IBM Cúram Social Program Management
Version 7.0.10

Cúram Content Management Interoperability Services Integration Guide

IBM

Note

Before using this information and the product it supports, read the information in "Notices" on page 17

Edition

This edition applies to IBM® Cúram Social Program Management v7.0.10 and to all subsequent releases unless otherwise indicated in new editions.

Contents

Figures

Tables

Chapter 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 content management system, attachments, Microsoft Word communications, and pro forma communications documents are stored and retrieved from the CMS.

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

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

For more information, see the *Cúram Communications Guide* and the *Configuring Integrated Case Management* related links.

Standards

SPM uses the Content Management Interoperability Services (CMIS) standard. CMIS integration is verified with IBM FileNet P8 v5.5.1. For more information about CMIS standards, see the *OASIS Standard* related link.

Related concepts

Cúram Communications Guide
Cúram Integrated Case Management Guide
Content Management Interoperability Services (CMIS) Version 1.0

Configuring SPM to use with a CMS

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

Registering and adding a CMS to the target system

To permit two-way communication between IBM SPM and a 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.

Service name

Description
Service name is the unique name of the service.

Select **Content Management Interoperability Service over Atompub**.

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 the CMS Atomic 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 the *Customization* related link.

Mandatory
No.

Invoking user password

Description
The CMS Atom 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 the *Customization* related link.

When the target system service is created, the login password is encrypted by IBM SPM encryption algorithms. The plain text password is never stored. IBM SPM 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 *Cryptography in Curam* related link.

Mandatory
No

Note: A configuration is only valid where **Content Management Interoperability Service over Atompub** is selected for one target system.

Related concepts

Customization

Cryptography in Cúram

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 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 proceeding 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 IBM SPM 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 IBM SPM 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 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 the *Customizing Metadata* related link.

Default value
False.

curam.cms.curam.dir

Description
The `curam.cms.curam.dir` property indicates the absolute path to the root directory for IBM SPM content within the global CMS repository that the IBM SPM 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 IBM SPM 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 the *Customization* related link.

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.

Related concepts

Customizing Metadata

Customization

Integrating a CMS with SPM

When integrate with a CMS is enabled, 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 IBM SPM database.

Customizing the CMS

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

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

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

Configuring the CMS

To use the SPM 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

	Type
caseReference	String
communicationDate	Date
documentReceiptDate	Date
documentType	String
documentTypeCode	String
participantDOB	Date
participantFirstName	String
participantLastName	String
participantReference	String
applicationReference	String

Gradual migration

Organizations that are using the SPM application and who are storing files in the application database can enable CMIS integration without performing an upfront migration of the content in the CMS repository.

The documents that are stored in the SPM application database remain there 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 as the 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 proceeding 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 CMS. Existing SPM security mechanisms provide access to documents that are stored in a CMS.

Attachments

Attachment records in the SPM 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 SPM security mechanisms, for example data-based security and location-based security. For more information about data-based and location-based security, see the *Cúram System Configuration Guide* and *Cúram Security Guide* related links. 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.

Related concepts

Configuring the system

Configuring security

Microsoft Word and pro forma communications

For Microsoft Word communications, the Microsoft Word document is stored in the CMS when the communication is created. For pro forma communications, the PDF file that is generated 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.

Metadata

You can store metadata with a document that is stored in the CMS. By default, any CMS that supports 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 but the rest of the document is usually accessible.

Attachment metadata

When CMIS is enabled in IBM SPM, by default metadata is stored for all SPM attachments. You can update custom attachment integration points to use the `CMISAccess` API functions to store and modify extra metadata properties along 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*

Figure 1. Class type tree

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.

Table 2. CMIS metadata descriptions (continued)

Metadata element	Description
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 Default Metadata

To use the metadata properties provided by default when CMIS is enabled, the CMS must be configured with a metadata schema to match the schema implemented on the application side. When a file is sent to the CMS through the CMIS API, contextual information including metadata is collected. The CMIS API does the following:

- Fetches all of the contextual information collected
- Derives whatever metadata it can
- Distinguishes metadata enabled via the system administration screen from other contextual information
- Calls a custom hook point which can wire up or derive additional, custom metadata
- Updates the metadata on the CMS

Configuring Metadata Properties on the CMS

To utilize the metadata support of CMIS, it is necessary to configure the CMS for metadata too. The following properties are required on the `CuramAttachment` document class on the CMS:

Table 3. CMIS Metadata Properties

Property	Type	Length
caseReference	String	40
communicationDate	Date	n/a
documentReceiptDate	Date	n/a
documentType	String	500
documentTypeCode	String	10
participantDOB	Date	n/a
participantFirstName	String	65
participantLastName	String	65
participantReference	String	18

Table 3. CMIS Metadata Properties (continued)

Property	Type	Length
applicationReference	String	10

Configuring Metadata Properties in the Cúram Application

A user with system administrator privileges can manage metadata properties within the system administration application. Each individual property can be enabled or disable and multiple display names and descriptions can be specified for each property to provide multi-language support.

