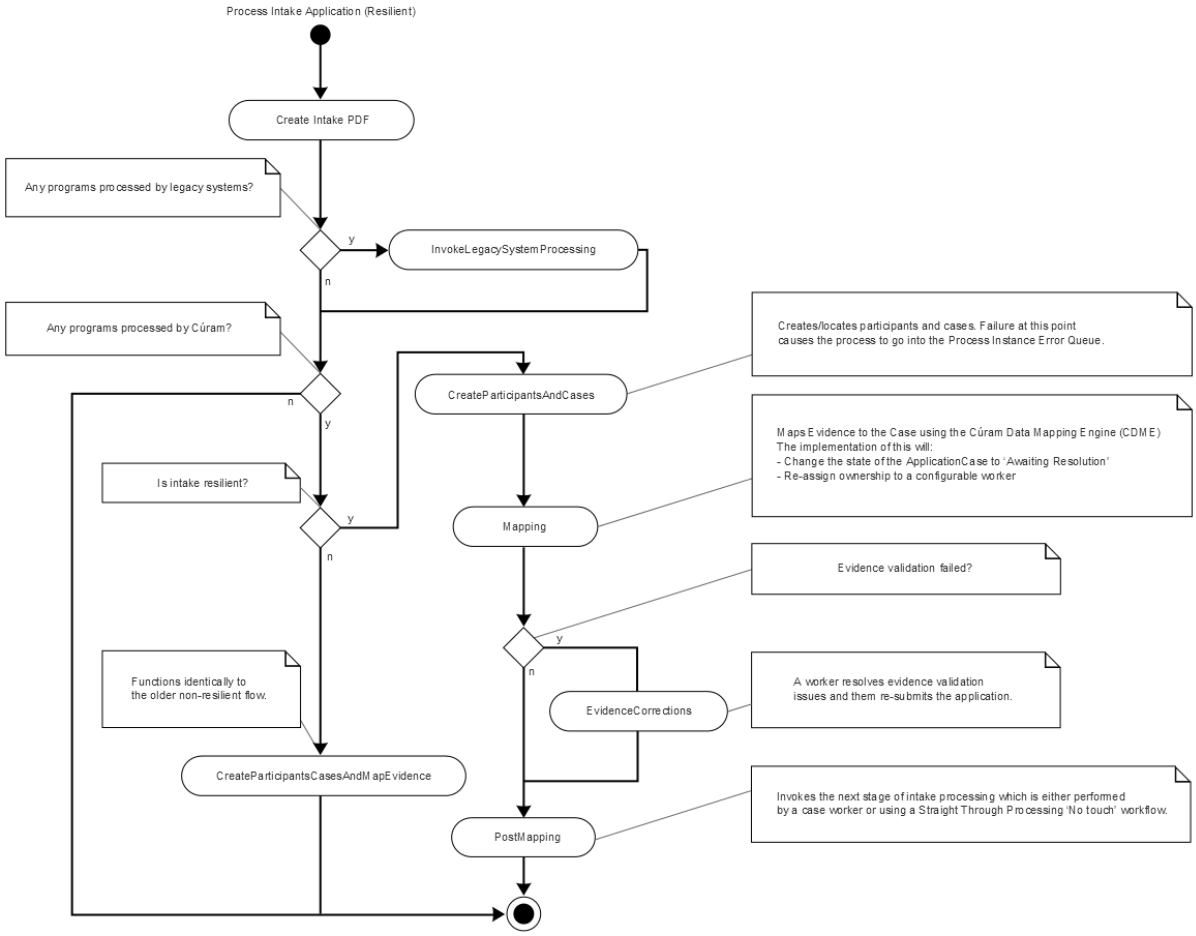


Figure 4: Intake application workflow

Create intake PDF

This automatic activity creates a PDF document based on the content of the application. For more information, see *Customizing the generic PDF for processed applications*.

InvokeLegacySystemProcessing

This automatic activity sends applications to legacy systems via Web Services. This path is taken only if there are legacy systems associated with at least one of the programs on the application.

CreateParticipantsAndCases

This automatic activity creates participants for the submitted application and then creates a case or cases for the various programs associated with the application. In most cases, an Application Case or Cases are created. This path is taken if the value of the configuration property `curam.intake.use.resilience` is set to true. For reasons of backward compatibility, this property is set to false by default, however it is strongly recommended that all production systems set this value to true. For more information on the implications of setting this value to true, see *Using events to extend intake application processing*.

Mapping

This automatic activity uses the Cúram Data Mapping Engine (CDME) to map data collected in the application script into Case Evidence. Under most circumstances this will proceed smoothly. In the

event that a validation issue occurs with the mapped evidence, this activity will be automatically re-tried. During the re-try, if there is a single Application Case, the validations will be disabled and a WDO flag `IntakeCaseDetails.mappingValidInd` set to false.

EvidenceCorrections
This manual task is invoked if the Mapping activity failed due to a validation error (`IntakeCaseDetails.mappingValidInd` set to false). The assignment of this task is configurable. For more information, see *Evidence issues intake strategy*. The caseworker or operator will resolve the evidence validation issues and then re-submit the application.

PostMapping
This automatic activity kicks off the next stage of application processing by invoking the event `IntakeApplication.IntakeApplicationEvents.postMapDataToCuram()`.

CreateParticipantsCasesAndMapEvidence
This path is followed when the configuration property `curam.intake.use.resilience` is set to false. This automatic activity behaves identically to the old, non-resilient workflow. It creates cases and participants and performs all evidence mapping in a single transaction. This makes the process less resilient in the event of a failure.

Customers can customize the workflow in the usual recommended manner as described in the `Cúram Development Compliance Guide` and `Cúram Workflow Management System Guide`. Note that customers should not make any changes to the enactment structs used by these workflows.

Related concepts

Customizing the generic PDF for processed applications
Use IBM Cúram Universal Access to map all intake applications to a generic PDF that records the values of all the information that the user enters.

Using events to extend intake application processing
The interface `IntakeApplication.IntakeApplicationEvents` contains events that are invoked when citizens submit an intake application for processing.

Related information

Evidence Issues Ownership Strategy

Customizing the generic PDF for processed applications

Use IBM Cúram Universal Access to map all intake applications to a generic PDF that records the values of all the information that the user enters.

This PDF is rendered by the XML Server. Customers can override the default formatting of the generic PDF as follows:

1. Copy `CURAM_DIR/EJBServer/components/Workspaceservices/Data_Manager/InitialData/XSLTEMPLATEINST.dmx` to `CURAM_DIR/EJBServer/components/custom/Data_Manager/InitialData`.
2. Edit `project\config\datamanager_config.xml`, replace the entry for:`CURAM_DIR/EJBServer/components/Workspaceservices/Data_Manager/InitialData/XSLTEMPLATEINST.dmx` with an entry for: `CURAM_DIR/EJBServer/components/custom/Data_Manager/InitialData/ XSLTEMPLATEINST.dmx`
3. Copy `CURAM_DIR/EJBServer/components/Workspaceservices/Data_Manager/InitialData/blob/WSXSLTEMPLATEINST001` to: `CURAM_DIR/EJBServer/components/custom/Data_Manager/InitialData/blob.`
4. Edit `WSXSLTEMPLATEINST001` to suit the needs of the project.

Using events to extend intake application processing

The interface `IntakeApplication.IntakeApplicationEvents` contains events that are invoked when citizens submit an intake application for processing.

Use these events to change the way that intake applications are handled, for example supplement or replace the standard CDME mapping or perform an action after an application has been sent to a remote system using web services. For more information, see the API Javadoc information for

`IntakeApplication.IntakeApplicationEvents` in `<CURAM_DIR>/EJBServer/components/WorkspaceServices/doc`.

The interface `IntakeProgramApplication.IntakeProgramApplicationEvents` contains events that are invoked at key stages during the processing of an application for a particular program. For information, see the API Javadoc information for `IntakeProgramApplication.IntakeProgramApplicationEvents` in `<CURAM_DIR>/EJBServer/components/WorkspaceServices/doc`.

Customizing the concern role mapping process

The `curam.workspaceservices.applicationprocessing.impl` package contains a ConcernRoleMappingStrategy API that provides a customization point into the online application process.

Use the ConcernRoleMappingStrategy API to implement custom behavior following the creation of each new concern role that is added to an application. For example, customers who have customized the prospect person entity might want to store information on that entity that cannot be mapped using the default CDME processing.

Enable the ConcernRoleMappingStrategy API

In the administration application, enable the ConcernRoleMappingStrategy API by setting the Enable Custom Concern Role Mapping property to true.

Procedure

1. Log in to the System Administration application as a user with system administration permissions.
2. Click **System Configurations** > **Application Data** > **Property Administration**.
3. In the **Application - Intake Settings** category.
4. Search for the property *curam.intake.enableCustomConcernRoleMapping*.
5. Edit the property to set its value to true.
6. Save the property.
7. Select **Publish**.

Use the ConcernRoleMappingStrategy API

When enabled, use the ConcernRoleMappingStrategy API to implement a strategy for mapping information to a custom concern role.

About this task

The `curam.workspaceservices.applicationprocessing.impl` package contains the ConcernRoleMappingStrategy API.

Procedure

1. Provide an implementation of the customization point.
2. Bind your custom implementation by creating or extending your custom mapping module as follows:

```
package com.myorg.custom;
class MyModule extends AbstractModule {
  @Override
  protected void configure() {

      bind(ConcernRoleMappingStrategy.class).to(
           MyCustomConcernRoleMapping.class);
```

3. If you did not already add your MyModule class to the ModuleClassName table by using an appropriate DMX file, add your MyModule class.

How to send applications to remote systems for processing

Use the Citizen Workspace to send applications to remote systems that use web services for processing.

An event `ReceiveApplicationEvents.receiveApplication` is raised before the application is sent to the remote system. The event can be used to edit the contents of the data store that is used to gather application data before transmission. For more information, refer to the API Javadoc for `ReceiveApplicationEvents`, which is in `<CURAM_DIR>/EJBServer/components/WorkspaceServices/doc`.

Customizing the Citizen Account

Users can use the Citizen Account to log in to a secure area where users can screen and apply for programs.

Users also use the Citizen Account to view information relevant to them, including individually tailored messages, system-wide announcements, updates on their payments, contact information for agency staff and outreach campaigns that might be relevant to them. The Citizen Account also provides a framework for customers to build their own pages or override the existing pages.

Security and the Citizen Account

Security must be a primary concern when you customize the citizen account customizations. All public-facing applications must be analyzed and tested before they are deployed. Users must contact IBM support to discuss unusual customizations that might have specific security issues.

Permission to call the server facade methods that serve data to citizen account pages is managed by the standard authorization model. For more information, see the *Server Developer* documentation. In addition to the standard authorization checks, each facade method that is called by a **Citizen Account** page must complete the following security checks to ensure the user who is associated with the transaction (the currently logged in user) has permission to access the data they are requesting:

- Ensure that the currently logged in user is of the correct type. They must be an external user with an `applicationCode` of `CITWSAPP`, and have an account of type `Linked`.
- Ensure that the currently logged in user has permission to access the specific records that they are reading. For instance, validate any page parameters that are passed in to ensure that the records requested are related to the currently logged in user in some way.

Ensure that the currently logged in user is the correct type

The `curam.citizenaccount.security.impl.CitizenAccountSecurity` API offers a method `performDefaultSecurityChecks` that ensures that the user is the correct type. This method checks the user type, and if not acceptable, writes a message to the logs and fails the transaction.

Note: This API needs to be called in the first line of every custom facade method before any processing or further validation takes place:

```
public CitizenPaymentInstDetailsList listCitizenPayments()
    throws AppException, InformationalException {

    // perform security checks
    citizenAccountSecurity.performDefaultSecurityChecks();

    // validate any page parameters (none in this case)

    // invoke business logic
    return citizenPayments.listPayments();
  }
```

Ensure that the logged in user has access to the requested records

A malicious user who is logged in to a valid linked account might send requests to the system to request other users' data. To prevent this intrusion from happening, all page parameters must be validated to ensure that they are somehow traceable back to the currently logged in user. How this conclusion is determined is different for each type of record.

For example, a **Payment** can be traced back to the **Participant** by way of the **Case** on which it was entered.

The curam.citizenaccount.security.impl.CitizenAccountSecurity application programming interface (API) offers methods to complete these checks for the types of records that are served to citizens by the initially configured pages. For specific information, review the Javadoc of this API. For custom pages that serve different types of data, additional checks must be implemented to validate the page parameters.

This process needs to be added to a custom security API and called by the facade methods in question. The methods need to check to see whether the record requested can be traced back to the currently logged in user, and if not, it needs to log the user name, method name, and other data. If these conditions are not met, the transaction needs to be failed immediately (as opposed to adding the issue to the validation helper and allowing the transaction to proceed):

```
if (paymentInstrument.getConcernRole().getID()
   != citizenWorkspaceAccountManager
     .getLoggedInUserConcernRoleID().getID()) {

 /**
 * the payment instrument passed in is not related
 * to the logged in user log the user name of the
 * current user, the method invoked and any other
 * pertinent data
 */

 // throw a generic message
 throw PUBLICUSERSECURITYExceptionCreator
   .ERR_CITIZEN_WORKSPACE_UNAUTHORISED_METHOD_INVOKATION();
 }
```

While as much information as possible regarding the infraction needs to be logged, it is important to ensure that the exceptions thrown do not display any information that might be useful to malicious users. A generic exception needs to be thrown that does not contain any information that relates to what went wrong. The curam.citizenaccount.security.impl.CitizenAccountSecurity API throws a generic message that states You are not privileged to access this page.

Messages

When a linked citizen logs in, messages are gathered from the system and from remote systems for display.

The curam.citizenmessages.impl.CitizenMessageController API gathers and displays messages. The API reads persisted messages by participant from the ParticipantMessage database table, and also raises the CitizenMessagesEvent.userRequestsMessages event, inviting listeners to add messages to a list it passes as part of the event parameter. The messages that are gathered from each source are sorted, turned into XML and returned to the citizen for display.

Configuring citizen messages

Global configurations are included that can be specified for **Citizen Messages**, such as enabling certain types and configuring their display order. The different types of messages also include their own configuration points. Specific information about how to customize the various message types is provided later.

The textual content of a message type also can be configured. Each message type has a related properties file that includes the localizable text entries for the various messages displayed for that type. These properties also include placeholders that are substituted for real values related to the citizen at run time.

The wording of this text can be customized, by inserting a different version of the properties file into the resource store. The following table defines which properties file need to be changed for each type of message:

Table 16: Message properties files

Message type	Property file name
Payments	CitizenMessageMyPayments.properties

Table 16: Message properties files (continued)	
Message type	**Property file name**
Application Acknowledgment	`CitizenMessageApplicationAcknowledgement.properties`
Verifications	`CitizenMessageVerificationMessages.properties`
Meetings	`CitizenMessageMeetingMessages.properties`
Referral	`CitizenMessagesReferral.properties`
Service Delivery	`CitizenMessagesServiceDelivery.properties`

You can also remove placeholders (which are populated with live data at run time) from the properties. However, there is currently no means to add further placeholders to existing messages. A custom type of message must be implemented in this situation.

Adding a new type of citizen message

Messages are gathered by the controller in two ways: the controller reads messages that were persisted to the database by using the `curam.citizenmessages.persistence.impl.ParticipantMessage` API, and also gathers them by raising the `curam.participantmessages.events.impl.CitizenMessagesEvent`

A decision needs to be made regarding whether to 'push' the messages to the database, or else have them generated dynamically by a listener that listens for the event that is raised when the citizen logs in. The specific requirements of the message type need to be considered, along with the benefits and drawbacks of each option.

Persisted messages

In this scenario, when something takes place in the system that might be of interest to the citizen, a message is persisted to the database. For example, when a meeting invitation is created, an event is fired. The initially configured meeting messages function listens for this event. If the meeting invitee is a participant with a linked account, a message is written to the `ParticipantMessage` table that informs the citizen that they are invited to the meeting.

One benefit of this approach is that little processing is done when the citizen logs in to see this message: the message is read from the database and displayed, as opposed to calculation that takes place that would determine whether the message was required. However, the implementation also needs to handle any changes to the underlying data that might invalidate or change the message, and take appropriate action.

For example, the meeting message function also listens for changes to meetings to ensure the meeting time, location, and similar, are up to date, and to send a new message to the citizen to inform the citizen that the location or time was changed.

Dynamic messages

These messages are generated when the citizen logs in, by event listeners that listen for the `curam.participantmessages.events.impl.CitizenMessagesEvent.userRequestsMessages` event.

Because the message is generated at runtime, code is not required to manage change over time. The message is generated based on the data within the system each time the citizen logs in. If some underlying data changes, the next time the citizen logs in, they will get the correct message.

A drawback to this approach is that significant processing might be required at run time to generate the message. Care must be taken to ensure that this processing does not adversely affect the load time of the **Citizen Account** dashboard.

Performance considerations must be evaluated against the requirements of the specific message type and the effort that is required to manage change to the data that the message is related to over time. For

example, the initially configured verification message is dynamic. When a citizen logs in, it checks to see whether any outstanding verifications exist for that citizen. This process is a relatively simple database read, whereas it would be complicated to listen for various events in the Verification Engine and ensure that an up-to-date message was stored in the database related to the participants' outstanding verifications. Alternatively, the meeting messages need to inform the citizen of changes to their meetings, so functionality had to be written to manage changes to the meeting record and its related message over time.

Implementing a new message type

Organizations can implement a dynamic message or a persisted message.

To implement a new message type, regardless of whether the message is persisted or is generated dynamically, complete the following steps.

Common tasks

- Add an entry to the `CT_ParticipantMessageType` codetable to represent the new message type, in administration to configure the new message type.
- Add a DMX entry for the ParticipantMessageConfig database table. This stores the type and sort order of the new message type and is used for administration. For example:

```
<row>
   <attribute name="PARTICIPANTMESSAGECONFIGID">
     <value>2110</value>
   </attribute>
   <attribute name="PARTICIPANTMESSAGETYPE">
     <value>PMT2001</value>
   </attribute>
   <attribute name="ENABLEDIND">
     <value>1</value>
   </attribute>
   <attribute name="SORTORDER">
     <value>5</value>
   </attribute>
   <attribute name="VERSIONNO">
     <value>1</value>
   </attribute>
 </row>
```

- Add a properties file to the App Resource store that contains the text properties and image reference for the message.
- Add an image for this message type to the resource store.

Implementing a dynamic message

To implement a dynamic style message, an event listener needs to be implemented to listen for the `CitzenMessagesEvent.userRequestsMessages` event. This event argument contains a reference to the Participant and a list, to which the listener adds `curam.participantmessages.impl.ParticipantMessage` Java™ objects. For further details please consult the Javadoc API for `CitzenMessagesEvent`. This can be found in `<CURAM_DIR>/EJBServer/components/core/doc`

Developers should also refer to the Javadoc API for `curam.participantmessages.impl.ParticipantMessage` and `curam.participantmessages.impl.ParticipantMessages` for a full explanation.

The message text is stored in a properties file in the resource store. A dynamic listener retrieves the relevant properties from the resource store, and create the ParticipantMessage object accordingly. The message text for a given message can include placeholders. Values for placeholders are added to ParticipantMessage objects as parameters. The CitizenMessagesController resolves these placeholders, replacing them with the real values related to the participant in question that have been added as parameters to the message object.

Take for example this entry from the CitizenMessageMyPayment.properties file:

```
Message.First.Payment=
  Your next payment is due on {Payment.Due.Date}
```

The actual payment due date of the payment in question is added to the ParticipantMessage object as a parameter (see example code below). The CitizenMessagesController then resolves the placeholders, populating the text with real values, and then turns the message into XML that is rendered on the citizen account (there is also a public CitizenMessageController method that will return all messages for a citizen as a list, please see the Javadoc information).

From curam.participantmessages.impl.ParticipantMessage API:

```
/**
 * Adds a parameter to the map. The paramReference
 * should be present in the message title or body so
 * it can be replaced by the paramValue before the message
 * is displayed.
 *
 * @param paramReference
 * a string place holder that is present in either the
 * message title or body. Used to indicate where the value
 * parameter should be positioned in a message.

 * @param paramValue
 * the value to be substituted in place of the place holder
 */
 public void addParameter(final String paramReference,
   final String paramValue) {

   parameters.put(paramReference, paramValue);
 }
```

The call to the method would look like this:

```
participantMessage.addParameter("Payment.Due.Date", "1/1/2011");
```

Messages can also include links. Similarly to placeholders, links are resolved at runtime. Links can use placeholder values as the text to be displayed for that link. A link is defined in a properties file as such:

```
Click {link:here:paymentDetails} to view the payment details.
```

In this example, "here" is the text to display, and "paymentDetails" refers to the name of the link that is to be inserted at that point in the text. Please see the Advisor Developer's Guide for more information. For a dynamic listener to populate this link with a target, it would create a curam.participantmessages.impl.ParticipantMessageLink object, specifying a target and a name for the link. The code would look like this:

```
ParicipantMessageLink participantMessageLink =
     new ParticipantMessageLink(false,
       "CitizenAccount_listPayments", "paymentDetails");

   participantMessage.addLink(participantMessageLink);
```

Before composing the message, the dynamic listener must check to ensure that the message type in question is currently enabled. The curam.participantmessages.configuration.impl.ParticipantMessageConfiguration record for that message type should be read, and the isEnabled method used to determine if this message type is enabled. If not, no further processing should occur.

* It is recommended to separate the code that listens for the event and the code that composes a specific message, to adhere to the philosophy of "doing one thing and doing it well".

Implementing a persisted message

To have a persisted message displayed to the citizen, it must be written to the database via the curam.citizenmessages.persistence.impl.ParticipantMessage API. Message arguments are handled by persisting a curam.advisor.impl.Parameter record and associating it with the ParticipantMessage record, and links by the curam.advisor.impl.Link API. Parameter names should map to placeholders contained within the message text. Link names should relate to the names of links specified in the message text. Please refer to the Javadoc information of curam.citizenmessages.persistence.impl.ParticipantMessage, curam.advisor.impl.Parameter and curam.advisor.impl.Link for more.

An expiry date time must be specified for each ParticipantMessage. After this date time, the message is no longer be displayed.

Messages can be removed from the database. If a message needs to be replaced with a modified version, or removed for another reason, this can be done via the `curam.citizenmessages.persistence.impl.ParticipantMessage` API.

Each message has a related ID and type. This is used to track the record that the message is related to. For example, meeting messages will store the Activity ID and a type of "Meeting". Messages can be read by participant, related ID and type via the `ParticipantMessageDAO`.

Before persisting the message, the dynamic listener checks to ensure that the message type in question is currently enabled. The `curam.participantmessages.configuration.impl.ParticipantMessageConfiguration` record for that message type should be read, and the `isEnabled` method used to determine if this message type is enabled. If not, no further processing should occur.

Customizing specific message types
Organizations can customize the default message to create a referral message or a service delivery message.

Referral message

This message type creates messages related to referrals. This is a dynamic message. When the citizen logs in, a message will be created for each referral that exists for the citizen in the system, provided that referral has a referral date of today or in the future, and provided that a related Service Offering has been specified for this referral. The properties file `EJBServer\components\CitizenWorkspace\data\initial\blob\prop\CitizenMessageReferral.properties` contains the properties for the referral message text, message parameters, links and images. This properties file is stored in the resource store. This resource is registered under the resource name `CitizenMessageReferral`. To change the message text of the message, or to remove placeholders or change links, a new version of this file must be uploaded into the resource store.

Service delivery message

This message type creates messages related to service deliveries. This is a dynamic message. When the citizen logs in, a message will be created for each service delivery that exists for the citizen in the system, provided that service delivery has a status of 'In Progress' or 'Not Started'. The properties file `EJBServer\components\CitizenWorkspace\data\initial\blob\prop\CitizenMessageServiceDelivery.properties` contains the properties for the service delivery message text, message parameters, links and images. This properties file is stored in the resource store. This resource is registered under the resource name `CitizenMessageServiceDelivery`. To change the message text of the message, or to remove placeholders or change links, a new version of this file must be uploaded into the resource store.

Payment messages
The payment message type creates messages based on the payments issued or canceled for a citizen.

The payment messages are persisted to the database. They replace each other, for example, if a payment is issued and then canceled, the payment issued message will be replaced with a payment canceled message. The properties file `EJBServer\components\CitizenWorkspace\data\initial\blob\prop\CitizenMessageMyPayments.properties` contains the properties for financial message text, message parameters, links and images. This properties file is stored in the resource store. This resource is registered in the resource name `CitizenMessageMyPayments`. To change the message text of financial messages, or to remove placeholders or change links, a new version of this file must be uploaded into the resource store. The table below describes the messages created when various events related to payments occur in the system, and the property in `CitizenMessageMyPayments.properties` that relates to each message created.

Table 17: Payment messages and related properties	
Payment event	**Message Property**
First payment issued on a case	Message.First.Payment
Latest payment issued	Message.Payment.Latest
Last payment issued	Message.Last.Payment
Payment canceled	Message.Cancelled.Payment
Payment reissued	Message.Reissue.Payment
Payment stopped (case suspended)	Message.Stopped.Payment
Payment / Case unsuspended	Message.Unsuspended.Payment

Customization of the payment messages expiry date

The number of days the payment for which the message will be displayed to the citizen can be configured using a system property. By default the property value is set to 10 days, however, this can be overridden from property administration.

Table 18: Payment message expiry property	
Name	**Description**
curam.citizenaccount.payment.message.expiry.days	The number of days the payment message will be displayed to the participant.

Meeting messages

The meeting message type creates messages based on meetings that citizens are invited to, provided that they are created by using the `curam.meetings.sl.impl.Meeting` API.

The API raises events that the meeting messages functionality consumes. There are other ways of creating Activity records without this API, but meetings created in these ways do not have related messages created as the events are not raised. These messages are persisted to the database. They replace each other, for example, if a meeting is scheduled and then the location is changed, the initial invitation message is replaced with one informing the citizen of the location change. The properties file `EJBServer\components\CitizenWorkspace\data\initial\blob\prop \CitizenMessageMeetingMessages.properties` contains the properties for the meeting messages text, message parameters, links and images. This properties file is stored in the resource store. This resource is registered in the resource name `CitizenMessageMeetingMessages`. To change the message text of meeting messages, or to remove placeholders or change links, a new version of this file must be uploaded into the resource store. Table 1 describes the messages created when various events related to meetings occur in the system, and the properties in `CitizenMessageMeetingMessages.properties` that relates to each message created. Different versions of the message text are displayed depending on whether the meeting is an all day meeting, whether a location has been specified, and whether the meeting organizer has contact details registered in the system. Accordingly, the property values in this table are approximations that relate to a range of properties within the properties file. Refer to the properties file for a full list of the message properties.

Table 19: Meeting messages	
Meeting event	**Message Properties**
Meeting invitation	Non.Allday.Meeting.Invitation.*, Allday.Meeting.Invitation.*
Meeting update	Non.Allday.Meeting.Update.*, Allday.Meeting.Update.*
Meeting canceled	Allday.Meeting.Update.*, Allday.Meeting.Cancellation.*

Customization of the meeting messages display date

The number of days before the meeting start date that the message should be displayed to the citizen can be configured using a system property. By default the property value is set to 10 days, however, this can be overridden from property administration.

The meeting message expires (it is no longer displayed to the citizen) at the end of the meeting, that is, the date time at which the meeting is scheduled to end.

Table 20: Meeting message display date property

Name	Description
curam.citizenaccount.meeting.message.effective.days	The number of days before the meeting start date that the message should be displayed to the citizen.

Application acknowledgment message
The application acknowledgment message type creates a message when an application is submitted by a citizen.

The message is persisted to the database. The properties file `EJBServer\components \CitizenWorkspace\data\initial\blob\prop \CitizenMessageApplicationAcknowledgment.properties` contains the properties for the messages text, message parameters, links and images. This properties file is stored in the resource store. This resource is registered under the resource name `CitizenMessageApplicationAcknowledgment`. To change the message text of the message, or to remove placeholders or change links, a new version of this file must be uploaded into the resource store.

Customization of application acknowledgment message expiry date

The number of days the Application Acknowledgment message will be displayed to the citizen can be configured using a system property. By default the property value is set to 10 days, however, this can be overridden from property administration.

Table 21: Application acknowledgment message expiry property

Name	Description
curam.citizenaccount. intake.application.acknowledgement.message.expiry.days	The number of days the application acknowledgment message will be displayed to the participant.

Artifacts with limited customization scope

A description of IBM Cúram Universal Access artifacts that have restrictions on their use. Customers that want to change these artifacts should consider alternatives or request an enhancement to Universal Access.

Model

Customers are not supported in making changes to any part of the Universal Access model. Changes in the model such as changing the data types of domains can cause failure of the Universal Access system and upgrade issues. This applies to the model files in the following packages:

- WorkspaceServices
- CitizenWorkspace
- CitizenWorkspaceAdmin

Code tables

See the *IBM Cúram Development Compliancy Guide* for a list of restricted code tables.

Related information

Developing Compliantly with Cúram Business Intelligence

Troubleshooting and support

Use this information to help you to troubleshoot issues with the Responsive Web Application or .

The supported assets can be installed, customized, and deployed separately from , before being integrated into the system.

When troubleshooting web applications that are integrated with , use this troubleshooting information in conjunction with the troubleshooting information for . For more information, see the *Troubleshooting and support* related link.

Citizen Engagement components and licensing

You can use and customize the new Universal Access web application for your organization, or develop your own custom web applications to complement the standard web client. Use this information to understand the components, supported assets, and licenses that you need.

Installable components

supported asset
The design system provides the foundational packages for building accessible and responsive web applications. It consists of a React UI component library, React development resources, and a style guide for creating web applications.

Responsive Web Application supported asset
The new web application, which you can use and customize for your organization. The responsive web application requires the and the application module.

application module
The (UA) application module includes REST APIs that expose interfaces to and IEG functions for consumption by the Responsive Web Application. requires the .

Licensing Universal Access (New)

To use and customize the new Universal Access web application, customers can buy the application module, which entitles the Universal Access Responsive Web Application asset, and , which entitles the asset. Customers can also buy Citizen Engagement, which includes the application module, the , and the assets.

Licensing the

To develop custom web applications to complement the standard web client, customers must buy the , which entitles the asset.

Citizen Engagement support strategy

The Citizen Engagement assets are expected to be released monthly, and they can be upgraded independently of the base product. Due to the more frequent release schedule, the asset support strategy is to maintain a single product line for the supported assets for both new features and maintenance. Where possible, all updates are planned for the latest version of the assets. Security and defect fixes will be delivered in the latest release only.

Support strategy for the application module

Where possible, REST APIs changes are delivered in refresh pack or other impact-free releases that impose no forced upgrade impact.

Support strategy for the supported assets

Assets versions are supported for the lifetime of the latest supported version available at the time of the asset release.

The assets use semantic versioning. As a general guideline, this means:

- MAJOR version for incompatible API changes
- MINOR version for adding functionality in a backwards-compatible manner
- PATCH version for backwards-compatible bug fixes

The assets will be full releases rather than delta releases regardless of version type.

Although new features (pages) can be delivered in any minor release, new pages will generally be delivered at the same time as the release that contains the new APIs for those features.

Compatibility

You can confirm compatibility between a version of the supported assets and the software by searching for your software version in Fix Central. Only compatible versions of the asset are available. In addition, the asset release notes will contain information on compatibility. For example, "This version is compatible with versions 7.0.3 and 7.0.4."

Examining log files

Log files are a useful resource for troubleshooting problems.

Examining the browser console logs

For JavaScript applications, you can examine the browser console logs for errors that might be relevant to investigating problems. For the exact details about how to locate the console logs within the browser, see your browser documentation.

Note: When you are developing applications with the , console logging information might also be displayed within the console that runs the start process for the application.

Examining the HTTP Server log files

When you deploy a built application on an HTTP Server, the built application introduces a new point with which logging is captured in your system topology. The and the include comprehensive logging system and related information.

For more information about troubleshooting the , see Troubleshooting IBM HTTP Server.

For more information about troubleshooting the , see Managing Oracle HTTP Server Logs.

Known limitations

Review the known limitations for the and , and, where available, workaround information.

Note: and are provided only in US English.

Symptom

Translation packs are not provided with and . Therefore, translations are not currently provided for the languages that are supported by the corresponding or IBM Universal Access products.

Resolution

Support for regionalizing or localizing your user interface (UI) is not affected.

Notices

This information was developed for products and services offered in the United States.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd. 19-21, Nihonbashi-Hakozakicho, Chuo-ku Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM 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 IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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.

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 IBM, 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. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Privacy Policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies or other similar technologies that collect each user’s name, user name, password, and/or other personally identifiable information for purposes of session management, authentication, enhanced user usability, single sign-on configuration and/or other usage tracking and/or functional purposes. These cookies or other similar technologies cannot be disabled.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “ Copyright and trademark information ” at http://www.ibm.com/legal/copytrade.shtml.

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.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, 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 names may be trademarks of their respective owners. Other company, product, and service names may be trademarks or service marks of others.

Part Number:

(1P) P/N: