

Cúram 8.1.2

Content Management Interoperability Services Integration Guide

Note

Before using this information and the product it supports, read the information in [Notices on page 27](#)

Edition

This edition applies to Cúram 8.1, 8.1.1, and 8.1.2.

© Merative US L.P. 2012, 2024

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

Contents

Note.....	iii
Edition.....	v
1 Integrating with a content management system (CMS).....	9
1.1 Configuring Cúram to use with a CMS.....	9
Registering and adding a CMS to the target system.....	9
Activating a CMS.....	11
1.2 Integrating a CMS with Cúram.....	13
Gradual migration.....	14
Attachments and securing documents in a CMS.....	15
Microsoft® Word and pro forma communications.....	15
1.3 Metadata.....	16
Using the default metadata.....	18
1.4 Customization.....	19
Customizing metadata.....	19
Custom directory structures and file-naming strategies.....	22
Single sign-on.....	24
1.5 CMIS validation error messages.....	25
Notices.....	27
Privacy policy.....	28
Trademarks.....	28

1 Integrating with a content management system (CMS)

Register a target system so that you can use a content management system (CMS) as a repository for documents. When you integrate with a CMS, attachments, Microsoft™ Word communications, and pro forma communications documents are stored and retrieved from the CMS.

Organizations use a content management system (CMS) to store and manage various types of content. Various configuration options are available to integrate Cúram with a CMS. The following list outlines the concepts that you must be familiar with before you start to configure Cúram to integrate with a CMS:

- The basic elements of case processing in the human services industry.
- The Cúram case management function.

For more information, see the *Communications Overview Guide* and the *Integrated Case Management Guide*.

Standards

Cúram uses the Content Management Interoperability Services (CMIS) standard. CMIS integration is verified with IBM® FileNet P8 v5.5.1 and IBM® FileNet CMIS v3.0.5. For more information about CMIS standards, see the *Content Management Interoperability Services (CMIS) Version 1.0* and the *Content Management Interoperability Services (CMIS) Version 1.1* related links.

Related concepts

[Content Management Interoperability Services \(CMIS\) Version 1.0](#)

[Content Management Interoperability Services \(CMIS\) Version 1.1](#)

1.1 Configuring Cúram to use with a CMS

Before you can use a content management system (CMS) as a repository for documents, two steps are required. You must register the CMS as a target system and then configure Cúram so that the application can communicate with the CMS.

Registering and adding a CMS to the target system

To permit two-way communication between Cúram and a content management system (CMS), register a target system. To add the CMS service to the target system, specify the service settings.

Register a target system

A system administrator must configure the details for the content management target system. The following steps outline how to register a target system:

1. Log in to the system administration application.
2. Select to create a new target system.
3. Enter a unique name for the target system.
4. Enter the URL of the target system.

CMS target system service settings

You must specify CMS service settings for a target system. To specify the CMS service settings, you must first check your CMS to determine which CMIS binding type (service name) is available and the associated URL for that binding type.

Service name

- **Description**

Service name is the unique name of the service.

Select **Content Management Interoperability Service over Atompublish** or **Content Management Interoperability Service over Browser Binding**.

Note: The Atompublish binding is available for all repositories that support CMIS version 1.0 and 1.1. The browser binding is available only since CMIS version 1.1 and only for repositories where this optional binding is implemented.

- **Mandatory**

Yes.

Extension

- **Description**

Extension is the URL extension that the system adds to the system address of the address. The extension provides the full location of the Content Management Interoperability Service (CMIS) repository.

- **Mandatory**

Yes.

Invoking user username

- **Description**

Invoking user username is the login username that the CMIS service requires. By default, the account is required to connect to the CMS. If the organization implemented CMIS single sign-on support and enabled single sign-on support in the application, leave the setting blank. For more information about single sign-on, see [Single sign-on on page 24](#).

- **Mandatory**

No.

Invoking user password

- **Description**

The CMIS service requires the login password. By default, the account is required to connect to the CMS. If the organization implemented CMIS single sign-on support and enabled single sign-on support in the application, leave the setting blank. For more information about single sign-on, see [Single sign-on on page 24](#).

When the target system service is created, the login password is encrypted by Cúram encryption algorithms. The plain text password is never stored. Cúram compares the encrypted values only for authentication. The password is not retrieved during later editing of the service. If you edit the service, the system displays the **Invoking User Password** field as empty.

For more information about encryption, see the *Security Guide*.

- **Mandatory**
No

Note: A configuration is valid only where exactly one target system service exists for the Content Management Interoperability Service. Creating a service definition does not test the connection to the service. The system administrator must ensure that all target system and service details are correct before they enable CMIS in Cúram. Any connection issues are reported when users interact with the CMS through any Cúram CMIS integration, for example, when users create, view, or update attachments.

Activating a CMS

After you add the content management system (CMS) service to the target system, configure and activate the CMS.

Content Management Interoperability Service (CMIS) properties

Use the system properties group to configure the details for the content management target system. Click **System Administration > Property Administration > Application Properties - Content Management Settings**. Use the following property information to update the Content Management Interoperability Service (CMIS) properties of the CMIS target system service.

curam.cms.enable

- **Description**
The `curam.cms.enable` property indicates whether the CMIS is enabled. **True** determines that the storage location for specific files is in a configured CMS. **False** determines that the storage location is in the Cúram database.
- **Default value**
True.

curam.cms.attachment.enable

- **Description**
The `curam.cms.attachment.enable` property indicates whether the CMIS is enabled. **True** determines that the storage location for attachments is in a configured CMS. **False** determines that the storage location is in the Cúram database.
- **Default value**
True.

curam.cms.proforma.enable

- **Description**
The `curam.cms.proforma.enable` property indicates whether storing pro forma communications to the CMS is enabled. **True** determines that the storage location for communications is in a configured CMS. **False** determines that the storage location is in the Cúram database.
- **Default value**
True.

curam.cms.metadata.enable

- **Description**

The `curam.cms.metadata.enable` property indicates whether CMIS metadata is enabled. **True** determines that the associated metadata is uploaded with any file that is uploaded to the CMS. **False** determines that the only the file is uploaded to the CMS. Before you set the property to **True**, ensure that the metadata properties are configured on the CMS. For more information about configuring the metadata properties, see [Customizing metadata on page 19](#).

- **Default value**

False.

curam.cms.curam.dir

- **Description**

The `curam.cms.curam.dir` property indicates the absolute path to the root directory for Cúram content within the global CMS repository that the Cúram application uses. The following list outlines the criteria for the absolute path:

- The absolute path must begin with /
- The absolute path must not end with /
- The absolute path must not contain any unsupported characters.

For more information about unsupported characters, see the documentation for your CMS.

- **Default value**

The set is based on the organization implementation, for example `/Curam`.

curam.cms.repository.name

- **Description**

The `curam.cms.repository.name` property is the single repository for the global CMS account that the Cúram application uses.

- **Default value**

The set is based on the organization implementation.

curam.cms.sso.enable

- **Description**

The `curam.cms.sso.enable` property indicates whether single sign-on (SSO) is enabled. **True** enables SSO support. **False** disables SSO support. **False** uses the default authentication. Use **True** only when the organization implemented the custom code for SSO. For more information, see [Single sign-on on page 24](#).

- **Default value**

False.

curam.cms.connectiontimeout

- **Description**

The `curam.cms.connectiontimeout` property indicates the number of milliseconds that is spent connecting with the configured CMS before the result is a connection failure. Set the property to any positive integer or to zero. When the property is set to zero, the property is ignored.

- **Default value**
0

curam.cms.readtimeout

- **Description**
The `curam.cms.readtimeout` property indicates the number of milliseconds that is spent reading from the configured CMS before the result is a read failure. Set the property to any positive integer or to zero. When the property is set to zero, the property is ignored.
- **Default value**
0

Note: Do not change the CMIS application properties after the system goes live. Changing the properties after the system goes live requires extensive analysis and offline reconfiguration of the system.

1.2 Integrating a CMS with Cúram

When you enable integration with a content management system (CMS), attachments, Microsoft Word communications, and pro forma communications documents are stored and retrieved from the CMS. Attachment, Microsoft Word communications, and pro forma communications documents are not stored in the Cúram database.

Customizing the CMS

To use a Content Management Interoperability Service (CMIS) with attachments, organizations must verify whether the organization overrode the Cúram attachment entity implementation. If the organization overrode the Cúram attachment entity implementation, then the organization can use the customization and the CMIS with attachments functionality. However, the organization must update the custom version to match the most recent Cúram version. Update the custom implementation so that it runs the same functions as the Cúram implementation. Update each custom function to call the Cúram implementation.

Note: CMIS integration is not supported for Cúram batch jobs.

Note: Cúram's CMIS integration supports versionable document types only. Cúram's CMIS integration does not support non-versionable document types.

Configuring the CMS

To use the Cúram functionality, you must configure the CMS. The following list outlines the classes that you must add:

- Add a `CuramDocument` class as a subclass of `cmis:document`.
- Add a `CuramAttachment` class as a subclass of `CuramDocument`.

The following table outlines the properties that are required on the `CuramAttachment` document class on the CMS.

Table 1: CMIS metadata properties

Property	Type
<i>caseReference</i>	String
<i>communicationDate</i>	Date
<i>documentReceiptDate</i>	Date
<i>documentType</i>	String
<i>documentTypeCode</i>	String
<i>participantDOB</i>	Date
<i>participantFirstName</i>	String
<i>participantLastName</i>	String
<i>participantReference</i>	String
<i>applicationReference</i>	String

Gradual migration

Organizations that are using the Cúram application and who are storing files in the application database can enable Content Management Interoperability Service (CMIS) integration without performing an upfront migration of the content in the content management system (CMS) repository.

The documents that are stored in the Social Program Management application database remain in the database and are fully accessible. Documents are saved to the CMS only after any changes are made to a document or after a document is created. After a document is saved to the CMS, the document is also removed from the application database.

Note: When you enable the CMIS, do not disable it. Documents that are stored on the CMS can't be accessed from within the application.

CMIS attachment integration points that read or modify content

You must create or update custom CMIS attachment integration points that read or modify content. Update integration points that read content to check whether the content exists on the CMS. Update the modify integration points by using the following pseudocode logic:

```

}if CMIS is enabled {
  if content exists on the CMS for the record in question {
    modify the contents on the content management system
  } else {
    create the contents on the content management system
    blank the application database copy (optional)
  }
} else CMIS is not enabled {
  maintain the database copy
}

```

Note: Custom pro forma communication integration points are not affected.

Attachments and securing documents in a CMS

When a user views the attachment, the attachment is retrieved from the content management system (CMS). Existing Cúram security mechanisms provide access to documents that are stored in a CMS.

Attachments

Attachment records in the Cúram database no longer store the associated file content. Instead, the file and any of the configured attachment metadata elements for the attachment are stored in the CMS. If the user uploads a new version of the file, the original document that is stored in the CMS is superseded by the new document. If the user updates any configured attachment metadata elements, the elements are also updated in the CMS. Selecting the attachment returns the most recent version of the file on the CMS. In CMS repositories that support version control of documents, superseded versions of documents are still available to view directly from the CMS application.

Securing documents in a CMS

Access to documents that are stored in a CMS are secured only through existing Cúram security mechanisms, for example data-based security and location-based security. For more information about data-based and location-based security, see the *Server Developer's Guide* and the *Security Guide*. Customers must provide their own means of securing alternative forms of access to documents in the CMS, for example security for accessing documents directly from the CMS.

Microsoft® Word and pro forma communications

For Microsoft® Word communications, the Microsoft® Word document is stored in the content management system (CMS) when the communication is created. For pro forma communications, the generated PDF file uses a pro forma template that is stored in the CMS when the status of the pro forma communication is set to **Sent**.

Microsoft® Word communications

The Microsoft® Word document in the CMS includes any configured attachment metadata elements that exist for the Microsoft® Word document. When a user opens the Microsoft® Word document, the Microsoft® Word document is retrieved from the CMS. When the user modifies the Microsoft® Word document, a new version of the document is stored in the CMS. When the Microsoft Word document is opened, the most recent version of the document is retrieved.

Pro forma communications

When the PDF file is stored in the CMS, any requests to preview the pro forma communication retrieves the file from the CMS.

Note: Except for the document title, no metadata is stored for pro forma communications.

1.3 Metadata

You can store metadata with a document that is stored in the content management system (CMS). By default, any CMS that supports Content Management Interoperability Service (CMIS) must save part of the `cmis:document` metadata for all files. Typically, the front end of a CMS highlights only the document title of the system metadata. However, the rest of the document is usually accessible.

Attachment metadata

When CMIS is enabled in Cúram, by default metadata is stored for all Social Program Management attachments. You can update custom attachment integration points to use the `CMISAccess` API functions to store and modify extra metadata properties with a file that is stored in a CMS. Attachments are stored as class type `CuramAttachment`. The class type `CuramAttachment` is a subclass of `CuramDocument`. The subclass `CuramDocument` is a subclass of `cmis:document`. Pro forma communication attachments are stored as `CuramDocument`. Pro forma communication attachments do not support metadata.

Metadata properties

Metadata is stored for all attachments, including attachments that are associated to recorded communications and to Microsoft Word communications. By default, you can store 10 metadata properties. The following class type tree lists the metadata properties:

- `cmis:document`
 - `CuramDocument`
 - `CuramAttachment`
 - *caseReference*
 - *communicationDate*
 - *documentReceiptDate*
 - *documentType*
 - *documentTypeCode*
 - *participantDOB*
 - *participantFirstName*
 - *participantLastName*
 - *participantReference*
 - *applicationReference*

Administrative users can enable or disable the storage of individual metadata properties. When administrative users enable or disable a metadata property, the change applies to all attachment integration points. The metadata that is stored depends on the attachment business flow. For example, where the case reference metadata property is enabled the following list outlines the effects:

- The property is stored as metadata for a case attachment created within an integrated case.
- The property is not stored for a participant attachment that is created for a person because it is not relevant.

Metadata descriptions

The following table describes each of the 10 default metadata properties. The table includes information about when and how the metadata is stored.

Table 2: CMIS metadata descriptions

Metadata element	Description
Case reference	Case reference is the case reference number of the case where the attachment, recorded communication, or Microsoft Word communication is created, if created within a case. For example, if an attachment is created within a service plan, the case reference of the service plan is stored. The case reference of the case where the service plan was created is not stored.
Participant reference	Participant reference is the participant reference number, which is stored as the primary alternate ID for the participant. The participant reference number is stored for all participant attachments and for attachments that are created in other contexts. For example, participant reference numbers that are created within a case where the attachment business flow permits the user to select a specific participant while the user creates the attachment.
Participant first name	Participant first name is the given name of a participant, which is stored as part of the primary alternate name for the participant. The metadata property is stored only for person and prospect person type participants.
Participant last name	Participant last name is the surname of a participant, which is stored as part of the primary alternate name for the participant. The metadata property is stored only for person and prospect person type participants.
Participant date of birth	Participant date of birth is the birth date of a participant. The metadata property is stored only for person and prospect person type participants.
Document type	Document type is the type of document, as captured in the Document Type field when the system is creating an attachment. The metadata property is stored for all attachments where the information is stored in the application database. The metadata property is not stored where a particular attachment business flow does not capture the information.
Document type code	The document type code is the code for the document type, as captured in the Document Type field when the system is creating an attachment. The metadata property is stored for all attachments where the information is stored in the application database. The metadata property is not stored where a particular attachment business flow does not capture the information.
Document receipt date	Document receipt date is the date that the document was received, as captured in the Receipt Date field when the system is creating an attachment. The metadata property is stored for all attachments where the information is stored in the application database. The metadata property is not stored where a particular attachment business flow does not capture the information.
Communication date	Communication date is the date of the communication to which the attachment is associated. The metadata property is stored only for attachments that are associated to recorded communications.
Application reference	Application reference is the reference number for the application case.

The metadata that is modified is determined by the attachment business flow and the data that is stored as metadata that can be modified in the application. For example, for integrated case attachments, the document type for the attachment can be modified in the application. Where the document type for the attachment is modified, it is modified in the CMS. However, the case reference of the attachment can't be modified. If an attachment record is initially created without an associated file, initially no file or metadata is stored in the CMS. However, if the attachment is then updated and a file is associated, then all metadata elements that are enabled and available are stored with the file in the CMS.

You can also set up a custom implementation by using the new APIs that includes extra metadata properties.

Note: The infrastructure updates the file content only if the content changed. The infrastructure updates the metadata properties only if the properties are supplied.

Using the default metadata

To use the metadata properties that are provided by default when the Content Management Interoperability Service (CMIS) is enabled, ensure that the content management system (CMS) is configured with a metadata schema that matches the schema that is implemented on the application side.

When a file is sent to the CMS through the CMIS API, contextual information, including metadata, is collected. The following list outlines the steps that are performed by the CMIS API:

- Fetches the collected contextual information.
- Derives as much metadata as possible.
- Distinguishes between metadata that is enabled by using the system administration screen and other contextual information.
- Calls a custom hook point that can wire up or derive extra, custom metadata.
- Updates the metadata on the CMS.

Configuring metadata properties on the CMS

To use the CMIS metadata support, you must configure the CMS for metadata. The properties that are required for the `CuramAttachment` document class on the CMS are listed in the following table.

Table 3: CMIS metadata properties

Property	Type	Length
<i>caseReference</i>	String	40
<i>communicationDate</i>	Date	n/a
<i>documentReceiptDate</i>	Date	n/a
<i>documentType</i>	String	500
<i>documentTypeCode</i>	String	10
<i>participantDOB</i>	Date	n/a
<i>participantFirstName</i>	String	65
<i>participantLastName</i>	String	65

Property	Type	Length
<i>participantReference</i>	String	18
<i>applicationReference</i>	String	10

Configuring metadata properties in the Cúram application

A user with system administrator privileges can manage metadata properties within the system administration application. Users can enable or disable each individual property. To provide multi-language support, users can specify multiple display names and descriptions for each property.

Note: By default, the metadata properties are enabled. Before the CMIS goes live, ensure that the properties are checked. If required, change the properties.

Note: When the CMIS is live, disabling a property through the application does not remove existing content from the CMS.

1.4 Customization

You can customize three main parts of the content management system (CMS): metadata, the directory structures and file-naming strategies, and the single sign-on.

Customizing metadata

Use an explicit method or a transaction-scoped method to collect extra metadata for a document.

Adding custom metadata properties to the documents that are stored on the CMS

In addition to the metadata properties that are stored by default, you can add custom metadata properties to the documents that are stored on the content management system (CMS). The following steps outline how to add the custom metadata properties to the documents that are stored on the CMS:

1. Ensure that the additional metadata properties are configured correctly on the CMS.
2. Implement and bound a custom hook point on the application side to derive or wire up the additional metadata.
3. Optionally, collect inputs for the custom hook point. You can collect the inputs at any point in the business flow.

There are two ways of collecting extra metadata for a document.

Explicit method

Make modelling changes and pass the metadata from function to function to the integration point. At the integration point, the custom code adds the metadata to the document that is to be stored. The method makes it easier to follow the information flow when you are reading code or when you are debugging code.

Transaction-scoped method

The transaction-scoped method means that metadata can be collected at any point in the business flow by using an object that is injected by Guice. When the metadata is collected by the Guice-injected object before the call to the Content Management Interoperability Service (CMIS) infrastructure, then the metadata is available to wire up in the custom hook point. Metadata that is wired up in the custom hook point is sent with the document to the CMS.

Configuring extra metadata properties on the CMS

Configure the document class on the CMS for the new custom metadata properties that are required. For more information about setting up document classes and configuring metadata properties, see [1.2 Integrating a CMS with Cúram on page 13](#) and [1.3 Metadata on page 16](#).

Note: When you add extra metadata properties, the properties are enabled. System administrators can't configure the properties. Add extra metadata properties so that the properties are to be stored, not disabled.

Note: The following code samples show how to add the metadata string properties `xyz_contextualReference` and `xyz_updatedBy`. By default, neither of the two properties are included with the Cúram application. You must configure the properties on the CMS.

Note: To start using the updated metadata schema over CMIS, some CMS implementations must be recycled.

Collecting metadata transactionally with Guice injection

The following code sample shows how to collect metadata using a transaction-scoped object injected by Guice. In the example, the metadata is collected within the transaction that is associated with the creation or modification of a document. When the document is either added to the CMS or updated on the CMS, the metadata that is stored in the transaction-scoped object is available within the custom hook point to wire it up to the document.

The following code sample shows the transactional collection of contextual information or metadata:

```
@Inject
private Provider<CMSMetadataInterface> cmsMetadataProvider;

public SampleConstructor() {
    GuiceWrapper.getInjector().injectMembers(this);
}

public sampleMethod() {
    ...
    CMSMetadataInterface cmsMetadata = cmsMetadataProvider.get();
    cmsMetadata.add("xyz_contextualReference", contextualReference);
    cmsMetadata.add("xyz_updatedBy",
        curam.util.transaction.TransactionInfo.getProgramUser());
    ...
}
```

Note: The name of a custom input to the custom hook point must, as shown in the preceding code, be prefixed by a string, for example "xyz_", where "xyz" is a customer-specific prefix.

For more information, see the *Business Intelligence and Analytics Guide*.

The following code sample shows the integration point for transactional method:

```
// save the contents to the content management system
cmisAccess.create(details.attachmentID, CMSLINKRELATEDTYPEEntry.ATTACHMENT,
attachmentDtls.attachmentContents.copyBytes(), attachmentDtls.attachmentName,
CMISNAMINGTYPEEntry.ATTACHMENT, null, CMSMETADATACLASSTYPEEntry.ATTACHMENT);
```

Adding or updating metadata explicitly

An alternative mechanism is to add the metadata explicitly. When you are reading or debugging code, then adding or updating metadata explicitly can make it easier to follow the data flow.

In the method where the document is created or modified, where there is metadata to be added to the default set, call the `CMISAccessImpl.addItemToExplicitMetadata` method. The method takes in three parameters: the list to which to add the metadata, the metadata name, and either a string value or a date value.

The following code example shows how to use `CMISAccessImpl.addItemToExplicitMetadata` and how to call the overloaded `CMISAccessImpl.create` with the explicit metadata list. Replicate where custom metadata must be added to a document that is being created. The following code example shows the explicit collection of contextual information or metadata:

```
CMSMetadataItemList metadataList = new CMSMetadataItemList();
cmisAccess.addItemToExplicitMetadata(metadataList, "xyz_contextualReference",
contextualReference);
```

The following code example shows the integration point for the explicit method:

```
// save the contents to the content management system
cmisAccess.create(details.attachmentID, CMSLINKRELATEDTYPEEntry.ATTACHMENT,
attachmentDtls.attachmentContents.copyBytes(), attachmentDtls.attachmentName,
CMISNAMINGTYPEEntry.ATTACHMENT, null, metadataList,
CMSMETADATACLASSTYPEEntry.ATTACHMENT);
```

Wiring up and deriving custom metadata

From the default set of metadata, you can derive extra custom metadata values.

To create a custom metadata hook, extend

`curam.core.sl.infrastructure.cmis.impl.CMSMetadataClassCustomDefaultImpl`.

Do not implement the underlying interface. By using a Guice module, bind the extra extension class to a code table on `CMSMetadataClassType`. The following Guice module code sample shows how to bind a sample implementation:

```
MapBinder<CMSMETADATACLASSTYPEEntry, CMSMetadataClassCustomInterface>
metadataStrategyCustomBinder
= MapBinder.newMapBinder(
    binder(), CMSMETADATACLASSTYPEEntry.class,
    CMSMetadataClassCustomInterface.class);

metadataStrategyCustomBinder.addBinding(CMSMETADATACLASSTYPEEntry.ATTACHMENT)
.to(SampleCustomMetadataImpl.class);
```

The following code sample shows how to extend the `CMSMetadataClassCustomDefaultImpl` class, which is used to derive and gather further metadata and then adds the metadata to the metadata to send to the CMS.

In the implementable functions, *properties* contains the enabled metadata properties. *otherData* contains any extra contextual information and any metadata that is disabled.

The following code is a sample for a custom metadata collection:

```
public class SampleCustomMetadataImpl extends CMSMetadataClassCustomDefaultImpl {

    public void collectMetadataForNewFile(Map<String, Object> properties,
        Map<String, Object> otherData) throws AppException,
        InformationalException {

        // Manually add the metadata property to the Map
        properties.put("xyz_updatedBy",
            curam.util.transaction.TransactionInfo.getProgramUser());

        // Wire up additional metadata collected in custom facade
        properties.put("xyz_contextualReference",
            otherData.get("xyz_contextualReference"));

        // Potentially derive more custom metadata
        // properties from the data available
        ...

    }

    public void collectMetadataForUpdate(Map<String, Object> properties,
        Map<String, Object> otherData) throws AppException,
        InformationalException {

        // Manually add the metadata property to the Map
        properties.put("xyz_updatedBy",
            curam.util.transaction.TransactionInfo.getProgramUser());

        // Wire up additional metadata collected in custom facade
        properties.put("xyz_contextualReference",
            otherData.get("xyz_contextualReference"));

        // Potentially derive more custom metadata
        // properties from the data available
        ...

    }

}
```

Custom directory structures and file-naming strategies

You can customize how Cúram organizes and names the files that it stores in the content management system (CMS).

The following list outlines the directory structures and file names that are used by default within the directory that is specified by the `curam.cms.curam.dir` application property:

- `<ROOT>Default/<extension>/YYYY/MM/DD/HHMMSS_<Milliseconds>/<filename>.<extension>`
- `<ROOT>Attachment/YYYY/MM/DD/<filename>.<extension>`
- `<ROOT>ProForma/YYYY/MM/DD/<filename>_HHMMSS_<Milliseconds>.<extension>`

The `CMISNamingType` code table determines where the files are stored on the CMS. Attachments are stored in the *Attachments* directory. Pro-formas are stored in the *ProForma* directory. Any other content types that are configured as the `CuramDocument` base class type are stored in

the *Default* directory. The content types do not exist in the product that is ready for immediate use.

Creating a custom-naming strategy

To create a custom-naming strategy, extend

`curam.core.sl.infrastructure.cmis.impl.CMISNamingDefaultImpl`. Do not implement the `CMISNamingInterface`. Bind this extra extension class, by using a Guice module, to a code table on `CMISNamingType`. The following Guice module code sample shows how a sample implementation can be bound:

```
MapBinder<CMISNAMINGTYPEEntry, CMISNamingInterface> namerBinder =
    MapBinder.newMapBinder(
        binder(), CMISNAMINGTYPEEntry.class, CMISNamingInterface.class);

namerBinder.addBinding(CMISNAMINGTYPEEntry.ATTACHMENT).to(
    SampleAttachmentCMISImpl.class);
```

The directory structure and file name are composed in the `CMISNamingInterface.getFilePath` method. The method returns the file path as an ordered list of folder names that are followed by the file name, including extension. Build the intended directory tree by adding the parts of the intended naming strategy to the list in the correct order. Add the file name or extension last.

Note: The `curam.cms.curam.dir` application property is added to the list within the CMIS API.

The following code shows the sample attachment-naming strategy implementation:

```
public class SampleAttachmentCMISImpl extends CMISNamingDefaultImpl {

    @Override
    public List<String> getFilePath(String filename, long relatedID, Object relatedData)
        throws AppException, InformationalException {

        List<String> filePathBelowRoot = new ArrayList<String>();

        filePathBelowRoot.add(
            CodeTable.getOneItem(CMISNAMINGTYPE.TABLERNAME, CMISNAMINGTYPE.ATTACHMENT));

        // Format current date time
        long currentTimeMillis = System.currentTimeMillis();
        DateTime currentDateTime = new DateTime(currentTimeMillis);

        String formattedDateTime = Locale.getFormattedDateTime(currentDateTime,
            Locale.Date_ISO8601_complete);

        filePathBelowRoot.add(formattedDateTime.substring(0, 4)); // YEAR
        filePathBelowRoot.add(formattedDateTime.substring(4, 6)); // MONTH
        filePathBelowRoot.add(formattedDateTime.substring(6, 8)); // DAY
        filePathBelowRoot.add(filename);

        return filePathBelowRoot;
    }
}
```

Parameters

The file name parameter of `getFilePath` is the full file name, including the extension but not prefixed by any directory structure. `getFilePath` is a method of `CMISNamingInterface` or `CMISNamingDefaultImpl`.

The `relatedID` parameter of `getFilePath` is the identifier for the applicable Cúram object on the Cúram database. For example, with the attachment integration, for immediate use, the related ID is the `attachmentID`. For communications, the related ID is `communicationID`.

The `relatedData` parameter of `getFilePath` is any object that the organization considers potentially useful to its naming-strategy implementation. No integrations with the product implement the feature, but organizations can use the parameter in their own naming strategies. By default, the `namingType` and the current date are available for the naming strategy.

Single sign-on

Single sign-on (SSO) is a mechanism that permits an authenticated user to access multiple different systems and services without requiring the user to log in to each system and service. Typically, authentication is shared between the systems through a transmission of security tokens.

Implementing SSO with Cúram's Content Management Interoperability Service (CMIS) functionality requires custom development to integrate with an organization's SSO system. Making configuration updates to Cúram is also required.

Custom development

Organizations must first implement a hook point to add custom cookies to the CMIS invocations that Cúram performs. For example, custom security tokens that the organization's SSO implementation uses. The organization must create a class that extends the `CMISAuthenticationProviderDefaultImpl` class that includes the method `getHTTPHeaders`. Bind the extra extension class, by using a Guice module, to `CMISAuthenticationProviderDefaultImpl`. The following Guice module code sample shows how to bind a sample implementation.

```
bind(CMISAuthenticationProviderDefaultImpl.class).to(
    CustomCMISAuthenticationProvider.class);
```

The following code sample shows how to create the custom SSO implementation.

```
public class CustomSSOImplementation extends CMISAuthenticationProviderDefaultImpl {
    @Override
    public Map<String, List<String>> getHTTPHeaders(String url) {
        String tokenName = // Get token name
        String tokenValue = // Get security token value

        Map<String, List<String>> tokenMap = new HashMap<String, List<String>>();
        List<String> tokenList = new List<String>();

        tokenList.add(tokenValue);
        tokenMap.put(tokenName, tokenList);

        return tokenMap;
    }
}
```

Enabling SSO support

You must then enable SSO support. Use the system administration application property `curam.cms.sso.enable` to administer Cúram SSO support. The following list outlines the actions that are performed by the CMIS API after you enable the `curam.cms.sso.enable` application property:

- Enables the transmission of cookies by using a CMIS to a content management system (CMS).
- Disables the transmission of the global CMS username and password credentials.
- Calls the custom hook point to populate custom cookies.

1.5 CMIS validation error messages

When you use a content management system (CMS) with Cúram, various validation error messages might be displayed.

The following table describes the Content Management Interoperability Service (CMIS) validation error messages.

Validation	When is the validation displayed?
An error occurred while interacting with the Content Management System. Please inform your system administrator.	A generic exception that produces an error from the CMIS infrastructure when there is no associated error handler.
No application property supplied for 'curam.cms.xxxx'. Please contact your administrator.	When the system is storing a file to the CMIS, a required application property is empty.
The application property 'curam.cms.curam.dir' must be in the form '/x/y/z'. Please contact your administrator.	When the system is storing a file to the CMIS, the following list outlines the potential sources of the validation: <ul style="list-style-type: none"> • The path doesn't start with / • The path ends with / • The path contains a folder name with an unsupported character.
The configured Content Management System is unavailable. Please contact your administrator.	When the system is storing a file to the CMIS, a connection to the target system cannot be made. The validation can be triggered by networking issues, by exceeding timeouts, or by incorrect target system settings.
The file was not found on the Content Management System.	When the system is retrieving a file at a specified location from the CMIS within Cúram, the file was deleted directly from the CMS application.
The file is no longer on the Content Management System. Please contact your administrator.	When the system is retrieving a file from CMIS within Social Program Management, the file was deleted directly from the CMS application.
The Content Management System has not been configured. Please contact your administrator.	When the system is accessing a CMS, the CMIS target system is not configured.
Multiple Content Management Systems have been configured. Please contact your administrator.	When the system is accessing a CMS, more than one CMIS target system is configured.
An error occurred gathering information about a file to send to the configured Content Management System. Please inform your system administrator.	When the system is compiling metadata about a file, one or more properties are not configured.
An error occurred gathering information about a file to send to the configured Content Management System. Please inform your system administrator.	When the system is compiling metadata about a file, a required metadata property is not provided.
An error occurred updating a file on the configured Content Management System. Please inform your system administrator.	When the system is trying to modify a file stored in the CMS, no changes are applied as all information is equal to the preexisting data or the file cannot be checked back in.

Validation	When is the validation displayed?
An error occurred reading a file on the configured Content Management System. Please inform your system administrator.	When the system is trying to read a file from the CMS, the file is not the found.
The file '%1s' could not be found within '%2s' on the Content Management System. Please inform your system administrator.	When the system is trying to replace a file on the CMS, the file is not found on the CMS.
The file '%1s' was found in multiple locations within '%2s' on the Content Management System. Modification is not supported in this scenario.	When the system is trying to replace a file on the CMS, more than one location within the root folder is specified.
A target system service already exists for Content Management Interoperability Service.	When a system administrator attempts to add more than one target system service for the Content Management Interoperability Service.

Notices

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

Applicability

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

Personal use

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

Commercial use

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

Rights

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Privacy policy

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

Trademarks

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

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

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

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

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

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

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

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