Note: These properties are enabled by default and should be checked, and modified if required, before going live with CMIS.

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

Customization

Overview

This chapter outlines how to customize:

1. Metadata
2. Directory Structures and File Naming Strategies
3. Single Sign On (SSO)

Customizing Metadata

In addition to the metadata properties that are stored by default, custom metadata properties can be added to the documents that are stored on the CMS.

In order to do this:

1. The additional metadata properties must be configured correctly on the CMS.
2. A custom hook point must be implemented and bound on the application side to derive or wire up the additional metadata.
3. Optionally, inputs for the custom hook point may be collected at any point along the business flow.

There are two ways of collecting extra metadata for a document, an explicit method and a transaction scoped method. Using the explicit method, it will be necessary to make modelling changes and to pass the metadata from function to function up to the integration point, where the custom code will add it to the document to be stored. This method makes it easier to follow information flow when reading or for debugging code. The transaction scoped method allows metadata to be collected anywhere along the business flow by using an object injected by Guice. As long as the metadata is collected by the Guice-injected object before the call to the CMIS infrastructure, then it will be available to wire up in the custom hook point. Metadata wired up in the custom hook point will be sent with the document to the CMS.

Configuring Additional Metadata Properties on the CMS

The Document Class on the CMS will need to be configured for the new custom metadata properties that are required. See the "Integrating a Content Management System with Cúram" and "Metadata" chapters for more information on setting up document classes and configuring metadata properties.

Note: Additional metadata properties added will be enabled and will not be configurable by the system administrator. It is assumed that if they are being implemented they are meant to be stored and not disabled.

Note: In the code samples in this chapter, we will show how to add the metadata *String* properties `xyz_contextualReference` and `xyz_updatedBy`, neither of which are included by default out of the box. These properties will need to be configured on the CMS.

Note: Some CMS implementations require recycling in order to begin using the updated metadata schema over CMIS.

Collecting Metadata Transactionally with Guice Injection

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

```
@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());
  ...
}
```

Figure 2. Transactional collection of contextual information or metadata

Note: The name of a custom input to the custom hook point, must, as above, be prefixed by a string such as "xyz_" where "xyz" is a customer specific prefix. Please refer to the *"Use Project-specific Prefixes in Artifact Names" section of the Curam Development Compliancy Guide* for more information.

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

Figure 3. Integration point for transactional method

Adding or Updating Metadata Explicitly

An alternative mechanism is to add the metadata explicitly. This method can be easier to follow the data flow when reading code or when debugging

In the method where the document is created or modified, if there is metadata to be added to the default set, the `CMISAccessImpl.addItemToExplicitMetadata` method should be called. This method takes in three parameters; the list that the metadata is to be added to, 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 this explicit metadata list. This will need to be replicated anywhere that custom metadata is to be added to a document that is being created.

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

Figure 4. Explicit collection of contextual information or metadata

```
// 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);
```

Figure 5. Integration point for explicit method

Wiring up and Deriving Custom Metadata

From the default set of metadata, it is possible to further derive additional custom metadata values.

To create a custom metadata hook, it is necessary to extend `curam.core.sl.infrastructure.cmis.impl.CMSMetadataClassCustomDefaultImpl`. The underlying interface itself is not to be implemented by organizations. This additional extension class must then be bound, using a Guice module, to a codetable code on `CMSMetadataClassType`. The following Guice Module code sample shows how a sample implementation may be bound.

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

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

Figure 6. Sample Guice Module Binding

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

In the implementable functions, *properties* contains the enabled metadata properties and *otherData* contains any additional contextual information and any metadata that has been disabled.

```
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
    ...
  }
}
```

Figure 7. Sample Custom Metadata Collection

Custom Directory Structures and File Naming Strategies

It is possible to customize the way that Cúram organizes and names the files that it stores in the CMS. By default, the following directory structures and file names are used (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` codetable determines where the files are stored on the CMS. Attachments are stored in the Attachments directory. ProFormas 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 (they do not exist in the product ready for immediate use).

To create a custom naming strategy, it is necessary to extend `curam.core.sl.infrastructure.cmis.impl.CMISNamingDefaultImpl`. The CMISNamingInterface itself is not to be implemented by organizations. This additional extension class must then be bound, by using a Guice module, to a codetable code 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);
```

Figure 8. Sample Guice Module Binding

The `CMISNamingInterface.getFilePath` method is where the directory structure and file name are composed. The method returns the filepath as an ordered list of folder names followed by the file name including extension. The wanted directory tree is built by adding the parts of the wanted naming strategy to the list in the correct order. The filename/extension is added last.

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

```
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.TABLENAME, 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;
  }
}
```

Figure 9. Sample Attachment Naming Strategy Implementation

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

The relatedID parameter of `getFilePath` is the identifier for the Cúram object in question on the Cúram database. For example, with the Attachment integration (with the product for immediate use), the related ID would be the attachmentID, and for Communications it would be communicationID.

The relatedData parameter of `getFilePath` is any object that the customer thinks can be useful to its naming strategy implementation. No integrations with the product for immediate use implement this feature but customers are free to use it 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 to allow an authenticated user to access multiple different systems and services without having to log in on each one. Typically, authentication is shared between the systems through a transmission of security tokens.

Implementing SSO with Cúram's CMIS functionality requires custom development in order to integrate with an organization's SSO system. It is also necessary to make some configuration updates to Cúram.

Custom Development

Organizations must first implement a hook point to add custom cookies to the CMIS Atompub invocations performed by Cúram - e.g. custom security tokens used by their SSO implementation. The organization should create a class which extends the `CMISAuthenticationProviderDefaultImpl` class which has the method `getHTTPHeaders`. This additional extension class must then be bound, using a Guice module, to `CMISAuthenticationProviderDefaultImpl`. The following Guice Module code sample shows how a sample implementation may be bound.

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

Figure 10. Sample Guice module linked binding

The following code sample illustrates 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;
  }
}
```

Figure 11. Sample SSO Hook Point

Configuration

Next, it is necessary to enable SSO support. This is administered within the application by use of a system administration application property, `curam.cms.sso.enable`. Enabling this application property effectively means that the CMIS API:

- Enables the transmission of cookies via CMIS to a CMS.
- Disables the transmission of the global CMS username and password credentials.
- Calls the custom hook point to populate custom cookies.

Appendix: Error messages

Validation Error Messages

The following table provides a list of possible validation error messages that may be encountered by a user when a CMS is used with Cúram.

Table 4. CMIS Validation Error Messages.

This table describes the CMIS Validation error messages.

Validation	**When is this displayed?**
An error occurred while interacting with the Content Management System. Please inform your system administrator.	A generic exception thrown from the CMIS infrastructure when there is no associated error handler.
No application property supplied for 'curam.cms.xxxx'. Please contact your administrator.	Upon storing a file to CMIS, when 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.	Upon storing a file to CMIS, when the path doesn't start with a '/', ends with a '/', or contains a folder name with an unsupported character.

Table 4. CMIS Validation Error Messages.

This table describes the CMIS Validation error messages.

(continued)

Validation	**When is this displayed?**
The configured Content Management System is unavailable. Please contact your administrator.	Upon storing a file to CMIS, when a connection to the target system cannot be made. May be triggered by genuine networking issues, by timeouts being exceeded or by incorrect target system settings.
The file was not found on the Content Management System.	Upon retrieving a file at a specified location from CMIS within Cúram which has been deleted directly from the CMS application.
The file is no longer on the Content Management System. Please contact your administrator.	Upon retrieving a file from CMIS within Cúram which has been deleted directly from the CMS application.
The Content Management System has not been configured. Please contact your administrator.	Upon accessing a CMS, when the CMIS ATOM target system has not been configured.
Multiple Content Management Systems have been configured. Please contact your administrator.	Upon accessing a CMS, when more than one CMIS ATOM target system configured.
An error occurred gathering information about a file to send to the configured Content Management System. Please inform your system administrator.	When compiling metadata about a file, one or more properties have not been configured.
An error occurred gathering information about a file to send to the configured Content Management System. Please inform your system administrator.	When compiling metadata about a file, a required metadata property has not been provided.
An error occurred updating a file on the configured Content Management System. Please inform your system administrator.	When attempting to modify a file stored in the CMS, there are no changes to be made as all information is equal to the preexisting data.
	Or, when attempting to modify a file stored in the CMS, the file cannot be checked back in.
An error occurred reading a file on the configured Content Management System. Please inform your system administrator.	When attempting to read a file from the CMS, the file could not be found.
The file '%1s' could not be found within '%2s' on the Content Management System. Please inform your system administrator.	When attempting to replace a file on the CMS, the file could not be 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 attempting to replace a file on the CMS, more than one location within the root folder is specified.

Notices

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

Part Number:

(1P) P/N: